DISS. ETH NO. 30696

ENABLING FAST, ACCURATE, AND EFFICIENT REAL-TIME GENOME ANALYSIS VIA NEW ALGORITHMS AND TECHNIQUES

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES
(Dr. sc. ETH Zurich)

presented by

CAN FIRTINA

B.Sc. and M.Sc. at Bilkent University

born on *19.04.1992*

accepted on the recommendation of

Prof. Dr. Onur Mutlu, examiner

Prof. Dr. Reetuparna Das, co-examiner

Dr. Hasindu Gamaarachchi, co-examiner

Prof. Dr. Benjamin Langmead, co-examiner

Prof. Dr. Heng Li, co-examiner

2024

This page is left blank intentionally.

Acknowledgments

This chapter is left blank intentionally.

Abstract

The advent of high-throughput sequencing (HTS) technologies has revolutionized genome analysis by enabling the rapid and cost-effective sequencing of large genomes. Despite these advancements, the increasing complexity and volume of genomic data present significant challenges related to accuracy, scalability, and computational efficiency. These challenges are mainly due to various forms of unwanted and unhandled variations in sequencing data, collectively referred to as noise. Addressing these challenges requires a deep understanding of different types of noise in genomic data and the development of techniques to mitigate the impact of noise on genome analysis.

In this dissertation, we aim to understand the types of noise that affect the genome analysis pipeline and address challenges posed by such noise by developing new computational techniques to tolerate or reduce noise for faster, more accurate, and scalable analysis of different types of sequencing data (e.g., raw electrical signals from nanopore sequencing).

First, we introduce BLEND, a noise-tolerant hashing mechanism that quickly identifies both exactly matching and highly similar sequences with arbitrary differences using a single lookup of their hash values. Second, to enable scalable and accurate analysis of noisy raw nanopore signals, we propose RawHash, a novel mechanism that effectively reduces noise in raw nanopore signals and enables accurate, real-time analysis using a hash-based similarity search technique. Third, we extend the capabilities of RawHash with RawHash2, an improved mechanism that 1) provides a better understanding of noise in raw nanopore signals to reduce it more effectively and 2) improves the robustness of mapping decisions. Fourth, we explore the broader implications and new applications of raw nanopore signal analysis by introducing Rawsamble, the first mechanism for all-vs-all overlapping of raw signals using hash-based search. Rawsamble enables the construction of *de novo* assemblies directly from raw signals without basecalling, which opens up new directions and uses for raw nanopore signal analysis.

This dissertation builds a comprehensive understanding of how noise in different types of genomic data affects the genome analysis pipeline and provides novel solutions to mitigate the impact of noise. Our findings demonstrate that by effectively tolerating and reducing noise using new computational techniques, we can 1) significantly improve the performance, accuracy, and scalability of genome analysis and 2) expand the scope of raw signal analysis by enabling new applications and directions. We hope and believe that the methods and insights presented in this dissertation will contribute to the invention and development of more robust, efficient, and capable genomic analysis tools, especially in the field of raw signal analysis.

Zusammenfassung

Die Einführung von Hochdurchsatz-Sequenzierungstechnologien (HTS) hat die Genomanalyse revolutioniert, indem sie eine schnelle und kostengünstige Sequenzierung grosser Genome ermöglicht. Trotz dieser Fortschritte stellen die zunehmende Komplexität und das Volumen genomischer Daten erhebliche Herausforderungen hinsichtlich Genauigkeit, Skalierbarkeit und rechnerischer Effizienz dar. Diese Herausforderungen ergeben sich hauptsächlich aus verschiedenen unerwünschten und unbehandelten Variationen in den Sequenzierungsdaten, die zusammenfassend als Rauschen bezeichnet werden. Die Bewältigung dieser Herausforderungen erfordert ein tiefes Verständnis der verschiedenen Arten von Rauschen in genomischen Daten sowie die Entwicklung von Techniken, um die Auswirkungen des Rauschens auf die Genomanalyse zu verringern.

In dieser Dissertation zielen wir darauf ab, die Arten von Rauschen zu verstehen, die die Genomanalyse beeinflussen, und die durch solches Rauschen verursachten Herausforderungen durch die Entwicklung neuer rechnerischer Techniken anzugehen, um das Rauschen zu tolerieren oder zu reduzieren und so eine schnellere, genauere und skalierbare Analyse verschiedener Arten von Sequenzierungsdaten (z. B. rohe elektrische Signale aus der Nanopore-Sequenzierung) zu ermöglichen.

Erstens führen wir BLEND ein, einen rauschresistenten Hashing-Mechanismus, der sowohl exakt übereinstimmende als auch hochgradig ähnliche Sequenzen mit beliebigen Unterschieden schnell durch einen einzigen Abruf ihrer Hash-Werte identifiziert. Zweitens schlagen wir RawHash vor, einen neuartigen Mechanismus, der das Rauschen in rohen Nanopore-Signalen effektiv reduziert und eine genaue Echtzeitanalyse durch eine auf Hash-Werten basierende Ähnlichkeitssuche ermöglicht, um eine skalierbare und genaue Analyse von rauschbehafteten Nanopore-Signalen zu gewährleisten. Drittens erweitern wir die Fähigkeiten von RawHash durch RawHash2, einen verbesserten Mechanismus, der 1) ein besseres Verständnis des Rauschens in rohen Nanopore-Signalen bietet, um es effektiver zu reduzieren, und 2) die Robustheit von Zuordnungsentscheidungen verbessert. Viertens untersuchen wir die breiteren Implikationen und neuen Anwendungen der Analyse von rohen Nanopore-Signalen, indem wir Rawsamble einführen, den ersten Mechanismus für das all-vs-all Überlappen von rohen Signalen unter Verwendung einer auf Hash-Werten basierenden Suche. Rawsamble ermöglicht die Konstruktion von *de novo* Assemblies direkt aus rohen Signalen ohne Basecalling, was neue Richtungen und Anwendungen für die Analyse von rohen Nanopore-Signalen eröffnet.

Diese Dissertation schafft ein umfassendes Verständnis dafür, wie Rauschen in verschiedenen Arten von genomischen Daten die Genomanalyse beeinflusst, und bietet neuartige Lösungen zur Minderung der Auswirkungen von Rauschen. Unsere Ergebnisse zeigen, dass durch die effektive Tolerierung und Reduktion von Rauschen mittels neuer rechnerischer Techniken 1) die Leistung, Genauigkeit und Skalierbarkeit der Genomanalyse erheblich verbessert werden kön-

nen und 2) der Anwendungsbereich der Analyse von rohen Signalen durch die Ermöglichung neuer Anwendungen und Richtungen erweitert werden kann. Wir hoffen und glauben, dass die in dieser Dissertation vorgestellten Methoden und Erkenntnisse zur Entwicklung robusterer, effizienterer und leistungsfähigerer Werkzeuge zur Genomanalyse beitragen werden, insbesondere im Bereich der Analyse von rohen Signalen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Genome analysis plays a crucial role in various fields such as personalized medicine [2–22], genome editing [23–50], evolutionary biology [51–65], cancer research [66–112], prenatal and neonatal screening [113–137], outbreak tracing [138–157], microbiome studies [158–182], agriculture [183–195], and forensics [196–214]. The advent of high-throughput sequencing (HTS) technologies, such as sequencing-by-synthesis (SBS) [215–223], Single Molecule Real-Time (SMRT) [224], and nanopore sequencing [225–247], has revolutionized genome analysis, enabling faster and more cost-effective sequencing of genomes by generating large amounts of genomic data at relatively low cost [248] compared to Sanger sequencing [249]. However, the analysis of genomic data is challenging due to a variety of reasons: 1) HTS technologies can only sequence relatively short fragments of genomes, called *reads*, whose locations in their corresponding genome are unknown [250–254], 2) these reads can contain sequencing errors [248, 252, 255–257], leading to differences from their original sequences, 3) the sequenced genome may not (and usually does not) exactly match recorded genomes in a reference database, known as *reference genomes*, due to variations between individuals within and across species [255, 258]. Despite significant improvements in computational tools since the 1980s [258] to overcome such challenges, the rapid growth in genomic data [259] has led to ever larger computational overheads in the genome analysis pipeline, posing large challenges for efficient, accurate and timely analysis of genomes [260–262].

Figure 1.1 shows multiple key steps of a genome analysis pipeline, each of which affects the accuracy, speed, and energy consumption of genome analysis.

First, a biological molecule (e.g., a DNA molecule [263]) is sequenced using an HTS technology to generate the raw sequencing data (e.g., measured raw electrical signals in nanopore sequencing) ❶. Among HTS technologies, nanopore sequencing uniquely allows for significant reductions in genome analysis time and cost by enabling early termination of sequencing,

either for individual reads or the entire run, based on real-time data analysis. This capability is known as *adaptive sampling* [264]. Such an analysis needs to be quick and, ideally, should be made as soon as the raw signal data is generated by the sequencer. Analyzing raw nanopore signals during sequencing at a computational speed that matches or exceeds the sequencer's data generation speed (i.e., throughput) is known as *real-time analysis* and is essential for applications such as adaptive sampling. Real-time analysis is mainly useful for 1) reducing the genome analysis latency by overlapping the sequencing with analysis, especially needed for urgent cases that require rapid analysis (e.g., neonatal screening [137]), and 2) reducing the sequencing time and cost by reducing unnecessary sequencing [265]. However, real-time analysis introduces unique challenges, particularly in meeting the stringent throughput and latency requirements demanded by nanopore sequencing. The analysis speed must keep pace with data generation to avoid bottlenecks, which is often difficult due to the large data volumes and the need for real-time decision-making during sequencing [266]. Section 2.1 provides detailed background on sequencing technologies.

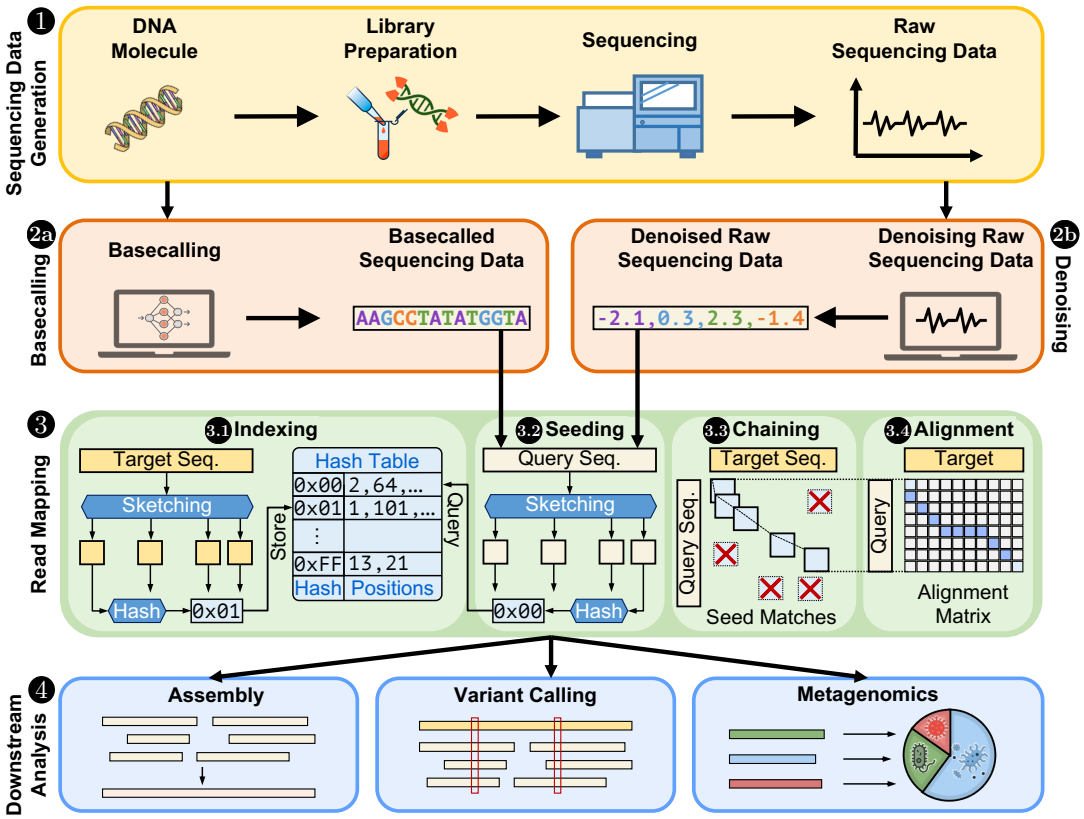

Figure 1.1: Key steps in the genome analysis pipeline.

Second, raw sequencing data is typically translated into sequences of nucleotide characters or *bases* (e.g., A, C, G, and Ts in DNA) via *basecalling* (2a). Basecalling tools [267–288] mainly rely on compute-intensive approaches that process large chunks of *noisy* and error-prone

raw data to accurately infer the actual nucleotide sequences [261, 287, 289]. Alternatively 2b, *raw sequencing data* can directly be analyzed *without* basecalling [138, 264–266, 290–302]. While direct analysis of raw data can avoid the computational overhead of basecalling, it introduces challenges due to increased noise, requiring specialized techniques for accurate denoising. These denoising techniques usually include time series analysis [286, 291, 303] and quantization [265, 266, 299].

Third, read mapping 3 aims to find similarities and differences between genomic sequence pairs (e.g., between sequenced *query* reads and *target* reference genomes of one or more species). To facilitate practical similarity identification, read mapping includes several steps. These steps include constructing (i.e., indexing 3.1) and using (i.e., seeding 3.2) a database [2, 265, 266, 292, 299, 304–411] for efficient similarity search by utilizing various sketching (i.e., sampling) [306, 310, 333, 360, 412–432] and hashing [304, 307–309, 315, 334, 358, 414, 416, 424, 433–459] methods. Filtering [357, 460–468] and co-linear chaining (i.e., sparse dynamic programming) [306, 469–482] steps 3.3 in read mapping aim to reduce the workload of the next computationally costly steps in read mapping by quickly identifying highly dissimilar or similar regions between query and target sequences. The alignment [483–501] step 3.4 identifies the exact differences and similarities between query and target sequences, which overall demand considerable processing power and memory due to the large scale of genomic sequences [258, 502, 503]. Section 2.2 provides detailed background on the keys steps in read mapping.

Fourth, the output generated in the read mapping step can be used in the subsequent steps of genome analysis (i.e., *downstream analysis* 4), such as *de novo* genome assembly (i.e., construction of a genome from scratch) [250, 251, 305, 504–532], variant calling (i.e., identifying genetic variations in an individual's genome compared to a reference genome) [533–570], and metagenomics (i.e., identifying and profiling organisms present in an environment) [571–591]. Downstream analysis often requires additional computationally intensive steps [261], including set intersection [592], graph processing [305,593], and the use of deep neural networks (DNNs) [540]. These additional steps further contribute to the overall computational overhead and energy consumption of the genome analysis pipeline [261]. Section 2.3 provides detailed background on key steps in the downstream genome analysis pipeline after read mapping.

Many algorithmic, software, and hardware techniques aim to address the computational challenges in the genome analysis pipeline. These works improve the performance and accuracy of the computational tools by 1) reducing overall computational and space complexity [266,483,485, 535,540], 2) eliminating useless work [266,267,275,275,299,306,357,460–468,593], 3) optimizing data structures and memory access patterns [324, 594–598], 4) exploiting parallelism and distributed computing in multi-core, many-core, and SIMD architectures [300,301,334,462,484,594, 596,599,599–648,648–660], and 5) designing specialized hardware accelerators for many steps in

genome analysis [138,195,262,267–284,297,300–302,461,463–467,484,490,593,594,596,598–779]. We provide a detailed description of these approaches in Sections 2.4 and 3.

1.1 Problem Discussion

Unfortunately, the increasing complexity of genomic data analysis introduces significant challenges in terms of accuracy, scalability, and computational efficiency. These challenges are mainly due to the heuristic techniques that need to analyze large volumes of sequencing data in the presence of various forms of *unwanted* and *unhandled* variations. In this dissertation, we refer to these variations collectively as noise. Noise can originate from sequencing errors, biological variations, or technical limitations in both the sequencing and analysis processes, which substantially impacts the heuristic decisions and sensitivity of genome analysis.

Noise in Seed Matching. A critical step in genome analysis is seed matching [780], where short sequence segments, called *seeds*, are identified between biological sequences (e.g., DNA reads) to find similarities. Traditional hashing methods used in seed matching [358, 440, 442–454] require *exact* matches of these seeds. The requirement for exact matching necessitates the use of limited parameter choices, which either increases the number of computationally costly steps or reduces the sensitivity in detecting relevant genomic regions.

Although identifying exactly matching seeds is crucial for finding similarities, seeds may still have arbitrary differences—what we define as *noise*—due to sequencing errors or biological variations. Conventional methods either 1) fail to detect these near-matches when the differences occur at arbitrary positions or 2) resort to computationally expensive steps to identify these differences. The lack of a mechanism that can efficiently *tolerate noise* during a single hash value lookup limits the performance, accuracy, and efficiency of genome analysis, as we demonstrate in Chapter 4.

Noise in Raw Nanopore Signals. Nanopore sequencing offers unique opportunities for genome analysis, particularly with its capability to stop the sequencing of a read or the entire sequencing run based on real-time analysis. However, the raw electrical signals generated by nanopore sequencers are inherently noisy [781–785]. Traditional approaches to handling this noise rely on computationally expensive basecalling that usually utilize GPUs [267, 268, 270, 272–284, 681, 721]. Although various works that analyze raw nanopore signals without basecalling [138, 264–266, 290–302] can provide better scalability and efficiency, many of these techniques fall short in speed and accuracy when analyzing larger genomes [266], such as human genomes. This is because these works handle noise in raw nanopore signals ineffectively (as shown in Chapters 5 and 6). *Understanding and reducing noise effectively* in raw nanopore signals provides the opportunity for enabling scalable and accurate analysis of raw nanopore signals without basecalling.

Limited Scope of Applications for Raw Nanopore Signal Analysis. Raw nanopore signals can be analyzed without the intermediate basecalling step; however, the scope of new applications, such as constructing genomes from scratch using raw nanopore signals, remains limited. This is mainly due to the lack of techniques that can enable new uses and directions in raw nanopore signal analysis, particularly in the absence of a reference genome (as Chapter 7 shows). The full potential of raw nanopore signal analysis remains largely untapped due to limited techniques designed for analyzing raw nanopore signals.

Impact on Genome Analysis Pipeline. The sources of noise discussed above lead to significant computational overhead and limit the applicability of current genome analysis methods (as Chapters 4, 5, 6 demonstrate). There is a pressing need for the development of new techniques that can effectively tolerate or reduce noise throughout the genome analysis pipeline. Such techniques can enable more accurate, scalable, and real-time genome analysis, opening up new directions and applications in the field (as we show in Chapters 7 and 8).

1.2 Goal

In this dissertation, our goal is to 1) *understand* how certain types of noise in sequencing data and its analysis impact the performance, accuracy, and scalability of the analysis and 2) build mechanisms to tolerate or reduce this noise for faster, scalable, more accurate, and real-time analysis of genomic sequencing data.

1.3 Thesis Statement

Our approach is encompassed by the following thesis statement:

> *The imperfections (i.e., noise) in DNA sequencing data and its analysis can be mitigated by*
> *1) building a better understanding of the types of noise, and*
> *2) developing new algorithms and techniques that can tolerate and reduce noise,*
> *thereby providing accurate, scalable, and real-time analysis of sequencing data and enabling new applications in genome analysis.*

1.4 Our Approach

To identify and address the challenges introduced by imperfections in genomic sequencing data and its analysis, we pursue three main strategies: 1) we integrate a noise-tolerant hashing mechanism in seeding to quickly identify highly similar seeds with arbitrary differences between them; 2) we build a detailed understanding of the variations in raw nanopore signals and reduce noise caused by these variations to enable hash-based search for raw nanopore signals; and

3) we enable new applications and directions by providing new overlap detection and assembly mechanisms for raw nanopore signals.

1.4.1 Tolerating Arbitrary Noise with Hash-based Fuzzy Seeding

Generating the hash values of seeds enables quickly identifying similarities between genomic sequences by matching seeds with a single lookup of their hash values. However, these hash values can be used only for finding exactly matching seeds, as conventional hashing methods [358, 440, 442–454] assign distinct hash values for different seeds, including highly similar seeds. Due to the limited parameter choices when finding only exactly matching seeds, the selected seeds either 1) increase the use of the computationally costly steps after seeding or 2) provide limited sensitivity.

To address these challenges, we introduce *BLEND*, the first mechanism that can identify *both* exactly matching and highly similar seeds with arbitrary differences (i.e., noise) using a single lookup of their hash values, called *fuzzy seeds*. To achieve this, BLEND 1) utilizes SimHash [433, 434], which can generate identical hash values for a similar list of values (i.e., vectors), and 2) introduces mechanisms that treat seeds as vectors, allowing SimHash to efficiently identify fuzzy seed matches.

We show the benefits of BLEND when it is used in read overlapping and read mapping. For read overlapping, BLEND is faster by 2.4×-83.9× (on average 19.3×), has a lower memory footprint by 0.9×-14.1× (on average 3.8×), and finds higher quality overlaps leading to accurate *de novo* assemblies than the state-of-the-art tool, minimap2. For read mapping, BLEND is faster by 0.8×-4.1× (on average 1.7×) than minimap2.

1.4.2 Enabling Hash-based Search of Raw Nanopore Signals by Reducing Noise

A unique and important feature of nanopore sequencing, adaptive sampling [264], can eject single DNA or RNA molecules (i.e., strands) from sequencers without fully sequencing them, which provides opportunities to computationally reduce the sequencing time and cost. However, due to the inherent noise in raw electrical signals, existing works analyzing these raw signals [138, 264–266, 290–302, 369, 370, 680, 786–790] either 1) require powerful computational resources such as GPUs to eliminate this noise by converting the signals to DNA bases [369, 370, 680, 786–790] or 2) lack scalability for large genomes due to their mechanisms when analyzing noisy raw signals without converting them bases [291, 292].

To understand and reduce noise in raw nanopore signals and to enable efficient analysis of these signals, we propose RawHash, the first mechanism that can accurately and efficiently perform real-time analysis of nanopore raw signals for large genomes using hash-based similarity

search. To enable this, RawHash provides accurate hash-based similarity search via effective noise reduction with quantization of the raw signals such that signals corresponding to the same genomic content can have the same quantized value and, subsequently, the same hash value.

We evaluate RawHash on three applications: 1) read mapping, 2) relative abundance estimation, and 3) contamination analysis. When compared to the state-of-the-art techniques, UNCALLED [292] and Sigmap [291], RawHash provides 1) 25.8× and 3.4× better average throughput and 2) significantly better accuracy for large genomes, respectively. Our evaluations show that RawHash is the only tool that can provide high accuracy and high throughput for analyzing large genomes in real-time compared to these prior state-of-the-art works [291, 292].

1.4.3 Better Noise Reduction for and Robust Real-Time Mapping of Raw Nanopore Signals

Real-time analysis of raw signals is essential to utilizing the unique features that nanopore sequencing provides, enabling the early stopping of the sequencing of a read or the entire sequencing run based on the analysis. The state-of-the-art mechanism we introduce, RawHash, offers the first hash-based efficient similarity identification between raw signals and a reference genome by effectively reducing noise in raw signals to enable quick matching between hash values.

To build a better understanding of noise and better mechanisms for reducing noise in raw nanopore signals, we introduce RawHash2, which provides major improvements over RawHash in several ways. First, to provide more sensitive noise reduction in raw signals, RawHash2 provides a new adaptive noise reduction technique based on the characteristics of raw nanopore signals rather than a fixed method (as proposed in RawHash) for reducing noise. Second, to perform similarity search with fewer and more useful seeds (in order to improve both performance and accuracy), RawHash2 1) provides a sampling mechanism, known as *minimizer sketching*, to reduce the volume of raw signals used in analysis, 2) provides a filter to reduce ambiguous matching seeds before chaining, 3) improves the sensitivity of the chaining algorithm, and 4) identifies the best mapping for a read based on a weighted decision mechanism that takes several metrics into account instead of only one (as in RawHash).

We evaluate RawHash2 in a similar evaluation setup we use for RawHash. RawHash2 provides better accuracy (on average by 10.57% and up to 20.25%) and better throughput (on average by 4.0× and up to 9.9×) than RawHash, while providing even more substantial performance and accuracy improvements compared to two other existing works, UNCALLED [292] (e.g., 26.5× better throughput on average) and Sigmap [291] (e.g., 19.2× better throughput on average).

1.4.4 Enabling New Applications by Overlapping and Assembling Raw Nanopore Signals

Existing works that directly analyze raw signals without basecalling cannot interpret raw signals directly if a reference genome is unknown due to several major challenges. These challenges mainly stem from a lack of accurate mechanisms to handle increased noise and data volume in pairwise raw signal comparison. Our goal is to enable the direct analysis of raw signals *without* a reference genome. To this end, we propose Rawsamble, the *first* mechanism that can 1) identify regions of similarity between all raw signal pairs, known as *all-vs-all overlapping*, using a hash-based search mechanism and 2) use these to construct genomes from scratch, called *de novo* assembly. To enable overlapping and assembly of signals, Rawsamble provides 1) an aggressive filtering mechanism to minimize the increased noise caused by overlapping two raw signals, 2) adjusts the chaining and outputting strategies to enable the generation of a large number of useful overlapping read pairs.

Our extensive evaluations across multiple genomes of varying sizes show that Rawsamble provides a significant speedup (on average by 16.36× and up to 41.59×) and reduces peak memory usage (on average by 11.73× and up to 41.99×) compared to a conventional genome assembly pipeline using the state-of-the-art tools for basecalling (Dorado [272]) and overlapping (minimap2 [306]) on a CPU. We find that 36.57% of overlapping pairs generated by Rawsamble are identical to those generated by minimap2. Using the overlaps from Rawsamble, we construct the first *de novo* assemblies *ever* constructed directly from raw signals without basecalling. We show that we can construct contiguous assembly segments (i.e., unitigs [504]) up to 2.7 million bases in length (half the genome length of *E. coli*). We identify other previously unexplored directions that can be enabled by finding overlaps and constructing *de novo* assemblies.

1.5 Contributions

This dissertation makes the following contributions:

1. We provide a detailed review and overview of the state-of-the-art in raw signal analysis, covering a comprehensive spectrum of existing methodologies and their limitations. We systematically analyze various methods for managing noise, computational efficiency, and accuracy in the analysis of raw nanopore signals. This thorough review serves as a foundational framework for our subsequent contributions and sets the stage for the novel techniques proposed in this dissertation. Our review also includes acceleration methods in genome analysis and key aspects of downstream analysis. Chapters 2 and 3 provide this comprehensive review.

2. We develop a detailed understanding of how noise in genomic sequencing data impacts computational mechanisms, hindering improvements in speed, accuracy, efficiency, and applicability in genome analysis. Based on our understanding of the sources and types of noise, we provide mechanisms to better handle noise in sequencing data, which enables new directions with faster, more accurate, and scalable techniques. Chapters 4-7 describe how we build this understanding and the resulting mechanisms.

3. We introduce BLEND, to enable quick identification of highly similar genomic sequences. BLEND is a novel mechanism that can quickly and efficiently find highly-similar seed matches between sequences with a single lookup of their hash values by designing a noise-tolerant hashing mechanism for fuzzy seed matching. Chapter 4 describes BLEND and its evaluations in detail. BLEND is open source [791]. A paper describing BLEND was published in the Nucleic Acids Research (NAR) Genomics and Bioinformatics journal in [334] and presented in the highlights track of the RECOMB 2023 conference [792].

4. We introduce RawHash, the first mechanism that, without basecalling, can accurately and quickly map raw nanopore signals to large genomes (e.g., a human genome) using 1) a novel quantization (i.e., bucketing) technique for reducing noise in raw signals and 2) a hash-based search mechanism. RawHash provides a fast solution that can match the throughput of a nanopore sequencer to perform analysis during sequencing (i.e., real-time analysis). Chapter 5 describes RawHash and its evaluations in detail. RawHash is open source [793]. RawHash was presented at the ISMB/ECCB 2023 conference [794], and a paper describing RawHash was published in the Bioinformatics journal [266] as part of the conference proceedings.

5. We propose *Sequence Until*, the first mechanism that can stop the entire sequencing run for relative abundance estimation when an accurate estimation can be made without sequencing the entire sample. *Sequence Until* provides opportunities for mechanisms that can perform real-time analysis, such as RawHash, to substantially reduce the sequencing time and costs without sacrificing accuracy. Chapter 5 describes *Sequence Until* and its evaluations in detail. *Sequence Until* is open source [793]. *Sequence Until* was presented at the ISMB/ECCB 2023 conference [794], and a paper describing *Sequence Until* was published in the Bioinformatics journal [266] as part of the conference proceedings.

6. We introduce RawHash2, an improved mechanism for 1) better reducing noise in raw nanopore signals with an adaptive noise reduction technique designed based on the characteristics of raw nanopore signals and 2) improving the seed matching and mapping mechanisms to achieve faster and more accurate real-time mapping of raw nanopore

signals to reference genomes. Chapter 6 describes RawHash2 and its evaluations in detail. RawHash2 is open source [793]. A paper describing RawHash2 was published in the Bioinformatics journal [265].

7. We propose Rawsamble, the first mechanism that can find all-vs-all overlapping of raw signals to enable new use cases for raw signal analysis, such as assembling raw signals without basecalling. Using the overlaps that Rawsamble generates, we construct the first *de novo* assemblies from raw signals without basecalling. Chapter 7 describes Rawsamble and its evaluations in detail. Rawsamble is open source [793]. Rawsamble was presented in the HitSeq track of the ISMB 2024 conference [795]. A preprint describing Rawsamble is available [296].

8. We describe the remaining challenges in overcoming the problems to enable new applications in raw signal and real-time analysis and discuss new directions enabled by the contributions described in this thesis. Chapter 8 discusses these challenges and directions, including the challenges for constructing *de novo* assemblies in real-time.

1.6 Outline

This dissertation is organized into 8 chapters. Chapter 2 gives relevant background information about sequencing technologies, the challenges, and steps to analyze the noisy sequencing data. Chapter 3 discusses related works that address the relevant problems in tolerating noise when mapping basecalled sequences and the works that aim to perform real-time analysis with or without basecalling the sequencing data. Chapter 4 introduces BLEND and its evaluations. Chapter 5 introduces RawHash and *Sequence Until* and their respective evaluations. Chapter 6 introduces RawHash2 and its evaluations. Chapter 7 proposes Rawsamble and provides its evaluations. Chapter 8 provides a summary of this dissertation as well as future research directions and concluding remarks.

Chapter 2

Background

This chapter provides an overview of the background material necessary to understand our discussions, analyses, and contributions. Section 2.1 describes the main sequencing technologies, real-time analysis, and the challenges in analyzing sequencing data. Section 2.2 provides the main steps taken to provide a practical sequence analysis. Section 2.3 describes the subsequent steps that can utilize the output from read mapping to perform genome analysis. Section 2.4 describes the software and hardware approaches for accelerating genome analysis. Section 2.5 provides a summary and suggests further reading for interested readers.

2.1 High-Throughput Sequencing Technologies

High-throughput sequencing (HTS) technologies produce raw sequencing data, the content of which depends on the type of sequencing technology. Figure 2.1 shows the flow of main steps for generating the sequencing data from HTS technologies. There are three main types of HTS technologies: sequencing by synthesis (SBS) [215–223], Single Molecule Real-Time (SMRT) [224], and nanopore sequencing [225–247].

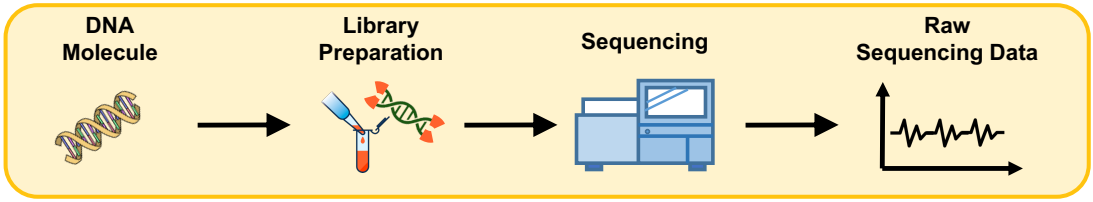

Figure 2.1: Main steps in sequencing data generation.

2.1.1 Sequencing by Synthesis (SBS)

Sequencing by synthesis (SBS) [215–223] is a widely adopted high-throughput sequencing technology that enables accurate and cost-effective sequencing of nucleic acid molecules. The SBS process involves several steps to determine the nucleotide sequences [219].

First, the nucleic acid molecule (e.g., DNA) is fragmented into smaller pieces, called *reads*, each around a few hundred nucleotides long (e.g., approximately 200 bases). To enable the attachment of these reads to a specialized glass slide known as a *flow cell* for sequencing, certain short nucleic acid molecules known as *adapters* are added to molecules' ends. Second, to increase the redundancy for improved reliability during sequencing, clusters of identical DNA fragments on the flow cell are created with amplification. Third, to sequence DNA fragments, a complementary DNA strand from a read is synthesized base by base. To do so, an enzyme called *DNA polymerase* adds a single fluorescently labeled nucleotide to the growing DNA strand in each sequencing *cycle*. To allow for the identification of each individual nucleotide, these labeled nucleotides emit a specific light at a certain frequency (i.e., a different light color for each nucleotide) when incorporated into the strand. Fourth, after each nucleotide is added, the flow cell is imaged to capture the fluorescent signal emitted by the incorporated nucleotide. This nucleotide incorporation and imaging cycle is repeated for each base in the DNA fragment. The sequence of the entire DNA fragment is determined base by base as the fluorescent signals are recorded across multiple cycles.

SBS generates a series of images as *raw data*, where each image corresponds to a single cycle of synthesis, and the color intensity at a specific position in the image represents a particular nucleotide. Translating the raw data into nucleotide sequences is called *basecalling*. Basecalling tools [796–811] developed for SBS aim to accurately associate these colors with their corresponding bases while correcting sequencing errors [812]. Although SBS generates highly accurate raw sequencing data, it also introduces certain limitations, primarily due to the cycling procedure. Specifically, SBS generates *short reads*, typically a few hundred bases long, which can complicate downstream analysis, especially when assembling large genomes or resolving complex genomic regions [252, 813].

2.1.2 Single Molecule Real-Time (SMRT) Sequencing

Single-molecule real-time (SMRT) sequencing is a high-throughput sequencing technology that enables direct observation of nucleic acid molecule synthesis without sequencing each nucleotide cycle by cycle (i.e., in real-time) [224]. Since the short read length limitations introduced by cycling procedures in SBS are not a factor in SMRT sequencing, longer reads up to a few thousand bases can be generated [814], which is crucial for resolving complex genomic regions [814]. SMRT sequencing mainly works in several steps [224].

First, similar to SBS, the nucleic acid molecule (e.g., DNA) is prepared by fragmenting it into smaller pieces (i.e., reads). These fragmented molecules are then circularized by adding special adapters to their ends. These adapters include hairpin loops that allow the DNA to form a closed circular molecule, which is essential for the continuous sequencing process in SMRT.

Second, SMRT sequencing introduces the circularized fragments into specialized wells, known as Zero-Mode Waveguides (ZMWs) [815]. Each ZMW contains a single DNA polymerase enzyme that initiates the synthesis of a complementary strand of DNA using the circularized DNA template.

Third, unlike SBS, which involves discrete sequencing cycles, SMRT sequencing operates continuously in real-time. As the DNA polymerase incorporates fluorescently labeled nucleotides into the growing DNA strand, each incorporation event is detected immediately by exciting the fluorescent label with a laser. The emitted light is captured by a highly sensitive detector, and this process occurs without interruption, producing a continuous stream of data in the form of a movie. Each of the four nucleotides (A, C, G, T) is labeled with a distinct fluorescent dye, allowing for their identification based on the emitted color.

Fourth, SMRT sequencing can be performed in different modes depending on the application. One common mode is Circular Consensus Sequencing (CCS) [224, 816], where the polymerase repeatedly sequences the same DNA fragment as it loops around the circular template. This approach generates multiple reads of the same fragment, which are combined to produce a highly accurate consensus sequence. Alternatively, in the Continuous Long Read (CLR) mode [817], the polymerase reads the DNA fragment continuously without repeated sequencing, which is useful for generating longer reads but may have lower accuracy due to the absence of the error correction provided by CCS.

SMRT sequencing generates a continuous series of images in a movie format while capturing the fluorescent signals in real-time. Although these images can be quickly converted into corresponding nucleotide sequences, the noise inherent in SMRT sequencing requires additional steps to correct sequencing errors [256, 818]. These steps include sequence alignment [306], consensus assembly construction [818], and assembly polishing [257, 819], which are crucial for ensuring the high accuracy of the final sequences, particularly in applications that benefit from the long-read capability of SMRT sequencing.

2.1.3 Nanopore Sequencing

Nanopore sequencing is a high-throughput sequencing technology that enables the sequencing of nucleic acid molecules (e.g., double-stranded DNA, *dsDNA*) as they pass through tiny nanoscale pores called *nanopores* [225–247], as shown in Figure 2.2. These pores are embedded in a *membrane* where an electric potential is applied to create a voltage difference between the two sides of the membrane. To facilitate the passing of nucleic acid molecules through the pores, the voltage is applied in a specific direction such that the upper part of the membrane (i.e., *cis* side) is negatively charged, and the lower part (i.e., *trans* side) is positively charged. Since DNA and RNA molecules are negatively charged, they are naturally driven through

the nanopore from the cis side to the trans side by this electric potential while disrupting *ionic current* flow around the nanopore. Due to the nature of its sequencing method, nanopore sequencing enables sequencing *ultra long* reads up to a few million bases [820], which is a unique feature compared to SBS and SMRT sequencing technologies. Figure 2.2 shows the three main steps in nanopore sequencing [1]: 1) guiding the movement of a nucleic acid molecule through the nanopore, 2) disrupting the ionic current flow, and 3) generating electrical signals from these disruptions.

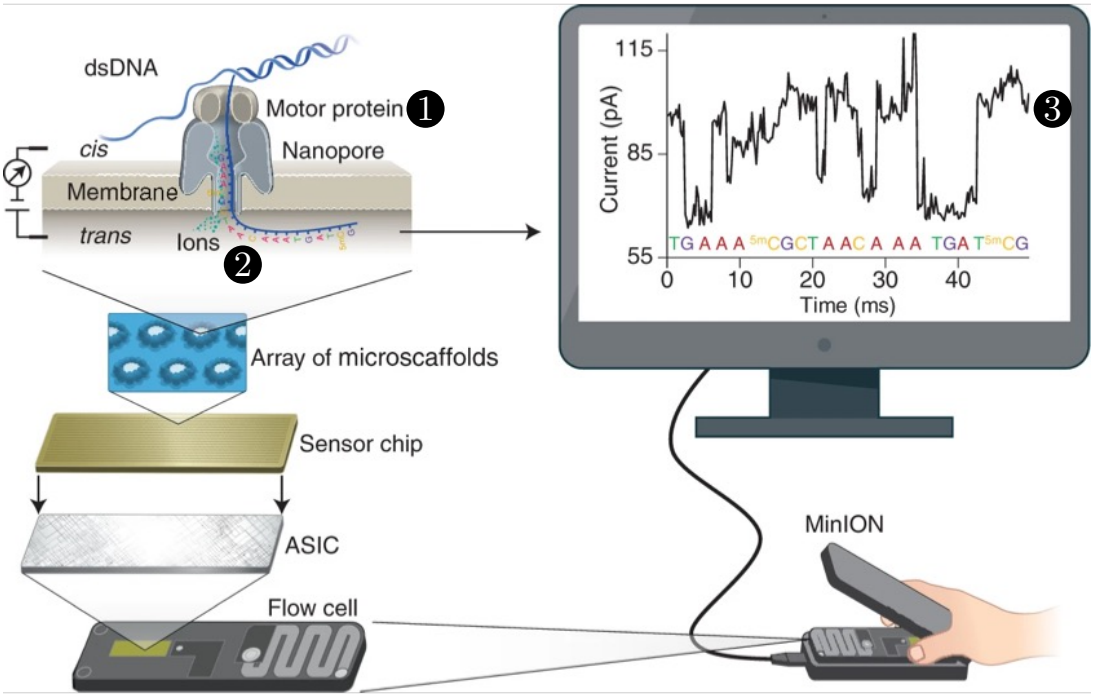

Figure 2.2: Structure of a nanopore sequencer and its sequencing steps. Image taken from [1].

First, nucleic acid molecules inserted into the flow cell of a nanopore sequencer are guided through the nanopore by a particular enzyme called a *motor protein* ❶. This motor protein attaches to one end of the molecule and controls its movement and speed through the nanopore, ensuring that the molecule is translocated at a controlled speed. Controlling the *translocation speed* is crucial, as it improves the accuracy and resolution of the sequencing process, which was a significant challenge in the early development of nanopore sequencing in the late 1980s [229].

Second, as the nucleic acid molecule passes through the nanopore, it partially obstructs the flow of ions around the nanopore ❷, causing characteristic changes in the ionic current. These changes are highly sensitive to the specific sequence of k nucleotides, called *k-mers*, passing through the nanopore, allowing the detection of the nucleotide sequence as the molecule translocates (i.e., moves) through the pore via sensing regions (i.e., narrow heads inside nanopores, also called reader heads) located near these k-mers [1,229]. The number (k) of nucleotides disrupting

the ionic current measurements is usually specific to the design nanopore (i.e., chemistry). This number k is usually between 6 and 9 [229, 821], determined based on the characteristics and the number of sensing regions of nanopores [1, 229]. However, several sources of variance, or *noise* [822], can affect the accuracy of these measurements. These sources include stochastic fluctuations in the ionic current [229], variations in the speed of the molecule as it translocates through the pore [823], and environmental factors such as temperature [824].

Third, from each nanopore, raw electrical signals are generated at a certain frequency (around 5,000 signals per second [247]) ❸. A high-throughput nanopore sequencer usually allows many reads to be sequenced simultaneously across multiple pores in the sequencer's flow cell [1]. These electrical signals contain the characteristics and content of the nucleic acid molecules passing through the pores.

Figure 2.3 shows the two subsequent steps that use *raw nanopore signals*: 1) *basecalling*, which translates these raw signals into nucleotide sequences (A, C, G, T), and 2) *raw signal analysis*, where the raw nanopore signals are analyzed without basecalling.

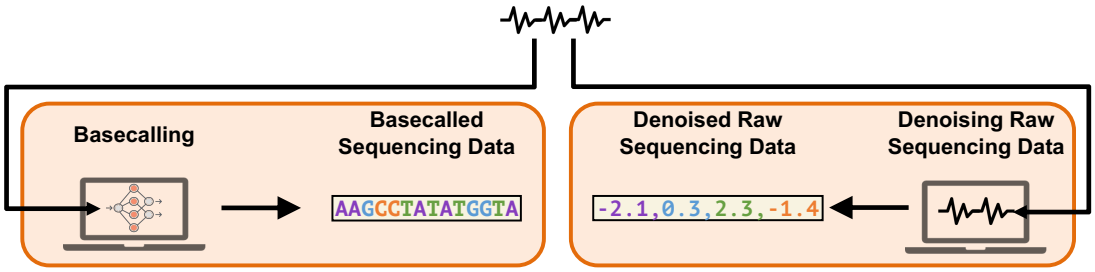

Figure 2.3: Main steps for processing raw sequencing data.

2.1.3.1 Basecalling

Many basecallers have been proposed specifically for nanopore sequencing [267–283, 663, 669–671, 680–684, 692, 711, 721, 722] , as opposed to basecallers designed for other sequencing technologies [796–811], due to two main reasons. First, the abundance of information and noise in raw signals makes the basecalling task more challenging compared to the relatively simpler and more direct signal-to-base conversion in SBS and SMRT sequencing methods [261, 275]. Second, nanopore sequencing provides unique opportunities that can be used to reduce the time and cost of sequence analysis [261, 266], which we explain in Section 2.1.3.3.

To translate the raw nanopore signals to sequences of nucleotide characters, modern basecalling techniques utilize deep learning models, i.e., deep neural networks (DNNs) [267–284], to improve the precision of identifying a nucleotide base from raw signals compared to traditional non-deep learning-based basecallers [285–288]. Deep learning models can accurately basecall raw signals because their architectures have advanced to effectively model and recognize spatial characteristics in the raw signal data [825].

Deep learning models used in basecalling generally consist of convolutional neural networks (CNNs) [284] and recurrent neural networks (RNNs) [276], or a combination thereof [272, 681]. These models can capture the spatial and temporal dependencies in the signal data in two ways. First, CNNs process the raw signal input to extract features indicative of the underlying nucleotide sequence. This feature extraction step is crucial, as it converts raw electrical signals into a structured format that deep learning models can use to predict the nucleotide sequence. Second, many basecalling models employ RNNs, specifically a bidirectional LSTM network [272, 681], to predict the sequence of nucleotides. The bidirectional approach allows the model to use the temporal context from signals to improve accuracy in noisy conditions such as variance due to translocation speed. This is particularly important in nanopore sequencing, where the signal may be affected by various sources of noise, such as stochastic fluctuations in the ionic current and variations in the speed of the molecule as it translocates through the pore.

To assign bases to a window of signals where the window length can be variable due to the varying translocation speed, several basecalling models use *decoder* mechanisms, usually in the form of a Connectionist Temporal Classification (CTC) layer [826], Conditional Random Field (CRF) decoders [827], or a combination of both [681, 825]. This flexibility in the window length is essential for identifying the variable lengths of signal segments that correspond to a particular nucleotide. These variabilities arise due to fluctuations in the ionic current and changes in the translocation speed of the molecule through the nanopore, as discussed in Section 2.1.3.

The complexity of deep neural networks for basecalling [275] often necessitates powerful computational resources, such as GPUs [267, 268, 270, 272–284, 681, 721] or custom hardware designs [269, 669–671, 683, 692], to perform the required computations quickly in a massively parallel manner. Such a large amount of computation is usually costly in terms of power and latency, mainly caused by data movements [711, 828, 829] and a large number of operations [275], which leads to challenges in using them in computationally (or power-wise) resource-constrained environments such as for in-field analysis with mobile devices [297].

2.1.3.2 Raw Signal Analysis

Nanopore sequencing generates raw signal data in the form of time series [781], which reflect the changes in ionic current as nucleic acids translocate through the nanopore. Analyzing these time series data effectively addresses the variable translocation speed of the nucleic acid molecules, a critical aspect that influences signal noise. Figure 2.4 shows several steps for processing raw nanopore signals to reduce certain types of noise in raw nanopore signals.

First, to reduce noise caused by variable translocation speed during sequencing, the raw signal data is segmented using time-series statistical analysis [266, 291, 830]. Segmentation ❶

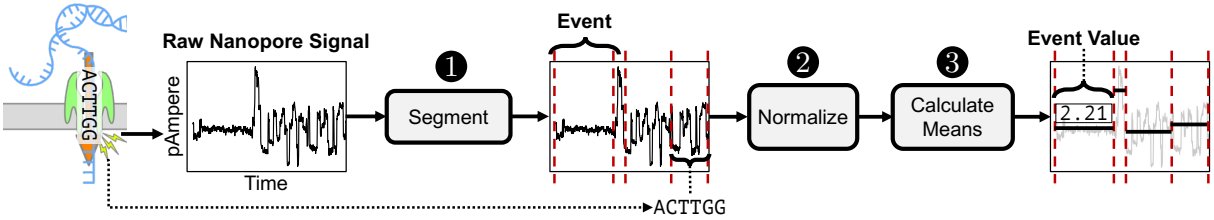

Figure 2.4: Main steps for raw signal analysis.

involves dividing the continuous signal into discrete parts or segments, called *events*, based on detected abrupt changes in the signal intensity and pattern. To identify these abrupt changes, statistical methods such as the t-test [303] are typically used. Each event is assumed to correspond to the sequencing of a specific k-mer that transiently occupies the nanopore (see Section 2.1.3), although these segmentation mechanisms have limitations in identifying particular types of noise related to translocation speed (i.e., skip and stay errors). The segmentation process is useful for identifying regions of the signal generated by specific k-mers to facilitate more accurate raw signal analysis without the translocation speed effect.

Second, to account for the fact that raw signals generated under different sequencing conditions (e.g., different nanopores within the same flow cell) may appear at different scales, signal values are normalized (e.g., by calculating their standard scores [291]) ❷. Signal values may appear at different scales due to variations between nanopores and within the same nanopore [823, 824]. Normalization adjusts the signal values to a common scale to harmonize these differences and enable consistent subsequent analyses.

Third, the mean or median values of the segmented regions are calculated from the normalized signal values. This step ❸ helps reduce the effects of uninformative outliers and transient signal fluctuations within the segment. Additional methods for outlier detection can be employed [266, 291] to exclude anomalous data points that may result from noise. The value generated for each event is called *event value*.

Despite these initial processing steps, the resulting series of normalized signal values remains noisy due to various types of variations (see Section 2.1.3) and potential skip and stay errors [299]. To further mitigate and eliminate this noise, various techniques have been developed, which we describe in Chapter 3.

Benefits of Raw Signal Analysis. Analyzing raw nanopore signals directly, rather than relying solely on basecalled sequences, offers several unique benefits that can enhance genomic analyses.

First, raw nanopore signals *capture subtle variations in the ionic current* that occur when modified nucleotides, such as methylated bases [233, 234, 239, 831, 832], pass through the

nanopore [833]. These modifications can have profound biological implications, affecting gene expression and regulation [834]. While specialized basecallers can infer certain modifications by incorporating models trained on known modification patterns [832, 834–848], they may not detect novel or rare modifications not represented in the training data [836]. Analyzing raw signals directly allows for a more comprehensive and potentially unbiased detection of nucleotide modifications.

Second, raw signal analysis can *improve accuracy in challenging genomic regions*, such as homopolymeric tracts (e.g., estimating polyA tails [849,850]) and repetitive sequences (e.g., short tandem repeats [851–853]). Although modern basecallers [267–284] provide highly accurate basecalling, these basecallers may still introduce errors in these challenging regions due to difficulties in accurately determining the exact number of repeating units [785, 853].

Third, analyzing raw signals can *reduce computational requirements* in certain applications [266]. Basecalling involves complex deep learning models [267–284] that require substantial computational resources, such as GPUs [267, 268, 270, 272–284, 681, 721] or specialized hardware [269, 669–671, 683, 692]. In resource-constrained environments, such as field portable sequencing setups [297, 789], these requirements may not be practical [297]. By bypassing the basecalling step and analyzing raw signals directly with more lightweight algorithms, it is possible to reduce computational overhead and enable scalable analyses that would otherwise be infeasible.

Fourth, raw signal analysis can *enable new directions in research* by leveraging the rich information contained in the signals. As new algorithms and techniques are developed for directly analyzing raw nanopore signals without basecalling [138, 233, 234, 239, 264–266, 290–302, 831, 832, 849–853], many applications that use basecalled sequences for the analysis (e.g., variant calling [533–570] or *de novo* assembly [250, 251, 305, 504–532]) can be performed using raw signals. These applications can utilize the rich information in raw signals to improve their accuracy while reducing the latency by avoiding the costly basecalling step. This advancement opens up possibilities for more accurate and comprehensive genomic studies, enhancing our understanding of genetics and molecular biology.

In summary, raw signal analysis offers unique advantages by providing access to richer information [290, 833, 848], improving accuracy in complex genomic regions, reducing computational requirements, and enabling new directions in genomic research. While some benefits can be partially achieved through advanced basecalling techniques, these may involve trade-offs such as loss of detail, increased computational demands, and potential inaccuracies. Therefore, analyzing raw nanopore signals directly can enhance genomic studies, particularly in applications where these factors are critical.

2.1.3.3 Unique Opportunities: Real-time Analysis and Adaptive Sampling

Computational techniques that can analyze the raw signals while they are generated at a speed that is as fast as the throughput of nanopore sequencing are called *real-time analysis techniques* [264, 266]. These signals can be analyzed in real-time with [369, 370, 680, 786–790] or without [138, 264–266, 290–302] basecalling, as explained in the earlier parts of this section. There are unique benefits and challenges to performing real-time analysis.

Benefits. Figure 2.5 shows the two unique benefits that real-time analysis offers. First, real-time analysis allows for overlapping sequencing time with analysis time ❶, as raw signals can be analyzed while they are being generated. This benefit enables reducing the overall latency of genome analysis. Second, computational mechanisms can stop the sequencing of a molecule or the entire sequencing run early without sequencing the entire molecule ❷ or the sample using techniques known as *adaptive sampling* with the Read Until [264] and Run Until [786] features. Adaptive sampling is useful for the selective sequencing of *useful* molecules while minimizing the sequencing of *useless* molecules, which can substantially reduce sequencing time and costs. To determine if a molecule is useful, computational mechanisms can start analyzing raw sequencing data with real-time analysis. Based on this analysis, the computational mechanism can determine if further sequencing of the nucleic acid molecule is necessary and can stop sequencing the molecule by *ejecting* it from the nanopore [264]. To eject the molecule, the electric potential is temporarily reversed or altered in such a way that the molecule is pushed back out of the nanopore or drawn back into the *cis* side (upper part of the membrane as explained in Section 2.1.3) of the membrane [264, 790]. This process can fully remove the molecule from the nanopore and allows the nanopore to capture a new molecule for sequencing.

Challenges. There are four key challenges in achieving effective adaptive sampling with real-time analysis. First, the sequencing of unnecessary reads must be stopped as early as possible to minimize wasted time and costs. Specifically, failing to stop sequencing a read after a certain amount of time (i.e., sequencing a read longer than around 5 seconds) significantly increases the risk of blocking the nanopore [783], rendering it unusable for the remainder of the sequencing run. The probability of blocking a nanopore usually increases with the increased sequenced length of a read, as longer molecules are more likely to be tangled during the ejection process. To prevent this, reads are not ejected after being sequenced for a certain period of time [786]. Second, the speed of analysis must keep pace with the data generation speed (i.e., the throughput of the sequencer). This is critical to avoid data backlog and ensure timely decision-making [266]. Third, the analysis techniques must effectively tolerate noise in raw nanopore signals to provide accurate results. Fourth, real-time analysis, particularly in

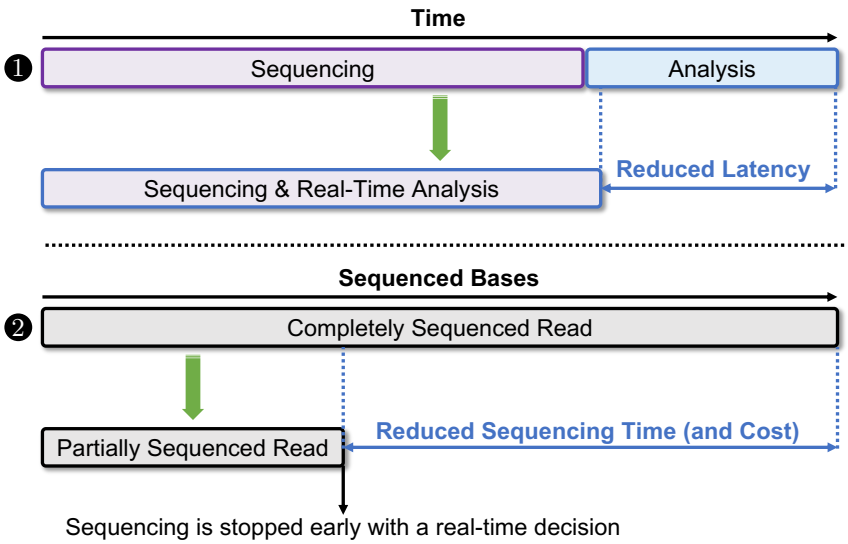

Figure 2.5: Two main benefits of real-time analysis with nanopore sequencing.

portable and resource-constrained environments, must be power-efficient and scalable. Low power consumption is crucial, as the analysis must handle large genomes and high-throughput data while operating within the limited power budgets of portable sequencing setups [297,789]. This necessitates the design of energy-efficient algorithms and hardware architectures that can support real-time, portable sequencing applications, which we discuss in Sections 2.4 and 3.2.2.

2.1.4 Challenges in Analyzing Sequencing Data

High-throughput sequencing (HTS) technologies have significantly advanced our ability to sequence genomes rapidly and cost-effectively; however, they also present inherent limitations that pose challenges for genomic analysis.

First, the reads generated by HTS technologies are produced without prior knowledge of their origin within the genome, except for specialized approaches that perform tagging and barcoding on the DNA to identify the origins of *certain* sequences of nucleotides [250–254]. This lack of contextual information necessitates accurately and efficiently identifying each read's origin. To achieve this, it is essential to compare the reads either 1) against each other, in a process known as *read overlapping* [305], or 2) against a known representative of a species' genome, referred to as a *reference genome*. This sequence similarity identification is usually performed in a step called *read mapping*. Given the large volume of sequencing data produced and the substantial size of certain reference genomes (e.g., a human genome), read mapping becomes challenging in terms of both scalability and accuracy [258].

Second, the reads produced by HTS technologies are prone to variations due to sequencing errors and natural mutations [248,252]. Accurately detecting and distinguishing these variations is crucial for the integrity of genomic analysis. However, identifying these variations in the vast

amount of data generated by HTS technologies is computationally intensive [258], requiring the development of scalable algorithms capable of handling large datasets while maintaining precision.

Addressing these limitations requires the development of efficient algorithms and data structures that can perform rapid similarity identifications, scale to handle the vast amount of data generated, and accurately identify sequence variations while maintaining computational efficiency.

2.2 Facilitating Practical Sequence Analysis with Read Mapping

The goal of read mapping is to identify similarities and differences between genomic sequences, such as between 1) a read (i.e., query sequence) and 2) either another read or a reference sequence representing a species, known as a *reference genome* (i.e., target sequence). Due to genomic variants and sequencing errors, differences and similarities between these sequences (i.e., matches, substitutions, insertions, and deletions) are identified mainly using an approximate string matching (ASM) algorithm to generate an *alignment score* that quantifies the degree of similarity between a pair of sequences (e.g., sequence segments from a reference genome and a read). This process is known as *sequence alignment*. To identify the degree of similarity, the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one sequence into another, known as *edit distance*, is usually measured as well as the corresponding edits. However, ASM algorithms [483–501] often have quadratic time and space complexity due to the dynamic programming nature of these algorithms [854]. This makes the use of sequence alignment computationally challenging between a large number of sequence pairs. To ease the identification of similarities within vast amounts of sequencing data, the read mapping process includes multiple steps designed to narrow down the search space and reduce computational overhead, as shown in Figure 2.6. These steps include:

- Constructing (i.e., indexing 3.1) and using (i.e., seeding 3.2) a database [2, 265, 266, 292, 299, 304–411]. These steps use various sketching (i.e., sampling) [306, 310, 333, 360, 412–432] and hashing [304, 307–309, 315, 334, 358, 414, 416, 424, 433–459] methods to selectively sample and hash sequencing data, which allows for rapid comparison. We describe 1) indexing and seeding in Section 2.2.1, 2) sketching in Section 2.2.2 and 3) hashing techniques in Section 2.2.3.

- Pre-alignment filtering [357, 460–468] and chaining [306, 469–482] 3.3, which discards dissimilar sequences early in the process to avoid unnecessary and expensive alignment computations. We describe these steps in Section 2.2.4.

- Sequence alignment [483–501] 3.4 to determine the alignment and the edit distance between sequence pairs. We describe this step in Section 2.2.5.

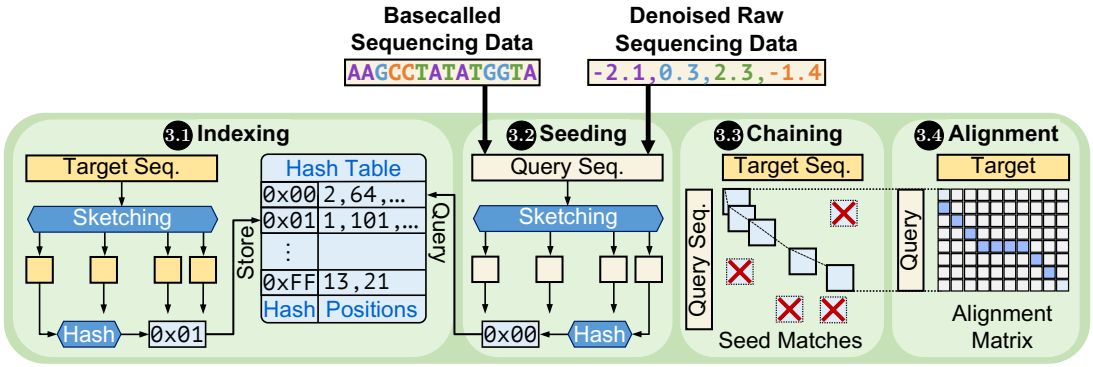

Figure 2.6: Main steps in read mapping.

2.2.1 Indexing and Seeding

To quickly find similar regions between genomic sequence pairs, read mapping typically utilizes data structures, such as a hash table [2, 265, 266, 304–361] or a suffix array-based structure [292, 299, 362–411]. These data structures enable efficient similarity identification between target and query sequence pairs within a pre-built database of the collection of target sequences (e.g., a reference genome). Such a database is called an *index* [855], and the process of constructing this index is called *indexing*.

To narrow down the search space between target and query sequences, an index allows for quick matching of short sequence segments (i.e., *seeds*). This process significantly reduces the need for computationally costly steps like sequence alignment by focusing on regions with matching seeds.

To find the matching seeds efficiently, a common approach is to match the hash values of seeds with a *single lookup* (for each seed) using a hash table as an index. Figure 2.7 shows an overview of how hash tables are used as an index to find seed matches between sequence pairs in two steps. First, certain sequence segments are determined by a sampling mechanism on the entire target sequence to store them in the hash table (3.1 in Figure 2.6). Such sampling mechanisms are called sketching techniques ❶, which we explain in Section 2.2.2. The hash values of these sampled sequence segments are stored as the *keys* of the hash table ❷, while their positional information, such as the sequence ID (e.g., a chromosome name), is stored as the *value* returned by the key ❸. Since there can be multiple sequence segments with the same hash value (i.e., either due to repeated sequence segments or hash collisions), the value is returned as a list of such positional information that contains all seeds with the same hash value. Second, certain sequence segments with the same sampling strategy are extracted from

query sequences to use them as seeds ❹ (3.2 in Figure 2.6). The hash values of these seeds are used to query the hash table ❺ to quickly find the positions where a seed from the query sequence appears in the target sequence with a single lookup. Such a query can quickly and substantially narrow down the search space from the entire target and query sequence to fewer regions (i.e., candidate regions) where seed matches appear.

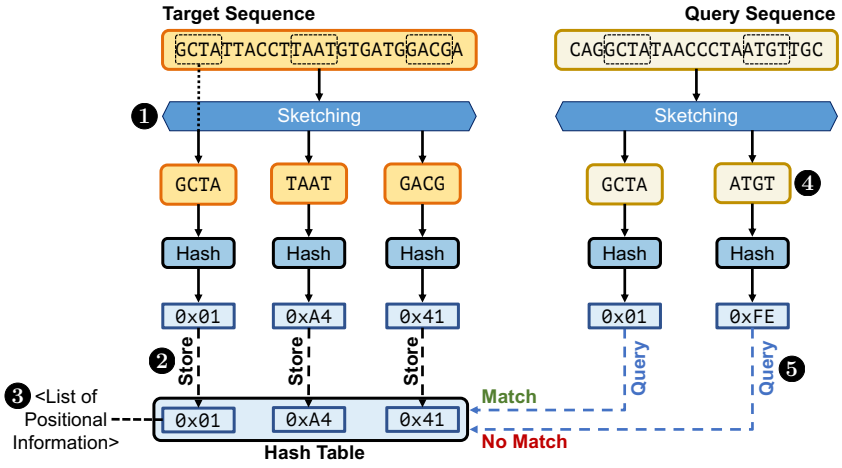

Figure 2.7: Steps in indexing (on the left side) and seeding (on the right side) to find matching sequence segments between target and query sequences using their hash values.

2.2.2 Sketching

Sketching (3.1 and 3.2 in Figure 2.6) aims to accurately represent an entire genomic sequence by sampling it, which enables reducing certain overheads (e.g., performance and memory overheads) of using the entire sequence while minimizing the useful information loss after sampling [334]. Although there are several sketching methods [306, 310, 333, 360, 412–432], one commonly used method is *minimizer sketching* [306, 415, 423]. Figure 2.8 shows the steps in minimizer sketching. First, subsequences of fixed length k, called k-mers, in a window of w ❶ overlapping k-mers[1] are extracted from a sequence ❷ to generate the hash values of these k-mers ❸. Second, among these k-mers, only one k-mer with the minimum hash value is sampled ❹ to represent the sequence segment that w-many overlapping k-mers cover. The window length can be tuned to increase the probability of matching k-mers between sequence pairs. The next window of k-mers is usually generated by removing the first k-mer and including the next consecutive k-mer of the sequence. Finding the minimizers using minimum values is effective since it builds on the assumption that many overlapping windows will share the same minimum value due to hitting the local minima in a large region [423]. This enables finding

[1]The term "overlapping k-mers" is commonly used interchangeably with "adjacent k-mers" or "consecutive k-mers." All these terms refer to the same set of k-mers: a list of k-mers generated by shifting one character at a time along the sequence.

the same local minima between similar sequence pairs even though these sequence pairs are not exactly the same.

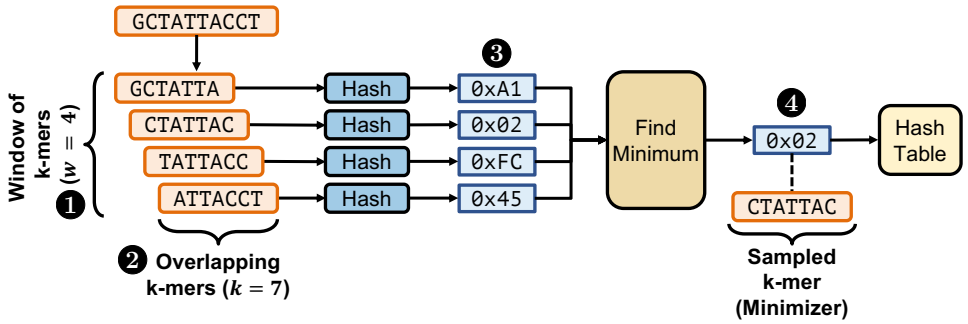

Figure 2.8: A sampling (i.e., sketching) mechanism, called *minimizer sketching*.

Similar to the minimizer sketching, the MinHash sketching technique [420] can be used [416] to sample k-mers based on their minimum hash values. To do this, n hash functions are used, generating n hash values for each k-mer in a sequence. For each hash function, only the k-mer with the minimum hash value is selected from the hash values generated by that function. Although (unlike minimizer sketching) MinHash does not provide window-based sampling like minimizer sketching, it guarantees sampling a fixed number of k-mers, determined by the number of hash functions [416].

MinHash is particularly effective for matching sequences of similar lengths, as the number of hash functions is constant across all sequences. However, when sequence lengths vary significantly, MinHash can generate too many seeds for shorter sequences [305].

There are other alternatives to minimizers and MinHash, such as syncmers [418], paired-minimizers [310, 419, 856], and HyperLogLog-based sketching [417, 421]. These methods offer different trade-offs in terms of performance, accuracy, and search space size (e.g., comparisons between entire genomic sequences). We discuss some of these approaches in Chapter 3.

2.2.3 Hashing Techniques

The main goal of using hashing techniques (3.1 and 3.2 in Figure 2.6) is to *compress* the input values from a larger space into a smaller space. The characteristics of hashing techniques used in genome analysis can broadly be defined in three ways. First, low-collision hash functions [358, 440, 442–454] are designed to avoid assigning the same hash values to different input values, called *collision*. Collisions of dissimilar seeds are incorrect seed matches (i.e., useless) and can lead to 1) unnecessary execution of computationally costly steps in read mapping (e.g., chaining and alignment) and 2) incorrect analysis. Although low-collision hash functions are useful for avoiding incorrect seed matches, using these hash functions requires finding *only* exactly matching seeds. The exact matching requirement imposes challenges

when determining the seed length. Longer seed lengths significantly decrease the probability of finding the exactly matching seeds between sequences due to genetic variations and sequencing errors. Short seed lengths (e.g., 8-21 bases) result in matching a large number of seeds due to both the repetitive nature of most genomes and the high probability of finding the same short seed frequently in a long sequence of DNA letters [460].

Second, masking operations [304,307–309,315,435,436,441,455–457] aim to alter the content of the original seed sequence or its corresponding hash value, usually to provide better locality for similar seeds (e.g., increasing the chance for assigning the same hash value for similar sequences) and to enable the use of efficient similarity estimations [441]. These approaches usually apply a pre-determined and fixed pattern (i.e., masks) to these sequences. Figure 2.9 shows the use of such a fixed pattern ❶ where certain characters are marked *X* (i.e., *don't care characters*) to mask them ❷ (this technique is called *spaced seeding* [304, 307–309, 315, 435, 436, 441, 455–457]). Although fixed patterns can specifically be useful for ignoring potential substitutions at particular positions between seeds, generating hash values from fixed patterns prevents ignoring differences (i.e., substitutions, insertions, and deletions) at arbitrary positions between sequences. *These substitutions and indels between seeds from basecalled sequences form a part of what we call* noise *in the scope of this thesis.*

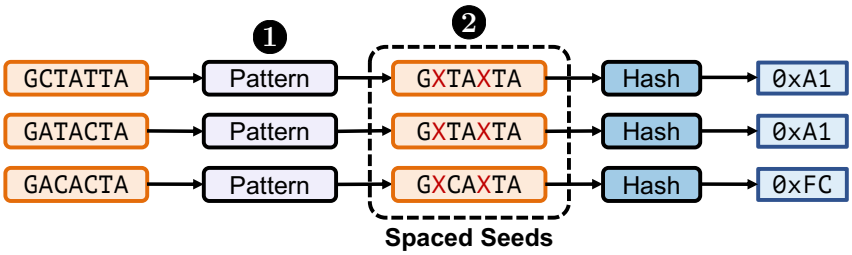

Figure 2.9: A spaced seeding technique. A pre-defined fixed pattern is applied to all three input sequences. Masked characters are highlighted by X in red. The first two input sequences generate the same output hash value since the characters where they differ from each other are masked, generating the same spaced seed and the same corresponding hash value of these seeds.

Third, SimHash [433, 434] is a locality-sensitive hashing (LSH) technique [304, 307, 334, 414, 416, 424, 433, 434, 437–439, 458, 459] that enables generating the same hash values for highly similar vectors (e.g., seeds) even when some arbitrary items between two similar vectors differ (e.g., noise at arbitrary positions). Figure 2.10 shows the main steps to calculate the SimHash value for a given input vector (e.g., vector of words). The input vector (e.g., a sentence) is provided as input ❶ to generate its hash value (i.e., SimHash value). To do so, the vector items are defined, such as all words in the sentence ❷. Each item (e.g., word) of the vector is hashed using a low-collision hash function ❸. These hash values are shown in their binary forms. A

bitwise sum is then computed across all the generated hash values ❹. This is done by adding the corresponding bits across all the hash values at each bit position. The result is a counter vector ❺, where each entry represents the net sum (difference between 1s and 0s) for that specific bit position. The SimHash value is generated from the counter vector ❻. For each position in the counter vector, if the sum is positive, the corresponding bit in the SimHash value is set to 1; if the sum is negative, the bit is set to 0. This results in a SimHash value for the given input vector. The SimHash technique enables generating the same hash values for highly similar vectors because a few different individual items (e.g., noise) may result in a counter vector with the same sign information, even though the exact count information may be different. This property allows SimHash to produce the same hash values despite minor variations between vectors.

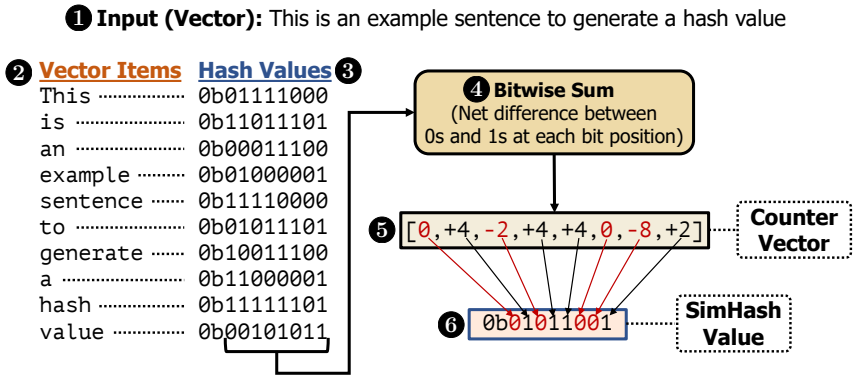

Figure 2.10: Generating a hash value for a vector of items using SimHash technique. Example input is a sentence (i.e., vector) where the hash values (shown in binary form) of these words (i.e., items) are used to generate the hash value for the entire sentence.

These properties of the SimHash technique enable estimating the cosine similarity between a pair of vectors [857] based on the Hamming distance [858] of their hash values that SimHash generates (i.e., *SimHash values*) [433, 859]. To efficiently find the pairs of SimHash values with a small Hamming distance, the number of matching most significant bits between different permutations of these SimHash values are computed [434]. This *permutation-based* approach enables exploiting the Hamming distance similarity properties of SimHash technique for various applications that find near-duplicate items [434, 860–863].

2.2.4 Filtering and Chaining

The computational cost of performing sequence alignment on a vast number of seed matches, where many sequences may not actually align, can be prohibitively expensive. This results in a significant waste of computational resources. To mitigate this, read mappers employ 1) pre-alignment filtering [357, 460–468] and 2) chaining techniques [306, 469–482] to eliminate

dissimilar sequence pairs before alignment and chain many useful seed matches into longer regions.

2.2.4.1 Filtering

Filtering techniques are used to quickly eliminate dissimilar sequences before they proceed to the computationally expensive alignment step. There are four main filtering techniques.

Frequency Filters. To quickly eliminate high-frequency, non-informative k-mers that may lead to incorrect seed matches, frequency filters are used [306]. These frequency filters usually calculate the average number of occurrences of a seed in an index and set a frequency threshold based on this calculation to identify if a seed appears frequently. By focusing on less frequent, more informative seeds, frequency filters help reduce the computational burden and improve the accuracy of the subsequent alignment steps.

Pigeonhole Principle. To quickly detect and discard dissimilar sequences from further analysis, the pigeonhole principle [460, 461, 464] assumes that if two sequences differ by up to E edits, they should still share *at least one common subsequence* among any set of $E + 1$ non-overlapping subsequences. If a sequence pair does not share a sufficient number of common subsequences, it can be filtered out by matching these subsequences, thereby avoiding unnecessary and costly sequence alignment.

Base Counting. To recognize similar regions by quickly estimating the edit distance between two sequences, the base counting approach [321, 359] compares the counts of individual nucleotide bases (A, C, G, T) between sequence pairs. The sum of the absolute differences in base counts provides an upper bound on the similarity identification measured based on edit distance. If this sum exceeds a certain threshold, the sequences are identified as dissimilar, and further alignment is avoided.

q-gram Filtering. To filter out dissimilar sequences by examining shared subsequences, the q-gram filtering approach [463] considers all overlapping substrings of length q (known as q-grams) within the sequence pairs. Since differences between sequences affect only a certain number of q-grams, the technique can estimate whether the sequences are sufficiently similar by quickly counting the number of shared q-grams. The sequence pair is filtered out if the number of shared q-grams is below a threshold.

2.2.4.2 Chaining

To identify the seed matches, known as *anchors*, that appear in close proximity in both target and query sequences are linked together in a step called *chaining* [306, 469–482] (3.3 in Figure 2.6). Identifying co-linear chains of anchors is useful as it can eliminate 1) the anchors that are not included in chains and 2) the chain of anchors that are unlikely to provide a mapping between target and query sequence pairs, which further reduces the search space (after the

filtering step) for the more computationally intensive sequence alignment step. Figure 2.11 shows a pair of anchors used in the chaining step to identify if these anchors should be linked together in the same chain. We explain the chaining process in several steps.

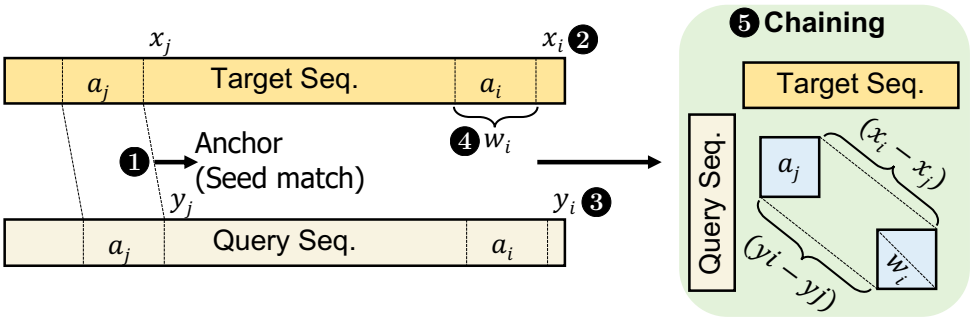

Figure 2.11: Chaining anchors (seed matches) between target and query sequences. The chaining approach calculates the distance between a pair of anchors to identify if they should be included in the same chain.

To explain the chaining process, we define each anchor ❶ as a 3-tuple (x, y, w), where x denotes the end position of the matching interval in the reference sequence ❷, y denotes the end position in the query sequence ❸, and w represents the length of the matching region (i.e., the span of the seed match) ❹. The goal of chaining is to calculate the optimal chain score $f(i)$ for each anchor i given a predecessor anchor j, which is determined through a dynamic programming (DP) approach ❺ in Equation 2.1:

$$f(i) = \max \left\{ \max_{i>j\geq 1} \{f(j) + \alpha(j, i) - \beta(j, i)\}, w_i \right\} \tag{2.1}$$

where $\alpha(j, i) = \min \left\{ \min\{y_i - y_j, x_i - x_j\}, w_i \right\}$ represents the number of matching bases that are covered by anchor i and not by predecessor anchor j, while the gap cost $\beta(j, i)$ penalizes gaps between anchors to ensure that only those anchors with minimal insertions or deletions chained together. The gap cost can be defined as:

$$\beta(j, i) = \gamma_c \left((y_i - y_j) - (x_i - x_j) \right) \tag{2.2}$$

where the gap penalty function $\gamma_c(l)$ is given by:

$$\gamma_c(l) = \begin{cases} 0.01 \cdot \bar{w} \cdot |l| + 0.5 \log_2 |l| & (l \neq 0) \\ 0 & (l = 0) \end{cases}$$

where $\bar{w}$ is the average seed length, and l is the length of the gap between the two anchors.

To efficiently compute the near-optimal chain score across many anchors, a heuristic is applied to reduce computational complexity. Instead of evaluating all possible predecessors for

each anchor, the algorithm can limit the number of comparisons to a fixed number of previous anchors. This heuristic significantly reduces the time complexity from $O(N^2)$ to $O(hN)$, where N is the number of anchors.

Backtracking and Identifying Primary Chains To identify the best chain among the several chains given a set of anchors, the backtracking step starts with an anchor with the highest score. The process continues by following the chain of predecessors for each anchor and marking them *used* until no further predecessor that is not used in another chain can be found. This backtracking step can identify multiple chains that represent similar regions between the sequence pairs.

This filtering and chaining process significantly improves the efficiency and accuracy of the read mapping process to enable the alignment of sequences in a computationally feasible manner, even when dealing with large genomic datasets.

2.2.5 Sequence Alignment

To accurately identify the positions and types of differences and similarities between sequence pairs, approximate string matching (ASM) approaches are commonly used for *sequence alignment* (3.4 in Figure 2.6). This process aims to identify differences and similarities, such as matches, substitutions, insertions, and deletions, by calculating the *edit distance* between sequence pairs—the minimum number of single-character edits required to transform one sequence into another. Widely used methods for performing sequence alignment are mainly based on the dynamic programming (DP) approach [483–501].

Figure 2.12 shows how a DP matrix is constructed and used for sequence alignment. Each cell in the matrix represents a score for a potential edit operation between characters from the two sequences being aligned. The score in each cell is computed based on three main scenarios: 1) aligning the current characters from both sequences (resulting in either a match or a substitution) ❶, 2) aligning the current character from one sequence by *opening* a gap in the other sequence (insertion or deletion) ❷, and 3) extending the open gap in the other sequence to handle insertions (or deletions) that appear in a row. ❸ The arrows in the matrix indicate how each cell's score is calculated based on the values from the adjacent cells.

To minimize the number of edits between sequence pairs, the alignment process usually penalizes operations that introduce and extend the existing open gaps. This is typically achieved using the *affine gap* scoring function [306]. The affine gap model applies two types of penalties: 1) a *gap opening penalty*, which usually incurs a higher cost when a new gap is introduced, and 2) a *gap extension penalty*, which mainly imposes a smaller incremental cost for extending an existing gap. These penalties help ensure that gaps are used judiciously in the alignment.

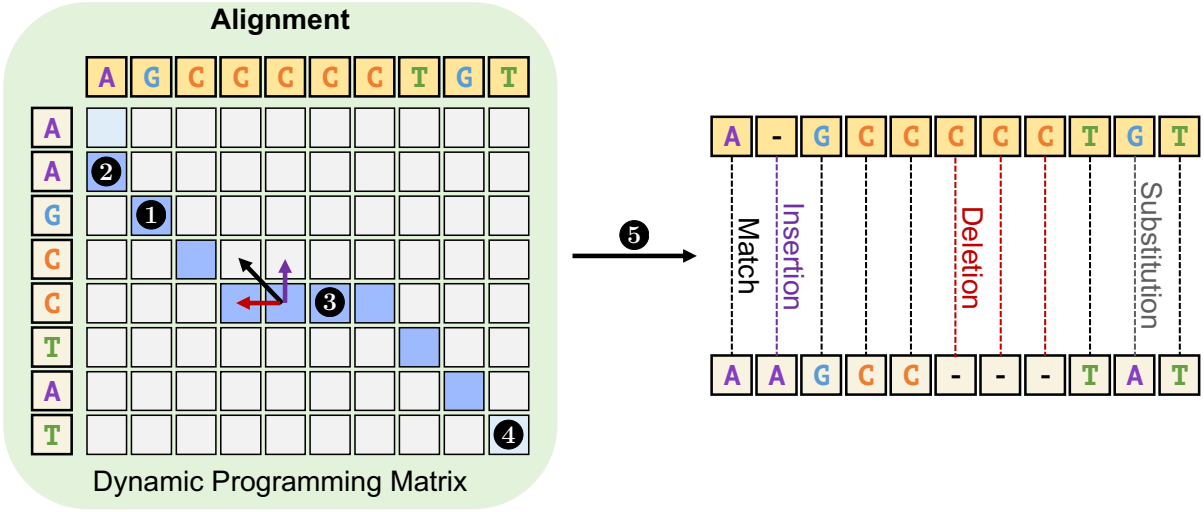

Figure 2.12: Dynamic Programming (DP) Matrix used to identify edit operations: Match, Substitution, Insertion, and Deletion. Each cell's value is calculated based on the values of adjacent cells, as indicated by the arrows. The optimal alignment path is highlighted in dark blue, while light blue cells indicate two subsequent anchors identified during the chaining step. The corresponding edit operations for the optimal alignment are shown on the right side of the DP Matrix.

The optimal alignment path is identified by backtracking from the cell with the highest score, as illustrated with cells marked by blue colors starting from the lower-right corner ❹ in Figure 2.12. This path shows the sequence of matches, substitutions, insertions, and deletions that results in the optimal alignment. The right side of Figure 2.12 shows the corresponding edit operations for the aligned sequences. Each operation is marked as matches (black lines), insertions (purple lines), deletions (red lines), and substitutions (grey lines). The final alignment score, which results from this backtracking process, provides a quantitative measure of sequence similarity by accounting for the types and number of edits needed to align the sequences.

DP methods provide a rigorous approach to perform sequence alignment. However, their computational complexity, typically $O(m \cdot n)$ [854] for sequences of length m and n, can become a limitation for long reads or large reference genomes. To improve efficiency, filtering and chaining steps are often used before alignment to narrow down the search space, which can prevent using the alignment process unnecessarily.

2.2.6 Necessity of Read Mapping

A fine-grained mapping of reads (i.e., basecalled sequences and raw sequencing data) by finding the base positions (i.e., *base-level read mapping*) where they map either within a corresponding genome or to another read is essential for various reasons. First, base-level mapping enables the detection of genetic variants such as single nucleotide polymorphisms (SNPs), insertions and deletions (indels), and structural variations by comparing the aligned

reads to a reference genome, as we discuss in Section 2.3.3. Second, read mapping facilitates genome assembly and reconstruction by identifying overlapping regions between reads based on their exact positions, which is crucial for *de novo* genome assembly to construct an organism's genome from scratch without a reference genome, as discussed in Section 2.3.1. Third, base-level mapping allows for accurate quantification of gene expression levels in RNA sequencing (RNA-Seq) experiments by mapping reads to specific genomic locations to measure transcript abundance, identify differentially expressed genes, and detect alternative splicing events [864, 865]. Fourth, read mapping is essential for *haplotype phasing* and detection of *de novo* mutations to determine which genetic variants are inherited together on the same chromosome and to study the detection of new mutations that arise in a single generation (*de novo* events) [58, 505, 866]. Fifth, base-level mapping enables the detection of *rare* or *low-frequency variants*, especially in heterogeneous samples such as cancer tissues, which is crucial for understanding tumor heterogeneity and developing personalized treatment strategies [867]. Read mapping is essential for many applications to achieve sensitive and fast analysis, even for scenarios where potential alternatives such as polymerase chain reaction (PCR) [868] can fall short compared to read mapping.

2.3 Further Steps in Sequence Analysis

2.3.1 *De novo* Assembly Construction

The goal of *de novo* assembly is to reconstruct a genome from reads without the use of a reference genome. This is achieved by identifying similarities between read pairs, known as *all-vs-all overlapping*, and stitching these reads together. There are mainly two reasons for constructing *de novo* assemblies. First, the reference genome may be unavailable or incomplete, particularly for non-model organisms or populations with high genetic diversity [869]. Second, analyzing an individual's genome without reference to a potentially divergent reference genome reduces biases introduced by reference-specific artifacts and assumptions [870].

All-vs-all Overlapping. To stitch the reads together from their ends, the first step in reconstructing a genome from sequencing reads involves identifying overlapping regions among all pairs of reads [305]. Typically, two reads are considered overlapping if they contain matching sequences from their ends (i.e., between the suffix of one read and the prefix of another). To identify these matching regions, read mapping steps discussed in Section 2.2 are mainly used.

Assembling Overlapping Reads. Constructing the assembly from the paths generated by overlapping reads is challenging due to the presence of numerous redundant and incorrect [305] overlapping pairs. To identify the underlying genome assembly, the overlap information between all read pairs is processed by constructing a graph from overlapping reads using a graph structure, such as an assembly graph (i.e., string graph) [532] or a de Bruijn graph [508].

These assembly graphs often contain errors [531], which are resolved through several steps collectively known as *graph cleaning*. First, incorrect or incomplete overlap information can result in direct connections (i.e., edges) between two reads, even when a longer path (i.e., multiple nodes and edges) with multiple overlapping read pairs connects them. These direct edges, known as *transitive edges*, are removed because they do not affect the overall connectivity of the graph [532]. Second, at least two sub-graphs with several nodes can exist in the graph such that they share the same start and end nodes. This creates alternative paths in the assembly graph called *bubble*. These bubbles can be resolved by either collapsing them, keeping only one path [305] or by presenting the alternative paths as different potential assemblies [505, 518]. These bubbles, often caused by variations and sequencing errors, are useful for identifying different copies of chromosomes in diploid and polyploid genomes. Third, the graph is traversed to identify paths that represent the original genome sequence, ensuring the assembled sequences are accurate and continuous. Additional steps can be used to simplify the graph further by identifying errors and artifacts in sequencing and during the overlapping process [305]. The resulting graph typically includes paths that are *trivial* to parse (i.e., without multiple edges from a node), representing the assembled genome by stitching the reads to each other as identified by the path in the graph.

2.3.2 Metagenomics Analysis

Metagenomics involves the study of genetic material recovered directly from environmental samples to allow for the analysis of microbial communities without the need for culturing individual species. This approach provides insights into the diversity and composition of microbial ecosystems.

Read Classification. The initial step in metagenomics analysis is the classification of sequencing reads, where DNA or RNA sequences obtained from the environment are assigned to known taxonomic groups. This classification is usually performed by aligning reads to a reference database of known genomes [571]. Alternatively, k-mer-based matching can quickly identify the closest taxonomic matches based on sequence similarity [572, 590]. The classification process enables the identification of the species present in the sample and helps map the genetic potential and functional roles of these organisms within their ecosystems.

Relative Abundance Estimation. Following classification, estimating the relative abundance of each identified species is a common next step in metagenomics. This process involves quantifying the number of reads assigned to each taxonomic group and normalizing these counts to mitigate biases introduced by variable gene copy numbers, read length, and genomic size among different organisms. Relative abundance data are useful for understanding the

structure of microbial communities and assessing how environmental changes or different conditions affect microbial composition and dynamics.

Challenges. Despite its potential, metagenomics faces several challenges. First, the sheer diversity of microbial communities, coupled with the vast amount of data generated, poses computational and analytical challenges [710]. The limited availability of complete reference genomes can hinder accurate taxonomic classification and functional prediction, especially for rapidly evolving microbial genomes [871]. Third, the presence of DNA from dead cells can complicate interpretations of community activity and function [872].

Addressing these challenges is critical for advancing our understanding of microbial ecosystems. Improvements in computational methods, database completeness, and data interpretation can allow for more accurate and comprehensive insights into microbial diversity, dynamics, and functional roles, further expanding the potential of metagenomics in environmental and clinical research.

2.3.3 Variant Calling

The objective of variant calling is to identify genomic variants between an individual's genome and a reference genome [533–570]. These variants are mainly categorized as single-nucleotide polymorphisms (SNPs), insertions, deletions, and larger structural variations (SVs). Accurate and efficient detection of these variants is vital for understanding of the genetic basis of diseases [66], population genetics [535], evolutionary studies [51], personalized medicine [873] and pharmacogenomics [874].

Variant calling involves processing the read mapping output and detecting variants. First, read mapping output is processed by sorting and optionally identifying duplicate information to minimize bias introduced during the *polymerase chain reaction* (PCR) step of sample preparation [875]. Second, mapped reads are analyzed to distinguish genuine variants from sequencing errors or misalignments using resource-intensive statistical techniques [535, 536, 538] or machine learning techniques [540]. Despite the high computational demands, variant calling plays a crucial role in advancing our understanding of genetic variations, driving research in genomics and personalized medicine. Ongoing improvements in variant calling accuracy, speed, and efficiency remain essential as the volume of sequencing data continues to grow.

2.4 Accelerating Genome Analysis

The goal of accelerating genome analysis is to reduce the time and energy required to process sequencing data and produce actionable insights. This acceleration would in turn reduce the overall latency and energy consumption of end-to-end genome analyses, while also improving throughput of analyses [262].

We categorize the strategies for accelerating genome analysis into two main categories: 1) acceleration with software (and algorithmic) optimizations as we discuss in Section 2.4.1, and 2) acceleration through hardware and software co-design as we discuss in Section 2.4.2.

We explain how these strategies are used in many steps in the genome analysis pipeline in Section 2.4.3.

2.4.1 Acceleration with Software Optimizations

Software-based optimizations aim to improve the performance (i.e., speed) of genome analysis tools by modifying the existing or designing new algorithms and data structures. These optimizations often (but not always) involve trade-offs between performance and accuracy, aiming to reduce computational load while maintaining acceptable (and ideally the same or better) levels of accuracy. We describe several common strategies below.

Reducing Computational and Space Complexity. Developing new algorithms that reduce the computational and space complexity of genome analysis tasks can directly accelerate the execution of an application [483, 485]. This usually involves replacing either part of the entire execution flow of an application with different sets of algorithms. For example, this can include replacing the computationally complex use of DNN approaches with simpler operations [266] or eliminating the use of costly operations with DNN approaches [535, 540].

Eliminating Unnecessary Computation. Eliminating unnecessary computations by identifying and discarding irrelevant or low-quality data early in the pipeline significantly reduces computational overhead and accelerates overall performance. We explain five common strategies to reduce the overall computations.

First, for DNN-based approaches (e.g., basecalling), a large number of parameters can be pruned without substantially reducing the accuracy to reduce computational overhead [275].

Second, data resolution can be reduced (i.e., quantization), which is useful, for example, for DNN-based works as they include a large number of complex operations with high precision [266, 275, 299].

Third, certain filters or sampling techniques can quickly eliminate computations and data that are likely useless in later steps by using cheaper computations to identify such useless computations and data (e.g., sketching and pre-alignment filters techniques as discussed in Sections 2.2.2 and 2.2.4, respectively) [267, 306, 310, 333, 357, 360, 412–432, 460–468]

Fourth, in many DP operations, such as in chaining [306] and when using profile Hidden Markov Models (pHMMs) [593], the connections to backward cells (or nodes) can usually be limited by how long a single connection can extend backward. This enables substantially reducing the high-overhead data-dependent computations that are usually performed iteratively.

Fifth, redundant computations are common when re-running a certain application on

slightly different data [502, 503, 876–882]. This is, for example, the case for read mapping when remapping reads from an older reference genome to a newer one [502]. In such cases, the similarities between the multiple versions of the data can be analyzed quickly to eliminate redundant computations by re-using the already-generated information from the earlier execution and identifying when it is necessary to perform additional computation on the newer data.

Data Structure and Data Access Pattern Optimizations. Optimizing the choice of data structures and their access patterns [324, 594] is another key method for accelerating genome analysis. Efficient data structures, such as compressed matrices [595, 596], graphs [597], and specialized indexing schemes [598], can significantly reduce memory usage and accelerate data retrieval.

Exploiting Parallelism. Leveraging parallelism is a fundamental approach to accelerating genome analysis. Modern commodity processors offer multi-core and many-core architectures that can be utilized to execute multiple analysis tasks that are multithreaded. Single Instruction, Multiple Data (SIMD) operations [334, 462, 756] can be used to perform the same operation on multiple data points simultaneously, making them highly effective for processing large genomic datasets. To significantly accelerate genome analysis tasks, massively parallel operations in genome analysis are exploited using 1) Graphics Processing Units (GPUs) [267,268,270,272–284,300,301,484,616–646,652–660,681,721], as they are well-suited for handling the large-scale parallel computations required in genomics, and 2) general distributed computing techniques [596, 599–615, 761].

2.4.2 Acceleration with Hardware and Software Co-design

Hardware-software (HW/SW) co-design, where hardware and software (and algorithm) are designed together to overcome the different limitations of and take advantage of the different capabilities of underlying hardware and software, can offer substantial improvements in both performance and energy efficiency. As such, these HW/SW co-designed acceleration approaches have formed a thriving area of research [260–262]. We discuss key challenges in software-only optimizations and potential hardware-software co-design solutions.

Challenges in Pure Software Optimizations. Many software optimization strategies still involve significant data movement between computation and memory units, which can create bottlenecks due to limited bandwidth and high power requirements [260, 661, 689, 710, 883–890]. Second, a substantial portion of the data processed in genome analysis pipelines can be useless [461, 463–467, 644, 661, 666, 673, 684, 689, 689, 690, 693, 708, 715, 716, 776], resulting in wasted compute cycles. Identifying and discarding useless computations earlier in the pipeline can save time and energy. However, even with software optimizations that identify

useless computations, data must still be moved through different levels of memory hierarchy, causing unnecessary delays [262, 891]. Third, to accommodate a wide range of applications, general-purpose processors usually provide limited computational resources and speed to execute effectively a particular operation [892, 893]. This makes it challenging to exploit the opportunities that a certain application provides (e.g., data access patterns with certain locality). Fourth, HTS technologies generate data at an ever-increasing rate, presenting a challenge to keep up with the output, especially in time-critical applications [260, 266].

To address these challenges, several hardware design strategies can be employed. These strategies almost always necessitate co-design of software along with the hardware.

Designing Customized Hardware. Application-Specific Integrated Circuits (ASICs) and Field-Programmable Gate Arrays (FPGAs) (and customized system on chip designs) can be used to design customized hardware to accelerate specific computational tasks [138, 269, 297, 302, 461, 464–466, 484, 490, 593, 621, 642, 661, 662, 669–679, 683, 692, 694–700, 703–707, 709, 723–725, 730, 732–734, 736, 740–744, 748, 750–753, 755, 764–770, 777–779]. FPGAs and ASICs are particularly important for designing special processing units (PUs) that are highly performant and efficient hardware. These PUs are specifically designed based on the needs of the application, especially when these applications provide a certain data flow such that intermediate data generated after the execution of each PU is moved to another PU that performs either the same or a different operation. In such cases, these PUs can be connected using a design known as *systolic arrays* [894, 895], which stream data efficiently between processing units, reducing the need for additional read and write operations to external memory. Such specialized designs avoid the overheads present in general-purpose hardware.

Memory-Centric Design. Memory-centric approaches, such as Processing-in-Memory (PIM) (i.e., Processing Using Memory (PUM) and Processing Near Memory (PNM)) [262, 597, 661, 828, 829, 855, 883–890, 896–998], can bring computation closer to the data to reduce the data movement overheads in genome analysis [195, 463, 467, 612, 627, 663–666, 668, 673, 682, 684–688, 690, 691, 693, 701, 702, 708, 711, 712, 714–720, 726–729, 731, 735, 737, 745, 749, 758–760, 762, 763, 771, 775, 776]. PUM approaches can perform computations directly within memory cells (e.g., by using analog operations within DRAM), reducing the need to transfer data between memory and processing units and exploiting the inherent parallelism present in memory arrays [890, 910, 976, 979, 980, 982]. PNM approaches place processing elements near memory to allow high-throughput data access at reduced latency and energy [890]. It is essential to understand the application's requirements to fully leverage PUM and PNM approaches, optimizing both computation [976, 982] and communication between memory-centric accelerators [890, 999].

Storage-Centric Design. Storage-centric designs integrate computational capabilities directly into storage devices, such as Solid-State Drives (SSDs) [689, 710, 747, 913]. To accelerate the

execution of an application, in-storage processing can leverage the higher internal SSD bandwidth to reduce data movement to external processing units by processing the data inside the SSD [689]. This can be achieved by utilizing the existing cores and DRAM within SSDs [1000], integrating specialized hardware inside SSDs [689], and performing computations with flash chips in SSDs [1001]. Storage-centric computation provides many opportunities, especially when the data reuse of an application is low, and the application can benefit from the increased bandwidth inside SSDs.

Extending Instruction Set Architecture with Specialized Instructions. Extending the Instruction Set Architecture (ISA) with specialized instructions can improve the performance of genome analysis tasks. By incorporating custom instructions tailored to specific genomic operations, such as DP operations in sequence alignment, it is possible to execute these tasks more efficiently, reducing the need for complex software routines and lowering the overall computational overhead [733,748,755,1002]. This approach involves designing custom logic and data handling mechanisms with special instructions to execute specific tasks directly within the processor, bypassing the limitations of general-purpose computation.

Exploiting Emerging Technologies. Emerging technologies such as analog computation using Resistive Random-Access Memory (ReRAM) [665, 666, 684, 711, 759], data search using Content Addressable Memory (CAM) arrays [727, 728, 731, 760], and optical computing [1003] provide opportunities for further acceleration. These technologies can potentially improve performance and energy efficiency for specific genome analysis tasks by leveraging novel computational paradigms.

By combining these hardware-centric approaches with software optimizations, highly efficient and scalable genome analysis pipelines capable of meeting the demands of modern sequencing technologies and applications can be created.

2.4.3 Accelerating the Common Steps in a Genome Analysis Pipeline

Hardware and software co-designed approaches are commonly utilized to accelerate several steps in the common basecalled analysis of genomes. These include accelerating the steps in read mapping and variant calling, which can be integrated into real-time analysis to reduce the latency of genome analysis. We explain these acceleration efforts in this section. We explain the acceleration efforts for basecalling and raw signal analysis in Sections 3.2.2 and 3.2.3, respectively.

2.4.3.1 Accelerating Read Mapping

Since read mapping is a crucial and computationally expensive step in many genome analysis pipelines, numerous works focus on accelerating it in various ways. First, a significant fraction of sequence pairs does *not* align, which leads to wasted computation and energy during

alignment [463]. To avoid this useless computation, various works propose *pre-alignment filtering*, another step in read mapping that can efficiently detect and eliminate highly dissimilar sequence pairs *without* using alignment. Most pre-alignment filtering works [461, 463–467, 644, 661, 666, 673, 689, 690, 693, 708, 715, 716, 776] provide algorithm-architecture co-design using FPGAs [461, 464–466, 661, 673], GPUs [644], and PIM [463, 467, 666, 673, 690, 693, 708, 715, 716, 776] to substantially accelerate the entire read mapping process by exploiting massive parallelism, efficient bitwise operations, and specialized hardware logic for detecting similarities among a large number of sequences. Apart from accelerating the filtering process, many works co-design hardware and software for the chaining [642, 643, 686, 734, 741, 755] (i.e., by using custom hardware design [642, 734, 741, 755], GPUs [642, 643] and PIM [686]) and seeding steps [594, 599, 647–651, 667, 713, 746, 757, 772–774] (i.e., by using custom hardware design [746], GPUs [648, 651] and PIM [667, 713, 757, 757, 772–774]) to accelerate the pre-alignment step in genome analysis.

Second, various works [689, 710, 747] such as GenStore [689], show that a large amount of sequencing data unnecessarily moves from a storage system (e.g., the solid-state drive (SSD)) to memory during read mapping, significantly increasing latency and energy consumption. To eliminate this wasteful data movement, available computation capabilities and the high bandwidth within the SSD, as well as the specialized logic that can be integrated within a storage system, can be exploited to process sequencing data in the storage system without moving to the main memory or the CPU, thereby eliminating unnecessary data movement in the system.

Third, numerous studies [484, 490, 593, 625–641, 664–666, 668, 675–678, 685, 688, 691, 694, 695, 697–702, 709, 717, 719, 733, 736, 737, 740–745, 754, 775], including GenASM [490] and Darwin [673], focus on accelerating the underlying ASM algorithm employed in sequence alignment through efficient algorithm-architecture co-design. They do so by exploiting systolic arrays [490, 676], GPUs [484, 625–641], customized hardware designs (and system on chip) [484, 490, 593, 675–678, 694, 695, 697–700, 709, 733, 736, 740–744], and PIM [627, 664–666, 668, 685, 688, 691, 701, 702, 717, 719, 737, 745, 775]. These works provide substantial speedups of up to multiple orders of magnitude compared to software baselines. Among these works, SeGraM [598] is the first to accelerate mapping sequences to graphs [598, 738, 739] that are used to reduce population bias and improve genome analysis accuracy by representing a large population (instead of a few individuals) within a single reference genome.

Fourth, several works focus on accelerating either all or some of the steps together [645, 646, 662, 672, 674, 684, 687, 696, 712, 718, 732, 734, 735, 748] using custom hardware accelerators [598, 738, 739], GPUs [645, 646] and PIM [684, 687, 712, 718, 735].

2.4.3.2 Accelerating *De novo* Assembly

De novo assembly pipeline commonly uses many steps in read mapping (i.e., seeding, sketching, and chaining). We discuss the acceleration efforts of these read mapping steps in Section 2.4.3.1.

There are unique challenges to *de novo* assembly, such as graph construction and cleaning (as discussed in Section 2.3.1) and k-mer counting [648]. Methods specifically designed for accelerating *de novo* assembly [596, 599–624, 762–771, 777–779] utilize specialized algorithms and hardware accelerators to efficiently handle large-scale assembly with distributed computing techniques [596, 599–615], custom hardware accelerators [621, 764–770, 777–779], GPUs [616–624], and PIM [612, 762, 763, 771]. Most of these accelerators target accelerating k-mer counting due to the nature of embarrassingly parallel operations in k-mer counting that can be performed independently on many-core architectures.

2.4.3.3 Accelerating Metagenomics Analysis

Metagenomic analysis poses significant computational challenges due to the large volume of data involved [710]. Efforts to optimize metagenomic analysis have focused on both software and hardware-based solutions.

Software Optimization of Metagenomics. Various software tools [571–591] aim to improve the efficiency of metagenomic analysis by optimizing the use of databases. Tools like MetaAlign [571] and Centrifuge [580, 589] aim for high accuracy with computationally expensive approaches. However, these tools often incur significant computational and I/O costs due to the large sizes of these databases [710]. To mitigate these issues, some tools apply sampling techniques to reduce database size, such as MetaPalette [581] and MetaCache [654]. While these techniques improve performance by reducing the amount of data that needs to be processed, they can lead to a loss in accuracy.

Hardware Acceleration of Metagenomics. Several studies [195, 654–660, 710, 714, 720, 723–731, 747, 759–761] have explored hardware acceleration to overcome the computational bottlenecks in metagenomics. GPUs have been widely used to speed up various stages of metagenomic analysis by offloading compute-intensive tasks [654–660]. FPGAs provide opportunities for acceleration, offering configurable hardware that can be tailored to specific tasks within the metagenomic pipeline [723–725, 730]. Processing-in-memory (PIM) techniques have been employed to address the I/O bottlenecks in metagenomic analysis [195, 714, 720, 726–729, 731, 759, 760]. These hardware-accelerated approaches significantly reduce computation time and energy consumption but often still face challenges related to the I/O overhead associated with large metagenomic datasets. MegIS [710] alleviates the data movement overhead in metagenomics analysis from the storage system [710, 747] 1) via its

efficient and cooperative pipeline between the host and the SSD and 2) by leveraging in-storage processing for metagenomics.

2.4.3.4 Accelerating Variant Calling

Variant callers [533–570] like GATK HaplotypeCaller [535] use costly probabilistic calculations to analyze the likelihood of specific variants in large sequencing datasets. DeepVariant [540], a DNN-based variant caller, processes read alignment data as images, a method that demands substantial GPU resources and memory due to the complexity of deep neural networks. Reducing computational requirements through algorithmic optimizations, parallelization, and efficient data representation is crucial for faster, more accurate genetic variant analyses.

To accelerate variant calling, several works propose algorithm-architecture co-designs [652, 653, 679, 703–707, 749–754, 758]. These include fast execution of Pair Hidden Markov Models (Pair HMMs) in FPGAs or ASICs [679, 703–707, 750–753], reducing data movement overheads in GPUs [652, 653], and pipelining processing steps with tools like elPrep [1004], system-on-chip designs [704], and PIM [749, 758].

2.5 Summary and Further Reading

Significant research efforts are essential across the entire sequence analysis pipeline. These efforts range from improvements in sequencing technologies (including device and circuit-level advancements) to improvements in read mapping and downstream analysis (spanning algorithmic, software, and hardware optimizations). Each step, from efficiently processing raw sequencing data to accurately assembling and analyzing genomes, plays a critical role in advancing our understanding of genetic information and its practical applications.

For further reading, we recommend that interested readers explore works on sequencing technologies [215–247, 249], algorithmic approaches [256, 257, 334, 379, 386, 411, 424, 433, 434, 437–439, 458, 572, 580, 585, 589, 813], and hardware accelerations [138, 195, 262, 267–284, 297, 300–302, 461, 463–467, 484, 490, 593, 594, 596, 598–779] for genome analysis.

Chapter 3

Related Work

Many prior works study noise in sequence analysis and develop techniques for understanding, tolerating, and reducing this noise. This section provides an overview of closely-related works and describes where the existing body of work falls short.

In Section 3.1, we describe the limitations of the existing seeding techniques to set the stage for how the work we discuss in Chapter 4 fills the gap to resolve the limitations we discuss.

In Section 3.2, we discuss the existing works that perform real-time analysis with and without basecalling. We identify their challenges. In Chapters 5 and 6, we discuss how our works overcome these challenges.

3.1 Tolerating Noise in Seeding – Fuzzy Seed Matching

Commonly used read mappers and other related works that perform sequence analysis, such as minimap2 [306], MHAP [416], Winnowmap2 [413, 1005], re$M_u$val [1006], and CAS [432] use sampling techniques to choose a subset of k-mers from all k-mers of a read without significantly reducing their accuracy. For example, minimap2 uses the minimizer sketching technique [415] to reduce overall storage (and memory) space requirements while also improving performance. These works use low-collision hash functions that require exact matches of seeds generated after their sketching mechanisms. As discussed in Section 2.2.3, using low-collision hash functions provides certain challenges when determining parameters that impact performance and accuracy. Although it is important to reduce the collision rate for assigning different hash values for dissimilar seeds for accuracy and performance reasons, choosing low-collision hash functions also makes it unlikely to assign the same hash value for similar seeds. Thus, seeds must exactly match to find matches between sequences with a single lookup.

Mitigating this exact-matching requirement allows similar seeds to share the same hash value, thereby significantly improving the performance and sensitivity of applications. These

applications can then tolerate substitutions and indels at arbitrary positions during seed matching, a capability we refer to as *fuzzy seed matching*.

There are typically two directions for tolerating mismatches (i.e., substitutions and indels) when finding seed matches: 1) mechanisms that tolerate substitutions at fixed positions [304, 307–309,315,435,436,441,455–457] and 2) mechanisms that tolerate substitutions and indels [310, 419, 856]. To our knowledge, our hash-based fuzzy seeding approach (explained in Chapter 4) is the first work that can efficiently find fuzzy matches that can allow arbitrary mismatches between seeds.

3.1.1 Spaced Seeds to Tolerate Substitutions

To allow substitutions when matching k-mers, a common approach is to *mask* certain characters of k-mers extracted from target and query sequences [304, 307–309, 315, 435, 436, 441, 455–457]. These masks are used as *don't care* characters such that the substitutions that appear at these positions are ignored. Predefined *patterns* determine the *fixed* masking positions for a given input k-mer. Seeds generated from masked k-mers are known as *spaced seeds* [315].

Tools such as ZOOM! [308] and SHRiMP2 [309] use spaced seeds to improve sensitivity when mapping short reads (i.e., Illumina paired-end reads). S-conLSH [304, 307] generates many spaced seeds from each k-mer using different masking patterns to improve the sensitivity when matching spaced seeds with locality-sensitive hashing techniques. There have been recent improvements in determining the masking patterns to improve the sensitivity of spaced seeds [435,436]. However, spaced seeds are limited to tolerating substitutions at certain positions because these techniques use 1) fixed patterns that allow substitutions only at certain positions of k-mers and 2) *low-collision hashing* techniques used for finding *only* exactly matching spaced seeds.

3.1.2 Linked k-mers to Tolerate Substitutions and Indels

Linking k-mers aims to allow both substitutions and indels when matching k-mers. A common approach is to select a few k-mers from an input sequence and link them to use these linked k-mers as seeds, such as paired-minimizers [419] and strobemers [310, 856] as Figure 3.1 shows. These approaches can ignore large gaps between the linked k-mers ❶. For example, the strobemer technique links ❷ a subset of selected k-mers of a sequence (e.g., by finding minimizers of the input sequence) to generate a strobemer sequence ❸, which is used as a seed. Strobemers enable masking some characters within sequences without requiring a fixed pattern, unlike spaced k-mers. This makes strobemers a more sensitive approach for detecting indels with varying lengths as well as substitutions. However, the nature of the hash function used in strobemers requires exact matches of *all* concatenated k-mers in strobemer sequences when matching seeds. Such an exact match requirement introduces challenges for

further improving the sensitivity of strobemers for detecting indels and substitutions between sequences.

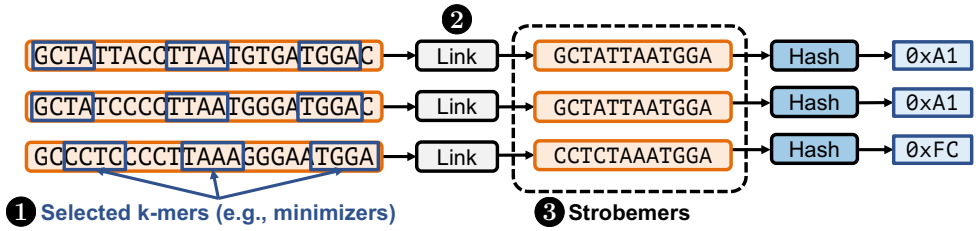

Figure 3.1: A simple example of the strobemers technique with three different input sequences. Sequences between selected k-mers are ignored to tolerate insertions and deletions. The first two input sequences generate the same hash values as output after generating their strobemer seeds even though these input sequences are not exactly matching.

We find that the existing approaches used for tolerating substitutions and indels require exact matches of the resulting seeds that these mechanisms generate (i.e., spaced seeds or strobemer seeds). Due to the limited parameter choices when finding only exactly matching seeds, the selected seeds either 1) increase the use of the computationally costly steps after seeding or 2) provide limited sensitivity. In Chapter 4, we explain our mechanism to find fuzzy seed matches while enabling highly similar seeds to mismatch at arbitrary positions.

3.2 Real-Time Analysis During Sequencing

Our works that can analyze raw nanopore signals without basecalling (as we discuss in Chapters 5, 6, and 7) aim to provide fast, scalable, accurate, and real-time genome analysis. There are several prior works that perform real-time analysis using 1) the Sequencing by Synthesis (SBS) technology with Illumina sequencers [1007–1013], 2) basecallers for nanopore sequencing data to perform basecalled sequence analysis [369, 370, 680, 786–790], and 3) raw nanopore signal analysis without basecalling [138, 264–266, 290–302]. Specialized hardware accelerators are commonly employed to perform real-time analysis [138, 267–283, 297, 300–302, 663, 669–671, 680–684, 692, 711, 721, 722].

To show the real-time analysis capabilities of other sequencing technologies, we discuss the works that can perform real-time analysis using the Sequencing by Synthesis (SBS) technology with Illumina sequencers [1007–1013] in Section 3.2.1.

We explain the works that perform real-time analysis with basecalling in Section 3.2.2. We argue these works are not scalable as they rely on computationally costly basecallers.

In Section 3.2.3, we show that there is a large body of recent work that focuses on the real-time analysis of raw nanopore signals without basecalling. We 1) describe the two prior works most related to our works we describe in Chapters 5, 6 and 7 and 2) explain the limitations of these prior works.

3.2.1 Real-time Analysis with SBS

There are multiple works [1007–1013] that analyze the raw sequencing data that SBS generates, specifically from Illumina sequencers, by basecalling the raw sequencing data from SBS followed by read mapping to reduce the overall genome analysis latency. HiLive [1007] and HiLive2 [1008] perform basecalling after a certain amount of sequencing cycles to enable accurate basecalling, followed by read mapping. Real-time basecalling and mapping can be used for many use cases, such as k-mer based taxonomic classification in LiveKraken [1009], variant calling [1008], molecular diagnosis for rapid treatment [1010], pathogen identification [1011, 1012], and privacy-preserving analysis [1013].

Our works described in Chapters 5, 6, and 7 are different from these works as our works focus on analyzing raw nanopore sequencing data to exploit the unique opportunities that nanopore sequencing provides.

3.2.2 Basecalled Real-time Analysis with Nanopore Sequencing

Various works [369, 370, 680, 786–790] perform real-time basecalling and further analysis using the raw nanopore signals generated during sequencing. Among these works, ReadFish [786], ReadBouncer [680], and RUBRIC [787] perform basecalling followed by read mapping. SPUMONI [369] and SPUMONI 2 [370] use basecalled sequences to perform binary classification between target and non-target species in a sample containing a large collection of genomes. To achieve this classification with a space-efficient mechanism, these tools utilize the r-index data structure [379, 385, 386] for indexing multiple genomes. Coriolis [788] enables performing real-time analysis for metagenomics classification in mobile devices after performing basecalling.

However, all of these works predominantly rely on GPUs [267, 268, 270, 272–284, 681, 721] for performing real-time basecalling during sequencing due to the high computational demands of DNN-based basecallers. This reliance presents significant challenges for achieving scalable, energy-efficient, and portable on-site analysis, particularly in resource-constrained environments. Additionally, DNN-based basecallers are mainly optimized to use long chunks of raw nanopore signal produced after a few seconds. Since stopping the sequencing decisions must made quickly in adaptive sampling (as we discuss in Section 2.1.3.3), using raw signals generated within a second or less than a second with these basecallers provides less accurate basecalling and analysis [291, 292].

3.2.2.1 Accelerating Basecalling

Basecallers developed for nanopore sequencing mainly use deep neural networks (DNNs) [267–284] to achieve high accuracy. However, the use of complex deep learning models

makes basecalling slow and memory-hungry [275, 684]. To accelerate basecalling, various algorithm-architecture co-design techniques have been proposed [267–283, 663, 669–671, 680–684, 692, 711, 721, 722]. First, to provide faster and more energy-efficient computations, multiple works use custom logic design to accelerate the basecalling operations on reconfigurable (i.e., FPGAs) or specialized (i.e., ASICs) hardware [269, 669–671, 683, 692]. Second, to perform a large number of arithmetic operations that are independent of each other (e.g., multiplications of many single-precision floating point operations), a large body of DNN-based basecallers exploit the massive parallelism that GPUs provide [267, 268, 270, 272–284, 681, 721]. To optimize these GPU-based basecallers, various works aim to reduce the unnecessary computations by 1) reducing the DNN parameters and precision [275] or 2) introducing pre-basecalling filters [267]. Third, PIM approaches [663, 682, 684, 711] can substantially accelerate DNN operations by performing 1) analog computation in-memory, 2) minimizing the data movement latency, and 3) exploiting large internal parallelism in memory arrays.

3.2.3 Real-time Raw Nanopore Signal Analysis

A number of works perform real-time genome analysis of raw nanopore signals by utilizing adaptive sampling [138, 264–266, 290–302].

SquiggleNet [290], DeepSelectNet [293], and RawMap [298] use machine learning techniques to classify raw nanopore signals to certain species without performing read mapping. Sigmoni [299] uses a similar strategy used in SPUMONI [369] and SPUMONI 2 [370] for classification without basecalling.

UNCALLED [292] and Sigmap [291] are the most relevant works to our raw nanopore signal analysis works (explained in Chapters 5, 6, and 7). These works map raw nanopore signals to a reference genome *without* using computationally costly basecallers.

To map raw signals to a reference genome, UNCALLED [292] calculates the probability of k-mers that each raw signal segment (i.e., event) can represent using a k-mer model that provides the expected event values for each possible k-mer. UNCALLED identifies the sequence of matching k-mers between the most probable k-mers of events and a reference genome using an FM-index [366]. However, as genome size increases, this probabilistic model struggles to accurately identify matching regions due to the sheer number of potential matches [292]. While UNCALLED remains highly accurate for small genomes (e.g., *E. coli* and *Yeast*), its accuracy declines significantly when applied to larger genomes, such as human genomes (as we demonstrate in Chapters 5 and 6).

To accurately map raw nanopore signals to genomes larger than *Yeast*, Sigmap [291] calculates the Euclidean distance between raw signals and a reference genome after converting the reference genome into its expected signal representation by using k-mer models. Distance

calculation enables identifying a segment of raw signals that are close in value to a particular segment in the reference genome. Identifying segments with a small distance between them can tolerate noise in raw signals, as a pair raw signal segment generated from the same identical content does not exactly match due to noise (as we discuss in Section 2.1.3) but expected to be close in value [291]. To calculate Euclidean distance between signal segments, Sigmap creates a vector from each n consecutive event values (i.e., n-dimensional vector space) from the reference genome (i.e., the indexing step) and measures the Euclidean distance between these vectors and the vectors generated from raw nanopore signals (i.e., the mapping step) using a k-d tree structure [1014]. While calculating the distance between raw signals and a reference genome can provide accurate mapping positions, this distance calculation is computationally expensive and suffers from the *curse of dimensionality* [1015]. This issue makes it increasingly difficult to efficiently scale the number of events within a single vector, limiting accuracy for large genomes. Increasing the dimension of vectors is needed to reduce the number of mapping positions in the reference genome as the genome size increases, which can substantially impact the overall runtime and accuracy of mapping.

We find that none of these earlier works can provide accurate and fast analysis for mapping raw nanopore signals to large genomes such as a human genome in real-time. Our work we discuss in Chapter 5 provides the first mechanism to map raw nanopore signals in real-time accurately and quickly to large genomes without basecalling.

3.2.3.1 Accelerating Raw Signal Analysis

To accelerate the costly computations that are commonly used in the raw signal analysis (e.g., alignment using dynamic time warping [1016]), several works [138, 297, 300–302] propose hardware-algorithm co-design. SquiggleFilter [138] provides an ASIC accelerator that quickly filters non-related raw electrical signals before basecalling for viral detection. HARU [297] is an FPGA accelerator that accelerates real-time selective genome sequencing on resource-constrained devices for detecting viral genomes. There are various other works that use GPUs [300,301] to accelerate raw signal analysis. All of these works mainly focus on accelerating the costly signal alignment algorithm known as dynamic time warping (DTW).

These works are orthogonal to the works we describe in Chapters 5 and 6, as the alignment step can be performed separately after quickly and accurately identifying the mapping positions for a raw signal.

Chapter 4

Tolerating Arbitrary Noise with Hash-based Fuzzy Seeding

In this chapter, we 1) explain the types of noise we find in seed matching that impact the sensitivity of sequence analysis and 2) introduce a mechanism that utilizes a noise-tolerant hashing mechanism to find fuzzy seed matching with single hash value lookups between seeds.

4.1 Background and Motivation

High-throughput sequencing (HTS) technologies have revolutionized the field of genomics due to their ability to produce millions of nucleotide sequences at a relatively low cost [248]. Although HTS technologies are key enablers of almost *all* genomics studies [533,814,1017–1020], HTS technology-provided data comes with two key shortcomings. First, HTS technologies produce short genome fragments called *reads*, which cover only a small region of a genome and contain from around one hundred to a million bases, depending on the technology [248]. Second, HTS technologies can misinterpret signals during sequencing and thus provide reads that contain *sequencing errors* [1021]. The average frequency of sequencing errors in a read highly varies from 0.1% up to 15% depending on the HTS technology [255, 1022–1025]. To address the shortcomings of HTS technologies, various computational approaches must be taken to process the reads into meaningful information accurately and efficiently. These include 1) read mapping [305,306,502,503,1026], 2) *de novo* assembly [505,506,1027], 3) read classification in metagenomic studies [571, 572, 1028], 4) correcting sequencing errors [257, 285, 1029].

At the core of these computational approaches, identifying sequence similarities through seed matching is essential to overcome the limitations of HTS technologies. However, identifying the similarities across *all* pairs of sequences is not practical due to the costly algorithms used to calculate the distance between two sequences, such as sequence alignment algorithms

using dynamic programming (DP) approaches [258, 261]. To practically identify similarities, it is essential to avoid calculating the distance between dissimilar sequence pairs. A common heuristic is to find matching *short* sequence segments, called *seeds*, between sequence pairs by using a hash table [2, 265, 266, 304–361]. Sequences that have no or few seed matches are quickly filtered out from performing costly sequence alignment. There are several techniques that generate seeds from sequences, known as *seeding techniques*. To find the matching seeds efficiently, a common approach is to match the hash values of seeds with a *single lookup* using a hash table that contains the hash values of all seeds of interest. The use of seeds drastically reduces the search space from all possible sequence pairs to similar sequence pairs to facilitate efficient distance calculations over many sequence pairs [415, 423, 854].

4.1.1 Motivation: Need for Tolerating Arbitrary Noise in Seeding

To our knowledge, there is no work that can *efficiently* find fuzzy matches of seeds *without* requiring 1) *exact matches* of all k-mers (i.e., any k-mer can mismatch) and 2) imposing high performance and memory space overheads. In this work, we observe that existing works have such a limitation mainly because they employ hash functions with low-collision rates when generating the hash values of seeds. Although it is important to reduce the collision rate for assigning different hash values for dissimilar seeds for accuracy and performance reasons, the choice of hash functions also makes it unlikely to assign the same hash value for similar seeds. Thus, seeds *must* exactly match to find matches between sequences with a single lookup. Mitigating this exact-matching requirement allows similar seeds to share the same hash value, significantly improving performance and sensitivity. This enables applications to tolerate substitutions and indels at arbitrary positions when matching seeds.

4.1.2 Overview of Fuzzy Seed Matching in BLEND

Our goal in this work is to enable finding *fuzzy* matches of seeds as well as exactly matching seeds between sequences (e.g., reads) with a single lookup of hash values of these seeds. To this end, we propose *BLEND*, the *first* efficient and accurate mechanism that can identify both exactly matching and highly similar seeds with a single lookup of their hash values. The **key idea** in BLEND is to assign the same hash value to highly similar seeds. To achieve this, BLEND 1) leverages the SimHash technique [433, 434], and 2) introduces mechanisms that adapt hash-based seeding techniques to SimHash for finding fuzzy seed matches with a single hash lookup. This provides us with two key benefits. First, BLEND can generate the same hash value for highly similar seeds *without* imposing exact matches of seeds, unlike existing seeding mechanisms that use hash functions with low-collision rates. Second, BLEND enables finding fuzzy seed matches with a single lookup of a hash value rather than 1) using various permutations to find the longest prefix matches [1030] or 2) matching many hash values

for calculating costly similarity scores (e.g., Jaccard similarity [1031]) that the conventional locality-sensitive hashing-based methods use, such as MHAP [416] or S-conLSH [304, 307]. These two ideas ensure that BLEND can efficiently find both 1) all exactly matching seeds that a seeding technique finds using a conventional hash function with a low-collision rate and 2) approximate seed matches that these conventional hashing mechanisms cannot find with a single lookup of a hash value.

Figure 4.1 shows two examples of how BLEND can replace the conventional hash functions that the seeding techniques use. The **key challenge** is to accurately and efficiently define the vector items derived from seeds for the SimHash technique. To address this, BLEND provides two mechanisms for converting seeds into vectors of items: 1) `BLEND-I` and 2) `BLEND-S`. To perform a sensitive detection of substitutions, `BLEND-I` uses all overlapping smaller k-mers of a potential seed sequence as the items of a vector for generating the hash value with SimHash. To allow mismatches between the linked k-mers that strobemers and similar seeding mechanisms use, `BLEND-S` uses only the linked k-mers as the vector with SimHash. We envision that BLEND can be integrated with any seeding technique that uses hash values for matching seeds with a single lookup by replacing their hash function with BLEND and using the proper mechanism for converting seeds into a vector of items.

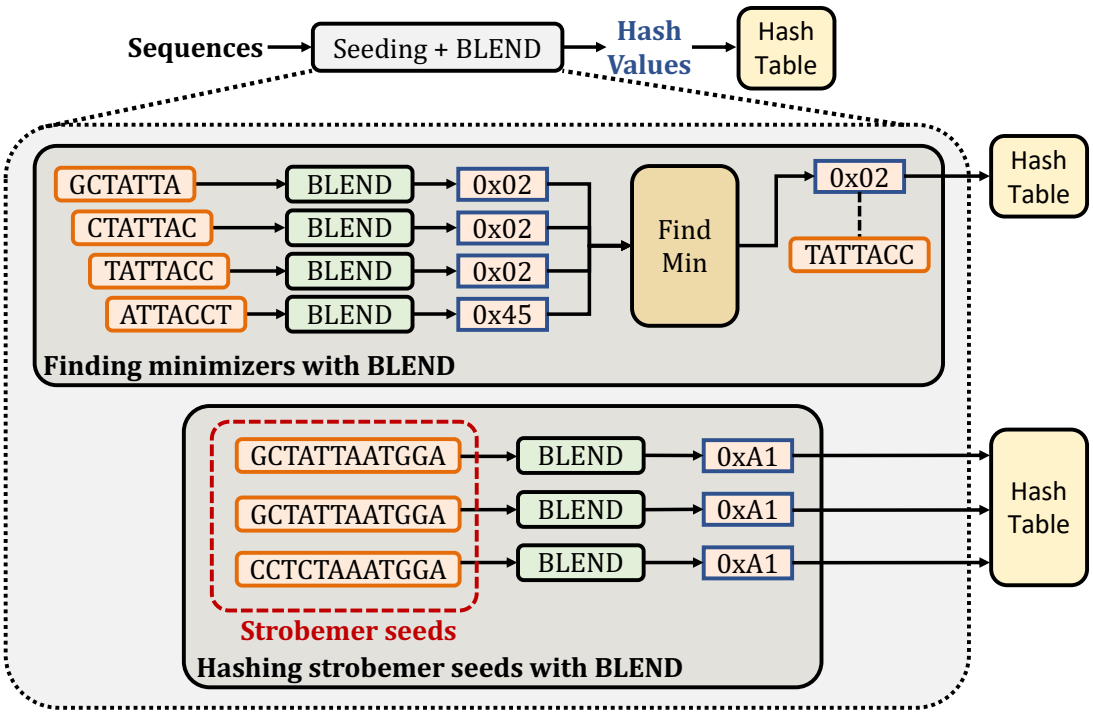

Figure 4.1: Replacing the hash functions in seeding techniques with BLEND.

4.1.3 Overview of Experimental Studies

Using erroneous (ONT and PacBio CLR), highly accurate (PacBio HiFi), and short (Illumina) reads, we experimentally show the benefits of BLEND on two important applications in genomics: 1) read overlapping and 2) read mapping. First, read overlapping identifies overlaps between all pairs of reads based on seed matches. These overlaps are crucial for generating assemblies of the sequenced genome [305, 1032]. We compare BLEND with minimap2 and MHAP by finding overlapping reads. We then generate the assemblies from the overlapping reads to compare the qualities of these assemblies. Second, read mapping uses seeds to find similar portions between a reference genome and a read before performing the read alignment. Aligning a read to a reference genome shows the edit operations (i.e., match, substitution, insertion, and deletions) to make the read identical to the portion of the reference genome, which is useful for downstream analysis (e.g., variant calling [1033]). We compare BLEND with minimap2, LRA [311], Winnowmap2, S-conLSH, and Strobealign by mapping long and paired-end short reads to their reference genomes. We evaluate the impact of long read mapping on downstream analysis by identifying structural variants (SVs) and assessing their detection accuracy.

4.2 BLEND Mechanism

We propose ***BLEND***, a mechanism that efficiently finds highly similar (fuzzy) seed matches using a single hash value lookup. BLEND allows for assigning identical hash values to highly similar seeds and integrates with hash-based seeding approaches (e.g., minimizers or strobemers) to find fuzzy seed matches with a single lookup.

Figure 4.2 shows the overview of steps to find fuzzy seed matches with a single lookup in three steps. First, BLEND starts with converting the input sequence it receives from a seeding technique (e.g., a strobemer sequence in Figure 4.1) to its vector representation as the SimHash technique generates the hash value of the vector using its items ❶. To enable effective and efficient integration of seeds with the SimHash technique, BLEND proposes two mechanisms for identifying the items of the vector of the input sequence: 1) `BLEND-I` and 2) `BLEND-S`. Second, after identifying the items of the vector, BLEND uses this vector with the SimHash technique to generate the hash value for the input sequence ❷. BLEND uses the SimHash technique as it allows for generating the same hash value for highly similar vectors. Third, BLEND uses the hash tables with the hash values it generates to enable finding fuzzy seed matches with a single lookup of their hash values ❸.

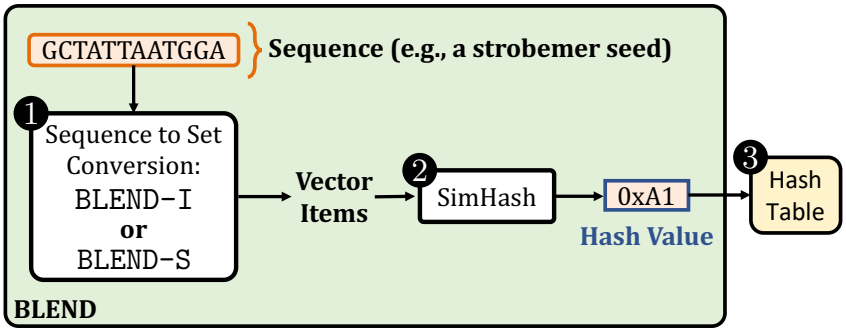

Figure 4.2: Overview of BLEND. ❶ BLEND uses `BLEND-I` or `BLEND-S` for converting a sequence into its vector of items. ❷ BLEND generates the hash value of the input sequence using its vector of items with the SimHash technique. ❸ BLEND uses hash tables for finding fuzzy seed matches with a single lookup of the hash values that BLEND generates.

4.2.1 Sequence to vector Conversion

Our goal is to convert the input sequences that BLEND receives from any seeding technique (Figure 4.1) to their proper vector representations so that BLEND can use the items of vectors for generating the hash values of input sequences with the SimHash technique. To achieve effective and efficient conversion of sequences into their vector representations in different scenarios, BLEND provides two mechanisms: 1) `BLEND-I` and 2) `BLEND-S`, as we show in Figures 4.3 and 4.4, respectively.

The goal of the first mechanism, `BLEND-I`, is to provide high sensitivity for a single character change in the input sequences that seeding mechanisms provide when generating their hash values such that two sequences are likely to have the same hash value if they differ by a few characters. `BLEND-I` has three steps. First, `BLEND-I` extracts *all* the overlapping k-mers of an input sequence, as shown in Figure 4.3. For simplicity, we use the *neighbors* term to refer to all the k-mers that `BLEND-I` extracts from an input sequence. Second, `BLEND-I` generates the hash values of these k-mers using any hash function. Third, `BLEND-I` uses the hash values of the k-mers as the vector items of the input sequence for SimHash. Although `BLEND-I` can be integrated with any seeding mechanism, we integrate it with the minimizer seeding mechanism, as shown in Figure 4.1 as proof of work.

The goal of the second mechanism, `BLEND-S`, is to allow indels and substitutions when matching the sequences such that two sequences are likely to have the same hash value if these sequences differ by a few k-mers. `BLEND-S` has three steps. First, `BLEND-S` uses *only* the selected k-mers that the strobemer-like seeding mechanisms find and link [310] as neighbors, as shown in Figure 4.4. `BLEND-S` allows for mismatches between linked k-mers in strobemer sequences because a single character difference does not propagate to other linked k-mers. This is unlike `BLEND-I`, where a single character difference affects multiple overlapping k-mers.

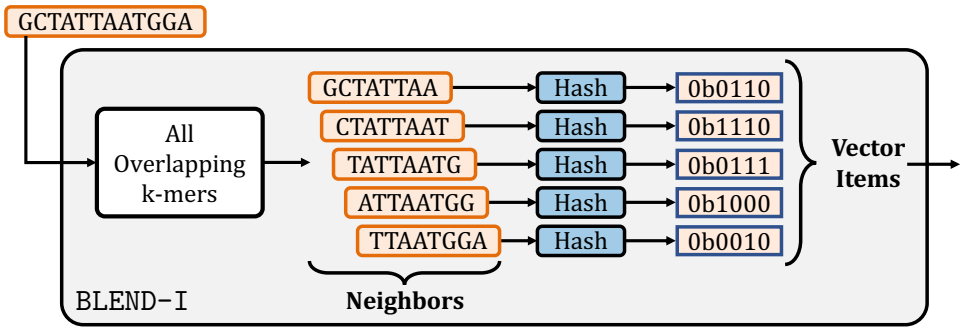

Figure 4.3: Overview of BLEND-I. BLEND-I uses the hash values of all the overlapping k-mers of an input sequence as the vector items.

To ensure the correctness of strobemer seeds when matching them based on their hash values, BLEND-S uses *only* the selected k-mers from the same strand. Second, BLEND-S generates the hash values of these linked k-mers using any hash function. Third, BLEND-S uses the hash values of all such selected k-mers as the vector items of the input sequence for SimHash.

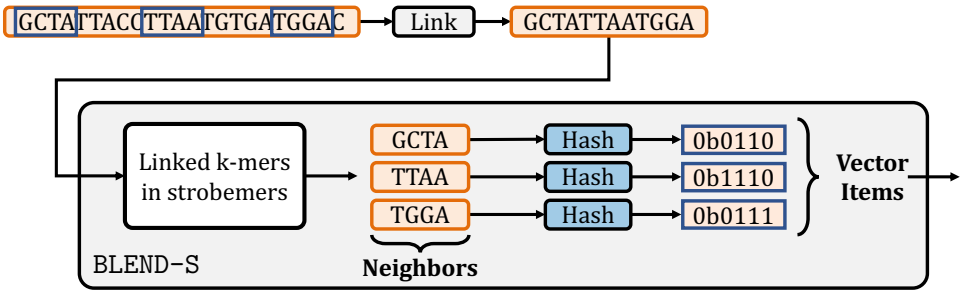

Figure 4.4: Overview of BLEND-S. BLEND-S uses the hash values of only the k-mers selected by the strobemer seeding mechanism.

4.2.2 Integrating the SimHash Technique

Our goal is to enable efficient comparisons of equivalence or high similarity between seeds with a single lookup by generating the same hash value for highly similar or equivalent seeds. To enable generating the same hash value for these seeds, BLEND uses the SimHash technique [433]. The SimHash technique takes a vector of items and generates a hash value for the vector using its items. The key benefit of the SimHash technique is that it allows generating the same hash value for highly similar vectors while enabling any *arbitrary* items to mismatch between vectors. To exploit the key benefit of the SimHash technique, BLEND efficiently and effectively integrates the SimHash technique with the vector items that BLEND-I or BLEND-S determine. BLEND uses these vector items for generating the hash values of seeds such that highly similar seeds can have the same hash value to enable finding fuzzy seed matches with a single lookup of their hash values.

BLEND employs the SimHash technique in three steps: 1) encoding the input vector items as bit vectors, 2) performing additions for the bit vector, resulting in a counter vector, and 3) decoding the counter vector to generate the hash value for the input vector that `BLEND-I` or `BLEND-S` determine, as we show in Figure 4.5. To enable efficient computations between vectors, BLEND uses SIMD operations when performing all these three steps. We provide the details of our SIMD implementation in Section 4.2.3.

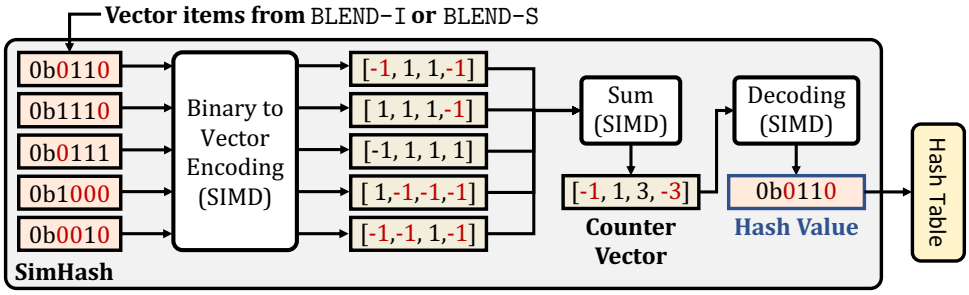

Figure 4.5: The overview of the steps in the SimHash technique for calculating the hash value of a given vector of items. The vector items are the hash values represented in their binary form. Binary to Vector Encoding converts these vector items to their corresponding bit vector representations. Sum performs the vector additions and stores the result in a separate vector that we call the *counter vector*. Decoding generates the hash value of the vector based on the values in the counter vector. BLEND uses SIMD operations for these three steps, as indicated by SIMD. We highlight in red how 0 bits are converted and propagated in the SimHash technique.

First, the goal of the *binary to vector encoding* step is to transform all the hash values of vector items from the binary form into their corresponding bit vector representations so that BLEND can efficiently perform the bitwise arithmetic operations that the SimHash technique uses in the vector space. For each hash value in the vector item, the encoding can be done in two steps. The first step creates a bit vector with n elements corresponding to an n-bit hash value. Initially, all elements in the bit vector are set to 1. For each bit position t of the hash value, the second step assigns -1 to the t^{th} element in the bit vector if the bit at position t is 0, as we highlight in Figure 4.5 with red colors of 0 bits and their corresponding -1 values in the vector space. For each hash value in vector items, the resulting bit vector includes 1 for the positions where the corresponding bit of a hash value is 1 and -1 for the positions where the bit is 0.

Second, the goal of the vector addition operation is to determine the bit positions where the number of 1 bits is greater than the number of 0 bits among the vector items, which we call determining the *majority* bits. The key insight in determining these majority bits is that highly similar vectors are likely to result in *similar* majority results because a few differences between two similar vectors are unlikely to change the majority bits at each position, given that there is a sufficiently large number of items involved in this majority calculation. To efficiently

determine the majority of bits at each position, BLEND counts the number of 1 and 0 bits at a position by using the bit vectors it generates in the vector encoding step, as shown with the addition step (Sum) in Figure 4.5. The vector addition step adds +1 or −1 values between bit vector elements and stores the results in a separate *counter* vector. The values in this counter vector show the majority of bits at each position of the vector items. Since BLEND assigns −1 for 0 bits and 1 for 1 bits, the majority of bits at a position is either 1) 1 if the corresponding value in the counter vector is greater than 0 or 2) 0 if the values are less than or equal to 0.

Third, to generate the hash value of a vector, BLEND uses the majority of bits that it determines by calculating the counter vector. To this end, BLEND decodes the counter vector into a hash value in its binary form, as shown in Figure 4.5 with the decoding step. The decoding operation is a simple conditional operation where each bit of the final hash value is determined based on its corresponding value at the same position in the counter vector. BLEND assigns the bit either 1) 1 if the value at the corresponding position of the counter vector is greater than 0 or 2) 0 if otherwise. Thus, each bit of the final hash value shows the majority voting result of input vector items of a seed. We use this final hash value for the input sequence that the seeding techniques provide because highly similar sequences are likely to have many characters or k-mers in common, which essentially leads to generating *similar* vector items by using `BLEND-I` or `BLEND-S`. Properly identifying the vector items of similar sequences enables BLEND to find similar majority voting results with the SimHash technique, which can lead to generating the same final hash value for similar sequences. This enables BLEND to find fuzzy seed matches with a single lookup using these hash values. We provide a step-by-step example of generating the hash values for two different seeds in Supplementary Section S4.1 and Supplementary Tables S4.1- S4.8.

4.2.3 SIMD Implementation of the SimHash Technique

Figure 4.6 shows the high-level execution flow when calculating the hash value of a seed from its vector items that `BLEND-I` or `BLEND-S` identifies, as explained in Sections 4.2.1 and 4.2.2. To efficiently perform the bitwise operations in the SimHash technique, BLEND utilizes the SIMD operations in three steps.

First, for each *hash value* in vector items, BLEND creates its corresponding *mask* using the `movemask_inverse` function, as shown in ❶. For each bit position t of the hash value, the `movemask_inverse` function assigns the bit at position t of the hash value to the bit position $t*8+7$ of a 256-bit SIMD register (i.e., the most significant bit positions of each 8-bit block), which BLEND uses it as a *mask* in the next steps. We assume 0-based indexing and the mask register is initially 0. Figure 4.7 shows how each bit in hash value propagates to the mask register in ❶. The `movemask_inverse` is an in-house implementation that performs the reverse behavior of

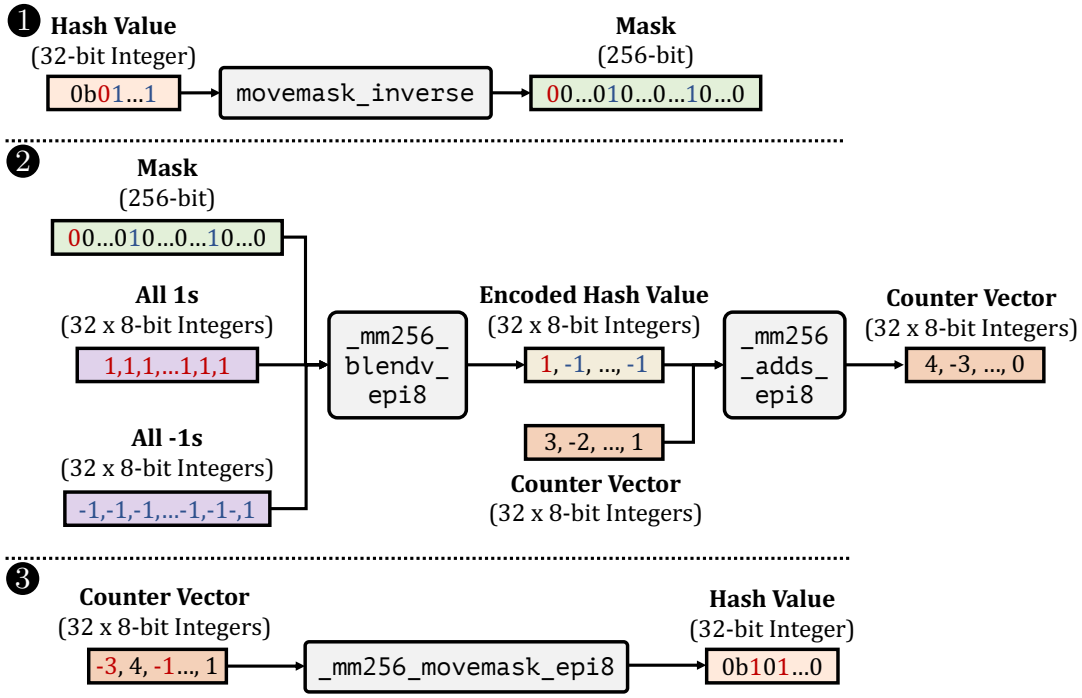

Figure 4.6: SIMD execution flow when generating the hash value of a seed from its vector items that `BLEND-I` or `BLEND-S` identifies. Colors highlight the propagation of bits and values to the outputs of functions.

the `_mm256_movemask_epi8`[1] SIMD function. Our function efficiently utilizes several other SIMD functions to perform the reverse behavior of `_mm256_movemask_epi8`.

Second, for each mask created in the first step, BLEND updates the values in the counter vector (explained in Section 4.2.2), as shown in ❷. To encode the hash value into its bit vector representation, BLEND uses the `_mm256_blendv_epi8`[2] SIMD function with 1) the mask register BLEND creates in the first step, 2) two 256-bit wide SIMD registers that include 32× 8-bit integers. For the first register, all 8-bit values are initialized to 1, and for the second register, all 8-bit values are initialized to -1. The `_mm256_blendv_epi8` function generates a new 256-bit register with 8-bit integers where each 8-bit block is copied from either the first or the second register based on the most significant value in the mask register. If the most significant value in the mask register is 0, the corresponding 8-bit block in the first register is copied. Otherwise, the 8-bit block in the second register is copied. We show in detail how the values in these registers propagate to the resulting 256-bit register in Figure 4.7 in ❷. BLEND, then, performs addition using the `_mm256_adds_epi8`[3] function between the register that the `_mm256_blendv_epi8`

[1] https://software.intel.com/sites/landingpage/IntrinsicsGuide/#text=_mm256_movemask_epi8&ig_expand=4874

[2] https://software.intel.com/sites/landingpage/IntrinsicsGuide/#text=_mm256_blendv_epi8&ig_expand=515

[3] https://software.intel.com/sites/landingpage/IntrinsicsGuide/#text=_mm256_adds_epi8&ig_expand=220

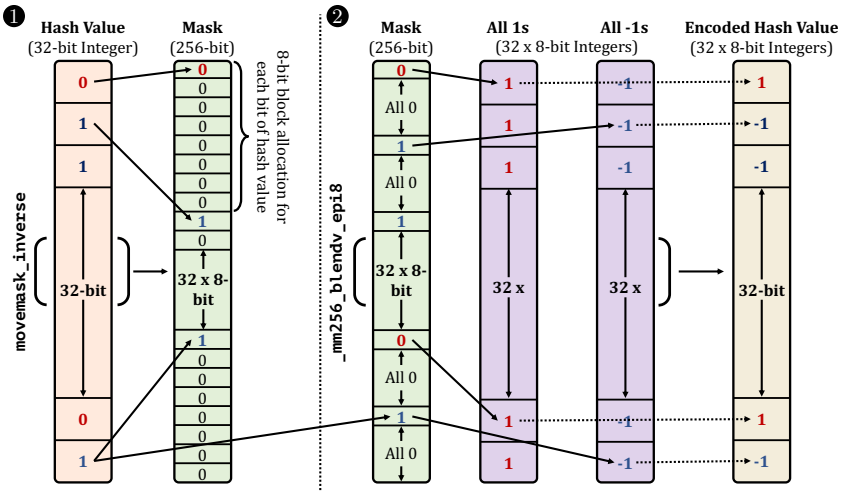

Figure 4.7: Details of the `movemask_inverse` and `_mm256_blendv_epi8` executions. Colors and arrows highlight the propagation of bits and values to the outputs of functions.

function generates and the 256-bit *counter vector* that includes 32× 8-bit integers. We assume that all the 8-bit values in the counter vector are initially 0. BLEND keeps updating the counter vector as it iterates through the vector items (i.e., hash values). The resulting value is written back to the counter vector to use it in the next iterations with the next vector item. We note that the current design encodes 1 bits as -1 and 0 bits as 1, which is the opposite case of our explanation in Section 4.2.2. Although we perform our encoding in an opposite way, BLEND generates the final hash values as originally explained. The reason for such a design change is due to the behavior of the function we use in the third step.

Third, BLEND converts the final result in the counter vector to its corresponding 32-bit hash value of a vector (explained in the decoding step of Section 4.2.2), as shown in ❸. BLEND uses the `_mm256_movemask_epi8` function that takes a 256-bit register of 8-bit blocks and assigns the corresponding bit accordingly in a 32-bit value. The behavior of this function is essentially the reverse behavior of the `movemask_inverse` function that we explain in the first step (i.e., reversing the arrows in Figure 4.7 ❶ can simulate the `_mm256_movemask_epi8` function). Thus, it assigns 1 to the bit position t of a hash value if the bit at position $t * 8 + 7$ (i.e., the most significant bit of an 8-bit integer) is 1. Since the most significant is 1 *only* for the negative values according to the signed integer value convention, this function creates the opposite behavior of our decoding step we explain in Section 4.2.2. We resolve this issue by performing the encoding in an opposite way in the second step, where the resulting counter vector includes negative values when the majority of bits at a bit position is 1. Thus, the final hash value contains the same bits as explained in our main paper.

Although we omit the details here, BLEND avoids performing redundant computations when calculating the hash values of each input sequence, as the vector items between each of

these input sequences are likely to be shared. For example, minimap2 generates a hash value for each k-mer in a sequence and selects the k-mer with the minimum hash value in a window of k-mers as the minimizer. Assuming that the vector items of each k-mer are l-mers, each k-mer differs by two l-mers at most with its next overlapping k-mer: one different l-mer contains the leading character of the first k-mer, and the other contains the trailing character of the next k-mer. Since there are two l-mer changes at most, BLEND calculates only the difference between subsequent k-mers. Thus, BLEND keeps a buffer in a first-in, first-out fashion such that the corresponding encoded hash value of the l-mer that is missing in the next k-mer is subtracted from the counter vector and popped from the queue while performing an addition *only* for the l-mer that is missing from the previous k-mer and pushing it into the queue.

We perform our operations on 256-bit wide SIMD registers. BLEND works on 8-bit integer blocks assigned for each bit in a hash value. Since our registers are 256-bit wide, BLEND uses 32-bit hash values when calculating the SimHash value of a seed. Our implementation allows working on up to 64-bit hash values by dividing the most and least significant 32 bits into two 32-bit hash values. Each 32-bit hash value can independently follow the three steps we show in Figure 4.6, and the final 64-bit value can be generated by the shift operations between the final 32-bit hash values. Although our approach is scalable to allow hash values with a larger number of bits, the current implementation does not support such flexible scaling and works on up to 64-bits.

4.2.4 Using a Hash Table for Quick Fuzzy Seed Matching

Our goal is to enable an efficient lookup of the hash values of seeds to find fuzzy seed matches with a single lookup. To this end, BLEND uses hash tables in two steps. First, BLEND stores the hash values of all the seeds of target sequences (e.g., a reference genome) in a hash table, usually known as the *indexing* step. Keys of the hash table are hash values of seeds, and the value that a key returns is a *list* of metadata information (i.e., seed length, position in the target sequence, and the unique name of the target sequence). BLEND keeps minimal metadata information for each seed sufficient to locate seeds in target sequences. Since similar or equivalent seeds can share the same hash value, BLEND stores these seeds using the same hash value in the hash table. Thus, a query to the hash table returns all fuzzy seed matches with the same hash value.

Second, BLEND iterates over all query sequences (e.g., reads) and uses the hash table from the indexing step to find fuzzy seed matches between query and target sequences. The query to the hash table returns the list of seeds of the target sequences that have the same hash value as the seed of a query sequence. Thus, the list of seeds that the hash table returns is the list of fuzzy seed matches for a seed of a query sequence as they share the same hash value. BLEND

can find fuzzy seed matches with a single lookup using the hash values it generates for the seeds from both query and target sequences.

BLEND finds fuzzy seed matches mainly for two important genomics applications: read overlapping and read mapping. For these applications, BLEND stores all the list of fuzzy seed matches between query and target sequences to perform *chaining* among fuzzy seed matches that fall in the same target sequence (overlapping reads) optionally, followed by alignment (read mapping) as described in minimap2 [306].

4.3 Results

4.3.1 Evaluation Methodology

We replace the mechanism in minimap2 that generates hash values for seeds with BLEND to find fuzzy seed matches when performing end-to-end read overlapping and read mapping. We also incorporate the `BLEND-I` and `BLEND-S` mechanisms in the implementation and provide the user to choose either of these mechanisms when using BLEND. We provide a set of default parameters we optimize based on sequencing technology and the application to perform (e.g., read overlapping). We explain the details of the BLEND parameters in Supplementary Table S4.15 and the parameter configurations we use for each tool and dataset in Supplementary Tables S4.16 and S4.17. We determine these default parameters empirically by testing the performance and accuracy of BLEND with different values for some parameters (i.e., k-mer length, number of k-mers to include in a seed, and the window length) as shown in Supplementary Table S4.12. We show the trade-offs between the seeding mechanisms `BLEND-I` and `BLEND-S` in Supplementary Figures S4.1 and S4.2 and Supplementary Tables S4.9 - S4.11 regarding their performance and accuracy.

For our evaluation, we use real and simulated read datasets as well as their corresponding reference genomes. We list the details of these datasets in Table 4.1. To evaluate BLEND in several common scenarios in read overlapping and read mapping, we classify our datasets into three categories: 1) highly accurate long reads (i.e., PacBio HiFi), 2) erroneous long reads (i.e., PacBio CLR and Oxford Nanopore Technologies), and 3) short reads (i.e., Illumina). We use PBSIM2 [1034] to simulate the erroneous PacBio and Oxford Nanopore Technologies (ONT) reads from the Yeast genome. To use realistic depth of coverage, we use SeqKit [1035] to down-sample the original *E. coli*, and *D. ananassae* reads to 100× and 50× sequencing depth of coverage, respectively.

We evaluate BLEND based on two use cases: 1) read overlapping and 2) read mapping to a reference genome. For read overlapping, we perform *all-vs-all overlapping* to find all pairs of overlapping reads within the same dataset (i.e., the target and query sequences are the same set of sequences). To calculate the overlap statistics, we report the overall number of overlaps, the

Table 4.1: Details of datasets used in evaluation.

Organism	Library	Reads (#)	Seq. Depth	SRA Accession	Reference Genome
Human CHM13	PacBio HiFi	3,167,477	16	SRR11292122-3	T2T-CHM13 (v1.1)
	ONT*	10,380,693	30	Simulated R9.5	T2T-CHM13 (v2.0)
Human HG002	PacBio HiFi	11,714,594	52	SRR10382244-9	GRCh37
D. ananassae	PacBio HiFi	1,195,370	50	SRR11442117	[1036]
Yeast	PacBio CLR*	270,849	200	Simulated P6-C4	GCA_000146045.2
	ONT*	135,296	100	Simulated R9.5	GCA_000146045.2
	Illumina MiSeq	3,318,467	80	ERR1938683	GCA_000146045.2
E. coli	PacBio HiFi	38,703	100	SRR11434954	[1036]
	PacBio CLR	76,279	112	SRR1509640	GCA_000732965.1

* We use PBSIM2 to generate the simulated PacBio and ONT reads.
We show the simulated chemistry under the SRA Accession column.

average length of overlaps, and the number of seed matches per overlap. To evaluate the quality of overlapping reads based on the accuracy of the assemblies we generate from overlaps, we use miniasm [305]. We use miniasm because it does not perform error correction when generating *de novo* assemblies, which allows us to directly assess the quality of overlaps without using additional approaches that externally improve the accuracy of assemblies. We use `mhap2paf.pl` package as provided by miniasm to convert the output of MHAP to the format miniasm requires (i.e., PAF). We use QUAST [1037] to measure statistics related to the contiguity, length, and accuracy of *de novo* assemblies, such as the overall assembly length, largest contig, NG50, and NGA50 statistics (i.e., statistics related to the length of the shortest contig at the half of the overall reference genome length), k-mer completeness (i.e., amount of shared k-mers between the reference genome and an assembly), number of mismatches per 100Kb, and GC content (i.e., the ratio of G and C bases in an assembly). We use dnadiff [362] to measure the accuracy of *de novo* assemblies based on 1) the average identity of an assembly when compared to its reference genome and 2) the fraction of overall bases in a reference genome that align to a given assembly (i.e., genome fraction). We compare BLEND to minimap2 [306] and MHAP [416] for read overlapping. For human genomes, MHAP either 1) requires more memory than available in our system (i.e., 1TB) or 2) produces an output so large that miniasm cannot generate the assembly due to memory constraints.

For read mapping, we map all reads in a dataset (i.e., query sequences) to their corresponding reference genome (i.e., target sequence). We evaluate read mapping in terms of accuracy, quality, and the effect of read mapping on downstream analysis by calling structural variants. We

compare BLEND with minimap2, LRA [311], Winnowmap2 [413, 1005], S-conLSH [304, 307], and Strobealign [856]. We do not evaluate 1) LRA, Winnowmap2, and S-conLSH for short reads as these tools do not support mapping paired-end short reads, 2) Strobealign for long reads as it is a short read aligner, 3) S-conLSH for the *D. ananassae* as S-conLSH crashes due to a segmentation fault when mapping reads to the *D. ananassae* reference genome, and 4) S-conLSH for mapping HG002 reads as its output cannot be converted into a sorted BAM file, which is required for variant calling. We do not evaluate the read mapping accuracy of LRA and S-conLSH because 1) LRA generates a CIGAR string with characters that the `paftools mapeval` tool cannot parse to calculate alignment positions, and 2) S-conLSH due to its poor accuracy results we observe in our preliminary analysis.

Read mapping accuracy. We measure 1) the overall read mapping error rate and 2) the distribution of the read mapping error rate with respect to the fraction of mapped reads. To generate these results, we use the tools in `paftools` provided by minimap2 in two steps. First, the `paftools pbsim2fq` tool annotates the read IDs with their true mapping information that PBSIM2 generates. The `paftools mapeval` tool calculates the error rate of read mapping tools by comparing the mapping regions that the read mapping tools find with their true mapping regions annotated in read IDs. The error rate shows the ratio of reads mapped to incorrect regions over the entire mapped reads.

Read mapping quality. We measure 1) the breadth of coverage (i.e., percentage of bases in a reference genome covered by at least one read), 2) the average depth of coverage (i.e., the average number of read alignments per base in a reference genome), 3) mapping rate (i.e., number of aligned reads) and 4) rate of properly paired reads for paired-end mapping. To measure the breadth and depth of coverage of read mapping, we use BEDTools [1038] and Mosdepth [1039], respectively. To measure the mapping rate and properly paired reads, we use BAMUtil [1040].

Downstream analysis. We use sniffles2 [534, 1041] to call structural variants (SVs) from the HG002 long read mappings. We use Truvari [1042] to compare the resulting SVs with the benchmarking SV set (i.e., the *Tier 1* set) released by the Genome in a Bottle (GIAB) consortium [1043] in terms of their true positives (*TP*), false positives (*FP*), false negatives (*FN*), precision ($P = TP/(TP + FP)$), recall ($R = TP/(TP + FN)$) and the F_1 scores ($F_1 = 2 \times (P \times R)/(P + R)$). False positives show the number of the called SVs missing in the benchmarking set. False negatives show the number of SVs in the benchmarking set missing from the called SV set. The Tier 1 set includes 12,745 sequence-resolved SVs that include the `PASS` filter tag. GIAB provides the high-confidence regions of these SVs with low errors. We follow the benchmarking strategy that GIAB suggests [1043], where we compare the SVs with the `PASS` filter tag within the high-confidence regions.

For both use cases, we use the `time` command in Linux to evaluate the performance and peak memory footprints. We provide the average speedups and memory overhead of BLEND compared to each tool, while dataset-specific results are shown in our corresponding figures. When applicable, we use the default parameters of all the tools suggested for certain use cases and sequencing technologies (e.g., mapping HiFi reads in minimap2). Since minimap2 and MHAP do not provide default parameters for read overlapping using HiFi reads, we use the parameters that HiCanu [507] uses for overlapping HiFi reads with minimap2 and MHAP. We provide the details regarding the parameters and versions we use for each tool in Supplementary Tables S4.16, S4.17, and S4.14. When applicable in read overlapping, we use the same window and the seed length parameters that BLEND uses in minimap2 and show the performance and accuracy results in Supplementary Figure S4.3 and Supplementary Table S4.13. For read mapping, the comparable default parameters in BLEND are already the same as in minimap2.

4.3.2 Empirical Analysis of Fuzzy Seed Matching

We evaluate the effectiveness of fuzzy seed matching by identifying non-identical seeds that share the same hash value (i.e., collisions) using both the low-collision hash function of minimap2 (`hash64`) and BLEND.

4.3.2.1 Finding minimizer collisions.

Our goal is to evaluate the effects of using a low-collision hash function and the BLEND mechanism on hash collisions between non-identical minimizers. We find minimizers using 1) the low collision hash function that minimap2 uses (i.e., `hash64`) and 2) the SimHash technique [433, 434] we use in BLEND. For BLEND, we use the `BLEND-I` technique to directly compare the minimizers found using BLEND and minimap2. We keep the seed length constant, 16. For BLEND, we use various numbers of immediately overlapping k-mers that BLEND extracts from seed sequences (i.e., *neighbors*), as explained in Section 4.2.1. To keep the seed length ($|S|$) constant with a varying number of neighbors (n), we calculate the k-mer length (k) we extract from seeds as follows: $|S| = n + k - 1$ where $|S|$ and n are known. For each tool and configuration, we report the overall number of minimizers we find, the number of minimizer pairs that generate the same hash value (i.e., *collision*), the ratio of collisions to all minimizers, and the average edit distance between the minimizer pairs that have the same hash value. We make our resulting dataset that includes the statistics shown in Figure 4.8 and Table 4.2 available at Zenodo[4].

Table 4.2 shows the overall statistics of the fuzzy seed matching, and Figure 4.8 shows the edit distance between non-identical seeds with hash collision when using minimap2 and BLEND. We make three key observations. First, BLEND significantly increases the ratio of hash

[4] https://doi.org/10.5281/zenodo.7317896

Table 4.2: Fuzzy seed matching statistics of minimizer seeds that we find using minimap2 and BLEND. The number of overlapping k-mers that BLEND extracts from seed sequences (i.e., neighbors or *n*) are annotated as BLEND-n

Tool	Number of Minimizers	Number of Collisions	Collision/Minimizer Ratio	Avg. Edit Distance Between Minimizers With Collision
minimap2	903,043	15,306	0.016949	9.327061
BLEND-3	1,014,173	18,224	0.017969	9.393437
BLEND-5	1,090,468	20,659	0.018945	9.213660
BLEND-7	1,140,254	23,591	0.020689	8.874698
BLEND-9	1,173,198	28,411	0.024217	8.495301
BLEND-11	1,186,687	35,500	0.029915	8.067549
BLEND-13	1,197,966	72,078	0.060167	8.075918

collisions between highly similar minimizer pairs (e.g., edit distance less than 3) compared to using a low-collision hash function in minimap2. This result shows that BLEND favors increasing the collisions for highly similar seeds (i.e., fuzzy seed matching) than uniformly increasing the number of collisions by keeping the same ratio across all edit distance values. Second, the number of collisions that minimap2 and BLEND find are similar to each other for the minimizer pairs that have a large edit distance between them (e.g., larger than 6). The only exception to this observation is BLEND-13, which substantially increases all collisions for any edit distance due to using many small k-mers (i.e., thirteen 4-mers) when generating the hash values of 16-character long seeds. We note that the number of collisions is significantly higher when the edit distance between minimizers is 2 compared to the collisions with edit distance 1. We argue that this may be due to the distribution of the edit distances between minimizer pairs where there may be significantly a large number of minimizer pairs with edit distance 2 than 1. Third, increasing the number of neighbors can effectively reduce the average edit distance between fuzzy seed matches with the cost of increasing the overall number of minimizer seeds, as shown in Table 4.2. We conclude that BLEND can effectively find highly similar seeds with the same hash value as it increases the ratio of collisions between similar seeds while providing a collision ratio similar to minimap2 for dissimilar seeds.

4.3.2.2 Identifying similar sequences.

Our goal is to find non-identical k-mer matches with the same hash value (i.e., fuzzy k-mer matches) between highly similar sequence pairs. To this end, we prepare a dataset that includes 25-character long sequences in four steps. First, we extract 25-character long non-overlapping sequences from the *E. coli* reference genome [1036] shown in Table 4.1, which we call *sampled sequences* for simplicity. To evenly sample these sequences, each sampled sequence is separated

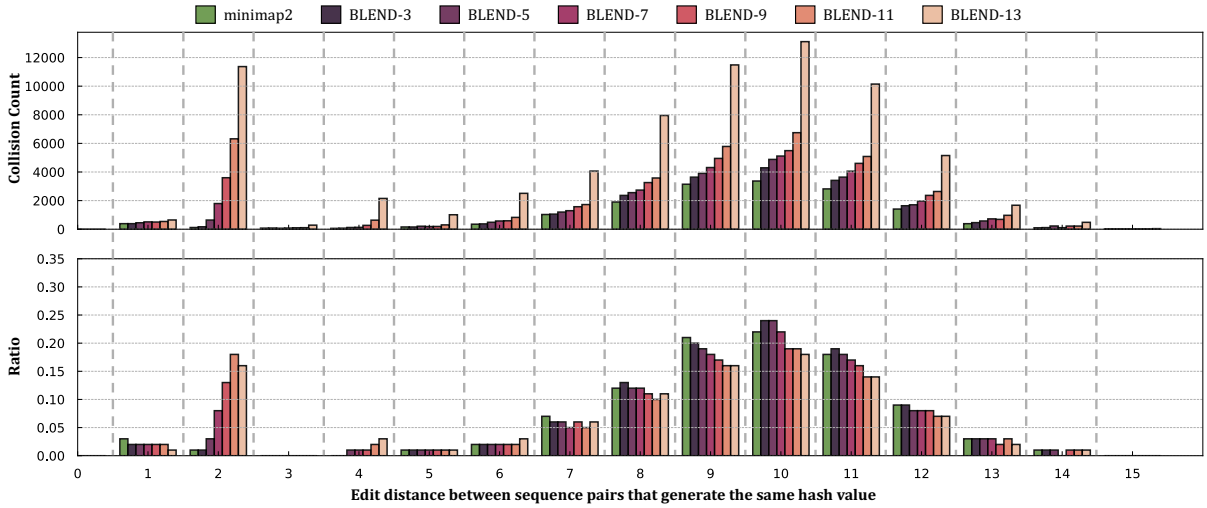

Figure 4.8: Fuzzy seed matching statistics. Collision count shows the number of non-identical seeds that generate the same hash value and the edit distance between these sequences. Ratio is the proportion of collisions between non-identical sequences at a certain edit distance over all collisions. BLEND-*n* shows the number of neighbors (*n*) that BLEND uses.

by 75 characters from the previous sampled sequence. Second, our goal is to find *all* sequences in the reference genome that are *similar* and non-identical to the sampled sequences. To achieve this, we use bowtie [365] and find *all* sequences in the *E. coli* reference genome that the sampled sequences align with at least one mismatch and, at most, three mismatches (i.e., at least ~ 88% similarity). Third, we extract the sequences from the reference regions that the sampled sequences align, which we call *aligned sequences*. Fourth, we prepare our dataset that contains 1,077 FASTA files and 4,130 25-character long sequences overall. Each FASTA file includes 1) a sampled sequence that has at least one alignment in the reference genome based on our mismatching criteria and 2) all aligned sequences that the sampled sequence is aligned to.

To find the non-identical k-mers with the same hash value in each FASTA file, we generate the hash values of all overlapping 16-mers of all sequences in a FASTA file. We use the low-collision hash function that minimap2 uses (i.e., `hash64`) and the `BLEND-I` technique in BLEND to generate these hash values. For BLEND, we use various numbers of neighbors when generating the hash values of 16-mers (see Section 4.3.2.1 for the relation between the number of neighbors and the seed length, which is 16 in our evaluation). In Table 4.3, we report the number of sequences in our dataset, the number of sequences that have at least one non-identical k-mer pair with the same hash value (i.e., collisions), the ratio of collisions to the overall number of sequences, and the average edit distance between k-mers with collision. We make our dataset available at Zenodo[5], which includes 1,077 FASTA files and the resulting

[5] https://doi.org/10.5281/zenodo.7319786

files that we generate the numbers we show in Table 4.3. These resulting files include the non-identical k-mers with the same hash value, the sequence pairs that we extract these k-mers from, and the edit distances with these k-mers.

Table 4.3: Fuzzy k-mer matching statistics of sequences that we find using minimap2 and BLEND. The number of overlapping k-mers that BLEND extracts from seed sequences (i.e., neighbors or *n*) are annotated as BLEND-n

Tool	Number of Sequences	Number of Sequences with Collision	Collision/Sequence Ratio	Avg. Edit Distance Between K-mers With Collision
minimap2	4,130	0	0	N/A
BLEND-3	4,130	0	0	N/A
BLEND-5	4,130	11	0.00263663	1.45455
BLEND-7	4,130	50	0.0119847	1.5
BLEND-9	4,130	77	0.0184564	2.01299
BLEND-11	4,130	273	0.0654362	2.80952
BLEND-13	4,130	329	0.0788591	2.20669

Table 4.3 shows the number and portion of similar sequence pairs that we can find using *only* fuzzy k-mer matches. We make two key observations. First, BLEND is the only mechanism that can identify similar sequences from their fuzzy k-mer matches since low-collision hash functions cannot increase the collision rates for high similarity matches. Second, BLEND can identify a larger number of similar sequence pairs with an increasing number of neighbors. For the number of neighbors larger than 5, the percentage of these similar sequence pairs that BLEND can identify ranges from 1.2% to 7.9% of the overall number of sequences we use in our dataset. We conclude that BLEND enables finding similar sequence pairs from fuzzy k-mer matches that low-collision hash functions cannot find.

4.3.3 Use Case 1: Read Overlapping

4.3.3.1 Performance.

Figure 4.9 shows the CPU time and peak memory footprint comparisons for read overlapping. We make the following five observations. First, BLEND provides an average speedup of 19.3× and 808.2× while reducing the memory footprint by 3.8× and 127.8× compared to minimap2 and MHAP, respectively. BLEND is significantly more performant and provides less memory overheads than MHAP because MHAP generates many hash values for seeds regardless of the length of the sequences, while BLEND allows sampling the number of seeds based on the sequence length with the windowing guarantees of minimizers and strobemer seeds. Second, when considering only HiFi reads, BLEND provides significant speedups by 40.3× and 1580.0× while reducing the memory footprint by 7.2× and 214.0× compared to minimap2 and MHAP,

respectively. HiFi reads allow BLEND to increase the window length (i.e., $w = 200$) when finding the minimizer k-mer of a seed, which improves the performance and reduces the memory overhead without reducing the accuracy. This is possible mainly because BLEND can find *both* fuzzy and exact seed matches, which enables BLEND to find *unique* fuzzy seed matches that minimap2 *cannot* find due to its exact-matching seed requirement. Third, we find that BLEND requires less than 16GB of memory space for almost all the datasets, making it largely possible to find overlapping reads even with a personal computer with relatively small memory space. BLEND has a lower memory footprint because 1) BLEND uses as many seeds as the number of minimizer k-mers per sequence to benefit from the reduced storage requirements that minimizer k-mers provide, and 2) the window length is larger than minimap2 as BLEND can tolerate increasing this window length with the fuzzy seed matches without reducing the accuracy. Fourth, when using erroneous reads (i.e., PacBio CLR and ONT), BLEND performs better than other tools with memory overheads similar to minimap2. The set of parameters we use for erroneous reads prevents BLEND from using large windows (i.e., $w = 10$ instead of $w = 200$) without reducing the accuracy of read overlapping. Smaller window lengths generate more seeds, which increases the memory space requirements. Fifth, we use the same parameters (i.e., the seed length and the window length) with minimap2 that BLEND uses to observe the benefits that BLEND provides with PacBio CLR and ONT datasets. We cannot perform the same experiment for the HiFi datasets because BLEND uses strobemer seeds of length 31, which minimap2 cannot support due to its minimizer seeds and the maximum seed length limitation in its implementation (i.e., max. 28). We use *minimap2-Eq* to refer to the version of minimap2, where it uses the parameters equivalent to the BLEND parameters for a given dataset in terms of the seed and window lengths. We show in Supplementary Figure S4.3 that minimap2-Eq performs, on average, $\sim$ 5% better than BLEND with similar memory space requirements when using the same set of parameters with the `BLEND-I` technique. Minimap2-Eq provides worse accuracy than BLEND when generating the ONT assemblies, as shown in Supplementary Table S4.13, while the erroneous PacBio assemblies are more accurate with minimap2-Eq. The main benefit of BLEND is to provide overall higher accuracy than both the baseline minimap2 and minimap-Eq, which we can achieve by finding unique fuzzy seed matches that minimap2 cannot find. We conclude that BLEND is significantly more memory-efficient and faster than other tools to find overlaps, especially when using HiFi reads with its ability to sample many seeds using large values of w without reducing the accuracy.

4.3.3.2 Overlap Statistics.

Figure 4.10 shows the overall number of overlaps, the average length of overlaps, and the average number of seed matches that each tool finds to identify the overlaps between reads.

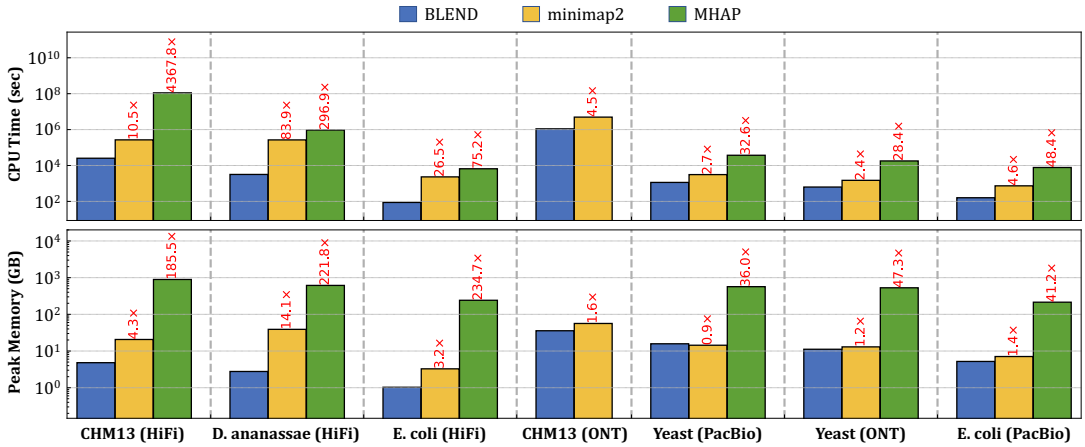

Figure 4.9: CPU time and peak memory footprint comparisons of read overlapping.

The total number of overlaps combined with the average number of seed matches per overlap determines the total number of seeds found by each method. We make the following four key observations. First, we observe that BLEND finds overlaps longer than minimap2 and MHAP can find in most cases. BLEND can 1) uniquely find the fuzzy seed matches that the exact-matching-based tools cannot find and 2) perform chaining on these fuzzy seed matches to increase the length of overlap using many fuzzy seed matches that are relatively close to each other. Finding more distinct seeds and chaining these seeds enable BLEND to find longer overlaps than other tools. Although these unique features of BLEND can lead to chaining longer overlaps, we also note that BLEND may not be able to find very short overlaps due to the larger window lengths it uses, which can also contribute to increasing the average length of overlaps. Second, BLEND uses significantly fewer seed matches per overlap than other tools, up to 27.3×, to find these longer overlaps. This is mainly because BLEND needs much fewer seeds per overlap as it uses 1) larger window lengths than minimap2 and 2) provides windowing guarantees, unlike MHAP. Third, finding fewer seed matches per overlap leads to 1) finding fewer overlaps than minimap2 and MHAP find and 2) reporting fewer seed matches overall. These overlaps that BLEND cannot find are mainly because of the strict parameters that minimap2 and MHAP use due to their exact seed matching limitation (e.g., smaller window lengths). BLEND can increase the window length while producing more accurate and complete assemblies than minimap2 and MHAP (Table 4.4). This suggests that minimap2 and MHAP find redundant overlaps and seed matches that have no significant benefits in generating accurate and complete assemblies from these overlaps. Fourth, the sequencing depth of coverage has a larger impact on the number of overlaps that BLEND can find compared to the impact on minimap2 and MHAP. We observe this trend when comparing the number of overlaps found using the PacBio (200× coverage) and ONT (100× coverage) reads of the Yeast genome. The gap

between the number of overlaps found by BLEND and other tools increases as the sequencing coverage decreases. This suggests that BLEND can be less robust to the sequencing depth of coverage. Such a trend does not impact the accuracy of the assemblies that we generate using the BLEND overlaps, while it provides lower NGA50 and NG50 values as shown in Table 4.4. We conclude that the performance and memory-efficiency improvements in read overlapping are proportional to the reduction in the seed matches that BLEND uses to find overlapping reads. Thus, finding fewer non-redundant seed matches can dramatically improve the performance and memory space usage without reducing the accuracy.

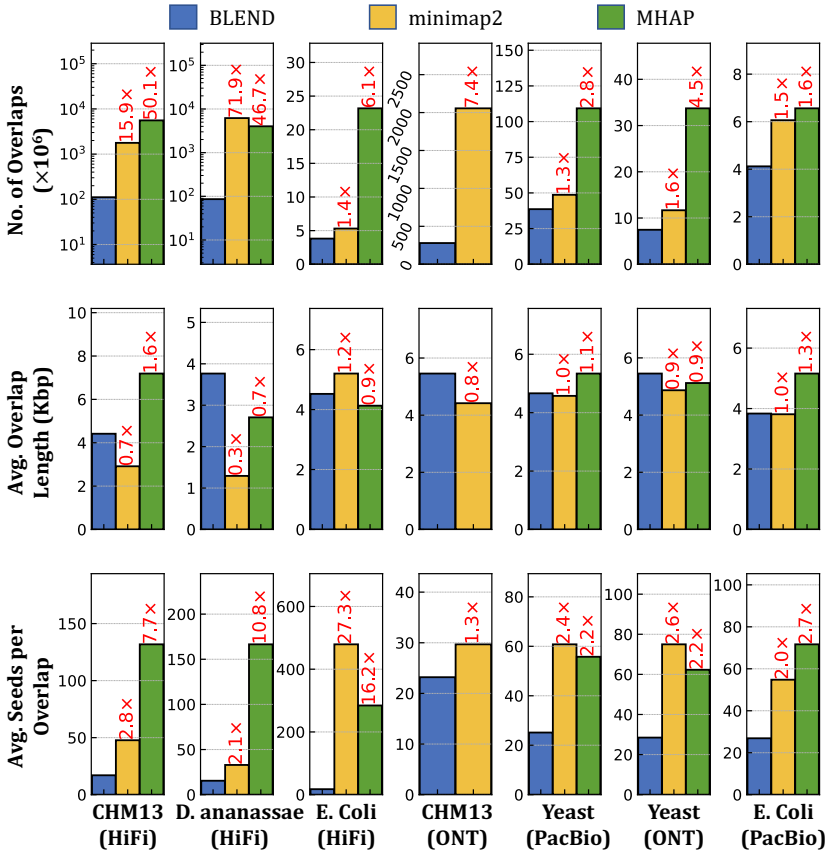

Figure 4.10: Average number and length of overlaps, and average number of seeds used to find a single overlap.

4.3.3.3 Assembly Quality Assessment.

Our goal is to assess the quality of assemblies generated using the overlapping reads found by BLEND, minimap2, and MHAP. Table 4.4 shows the statistics related to the accuracy of assemblies (i.e., the six statistics on the leftmost part of the table) and the statistics related to assembly length and contiguity (i.e., the four statistics on the rightmost part of the table) when compared to their respective reference genomes. We make the following five key observations based on the accuracy results of assemblies. First, we observe that assemblies generated using

the overlapping reads found by BLEND are more accurate in terms of average identity and k-mer completeness compared to those generated by minimap2 and MHAP. These results show that the assemblies we generate using the BLEND overlaps are more similar to their corresponding reference genome. BLEND can find unique and accurate overlaps using fuzzy seed matches that lead to more accurate *de novo* assemblies than the minimap2 and MHAP overlaps due to their lack of support for fuzzy seed matching. Second, we observe that assemblies generated using BLEND overlaps usually cover a larger fraction of the reference genome than minimap2 and MHAP overlaps. Third, although the average identity and genome fraction results seem mixed for the PacBio CLR and ONT reads such that BLEND is best in terms of either average identity or genome fraction, we believe these two statistics should be considered together (e.g., by multiplying both results). This is because a highly accurate but much smaller fraction of the assembly can align to a reference genome, giving the best results for the average identity. We observe that this is the case for the *D. ananassae* and *Yeast* (PacBio CLR) genomes such that MHAP provides a very high average identity only for the much smaller fraction of the assemblies than the assemblies generated using BLEND and minimap2 overlaps. Thus, when we combine average identity and genome fraction results, we observe that BLEND consistently provides the best results for all the datasets. Fourth, BLEND usually provides the best results in terms of the aligned length and the number of mismatches per 100Kb. In some cases, QUAST cannot generate these statistics for the MHAP results as a small portion of the assemblies aligns the reference genome when the MHAP overlaps are used. Fifth, we find that assemblies generated from BLEND overlaps are less biased than minimap2 and MHAP overlaps, based on the average GC content results that are mostly closer to their corresponding reference genomes. We conclude that BLEND overlaps yield assemblies with higher accuracy and less bias than the assemblies that the minimap2 and MHAP overlaps generate in most cases.

Table 4.4 shows the results related to assembly length and contiguity on its rightmost part. We make the following three observations. First, we show that BLEND yields assemblies with better contiguity when using HiFi reads based on the largest NG50, NGA50, and contig length results compared to minimap2 with the exception of the human genome. Second, minimap2 provides better contiguity for the human genomes and erroneous reads. Third, the overall length of all assemblies is mostly closer to the reference genome assembly. We conclude that minimap2 provides better contiguity for the assemblies from erroneous and human reads while BLEND is usually better suited for using the HiFi reads.

Table 4.4: Assembly quality comparisons.

Dataset	Tool	Average Identity (%)	Genome Fraction (%)	K-mer Compl. (%)	Aligned Length (Mbp)	Mismatch per 100Kbp (#)	Average GC (%)	Assembly Length (Mbp)	Largest Contig (Mbp)	NGA50 (Kbp)	NG50 (Kbp)
CHM13	BLEND	**99.8526**	**98.4847**	**90.15**	3,092.54	**22.02**	**40.78**	**3,095.21**	22.8397	5,442.25	5,442.31
(HiFi)	minimap2	99.7421	97.1493	83.05	**3,094.79**	55.96	40.71	3,100.97	**47.1387**	**7,133.43**	**7,134.31**
	MHAP	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	Reference	100	100	100	3,054.83	0.00	40.85	3,054.83	248.387	154,260	154,260
D. ananassae	BLEND	**99.7856**	**97.2308**	**86.43**	240.391	**143.13**	**41.75**	**247.153**	**6.23256**	**792.407**	**798.913**
(HiFi)	minimap2	99.7044	96.3190	72.33	**289.453**	191.53	41.68	298.28	4.43396	273.398	278.775
	MHAP	99.5551	0.7276	0.21	2.29	239.76	42.07	2.34951	0.028586	N/A	N/A
	Reference	100	100	100	213.805	0.00	41.81	213.818	30.6728	26,427.4	26,427.4
E. coli	BLEND	**99.8320**	**99.8801**	**87.91**	**5.12155**	**3.77**	**50.53**	5.12155	**3.41699**	**3,416.99**	**3,416.99**
(HiFi)	minimap2	99.7064	99.8748	79.27	5.09249	19.71	50.47	**5.09436**	3.08849	3,087.05	3,087.05
	MHAP	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	Reference	100	100	100	5.04628	0.00	50.52	5.04628	4.94446	4,944.46	4,944.46
CHM13	BLEND	N/A	N/A	**29.26**	**2,891.28**	**4,077.53**	**41.32**	2,897.87	25.2071	5,061.52	5,178.59
(ONT)	minimap2	N/A	N/A	28.32	2,860.26	4,660.73	41.36	**2,908.55**	**66.7564**	**13,189.2**	**13,820.3**
	Reference	100	100	100	3,117.29	0.00	40.75	3,117.29	248.387	150,617	150,617
Yeast	BLEND	89.1677	**97.0854**	**33.81**	**12.3938**	2,672.37	38.84	12.4176	1.54807	635.966	636.669
(PacBio)	minimap2	88.9002	96.9709	33.38	12.0128	2,684.38	38.85	**12.3325**	**1.56078**	**810.046**	**828.212**
	MHAP	**89.2182**	88.5928	32.39	10.9039	**2,552.05**	**38.81**	10.9896	1.02375	85.081	436.285
	Reference	100	100	100	12.1571	0.00	38.15	12.1571	1.53193	924.431	924.431
Yeast	BLEND	**89.6889**	99.2974	**35.95**	**12.3222**	**2,529.47**	**38.64**	**12.3225**	1.10582	793.046	793.046
(ONT)	minimap2	88.9393	**99.6878**	34.84	12.304	2,782.59	38.74	12.3725	**1.56005**	**796.718**	**941.588**
	MHAP	89.1970	89.2785	33.58	10.8302	2,647.19	38.84	10.9201	1.44328	118.886	618.908
	Reference	100	100	100	12.1571	0.00	38.15	12.1571	1.53193	924.431	924.431
E. coli	BLEND	**88.5806**	**96.5238**	**32.32**	**5.90024**	**1,857.56**	**49.81**	6.21598	2.40671	**769.981**	2,060.4
(PacBio)	minimap2	88.1365	92.7603	30.74	5.37728	2,005.72	49.66	**6.02707**	**3.77098**	367.442	**3,770.98**
	MHAP	88.4883	90.5533	31.32	5.75159	1,999.48	49.69	6.26216	1.04286	110.535	456.01
	Reference	100	100	100	5.6394	0.00	50.43	5.6394	5.54732	5,547.32	5,547.32

Best results are highlighted with **bold** text. For most metrics, the best results are the ones closest to the corresponding value of the reference genome.
The best results for *Aligned Length* are determined by the highest number within each dataset. We do not highlight the reference results as the best results.
N/A indicates that we could not generate the corresponding result because tool, QUAST, or dnadiff failed to generate the statistic.

4.3.4 Use Case 2: Read Mapping

4.3.4.1 Performance.

Figure 4.11 shows the CPU time and the peak memory footprint comparisons when performing read mapping to the corresponding reference genomes. We make the following four key observations. First, we observe that BLEND provides an average speedup of 1.7×, 6.8×, 4.3×, and 13.3× over minimap2, LRA, Winnowmap2, and S-conLSH, respectively. While BLEND outperforms most of these tools, the speedups observed are generally lower than those in read overlapping. Read mapping includes an additional computationally costly step that read overlapping skips, which is the read alignment. The extra overhead of read alignment slightly hinders the benefits that BLEND provides that we observe in read overlapping. Second, we find that LRA and minimap2 require 0.6× and 1.0× of the memory space that BLEND uses, while Winnowmap2 and S-conLSH have a larger memory footprint by 1.5× and 1.6×, respectively. BLEND cannot provide similar reductions in the memory overhead that we observe in read overlapping due to the narrower window length ($w = 50$ instead of $w = 200$) it uses to find the minimizer k-mers for HiFi reads. Using a narrow window length generates more seeds to store in a hash table, which proportionally increases the peak memory space requirements. Third, BLEND provides performance and memory usage similar to minimap2 when mapping the erroneous ONT and PacBio reads because BLEND uses the same parameters as minimap2

for these reads (i.e., same w and seed length). Fourth, Strobealign is the best-performing tool for mapping short reads with the cost of larger memory overhead. We conclude that BLEND, on average, 1) performs better than all tools for mapping long reads and 2) provides a memory footprint similar to or better than minimap2, Winnowmap2, S-conLSH, and Strobealign, while LRA is the most memory-efficient tool.

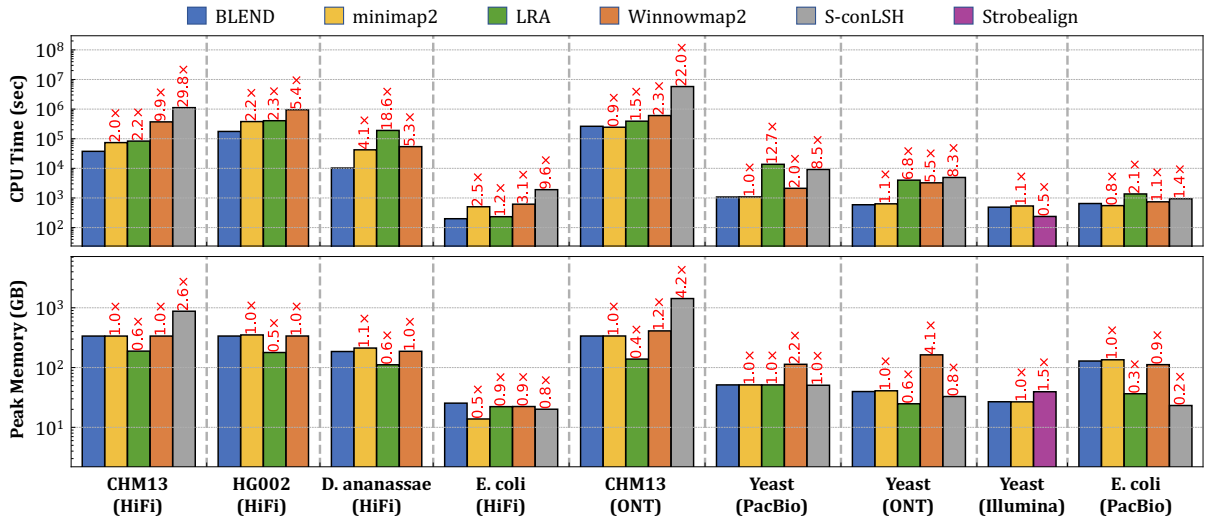

Figure 4.11: CPU time and peak memory footprint comparisons of read mapping.

4.3.4.2 Read Mapping Accuracy.

Table 4.5 and Figure 4.12 show the overall read mapping accuracy and fraction of mapped reads with their average mapping accuracy, respectively. We make two observations. First, we observe that BLEND generates the most accurate read mapping in most cases, while minimap2 provides the most accurate read mapping for the human genome. These two tools are on par in terms of their read mapping accuracy and the fraction of mapped reads. Second, although Winnowmap2 provides more accurate read mapping than minimap2 for the PacBio reads from the Yeast genome, Winnowmap2 always maps a smaller fraction of reads than those BLEND and minimap2 map. We conclude that although the results are mixed, BLEND is the only tool that generates either the most or the second-most accurate read mapping in all datasets, providing the overall best accuracy results.

4.3.4.3 Read Mapping Quality.

Our goal is to assess the quality of read mappings in terms of four metrics: average depth of coverage, breadth of coverage, number of aligned reads, and the ratio of the paired-end reads that are properly paired in mapping. Table 4.6 shows the quality of read mappings based on these metrics when using BLEND, minimap2, LRA, Winnowmap2, and Strobealign. We exclude S-conLSH from the read mapping quality comparisons as we cannot convert its SAM output

Table 4.5: Read mapping accuracy comparisons.

Dataset	Overall Error Rate (%)		
	BLEND	minimap2	Winnowmap2
CHM13 (ONT)	1.5168427	**1.4914009**	1.7001222
Yeast (PacBio)	**0.2403134**	0.2504307	0.2474206
Yeast (ONT)	**0.2386617**	0.2468770	0.2534777

Best results are highlighted with **bold** text.

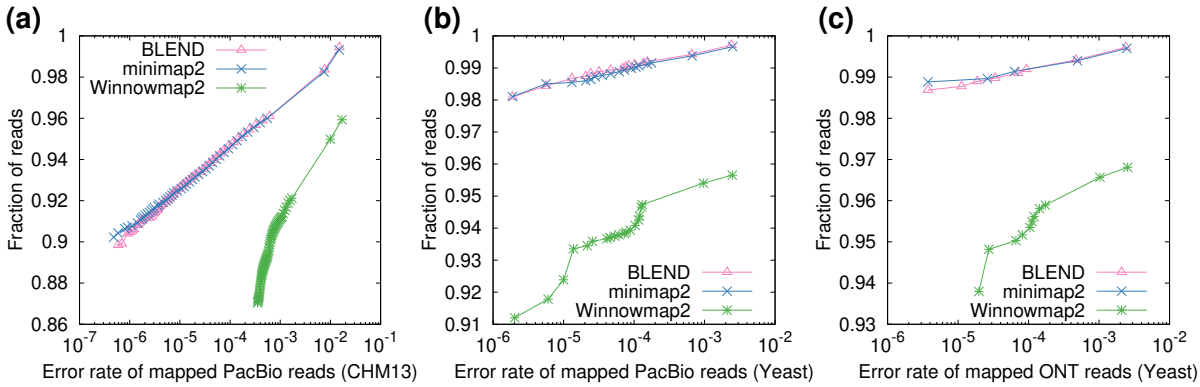

Figure 4.12: Fraction of simulated reads with an average mapping error rate. Reads are binned by their mapping quality scores. There is a bin for each mapping quality score as reported by the read mapper, and bins are sorted based on their mapping quality scores in descending order. For each tool, the n^{th} data point from the left side of the x-axis shows the rate of incorrectly mapped reads among the reads in the first n bins. We show the number of reads in these bins in terms of the fraction of the overall number of reads in the dataset. The data point with the largest fraction shows the average mapping error rate of all mapped reads.

to BAM format to properly index the BAM file due to issues with its SAM output format. We make five observations.

First, all tools cover a large portion of the reference genomes based on the breadth of coverage of the reference genomes. Although LRA provides the lowest breadth of coverage in most cases compared to the other tools, it also provides the best breadth of coverage after mapping the human HG002 reads. This result shows that these tools are less biased in mapping reads to particular regions with their high breadth of coverage, and the best tool for covering the largest portion of the genome depends on the dataset.

Second, both BLEND and minimap2 map an almost complete set of reads to the reference genome for all the datasets, while Winnowmap2 suffers from a slightly lower number of aligned reads when mapping erroneous PacBio CLR and ONT reads. The only exception to this observation is the HG002 dataset, where BLEND provides a smaller number of aligned reads compared to other tools, while BLEND provides the same breadth of coverage as minimap2.

Table 4.6: Read mapping quality comparisons.

Dataset	Tool	Average Depth of Cov. (×)	Breadth of Coverage (%)	Aligned Reads (#)	Properly Paired (%)
CHM13 (HiFi)	BLEND	**16.58**	**99.991**	3,171,916	NA
	minimap2	**16.58**	**99.991**	**3,172,261**	NA
	LRA	16.37	99.064	3,137,631	NA
	Winnowmap2	**16.58**	99.990	3,171,313	NA
HG002 (HiFi)	BLEND	51.25	92.245	11,424,762	NA
	minimap2	53.08	92.242	12,407,589	NA
	LRA	52.48	**92.275**	**13,015,195**	NA
	Winnowmap2	**53.81**	92.248	12,547,868	NA
D. ananassae (HiFi)	BLEND	57.37	99.662	1,223,388	NA
	minimap2	**57.57**	**99.665**	1,245,931	NA
	LRA	57.06	99.599	1,235,098	NA
	Winnowmap2	57.40	99.663	**1,249,575**	NA
E. coli (HiFi)	BLEND	**99.14**	99.897	39,048	NA
	minimap2	**99.14**	99.897	**39,065**	NA
	LRA	99.10	99.897	39,063	NA
	Winnowmap2	**99.14**	99.897	39,036	NA
CHM13 (ONT)	BLEND	**29.34**	**99.999**	**10,322,767**	NA
	minimap2	29.33	**99.999**	10,310,182	NA
	LRA	28.84	99.948	9,999,432	NA
	Winnowmap2	28.98	99.936	9,958,402	NA
Yeast (PacBio)	BLEND	**195.87**	**99.980**	**270,064**	NA
	minimap2	195.86	**99.980**	269,935	NA
	LRA	194.65	99.967	267,399	NA
	Winnowmap2	192.35	99.977	259,073	NA
Yeast (ONT)	BLEND	**97.88**	**99.964**	**134,919**	NA
	minimap2	**97.88**	**99.964**	134,885	NA
	LRA	97.25	99.952	132,862	NA
	Winnowmap2	97.04	99.963	130,978	NA
Yeast (Illumina)	BLEND	**79.92**	**99.975**	6,493,730	95.88
	minimap2	79.91	99.974	6,492,994	95.89
	Strobealign	**79.92**	99.970	**6,498,380**	**97.59**
E. coli (PacBio)	BLEND	**97.51**	100	83,924	NA
	minimap2	97.29	100	**85,326**	NA
	LRA	93.61	100	80,802	NA
	Winnowmap2	89.78	100	69,884	NA

Best results are highlighted with **bold** text.

Properly paired rate is only available for paired-end Illumina reads.

We investigate if such a smaller number of aligned reads leads to a coverage bias genome-wide in Supplementary Figures S4.4, S4.5, and S4.6. We find that the distribution of the depth of coverage of BLEND is mostly similar to minimap2. There are a few regions in the reference genome where minimap2 provides substantially higher coverage than BLEND provides, as we show in Supplementary Figure S4.6, which causes BLEND to align a smaller number of reads than minimap2 aligns. Since these regions are still covered by both BLEND and minimap2 with different depths of coverage, these two tools generate the same breadth of coverage without leading to no significant coverage bias genome-wide.

Third, we find that all the tools generate read mappings with a depth of coverage significantly close to their sequencing depth of coverage. This shows that almost all reads map to the reference genome evenly. Fourth, Strobealign generates the largest number of 1) short reads mappings

to the reference genome and 2) properly paired reads compared to BLEND and minimap2. Strobealign can map more reads using less time (Figure 4.11, which makes its throughput much higher than BLEND and minimap2. Fifth, although Strobealign can map more reads, it covers the smallest portion of the reference genome based on the breadth of coverage compared to BLEND and minimap2. This suggests that Strobealign provides a higher depth of coverage at certain regions of the reference genome than BLEND and minimap2 while leaving larger gaps in the reference genome. We conclude that the read mapping qualities of BLEND, minimap2, and Winnowmap2 are highly similar, while LRA provides slightly worse results. It is worth noting that BLEND provides a better breadth of coverage than minimap2 provides in most cases while using the same parameters in read mapping. BLEND does this by finding unique fuzzy seed matches that the other tools cannot find due to their exactly matching seed requirements.

4.3.4.4 Downstream Analysis.

To evaluate the effect of read mapping on downstream analysis, we call SVs from the HG002 long read mappings that BLEND, minimap2, LRA, and Winnowmap2 generate. Table 4.7 shows the benchmarking results. We make two key observations. First, we find that BLEND provides the best overall accuracy in downstream analysis based on the best F_1 score compared to other tools. This is because BLEND provides the best true positive and false negative numbers while providing the second-best false positive numbers after LRA. These two best values overall contribute to achieving the best recall and second-best precision that is on par with the precision LRA provides. Second, although LRA generates the second-best F_1 score, it provides the worst recall results due to the largest number of false negatives. We conclude that BLEND is consistently either the best or second-best in terms of the metrics we show in Table 4.7, which leads to providing the best overall F_1 accuracy in structural variant calling.

Table 4.7: Benchmarking the structural variant (SV) calling results.

	HG002 SVs (High-confidence Tier 1 SV Set)					
Tool	**TP (#)**	**FP (#)**	**FN (#)**	**Precision**	**Recall**	F_1
BLEND	**9,229**	855	**412**	0.9152	**0.9573**	**0.9358**
minimap2	9,222	915	419	0.9097	0.9565	0.9326
LRA	9,155	**830**	486	**0.9169**	0.9496	0.9329
Winnowmap2	9,170	1029	471	0.8991	0.9511	0.9244

Best results are highlighted with **bold** text.

4.4 Discussion

We demonstrate that there are usually too many redundant *short* and *exactly matching* seeds used to find overlaps between sequences, as shown in Figure 4.10. These redundant seeds

usually exacerbate the performance and peak memory space requirement problems that read overlapping and read mapping suffer from as the number of chaining and alignment operations proportionally increases with the number of seed matches between sequences [306]. Such redundant computations have been one of the main limitations against developing population-scale genomics analysis due to the high runtime of a single high-coverage genome analysis.

There has been a clear interest in using long or fuzzy seed matches because of their potential to find similarities between target and query sequences efficiently and accurately [258]. To achieve this, earlier works mainly focus on either 1) chaining the exact k-mer matches by tolerating the gaps between them to increase the seed region or 2) linking multiple consecutive minimizer k-mers such as strobemer seeds. Chaining algorithms are becoming a bottleneck in read mappers as the complexity of chaining is determined by the number of seed matches [642]. Linking multiple minimizer k-mers enables tolerating indels when finding the matches of short sequence segments between genomic sequence pairs, but these seeds (e.g., strobemer seeds) should still exactly match due to the nature of the hash functions used to generate the hash values of seeds. This requires the seeding techniques to generate exactly the same seed to find either exactly matching or approximate matches of short sequence segments. We state that any arbitrary k-mer in the seeds should be tolerated to mismatch to improve the sensitivity of any seeding technique, which has the potential for finding more matching regions while using fewer seeds. Thus, we believe BLEND solves the main limitation of earlier works such that it can generate the same hash value for similar seeds to find fuzzy seed matches with a single lookup while improving the performance, memory overhead, and accuracy of the applications that use seeds.

We hope that BLEND advances the field and inspires future work in several ways, some of which we list next. First, we observe that BLEND is *most effective* when using high coverage and highly accurate long reads. Thus, BLEND is already ready to scale for longer and more accurate sequencing reads. Second, the vector operations are suitable for hardware acceleration to improve the performance of BLEND further. Such an acceleration is mainly useful when a massive amount of k-mers in a seed are used to generate the hash value for a seed, as these calculations can be done in parallel. We already provide the SIMD implementation to calculate the hash values BLEND. We encourage implementing our mechanism for the applications that use seeds to find sequence similarity using processing-in-memory and near-data processing [195, 463, 467, 490, 598, 686, 687, 689, 690, 693, 702, 708, 719], GPUs [636, 643, 1044], and FPGAs and ASICs [465, 661, 662, 673, 698, 1045] to exploit the massive amount of embarrassingly parallel bitwise operations in BLEND to find fuzzy seed matches. Third, we believe it is possible to apply the hashing technique we use in BLEND for many seeding techniques with a proper design. We already show we can apply SimHash in regular minimizer k-mers or strobemers.

Strobemers can be generated using k-mer sampling strategies other than minimizer k-mers, which are based on syncmers and random selection of k-mers (i.e., randstrobes) [856]. It is worth exploring and rethinking the hash functions used in these seeding techniques. Fourth, potential machine learning applications can be used to generate more sensitive hash values for fuzzy seed matching based on learning-to-hash approaches [1046] and recent improvements on SimHash for identifying nearest neighbors in machine learning and bioinformatics [437–439].

4.4.1 Limitations

We identify two main limitations of our work that requires further improvements. First, BLEND may generate the same hash values for 1% – 8% of all the similar sequence pairs in a dataset, as we show in Table 4.3. These 1% – 8% of similar sequence pairs that cannot be found using low-collision hash functions can be significant in improving the accuracy and performance of some genomics applications. However, such a percentage may also be considered low for other use cases. We observe that increasing the number of neighbors (n) can increase the percentage of similar sequence pairs that BLEND can find with the cost of causing more collisions for dissimilar sequence pairs. A newer generation of the SimHash-like hash functions such as DenseFly [439] or FlyHash [1047] has the potential to improve the rate of similar sequence pairs with the same hash value. Second, the advantage of BLEND is mainly observed when using highly accurate and long reads with high sequencing depth of coverage in read overlapping and downstream analysis, while the improvements are lower in other datasets. Although BLEND scales better as the sequencing technologies become cheaper and generate longer and highly accurate reads, it is also essential to further improve its accuracy and performance for existing read datasets with erroneous long reads and short reads. This requires further optimizations in the parameter settings for erroneous long reads and short reads. We leave these two limitations as future work along with the other potential future works that we discuss earlier.

4.5 Summary

We propose BLEND, a mechanism that can efficiently find fuzzy seed matches between sequences to improve the performance, memory space efficiency, and accuracy of two important applications significantly: 1) read overlapping and 2) read mapping. Based on the experiments we perform using real and simulated datasets, we make six key observations. First, for read mapping, BLEND provides an average speedup of 19.3× and 808.2× while reducing the peak memory footprint by 3.8× and 127.8× compared to minimap2 and MHAP. Second, we observe that BLEND finds longer overlaps, in general, while using significantly fewer seed matches by up to 27.3× to find these overlaps. Third, we find that we can usually generate more *accurate* assemblies when using the overlaps that BLEND finds than those found by minimap2 and

MHAP. Fourth, for read mapping, we find that BLEND, on average, provides speedup by 1) 1.7×, 6.8×, 4.3×, and 13.3× compared to minimap2, LRA, Winnowmap2, and S-conLSH, respectively. Fifth, Strobealign performs best for short read mapping, while BLEND provides better memory space usage than Strobealign. Sixth, we observe that BLEND, minimap2, and Winnowmap2 provide both high quality and better accuracy in read mapping in all datasets, while BLEND and LRA provide the best SV calling results in terms of downstream analysis accuracy. We conclude that BLEND can use fewer fuzzy seed matches to significantly improve the performance and reduce the memory overhead of read overlapping without losing accuracy, while BLEND, on average, provides better performance and a similar memory footprint in read mapping without reducing the read mapping quality and accuracy. We hope that BLEND will lead to the design of more accurate genome analysis techniques with the ability to find fuzzy seed matches as well as exact matching seeds.

S4 Supplementary Materials

S4.1 A Real Example of Generating the Hash Values of Seeds S_k and S_l

Our goal is to show how the k-mer length k and the number of k-mers to include in a seed, n, affect the final hash value. To this end, we use the following two seeds as found in the Yeast reference genome: S_k : CGGATGCTACAGTATATACCA and S_l : ATGCTACAGTATATACCATCT. Both seeds are 21-character long. We use two different parameter settings when generating the hash values of these seeds. The first setting uses $k = 7$ as the k-mer length and $n = 15$ as the number of immediately overlapping k-mers to include in a seed so that we can generate the 21-character long seeds S_l and S_k. The second setting uses $k = 15$ as the k-mer length and $n = 7$ as the number of k-mers to include in a seed. We use the hash64 hash function as provided in the minimap2 implementation to generate the hash values of the k-mers of seeds.

In Supplementary Tables S4.1 - S4.8 we show k-mers, the hash values of the k-mers in their binary form, and the gradual change in the counter vectors used to calculate the hash values for seeds S_k and S_l. We update the counter vectors based on the bits in the hash values of each k-mer. Finally, we show the hash values of S_k and S_l in the last rows of each table. In Supplementary Tables S4.1- S4.4, we use $k = 7$ as the k-mer length and $n = 15$ as the number of immediately overlapping k-mers to include in a seed. In Supplementary Tables S4.5- S4.8, we use $k = 15$ as the k-mer length and $n = 7$ as the number of k-mers to include in a seed.

We make two key observations. First, we observe that the hash values of S_k and S_l are equal ($B(S_k) = B(S_l)$ =0b11000100 01101100 11101001 10110100) when we use a short k-mer with high number of neighbors even though these two seeds differ by 3 k-mers. Second, the hash values of these two seeds are not equal when we use fewer neighbors with larger k-mers. For S_k we find the hash value $B(S_k)$ =0b01101000 01000001 01110100 11000000 and for S_l we find $B(S_l)$ =0b00101101 10110000 01111100 01010011. We note that the bit positions with large values in their corresponding counter vectors are less likely to differ between two seeds when the seeds have a large number of k-mers in common. This motivates as to design more intelligent hash functions that are aware of the values in the counter vectors to increase the chance of generating the hash value for similar seeds.

Table S4.1: Hash Values of the k-mers of seed S_k: CGGATGCTACAGTATATACCA for $k = 7$ and $n = 15$. We show the most significant 16 bits of the counter vector $C(S_k)$. Last row shows the most significant 16 bits of the hash value of the seed.

K-mer	Hash Value	C[31]	C[30]	C[29]	C[28]	C[27]	C[26]	C[25]	C[24]	C[23]	C[22]	C[21]	C[20]	C[19]	C[18]	C[17]	C[16]
CGGATGC	0b 10100000 01111111 10000110 10110101	1	-1	1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	1	1
GGATGCT	0b 10101101 11110000 01110100 11010000	2	-2	2	-2	0	0	-2	0	0	2	2	2	0	0	0	0
GATGCTA	0b 01000010 01001011 11011001 10011011	1	-1	1	-3	-1	-1	-1	-1	-1	3	1	1	1	-1	1	1
ATGCTAC	0b 11001100 01110101 01010011 00100110	2	0	0	-4	0	0	-2	-2	-2	4	2	2	0	0	0	2
TGCTACA	0b 10110001 01101010 10101001 10100111	3	-1	1	-3	-1	-1	-3	-1	-3	5	3	1	1	-1	1	1
GCTACAG	0b 11000101 11010111 11010101 00100101	4	0	0	-4	-2	0	-4	0	-2	6	2	2	0	0	2	2
CTACAGT	0b 11001001 01111101 01001010 10110101	5	1	-1	-5	-1	-1	-5	1	-3	7	3	3	1	1	1	3
TACAGTA	0b 00101011 01101111 11111000 11111000	4	0	0	-6	0	-2	-4	2	-4	8	4	2	2	2	2	4
ACAGTAT	0b 11100100 01001110 01110101 00011010	5	1	1	-7	-1	-1	-5	1	-5	9	3	1	3	3	3	3
CAGTATA	0b 10010010 00001101 00100011 10110100	6	0	0	-6	-2	-2	-4	0	-6	8	2	0	4	4	2	4
AGTATAT	0b 01110100 00110000 10101100 00000000	5	1	1	-5	-3	-1	-5	-1	-7	7	3	1	3	3	1	3
GTATATA	0b 11001111 10110000 11001001 10010110	6	2	0	-6	-2	0	-4	0	-6	6	4	2	2	2	0	2
TATATAC	0b 10000001 00001000 00101111 01111111	7	1	-1	-7	-3	-1	-5	1	-7	5	3	1	3	1	-1	1
ATATACC	0b 11001100 11100000 00101000 11011010	8	2	-2	-8	-2	0	-6	0	-6	6	4	0	2	0	-2	0
TATACCA	0b 00110100 00000100 11110100 10010100	7	1	-1	-7	-3	1	-7	-1	-7	5	3	-1	1	1	-3	-1
		B[31]	B[30]	B[29]	B[28]	B[27]	B[26]	B[25]	B[24]	B[23]	B[22]	B[21]	B[20]	B[19]	B[18]	B[17]	B[16]
		1	1	0	0	0	1	0	0	0	1	1	0	1	1	0	0

Best results are highlighted with **bold** text.

Table S4.2: Hash Values of the k-mers of seed S_k: CGGATGCTACAGTATATACCA for $k = 7$ and $n = 15$. We show the least significant 16 bits of the counter vector $C(S_k)$. Last row shows the least significant 16 bits of the hash value of the seed.

K-mer	Hash Value	C[15]	C[14]	C[13]	C[12]	C[11]	C[10]	C[9]	C[8]	C[7]	C[6]	C[5]	C[4]	C[3]	C[2]	C[1]	C[0]
CGGATGC	0b 10100000 01111111 10000110 10110101	1	-1	-1	-1	-1	1	1	-1	1	-1	1	1	-1	1	-1	1
GGATGCT	0b 10101101 11110000 01110100 11010000	0	0	0	0	-2	2	0	-2	2	0	0	2	-2	0	-2	0
GATGCTA	0b 01000010 01001011 11011001 10011011	1	1	-1	1	-1	1	-1	-1	3	-1	-1	3	-1	-1	-1	1
ATGCTAC	0b 11001100 01110101 01010011 00100110	0	2	-2	2	-2	0	0	0	2	-2	0	2	-2	0	0	0
TGCTACA	0b 10110001 01101010 10101001 10100111	1	1	-1	1	-1	-1	-1	1	3	-3	1	1	-3	1	1	1
GCTACAG	0b 11000101 11010111 11010101 00100101	2	2	-2	2	-2	0	-2	2	2	-4	2	0	-4	2	0	2
CTACAGT	0b 11001001 01111101 01001010 10110101	1	3	-3	1	-1	-1	-1	1	3	-5	3	1	-5	3	-1	3
TACAGTA	0b 00101011 01101111 11111000 11111000	2	4	-2	2	0	-2	-2	0	4	-4	4	2	-4	2	-2	2
ACAGTAT	0b 11100100 01001110 01110101 00011010	1	5	-1	3	-1	-1	-3	1	3	-5	3	3	-3	1	-1	1
CAGTATA	0b 10010010 00001101 00100011 10110100	0	4	0	2	-2	-2	-2	2	4	-6	4	4	-4	2	-2	0
AGTATAT	0b 01110100 00110000 10101100 00000000	1	3	1	1	-1	-1	-3	1	3	-7	3	3	-5	1	-3	-1
GTATATA	0b 11001111 10110000 11001001 10010110	2	4	0	0	0	-2	-4	2	4	-8	2	4	-6	2	-2	-2
TATATAC	0b 10000001 00001000 00101111 01111111	1	3	1	-1	1	-1	-3	3	3	-7	3	5	-5	3	-1	-1
ATATACC	0b 11001100 11100000 00101000 11011010	0	2	2	-2	2	-2	-4	2	4	-6	2	6	-4	2	0	-2
TATACCA	0b 00110100 00000100 11110100 10010100	1	3	3	-1	1	-1	-5	1	5	-7	1	7	-5	3	-1	-3
		B[15]	B[14]	B[13]	B[12]	B[11]	B[10]	B[9]	B[8]	B[7]	B[6]	B[5]	B[4]	B[3]	B[2]	B[1]	B[0]
		1	1	1	0	1	0	0	1	1	0	1	1	0	1	0	0

Best results are highlighted with **bold** text.

Table S4.3: Hash Values of the k-mers of seed S_l: ATGCTACAGTATATACCATCT for $k = 7$ and $n = 15$. We show the most significant 16 bits of the counter vector $C(S_l)$. Last row shows the most significant 16 bits of the hash value of the seed.

K-mer	Hash Value	C[31]	C[30]	C[29]	C[28]	C[27]	C[26]	C[25]	C[24]	C[23]	C[22]	C[21]	C[20]	C[19]	C[18]	C[17]	C[16]
ATGCTAC	0b 11001100 01110101 01010011 00100110	1	1	-1	-1	1	1	-1	-1	-1	1	1	1	-1	1	-1	1
TGCTACA	0b 10110001 01101010 10101001 10100111	2	0	0	0	0	0	-2	0	-2	2	2	0	0	0	0	0
GCTACAG	0b 11000101 11010111 11010101 00100101	3	1	-1	-1	-1	1	-3	1	-1	3	1	1	-1	1	1	1
CTACAGT	0b 11001001 01111101 01001010 10110101	4	2	-2	-2	0	0	-4	2	-2	4	2	2	0	2	0	2
TACAGTA	0b 00101011 01101111 11111000 11111000	3	1	-1	-3	1	-1	-3	3	-3	5	3	1	1	3	1	3
ACAGTAT	0b 11100100 01001110 01110101 00011010	4	2	0	-4	0	0	-4	2	-4	6	2	0	2	4	2	2
CAGTATA	0b 10010010 00001101 00100011 10110100	5	1	-1	-3	-1	-1	-3	1	-5	5	1	-1	3	5	1	3
AGTATAT	0b 01110100 00110000 10101100 00000000	4	2	0	-2	-2	0	-4	0	-6	4	2	0	2	4	0	2
GTATATA	0b 11001111 10110000 11001001 10010110	5	3	-1	-3	-1	1	-3	1	-5	3	3	1	1	3	-1	1
TATATAC	0b 10000001 00001000 00101111 01111111	6	2	-2	-4	-2	0	-4	2	-6	2	2	0	2	2	-2	0
ATATACC	0b 11001100 11100000 00101000 11011010	7	3	-3	-5	-1	1	-5	1	-5	3	3	-1	1	1	-3	-1
TATACCA	0b 00110100 00000100 11110100 10010100	6	2	-2	-4	-2	2	-6	0	-6	2	2	-2	0	2	-4	-2
ATACCAT	0b 00000111 10111111 11010101 01001100	5	1	-3	-5	-3	3	-5	1	-5	1	3	-1	1	3	-3	-1
TACCATC	0b 01010110 11100111 00100010 11001101	4	2	-4	-4	-4	4	-4	0	-4	2	4	-2	0	4	-2	0
ACCATCT	0b 00010010 11001000 11001010 11100111	3	1	-5	-3	-5	3	-3	-1	-3	3	3	-3	1	3	-3	-1
		B[31]	B[30]	B[29]	B[28]	B[27]	B[26]	B[25]	B[24]	B[23]	B[22]	B[21]	B[20]	B[19]	B[18]	B[17]	B[16]
		1	1	0	0	0	1	0	0	0	1	1	0	1	1	0	0

Best results are highlighted with **bold** text.

Table S4.4: Hash Values of the k-mers of seed S_l: ATGCTACAGTATATACCATCT for $k = 7$ and $n = 15$. We show the least significant 16 bits of the counter vector $C(S_l)$. Last row shows the least significant 16 bits of the hash value of the seed.

K-mer	Hash Value	C[15]	C[14]	C[13]	C[12]	C[11]	C[10]	C[9]	C[8]	C[7]	C[6]	C[5]	C[4]	C[3]	C[2]	C[1]	C[0]
ATGCTAC	0b 11001100 01110101 01010011 00100110	-1	1	-1	1	-1	-1	1	1	-1	-1	1	-1	-1	1	1	-1
TGCTACA	0b 10110001 01101010 10101001 10100111	0	0	0	0	0	-2	0	2	0	-2	2	-2	-2	2	2	0
GCTACAG	0b 11000101 11010111 11010101 00100101	1	1	-1	1	-1	-1	-1	3	-1	-3	3	-3	-3	3	1	1
CTACAGT	0b 11001001 01111101 01001010 10110101	0	2	-2	0	0	-2	0	2	0	-4	4	-2	-4	4	0	2
TACAGTA	0b 00101011 01101111 11111000 11111000	1	3	-1	1	1	-3	-1	1	1	-3	5	-1	-3	3	-1	1
ACAGTAT	0b 11100100 01001110 01110101 00011010	0	4	0	2	0	-2	-2	2	0	-4	4	0	-2	2	0	0
CAGTATA	0b 10010010 00001101 00100011 10110100	-1	3	1	1	-1	-3	-1	3	1	-5	5	1	-3	3	-1	-1
AGTATAT	0b 01110100 00110000 10101100 00000000	0	2	2	0	0	-2	-2	2	0	-6	4	0	-4	2	-2	-2
GTATATA	0b 11001111 10110000 11001001 10010110	1	3	1	-1	1	-3	-3	3	1	-7	3	1	-5	3	-1	-3
TATATAC	0b 10000001 00001000 00101111 01111111	0	2	2	-2	2	-2	-2	4	0	-6	4	2	-4	4	0	-2
ATATACC	0b 11001100 11100000 00101000 11011010	-1	1	3	-3	3	-3	-3	3	1	-5	3	3	-3	3	1	-3
TATACCA	0b 00110100 00000100 11110100 10010100	0	2	4	-2	2	-2	-4	2	2	-6	2	4	-4	4	0	-4
ATACCAT	0b 00000111 10111111 11010101 01001100	1	3	3	-1	1	-1	-5	3	1	-5	1	3	-3	5	-1	-5
TACCATC	0b 01010110 11100111 00100010 11001101	0	2	4	-2	0	-2	-4	2	2	-4	0	2	-2	6	-2	-4
ACCATCT	0b 00010010 11001000 11001010 11100111	1	3	3	-3	1	-3	-3	1	3	-3	1	1	-3	7	-1	-3
		B[15]	B[14]	B[13]	B[12]	B[11]	B[10]	B[9]	B[8]	B[7]	B[6]	B[5]	B[4]	B[3]	B[2]	B[1]	B[0]
		1	1	1	0	1	0	0	1	1	0	1	1	0	1	0	0

Best results are highlighted with **bold** text.

Table S4.5: Hash Values of the k-mers of seed S_k: CGGATGCTACAGTATATACCA for $k = 15$ and $n = 7$. We show the most significant 16 bits of the counter vector $C(S_k)$. Last row shows the most significant 16 bits of the hash value of the seed.

K-mer	Hash Value	C[31]	C[30]	C[29]	C[28]	C[27]	C[26]	C[25]	C[24]	C[23]	C[22]	C[21]	C[20]	C[19]	C[18]	C[17]	C[16]
CGGATGCTACAGTAT	0b 01001010 11101011 00100110 11001101	-1	1	-1	-1	1	-1	1	-1	1	1	1	-1	1	-1	1	1
GGATGCTACAGTATA	0b 01101100 01000011 11111000 11000000	-2	2	0	-2	2	0	0	-2	0	2	0	-2	0	-2	2	2
GATGCTACAGTATAT	0b 01011000 01000101 00110001 11011000	-3	3	-1	-1	3	-1	-1	-3	-1	3	-1	-3	-1	-1	1	3
ATGCTACAGTATATA	0b 11100001 01110100 01100010 01000010	-2	4	0	-2	2	-2	-2	-2	-2	4	0	-2	-2	0	0	2
TGCTACAGTATATAC	0b 10111100 10011010 00111111 01011011	-1	3	1	-1	3	-1	-3	-3	-1	3	-1	-1	-1	-1	1	1
GCTACAGTATATACC	0b 11101010 01001100 01000100 11100001	0	2	2	-2	4	-2	-2	-4	-2	4	-2	-2	0	0	0	0
CTACAGTATATACCA	0b 00101001 10010001 11111100 01010000	-1	1	3	-3	5	-3	-3	-3	-1	3	-3	-1	-1	-1	-1	1
Seed CGGATGCTACAGTATATACCA		B[31]	B[30]	B[29]	B[28]	B[27]	B[26]	B[25]	B[24]	B[23]	B[22]	B[21]	B[20]	B[19]	B[18]	B[17]	B[16]
		0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	1

Best results are highlighted with **bold** text.

Table S4.6: Hash Values of the k-mers of seed S_k: CGGATGCTACAGTATATACCA for k = 15 and n = 7. We show the least significant 16 bits of the counter vector $C(S_k)$. Last row shows the least significant 16 bits of the hash value of the seed.

K-mer	Hash Value	C[15]	C[14]	C[13]	C[12]	C[11]	C[10]	C[9]	C[8]	C[7]	C[6]	C[5]	C[4]	C[3]	C[2]	C[1]	C[0]
CGGATGCTACAGTAT	0b 01001010 11101011 00100110 11001101	-1	-1	1	-1	-1	1	1	-1	1	1	-1	-1	1	1	-1	1
GGATGCTACAGTATA	0b 01101100 01000011 11111000 11000000	0	0	2	0	0	0	0	-2	2	2	-2	-2	0	0	-2	0
GATGCTACAGTATAT	0b 01011000 01000101 00110001 11011000	-1	-1	3	1	-1	-1	-1	-1	3	3	-3	-1	1	-1	-3	-1
ATGCTACAGTATATA	0b 11100001 01110100 01100010 01000010	-2	0	4	0	-2	-2	0	-2	2	4	-4	-2	0	-2	-2	-2
TGCTACAGTATATAC	0b 10111100 10011010 00111111 01011011	-3	-1	5	1	-1	-1	1	-1	1	5	-5	-1	1	-3	-1	-1
GCTACAGTATATACC	0b 11101010 01001100 01000100 11100001	-4	0	4	0	-2	0	0	-2	2	6	-4	-2	0	-4	-2	0
CTACAGTATATACCA	0b 00101001 10010001 11111100 01010000	-3	1	5	1	-1	1	-1	-3	1	7	-5	-1	-1	-5	-3	-1
Seed CGGATGCTACAGTATATACCA		B[15]	B[14]	B[13]	B[12]	B[11]	B[10]	B[9]	B[8]	B[7]	B[6]	B[5]	B[4]	B[3]	B[2]	B[1]	B[0]
		0	1	1	1	0	1	0	0	1	1	0	0	0	0	0	0

Best results are highlighted with **bold** text.

Table S4.7: Hash Values of the k-mers of seed S_l: ATGCTACAGTATATACCATCT for k = 15 and n = 7. We show the most significant 16 bits of the counter vector $C(S_l)$. Last row shows the most significant 16 bits of the hash value of the seed.

K-mer	Hash Value	C[31]	C[30]	C[29]	C[28]	C[27]	C[26]	C[25]	C[24]	C[23]	C[22]	C[21]	C[20]	C[19]	C[18]	C[17]	C[16]
ATGCTACAGTATATA	0b 11100001 01110100 01100010 01000010	1	1	1	-1	-1	-1	-1	1	-1	1	1	1	-1	1	-1	-1
TGCTACAGTATATAC	0b 10111100 10011010 00111111 01011011	2	0	2	0	0	0	-2	0	0	0	0	2	0	0	0	-2
GCTACAGTATATACC	0b 11101010 01001100 01000100 11100001	3	1	3	-1	1	-1	-1	-1	-1	1	-1	1	1	1	-1	-3
CTACAGTATATACCA	0b 00101001 10010001 11111100 01010000	2	0	4	-2	2	-2	-2	0	0	0	-2	2	0	0	-2	-2
TACAGTATATACCAT	0b 00001110 00100000 11011100 11110110	1	-1	3	-3	3	-1	-1	-1	-1	-1	-1	1	-1	-1	-3	-3
ACAGTATATACCATC	0b 00101111 10111010 00010000 11011111	0	-2	4	-4	4	0	0	0	0	-2	0	2	0	-2	-2	-4
CAGTATATACCATCT	0b 01100101 11100111 10111011 00111011	-1	-1	5	-5	5	1	-1	1	1	-1	1	1	-1	-1	-1	-3
Seed ATGCTACAGTATATACCATCT		B[31]	B[30]	B[29]	B[28]	B[27]	B[26]	B[25]	B[24]	B[23]	B[22]	B[21]	B[20]	B[19]	B[18]	B[17]	B[16]
		0	0	1	0	1	1	0	1	1	0	1	1	0	0	0	0

Best results are highlighted with **bold** text.

Table S4.8: Hash Values of the k-mers of seed S_l: ATGCTACAGTATATACCATCT for k = 15 and n = 7. We show the least significant 16 bits of the counter vector $C(S_l)$. Last row shows the least significant 16 bits of the hash value of the seed.

K-mer	Hash Value	C[15]	C[14]	C[13]	C[12]	C[11]	C[10]	C[9]	C[8]	C[7]	C[6]	C[5]	C[4]	C[3]	C[2]	C[1]	C[0]
ATGCTACAGTATATA	0b 11100001 01110100 01100010 01000010	-1	1	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	1	-1
TGCTACAGTATATAC	0b 10111100 10011010 00111111 01011011	-2	0	2	0	0	0	2	0	-2	2	-2	0	0	-2	2	0
GCTACAGTATATACC	0b 11101010 01001100 01000100 11100001	-3	1	1	-1	-1	1	1	-1	-1	3	-1	-1	-1	-3	1	1
CTACAGTATATACCA	0b 00101001 10010001 11111100 01010000	-2	2	2	0	0	2	0	-2	-2	4	-2	0	-2	-4	0	0
TACAGTATATACCAT	0b 00001110 00100000 11011100 11110110	-1	3	1	1	1	3	-1	-3	-1	5	-1	1	-3	-3	1	-1
ACAGTATATACCATC	0b 00101111 10111010 00010000 11011111	-2	2	0	2	0	2	-2	-4	0	6	-2	2	-2	-2	2	0
CAGTATATACCATCT	0b 01100101 11100111 10111011 00111011	-1	1	1	3	1	1	-1	-3	-1	5	-1	3	-1	-3	3	1
Seed ATGCTACAGTATATACCATCT		B[15]	B[14]	B[13]	B[12]	B[11]	B[10]	B[9]	B[8]	B[7]	B[6]	B[5]	B[4]	B[3]	B[2]	B[1]	B[0]
		0	1	1	1	1	1	0	0	0	1	0	1	0	0	1	1

Best results are highlighted with **bold** text.

S4.2 Parameter Exploration

S4.2.1 The trade-off between BLEND-I and BLEND-S

Our goal is to show the performance and accuracy trade-offs between the seeding techniques that BLEND supports: BLEND-I and BLEND-S. In Supplementary Figures S4.1 and S4.2, we show the performance and peak memory usage comparisons when using BLEND-I and BLEND-S as the seeding technique by keeping all the other relevant parameters identical (e.g., number of k-mers to include in a seed n, window length w). In Supplementary Table S4.9, we show the assembly quality comparisons in terms of the accuracy and contiguity of the assemblies that we generate using the overlaps that BLEND-I and BLEND-S find. In Supplementary Tables S4.10 and S4.11, we show the read mapping quality and accuracy results using these two seeding techniques, respectively.

We also show the values for different parameters we test with BLEND in Supplementary Table S4.12. We determine the default parameters of BLEND empirically based on the combination of best performance, memory overhead, and accuracy results.

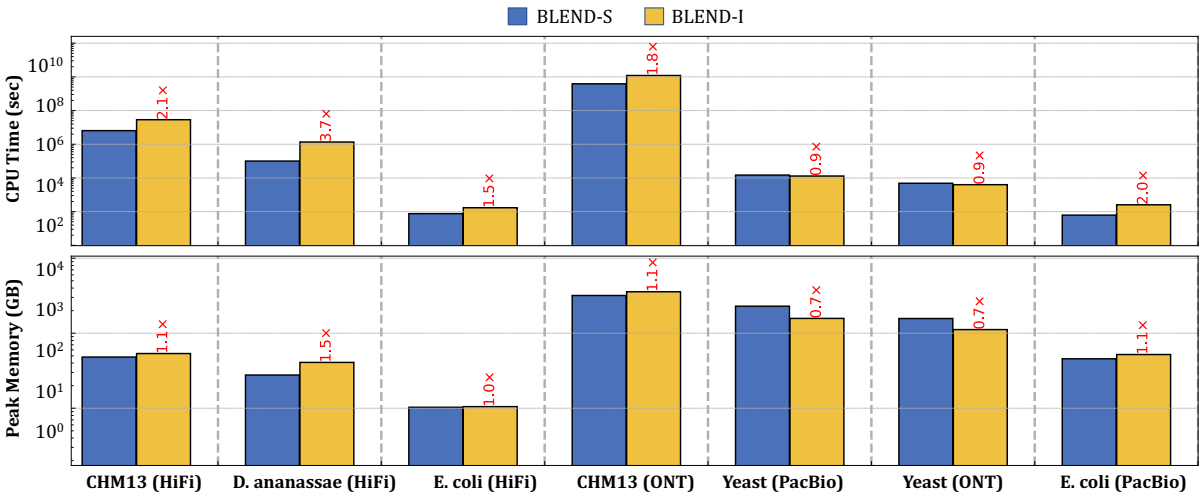

Figure S4.1: CPU time and peak memory footprint comparisons of read overlapping.

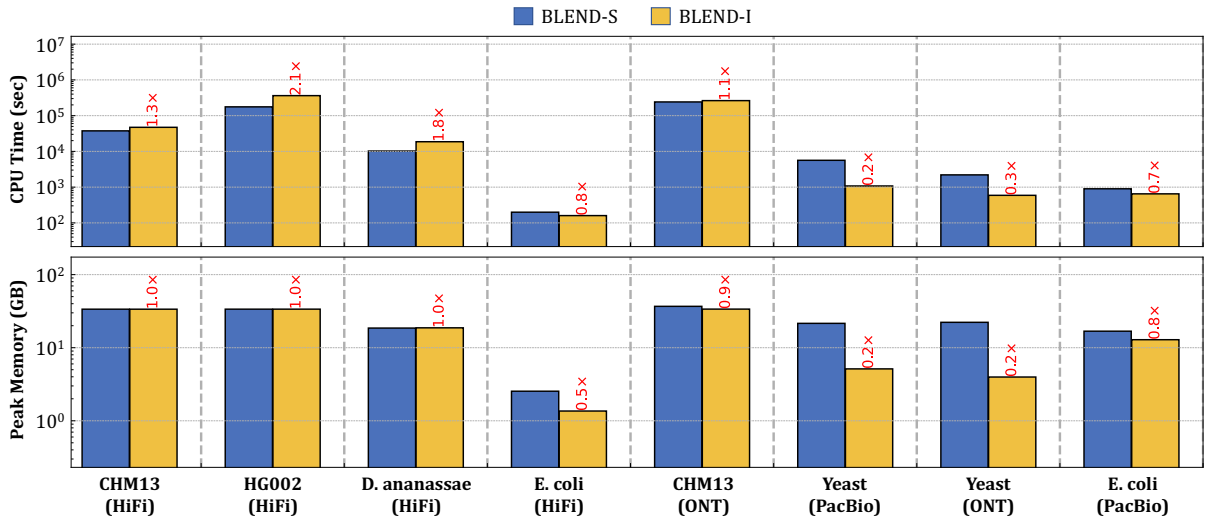

Figure S4.2: CPU time and peak memory footprint comparisons of read mapping.

Table S4.9: Assembly quality comparisons between `BLEND-I` and `BLEND-S`.

Dataset	Tool	Average Identity (%)	Genome Fraction (%)	K-mer Compl. (%)	Aligned Length (Mbp)	Mismatch per 100Kbp (#)	Average GC (%)	Assembly Length (Mbp)	Largest Contig (Mbp)	NGA50 (Kbp)	NG50 (Kbp)
CHM13 (HiFi)	BLEND-I	99.7535	96.7203	83.65	3,054.49	48.49	**40.79**	**3,059.29**	**41.8342**	**8,507.53**	**8,508.92**
	BLEND-S	**99.8526**	**98.4847**	**90.15**	**3,092.54**	**22.02**	40.78	3,095.21	22.8397	5,442.25	5,442.31
	Reference	100	100	100	3,054.83	0.00	40.85	3,054.83	248.387	154,260	154,260
D. ananassae (HiFi)	BLEND-I	99.6890	97.2290	77.85	**270.218**	233.18	41.95	280.388	5.01099	356.745	356.745
	BLEND-S	**99.7856**	**97.2308**	**86.43**	240.391	**143.13**	**41.75**	**247.153**	**6.23256**	**792.407**	**798.913**
	Reference	100	100	100	213.805	0.00	41.81	213.818	30.6728	26,427.4	26,427.4
E. coli (HiFi)	BLEND-I	99.6902	**99.8824**	79.36	5.04157	17.92	**50.52**	**5.04263**	**4.94601**	**4,025.48**	**4,946.01**
	BLEND-S	**99.8320**	99.8801	**87.91**	**5.12155**	**3.77**	50.53	5.12155	3.41699	3,416.99	3,416.99
	Reference	100	100	100	5.04628	0.00	50.52	5.04628	4.94446	4,944.46	4,944.46
CHM13 (ONT)	BLEND-I	N/A	N/A	**29.26**	**2,891.28**	4,077.53	**41.32**	**2,897.87**	**25.2071**	**5,061.52**	**5,178.59**
	BLEND-S	N/A	N/A	0	0.010546	**3,250.70**	51.30	0.010548	0.010548	0	0
	Reference	100	100	100	3,117.29	0.00	40.75	3,117.29	248.387	150,617	150,617
Yeast (PacBio)	BLEND-I	89.1677	**97.0854**	**33.81**	12.3938	**2,672.37**	38.84	**12.4176**	**1.54807**	**635.966**	**636.669**
	BLEND-S	**90.3347**	83.8814	33.17	**22.9473**	4,795.58	**38.71**	22.9523	0.265118	114.125	116.143
	Reference	100	100	100	12.1571	0.00	38.15	12.1571	1.53193	924.431	924.431
Yeast (ONT)	BLEND-I	89.6889	**99.2974**	**35.95**	**12.3222**	2,529.47	38.64	**12.3225**	**1.10582**	**793.046**	**793.046**
	BLEND-S	**91.0865**	7.9798	4.90	0.898565	**2,006.91**	**38.35**	0.899654	0.043321	0	0
	Reference	100	100	100	12.1571	0.00	38.15	12.1571	1.53193	924.431	924.431
E. coli (PacBio)	BLEND-I	88.5806	**96.5238**	**32.32**	**5.90024**	1,857.56	**49.81**	**6.21598**	**2.40671**	**769.981**	**2,060.4**
	BLEND-S	**90.3551**	36.6230	17.07	2.10137	**1,299.50**	48.91	2.10704	0.095505	0	0
	Reference	100	100	100	5.6394	0.00	50.43	5.6394	5.54732	5,547.32	5,547.32

Best results are highlighted with **bold** text. For most metrics, the best results are the ones closest to the corresponding value of the reference genome.
The best results for *Aligned Length* are determined by the highest number within each dataset. We do not highlight the reference results as the best results.
N/A indicates that we could not generate the corresponding result because tool, QUAST, or dnadiff failed to generate the statistic.

Table S4.10: Read mapping quality comparisons between BLEND-I and BLEND-S.

Dataset	Tool	Average Depth of Cov. (×)	Breadth of Coverage (%)	Aligned Reads (#)	Properly Paired (%)
CHM13 (HiFi)	BLEND-I	16.58	99.991	**3,172,305**	NA
	BLEND-S	16.58	99.991	3,171,916	NA
HG002 (HiFi)	BLEND-I	**51.25**	**92.245**	6,813,886	NA
	BLEND-S	11.24	13.860	**11,424,762**	NA
D. ananassae (HiFi)	BLEND-I	**57.51**	99.650	**1,249,666**	NA
	BLEND-S	57.37	**99.662**	1,223,388	NA
E. coli (HiFi)	BLEND-I	99.14	99.897	**39,064**	NA
	BLEND-S	99.14	99.897	39,048	NA
CHM13 (ONT)	BLEND-I	**29.34**	**99.999**	**10,322,767**	NA
	BLEND-S	17.51	99.700	5,760,401	NA
Yeast (PacBio)	BLEND-I	**195.87**	**99.980**	**270,064**	NA
	BLEND-S	142.31	99.975	179,039	NA
Yeast (ONT)	BLEND-I	**97.88**	**99.964**	**134,919**	NA
	BLEND-S	59.57	99.906	75,110	NA
E. coli (PacBio)	BLEND-I	**97.51**	100	**83,924**	NA
	BLEND-S	56.87	100	40,694	NA

Best results are highlighted with **bold** text.

Properly paired rate is only available for paired-end Illumina reads.

Table S4.11: Read mapping accuracy comparisons between BLEND-I and BLEND-S.

Dataset	Overall Error Rate (%)	
	BLEND-I	BLEND-S
CHM13 (ONT)	**1.5168427**	5.996888
Yeast (PacBio)	**0.2403134**	0.6959378
Yeast (ONT)	**0.2386617**	0.6284117

Best results are highlighted with **bold** text.

Table S4.12: Performance, memory, and accuracy comparisons using different parameter settings in BLEND.

Tool	K-mer Length (k)	# of k-mers in a Seed (n)	Window Length (w)	CPU Time (seconds)	Peak Memory (KB)	Average Identity (%)	Genome Fraction (%)
BLEND	9	11	200	62.38	1,115,384	99.7255	99.8502
BLEND	9	13	200	58.13	994,120	99.7294	99.7808
BLEND	9	15	200	49.79	1,030,148	99.7411	99.7619
BLEND	9	17	200	45.03	960,080	99.7302	99.7460
BLEND	9	21	200	36.84	976,456	99.7257	99.6640
BLEND	15	5	200	83.05	1,168,612	99.6735	99.7625
BLEND	15	7	200	74.93	1,137,360	99.7009	99.5874
BLEND	15	11	200	58.09	1,051,912	99.7149	99.1166
BLEND	19	5	200	77.16	1,130,604	99.7312	99.8802
BLEND	19	7	200	50.50	1,078,596	99.7880	99.8424
BLEND	19	11	200	46.26	977,060	99.8078	99.6438
BLEND	21	5	200	67.85	1,116,684	99.7472	99.8835
BLEND	21	7	200	61.63	1,042,724	99.7969	99.8605
BLEND	21	11	200	42.35	969,184	99.8340	99.7515
BLEND	25	5	200	65.61	1,057,804	99.7769	99.8818
BLEND	25	7	200	54.88	1,029,888	99.8320	99.8801
BLEND	25	11	200	37.01	936,260	99.8646	99.8001
BLEND	25	15	200	29.83	866,208	99.8838	99.7307
BLEND	25	17	200	29.59	826,456	99.8784	99.7521
BLEND	25	21	200	26.09	791,736	99.8774	99.6955
BLEND	9	11	50	263.82	1,786,516	99.7013	99.8612
BLEND	9	13	50	411.24	1,805,800	99.6995	99.8573
BLEND	9	15	50	271.00	1,729,784	99.6798	99.8517
BLEND	9	17	50	238.52	1,690,912	99.6690	99.8083
BLEND	9	21	50	206.76	1,725,168	99.6496	99.8150
BLEND	15	5	50	330.84	1,785,456	99.6634	99.8604
BLEND	15	7	50	337.95	1,812,052	99.6280	99.8177
BLEND	15	11	50	236.82	1,803,816	99.5831	99.6893
BLEND	19	5	50	328.67	1,692,248	99.7077	99.8794
BLEND	19	7	50	295.57	1,713,940	99.7188	99.8579
BLEND	19	11	50	201.79	1,700,412	99.7015	99.8578
BLEND	21	5	50	378.58	1,625,388	99.7120	99.8832
BLEND	21	7	50	278.56	1,695,476	99.7333	99.8832
BLEND	21	11	50	189.33	1,694,820	99.7623	99.8594
BLEND	25	5	50	323.69	1,685,304	99.7272	99.8831
BLEND	25	7	50	211.78	1,647,984	99.7722	99.8831
BLEND	25	11	50	170.60	1,683,736	99.8094	99.8866
BLEND	25	15	50	142.42	1,622,452	99.8170	99.8576
BLEND	25	17	50	103.96	1,590,776	99.8073	99.8206
BLEND	25	21	50	109.62	1,548,228	99.7792	99.7880
BLEND	9	11	20	837.50	2,769,552	99.6916	99.8784
BLEND	9	13	20	813.50	2,765,480	99.6834	99.8785
BLEND	9	15	20	764.91	2,795,848	99.6797	99.8756
BLEND	9	17	20	739.52	2,823,188	99.6801	99.8802

We use the *E.coli* dataset for all these runs

S4.2.2 The trade-off between BLEND and minimap2

Our goal is to compare BLEND and minimap2 using the same set of parameters that BLEND uses when generating its results. To achieve this, we control the following two conditions. First, we ensure that we use the same seeding technique that minimap2 uses. To this end, we use the BLEND-I seeding technique, which uses minimizers as seeds. We should note that BLEND-I does not always provide the best results in terms of performance or accuracy for the HiFi reads as the default seeding technique is BLEND-S for HiFi datasets in BLEND.

Second, we use the same seed length when we compare BLEND with minimap2. In minimap2, the seed length is the same as the k-mer length as minimap2 finds the minimizer k-mers from the hash values of k-mers. The seed length in BLEND-I is determined by *both* the k-mer length and the number of k-mers that we include in a seed (i.e., n). For example, BLEND uses the BLEND-I seeding technique with the k-mer length $k = 19$ and the number of neighbors $n = 5$ for the PacBio reads. Combining immediately overlapping 5-many 19-mers generates seeds with length $19 + 5 - 1 = 23$. Thus, BLEND-I uses seeds of length 23 based on these parameters. Supplementary Table S4.15 shows the seed length calculation for both BLEND-I and BLEND-S. We calculate the seed lengths for the datasets where BLEND uses BLEND-I as the default option (i.e., the PacBio and ONT datasets) in read overlapping. We note that BLEND uses the same seed length and window length as in minimap2 for mapping long reads. Thus, we do not report the read mapping results in this section, which are already reported in the main paper when comparing BLEND with minimap2. To run minimap2 with the same parameter conditions, we apply the same seed length and the window length that BLEND uses to minimap2 using the k and w parameters, respectively. We show these parameters in Supplementary Table S4.16 (minimap-Eq). In the results we show below, minimap-Eq indicates the runs of minimap2 when using the same set of parameters that BLEND uses with the BLEND-I technique.

In Supplementary Figure S4.3, we show the performance and peak memory comparisons when using BLEND with the BLEND-I seeding technique, minimap2, and minimap2-Eq. In Supplementary Table S4.13, we show the assembly quality comparisons in terms of the accuracy and contiguity of the assemblies that we generate using the overlaps that each tool finds.

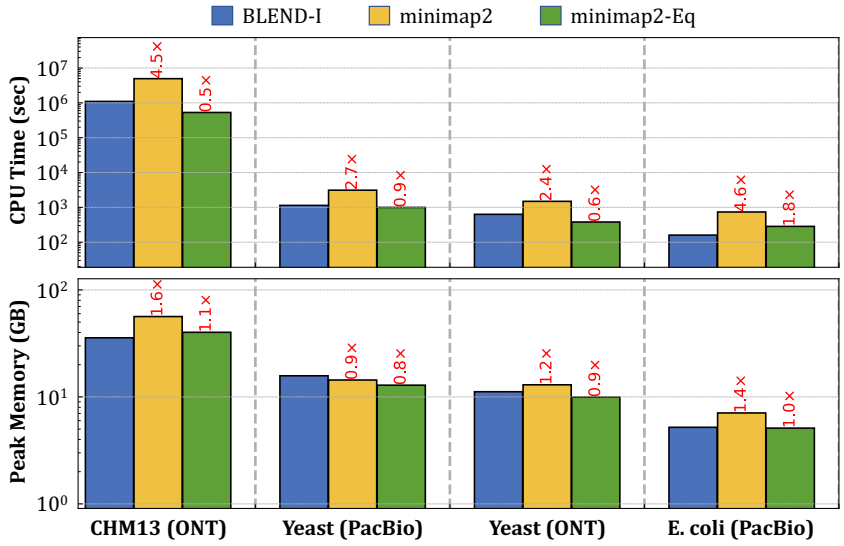

Figure S4.3: CPU time and peak memory footprint comparisons of read overlapping.

Table S4.13: Assembly quality comparisons when using the parameters equivalent to `BLEND-I`.

Dataset	Tool	Average Identity (%)	Genome Fraction (%)	K-mer Compl. (%)	Aligned Length (Mbp)	Mismatch per 100Kbp (#)	Average GC (%)	Assembly Length (Mbp)	Largest Contig (Mbp)	NGA50 (Kbp)	NG50 (Kbp)
CHM13 (ONT)	BLEND-I	N/A	N/A	29.26	2,891.28	4,077.53	**41.32**	2,897.87	25.2071	5,061.52	5,178.59
	minimap2	N/A	N/A	28.32	2,860.26	4,660.73	41.36	**2,908.55**	**66.7564**	**13,189.2**	**13,820.3**
	minimap2-Eq	N/A	N/A	**29.32**	**3,117.29**	**4,025.22**	**41.32**	2,882.94	24.6651	3,634.05	3,653.47
	Reference	100	100	100	3,117.29	0.00	40.75	3,117.29	248.387	150,617	150,617
Yeast (PacBio)	BLEND-I	89.1677	97.0854	33.81	**12.3938**	2,672.37	38.84	12.4176	1.54807	635.966	636.669
	minimap2	88.9002	96.9709	33.38	12.0128	2,684.38	38.85	**12.3325**	**1.56078**	**810.046**	**828.212**
	minimap2-Eq	**89.2166**	**97.2674**	**33.93**	12.3886	**2,653.08**	**38.82**	12.4241	1.53435	643.136	781.136
	Reference	100	100	100	12.1571	0.00	38.15	12.1571	1.53193	924.431	924.431
Yeast (ONT)	BLEND-I	**89.6889**	99.2974	**35.95**	**12.3222**	2,529.47	**38.64**	**12.3225**	1.10582	793.046	793.046
	minimap2	88.9393	**99.6878**	34.84	12.304	2,782.59	38.74	12.3725	**1.56005**	**796.718**	**941.588**
	minimap2-Eq	89.6653	97.3273	35.62	11.826	**2,465.87**	**38.64**	11.8282	1.07367	605.201	677.415
	Reference	100	100	100	12.1571	0.00	38.15	12.1571	1.53193	924.431	924.431
E. coli (PacBio)	BLEND-I	88.5806	96.5238	32.32	**5.90024**	1,857.56	**49.81**	6.21598	2.40671	769.981	2,060.4
	minimap2	88.1365	92.7603	30.74	5.37728	2,005.72	49.66	**6.02707**	3.77098	367.442	3,770.98
	minimap2-Eq	**88.6371**	**96.8540**	**32.33**	5.82218	**1,816.29**	49.76	6.05821	**3.77318**	**1,119.04**	**3,773.18**
	Reference	100	100	100	5.6394	0.00	50.43	5.6394	5.54732	5,547.32	5,547.32

Best results are highlighted with **bold** text. For most metrics, the best results are the ones closest to the corresponding value of the reference genome.
The best results for *Aligned Length* are determined by the highest number within each dataset. We do not highlight the reference results as the best results.
N/A indicates that we could not generate the corresponding result because tool, QUAST, or dnadiff failed to generate the statistic.

S4.3 The Genome-wide Coverage Comparison

We map the HG002 reads to the human reference genome (GRCh37) using BLEND and minimap2. Supplementary Figures S4.4 and S4.5 show the depth of mapping coverage at each position of the reference genome chromosomes for BLEND and minimap2 on the left and right sides of the figures, respectively. To calculate the position-wise depth of coverage, we use the `multiBamSummary` tool from the deepTools2 package [1048]. The `multiBamSummary` tool divides the reference genome into consecutive bins of equal size (10,000 bases) to calculate the genome-wide coverage in fine granularity. For positions where the coverage is higher than 500×, we set the coverage to 500× for visibility reasons as there are only a negligible amount of such regions where either BLEND or minimap2 exceeds this threshold without the other one exceeding it.

To find the positions where the depth of coverage significantly differs between BLEND and minimap2, we subtract the minimap2 coverage from the BLEND coverage for each chromosome position that we show in Figures S4.4 and S4.5. We show the coverage differences in Figure S4.6, where the positive values show the positions that minimap2 has a higher depth of coverage than BLEND, and negative values show the positions that BLEND has a higher coverage.

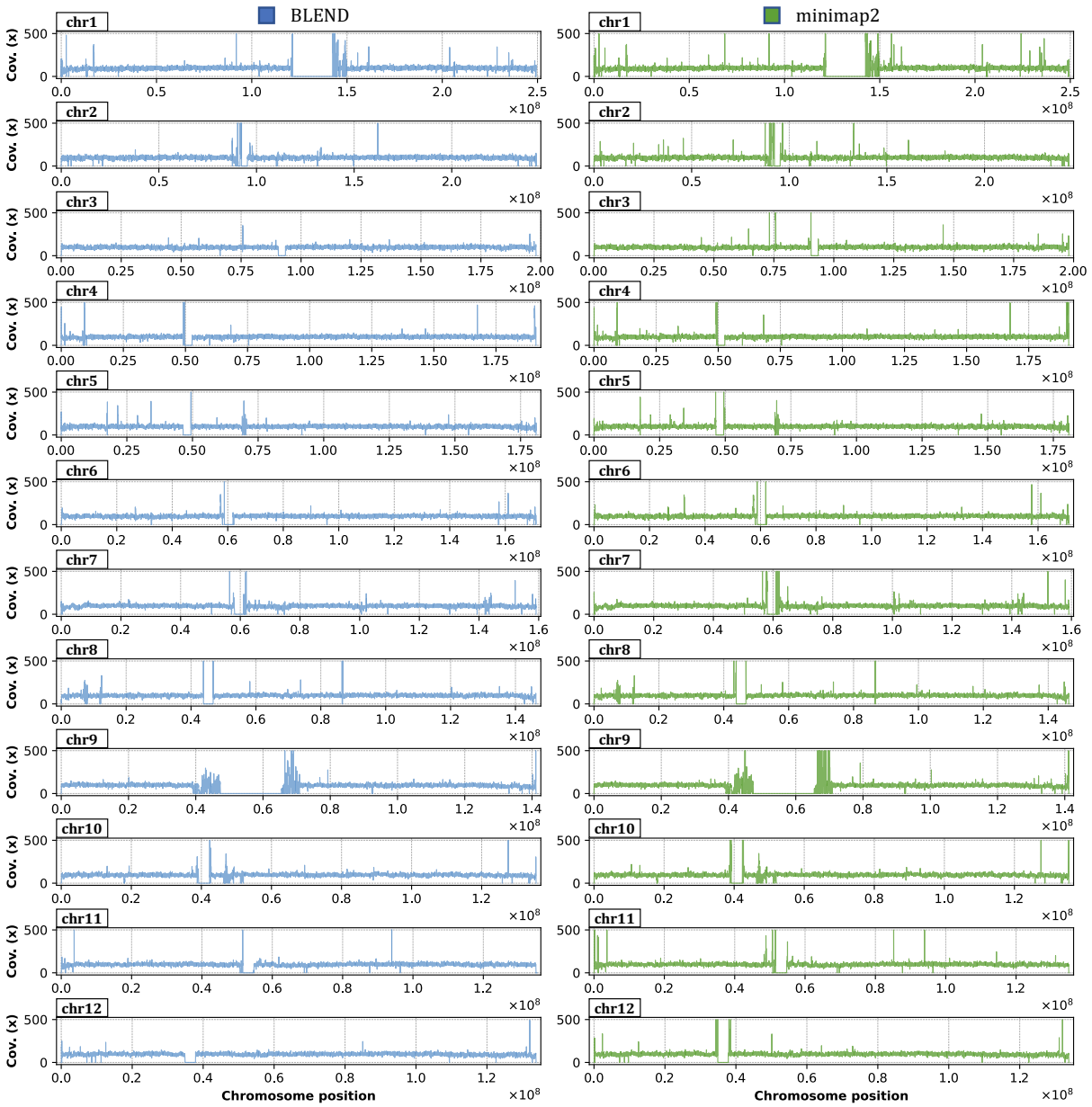

Figure S4.4: Depth of coverage at each position (binned) of the GRCh37 reference genome (chromosomes 1 to 12) after mapping the HG002 reads using BLEND and minimap2. We label the chromosomes on the top left corner of each plot.

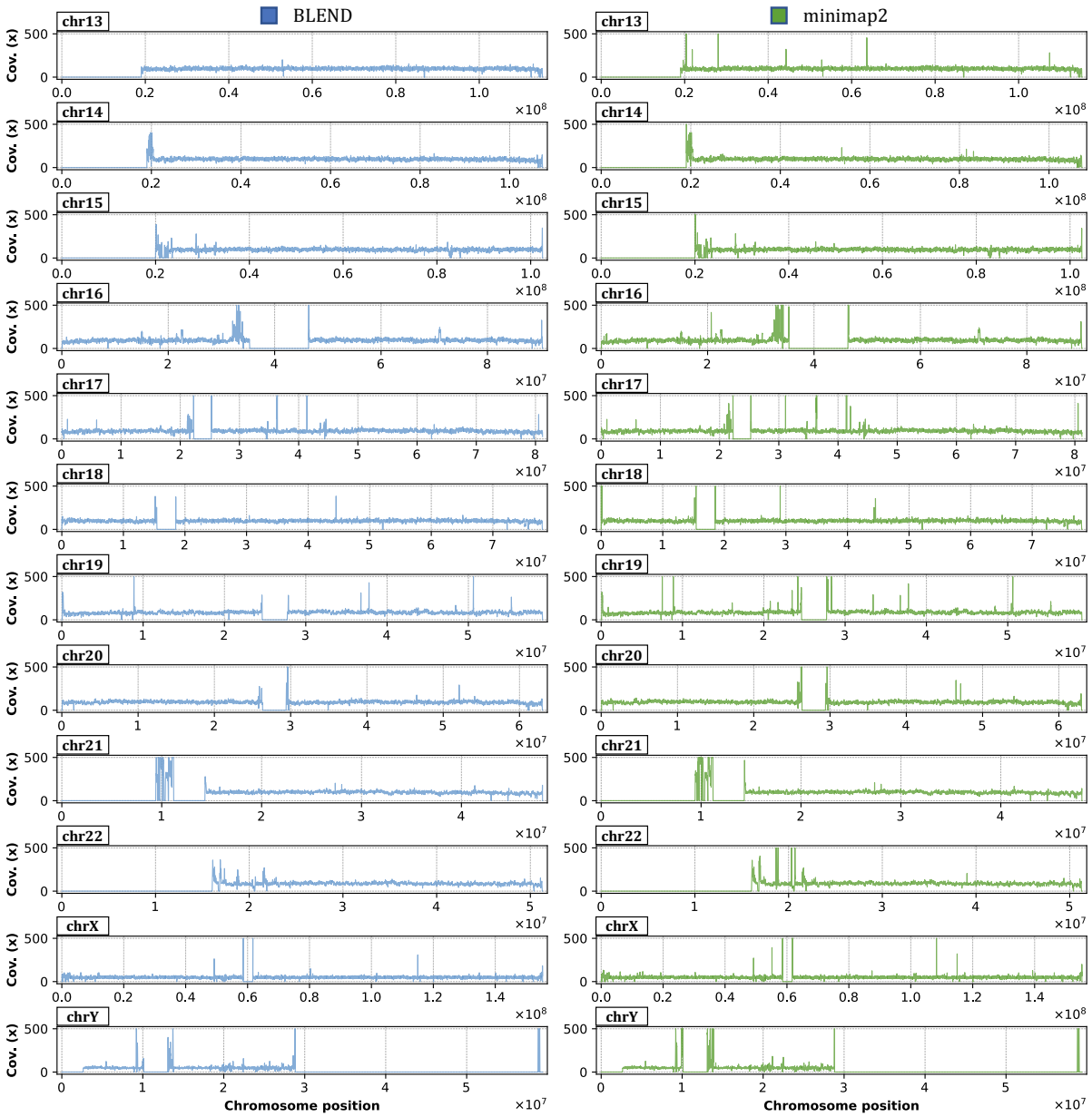

Figure S4.5: Depth of coverage at each position (binned) of the GRCh37 reference genome (chromosomes 13 to Y) after mapping the HG002 reads using BLEND and minimap2. We label the chromosomes on the top left corner of each plot.

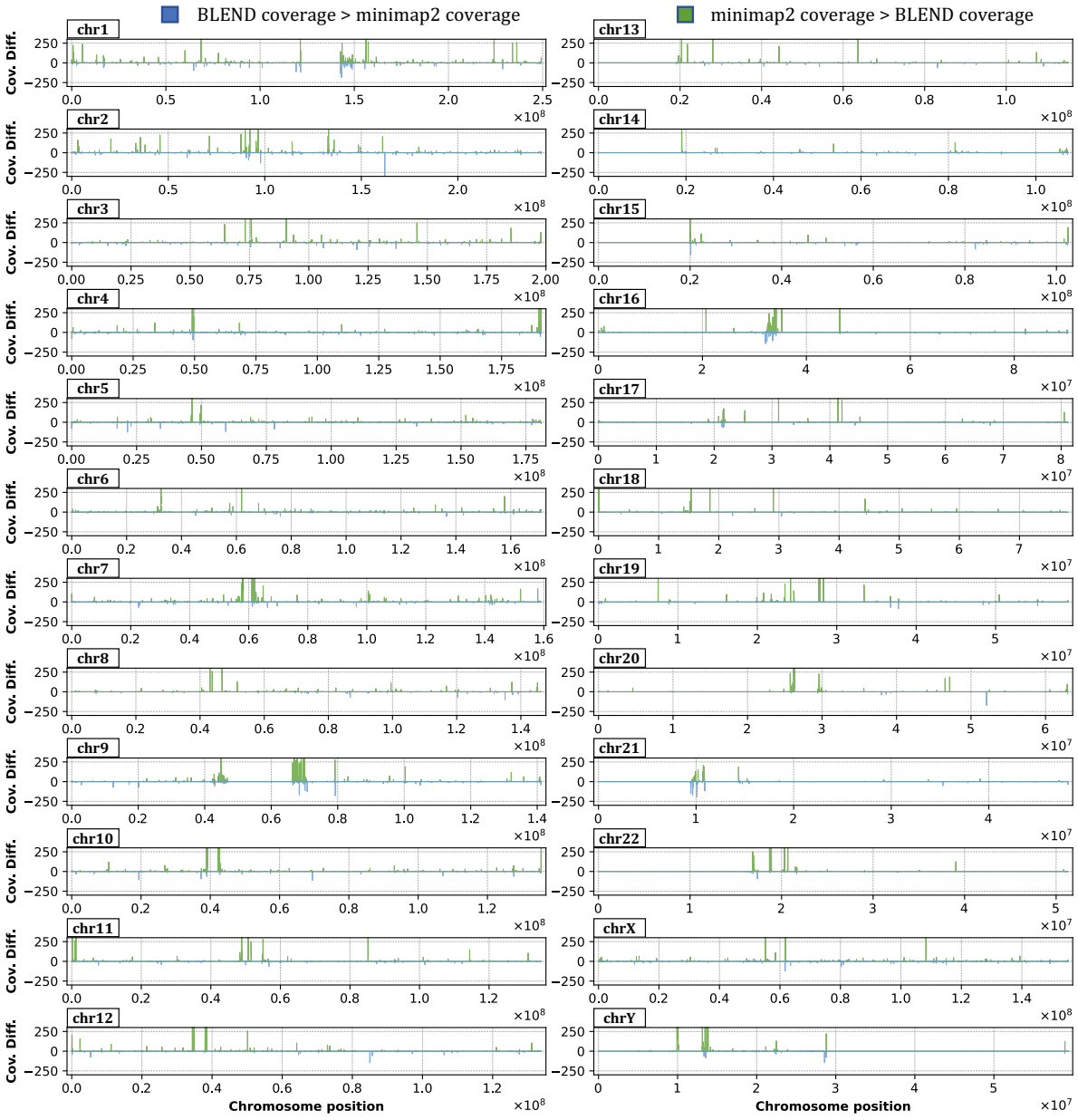

Figure S4.6: Difference between the depth of coverage of minimap2 and BLEND. Positive values show the positions where minimap2 has higher coverage and negative values show the positions where BLEND has higher coverage. We label the chromosomes on the top left corner of each plot.

S4.4 Parameters and Tool Versions

In Supplementary Table S4.14, we show the version numbers of each tool. When calculating the performance and peak memory usage, we use the time command from Linux and append the following command to the beginning of each of our runs: `/usr/bin/time -vp`.

Supplementary Table S4.15 shows the parameters we use in BLEND and their definition. Since BLEND uses the minimap2 implementation as a baseline, the rest of the parameters we do not show in Supplementary Table S4.15 can be found on the manual page of minimap2[6]. In Supplementary Table S4.16, we show the parameters we use with BLEND, minimap2, and MHAP [416] for read overlapping. Since there are no default parameters for minimap2 and MHAP when using the HiFi reads, we used the parameters as suggested by the HiCanu tool [507]. We found these parameters in the source code of Canu. For minimap2 and MHAP, the HiFi parameters are found in the GitHub pages[7,8], respectively. In Supplementary Table S4.16, *minimap-Eq* shows the parameters that are equivalent to the parameters we use with BLEND without the fuzzy seed matching capability.

In Supplementary Table S4.17, we show the parameters we use with BLEND, minimap2 [306], LRA [311], Winnowmap2 [413, 1005], S-conLSH [304, 307], and Strobealign [856] for read mapping.

Table S4.14: Versions of each tool.

Tool	Version	GitHub or Conda Link to the Version
BLEND	1.0	https://github.com/CMU-SAFARI/BLEND
minimap2	2.24	https://github.com/lh3/minimap2/releases/tag/v2.24
MHAP	2.1.3	https://anaconda.org/bioconda/mhap/2.1.3/download/noarch/mhap-2.1.3-hdfd78af_1.tar.bz2
LRA	1.3.2	https://anaconda.org/bioconda/lra/1.3.2/download/linux-64/lra-1.3.2-ha140323_0.tar.bz2
Winnowmap2	2.03	https://anaconda.org/bioconda/Winnowmap/2.03/download/linux-64/Winnowmap2-2.03-h2e03b76_0.tar.bz2
S-conLSH	2.0	https://github.com/anganachakraborty/S-conLSH-2.0/tree/292fbe0405f10b3ab63fc3a86cba2807597b582e
Strobealign	0.7.1	https://anaconda.org/bioconda/strobealign/0.7.1/download/linux-64/strobealign-0.7.1-hd03093a_1.tar.bz2

[6] https://lh3.github.io/minimap2/minimap2.html

[7] https://github.com/marbl/canu/blob/404540a944664cfab00617f4f4fa37be451b34e0/src/pipelines/canu/OverlapMMap.pm#L63-L65

[8] https://github.com/marbl/canu/blob/404540a944664cfab00617f4f4fa37be451b34e0/src/pipelines/canu/OverlapMhap.pm#L100-L131

Table S4.15: Definition of parameters in BLEND

Parameter	Definition
–strobemers	Use the `BLEND-S` mechanism when generating the list of k-mers of a seed
–immediate	Use the `BLEND-I` mechanism when generating the list of k-mers of a seed
-H	Use homopolymer-compressed k-mers
-w INT	Window size used when finding minimizers.
-k INT	k-mer size used when generating the list of k-mers of a seed
–neighbors INT	Number of k-mers included in the list of seeds. Combination of both -k (k) and –neighbors (n) determines the seed length. Seed length in `BLEND-S` is calculated as: $k \times n$ Seed length in `BLEND-I` is calculated as: $k + (n - 1)$
–fixed-bits INT	Bit length of hash values that BLEND generates for each seed. Setting it to $2 \times k$ is the default behavior.
-t INT	Number of CPU threads to use.
-x STR	Preset for setting the default parameters given the use case (STR)
-x map-ont	Preset for mapping ONT reads. It uses the following parameters: –immediate -w 10 -k 9 –neighbors 7 –fixed-bits 30
-x map-pb	Preset for mapping erroneous PacBio reads. It uses the following parameters: –immediate -H -w 10 -k 13 –neighbors 7 –fixed-bits 32
-x map-hifi	Preset for mapping accurate long (HiFi) reads. It uses the following parameters: –strobemers -w 50 -k 19 –neighbors 5 –fixed-bits 38
-x sr	Preset for mapping short reads. It uses the following parameters: –immediate -w 11 -k 21 –neighbors 5 –fixed-bits 32
-x ava-ont	Preset for overlapping ONT reads. It uses the following parameters: –immediate -w 10 -k 15 –neighbors 5 –fixed-bits 30
-x ava-pb	Preset for overlapping erroneous PacBio reads. It uses the following parameters: –immediate -H -w 10 -k 19 –neighbors 5 –fixed-bits 38
-x ava-hifi	Preset for overlapping accurate long (HiFi) reads. It uses the following parameters: –strobemers -w 200 -k 25 –neighbors 7 –fixed-bits 50

Table S4.16: Parameters* we use in our evaluation for each tool and dataset in read overlapping.

Tool	Dataset	Parameters
BLEND	*CHM13 (HiFi)*	-x ava-hifi -t 32
BLEND	*D. ananassae (HiFi)*	-x ava-hifi -t 32
BLEND	*E. coli (HiFi)*	-x ava-hifi -t 32
BLEND	*CHM13 (ONT)*	-x ava-ont -t 32
BLEND	*Yeast (PacBio)*	-x ava-pb -t 32
BLEND	*Yeast (ONT)*	-x ava-ont -t 32
BLEND	*E. coli (PacBio)*	-x ava-pb -t 32
minimap2	*CHM13 (HiFi)*	-x ava-pb -Hk21 -w14 -t 32
minimap2	*D. ananassae (HiFi)*	-x ava-pb -Hk21 -w14 -t 32
minimap2	*E. coli (HiFi)*	-x ava-pb -Hk21 -w14 -t 32
minimap2	*CHM13 (ONT)*	-x ava-ont -t 32
minimap2	*Yeast (PacBio)*	-x ava-pb -t 32
minimap2	*Yeast (ONT)*	-x ava-ont -t 32
minimap2	*E. coli (PacBio)*	-x ava-pb -t 32
minimap2-Eq	*CHM13 (ONT)*	-x ava-ont -k19 -w10 -t 32
minimap2-Eq	*Yeast (PacBio)*	-x ava-pb -k23 -w10 -t 32
minimap2-Eq	*Yeast (ONT)*	-x ava-ont -k19 -w10 -t 32
minimap2-Eq	*E. coli (PacBio)*	-x ava-pb -k23 -w10 -t 32
MHAP	*CHM13 (HiFi)*	–store-full-id –ordered-kmer-size 18 –num-hashes 128 –num-min-matches 5 –ordered-sketch-size 1000 –threshold 0.95 –num-threads 32
MHAP	*D. ananassae (HiFi)*	–store-full-id –ordered-kmer-size 18 –num-hashes 128 –num-min-matches 5 –ordered-sketch-size 1000 –threshold 0.95 –num-threads 32
MHAP	*E. coli (HiFi)*	–store-full-id –ordered-kmer-size 18 –num-hashes 128 –num-min-matches 5 –ordered-sketch-size 1000 –threshold 0.95 –num-threads 32
MHAP	*Yeast (PacBio)*	–store-full-id –num-threads 32
MHAP	*Yeast (ONT)*	–store-full-id –num-threads 32
MHAP	*E. coli (PacBio)*	–store-full-id –num-threads 32

* For the definitions of the parameters we use in BLEND, please see Supplementary Table S4.15

Table S4.17: Parameters we use in our evaluation for each tool and dataset in read mapping.

Tool	Dataset	Parameters
BLEND	*CHM13 (HiFi)*	-ax map-hifi -t 32 –secondary=no
BLEND	*HG002 (HiFi)*	-ax map-hifi -t 32 –secondary=no
BLEND	*D. ananassae (HiFi)*	-ax map-hifi -t 32 –secondary=no
BLEND	*E. coli (HiFi)*	-ax map-hifi -t 32 –secondary=no
BLEND	*CHM13 (ONT)*	-ax map-ont -t 32 –secondary=no
BLEND	*Yeast (PacBio)*	-ax map-pb -t 32 –secondary=no
BLEND	*Yeast (ONT)*	-ax map-ont -t 32 –secondary=no
BLEND	*Yeast (Illumina)*	-ax sr -t 32
BLEND	*E. coli (PacBio)*	-ax map-pb -t 32 –secondary=no
minimap2	*CHM13 (HiFi)*	-ax map-hifi -t 32 –secondary=no
minimap2	*HG002 (HiFi)*	-ax map-hifi -t 32 –secondary=no
minimap2	*D. ananassae (HiFi)*	-ax map-hifi -t 32 –secondary=no
minimap2	*E. coli (HiFi)*	-ax map-hifi -t 32 –secondary=no
minimap2	*CHM13 (ONT)*	-ax map-ont -t 32 –secondary=no
minimap2	*Yeast (PacBio)*	-ax map-pb -t 32 –secondary=no
minimap2	*Yeast (ONT)*	-ax map-ont -t 32 –secondary=no
minimap2	*Yeast (Illumina)*	-ax sr -t 32
minimap2	*E. coli (PacBio)*	-ax map-pb -t 32 –secondary=no
Winnowmap2	*CHM13 (HiFi)*	meryl count k=15 meryl print greater-than distinct=0.9998 -ax map-pb -t 32
Winnowmap2	*HG002 (HiFi)*	meryl count k=15 meryl print greater-than distinct=0.9998 -ax map-pb -t 32
Winnowmap2	*D. ananassae (HiFi)*	meryl count k=15 meryl print greater-than distinct=0.9998 -ax map-pb -t 32
Winnowmap2	*E. coli (HiFi)*	meryl count k=15 meryl print greater-than distinct=0.9998 -ax map-pb -t 32
Winnowmap2	*CHM13 (ONT)*	meryl count k=15 meryl print greater-than distinct=0.9998 -ax map-ont -t 32
Winnowmap2	*Yeast (PacBio)*	meryl count k=15 meryl print greater-than distinct=0.9998 -ax map-pb-clr -t 32
Winnowmap2	*Yeast (ONT)*	meryl count k=15 meryl print greater-than distinct=0.9998 -ax map-ont -t 32
Winnowmap2	*E. coli (PacBio)*	meryl count k=15 meryl print greater-than distinct=0.9998 -ax map-pb-clr -t 32
LRA	*CHM13 (HiFi)*	align -CCS -t 32 -p s
LRA	*HG002 (HiFi)*	align -CCS -t 32 -p s
LRA	*D. ananassae (HiFi)*	align -CCS -t 32 -p s
LRA	*E. coli (HiFi)*	align -CCS -t 32 -p s
LRA	*CHM13 (ONT)*	align -ONT -t 32 -p s
LRA	*Yeast (PacBio)*	align -CLR -t 32 -p s
LRA	*Yeast (ONT)*	align -ONT -t 32 -p s
LRA	*E. coli (PacBio)*	align -CLR -t 32 -p s
S-conLSH	*CHM13 (HiFi)*	–threads 32 –align 1
S-conLSH	*E. coli (HiFi)*	–threads 32 –align 1
S-conLSH	*CHM13 (ONT)*	–threads 32 –align 1
S-conLSH	*Yeast (PacBio)*	–threads 32 –align 1
S-conLSH	*Yeast (ONT)*	–threads 32 –align 1
S-conLSH	*E. coli (PacBio)*	–threads 32 –align 1
Strobealign	*Yeast (Illumina)*	-t 32

* For the definitions of the parameters we use in BLEND, please see Supplementary Table S4.15

Chapter 5

Enabling Hash-based Search of Raw Nanopore Signals by Reducing Noise

In the previous chapter, we present a mechanism for generating the same hash value when the basecalled seeds are highly similar to each other to improve the sensitivity of sequence analysis by tolerating noise in seeds (i.e., differences between seeds). Our goal in this chapter is to provide a similar mechanism for raw nanopore signals such that **noise in these signals are effectively reduced** to enable generating the same hash values between similar raw signals.

5.1 Background and Motivation

High-throughput sequencing (HTS) devices can generate a large amount of genomic data at a relatively low cost. HTS can be used to analyze a wide range of samples, from small amounts of DNA or RNA to entire genomes. Oxford Nanopore Technologies (ONT) is one of the most widely-used HTS technologies that can sequence long genomic regions, called *reads*, with up to a few million bases. ONT devices use the nanopore sequencing technique, which involves passing a single DNA or RNA strand through a tiny pore, *nanopore* or channel, at an average speed of 450 bases per second [292] and measuring the electrical current as the strand passes through. Nanopore sequencing enables two key features. First, nanopores provide the electrical raw signals in *real-time* as the DNA strand passes through a nanopore. Second, nanopore sequencing provides a functionality, known as *Read Until* [264], that can partially sequence DNA strands without fully sequencing them. These two features of nanopores provide opportunities for 1) real-time genome analysis and 2) significantly reducing sequencing time and cost.

Real-time analysis of nanopore raw signals using Read Until can reduce the sequencing time and cost per read by terminating the sequencing whenever sequencing the full read is not necessary. The freed-up nanopore can then be used to sequence a different read. A purely computational mechanism can send a signal to *eject* a read from a nanopore by reversing the voltage if the partial sequencing of a read meets certain conditions for particular genome analysis, such as 1) reaching a desired coverage for a species in a sample [786] or 2) identifying that a read does not originate from a certain genome of interest (i.e., a target region) [291, 292] and hence, does not need to be fully sequenced. By terminating the sequencing of reads that do not correspond to the target region, the sequencer can spend time and resources on higher coverage sequencing of the reads that correspond to the target. This process is referred to as *nanopore adaptive sampling*. By providing high coverage at target regions and avoiding unessential sequencing of reads outside those regions, this approach can improve the quality of sequencing and the downstream analysis utilizing the obtained data.

5.1.1 Motivation: Need for Scalable Mechanisms Raw Nanopore Signal Analysis

To effectively utilize adaptive sampling in nanopore sequencing, it is crucial to have computational methods that can accurately analyze the raw output signals from nanopores in real-time. These methods must provide 1) low latency and 2) throughput matching or exceeding that of the sequencer [138, 291, 292]. Several works propose adaptive sampling methods for real-time analysis of raw nanopore signals [138, 290–293, 297, 298, 680, 786, 787]. However, these works have three key limitations. First, most techniques mainly use powerful computational resources, such as GPUs [290, 786], or specialized hardware [138, 297] due to the use of computationally-intensive algorithms such as basecalling as we explain in detail in Chapter 3. This can make real-time genome analysis challenging for portable and low-cost nanopore-based sequencers, such as the ONT Flongle or MinION, which are not typically equipped with such resources. Therefore these techniques introduce challenges for using them in resource-constrained environments. Second, the sheer size of genomic data at the scale of large genomes (e.g., human genome) makes it challenging to process the data in real-time. This is because such large genomes require efficient and accurate similarity identification across a large number of regions. This renders many current methods [291, 292] inaccurate or useless for large genomes as they cannot either provide accurate results or match the throughput of nanopores for these genomes. Third, machine learning models used in past works [290, 293, 680, 786, 787] to analyze raw nanopore signals often require retraining or reconfiguring the model to improve accuracy for a certain experiment, which can be a barrier to flexibly and easily performing real-time analysis without retraining or reconfiguring these models. To our knowledge, there is no work

that can efficiently and accurately perform real-time analysis of raw nanopore signals on a large scale (e.g., whole-genome analysis for human) without requiring powerful computational resources, which can easily and flexibly be applied to a wide range of applications that could benefit from real-time nanopore raw signal analysis.

5.1.2 Challenge: Efficiently Handling Noise in Raw Nanopore Signal Analysis

Our **goal** is to enable efficient and accurate real-time genome analysis for large genomes. To this end, we propose ***RawHash***, the *first* mechanism that can efficiently and accurately perform real-time analysis of raw nanopore signals for large genomes in resource-contained environments without requiring computationally-intensive algorithms such as basecalling. Our **key idea** is to encode regions of the raw nanopore signal into hash values such that similar signal regions can efficiently be identified by matching their hash values, facilitating efficient similarity identification between signals. However, enabling accurate hashing-based similarity identification in the raw signal domain is challenging because raw signals corresponding to the same DNA content are unlikely to have exactly the same signal amplitudes. This is because the raw signals generated by nanopores can vary each time the same DNA fragment is sequenced due to several factors impacting nanopores during sequencing, such as variations in the properties of the nanopores or the conditions in which the sequencing is performed [286]. Although the similarity identification of raw signals is possible via calculating the Euclidean distance between a sequence of signals in a multi-dimensional space [291], such an approach can become impractical when dealing with larger sequences as the number of dimensions increases with the length of the sequences. This increase in dimensionality can lead to computational complexity and the curse of dimensionality, making it expensive and impractical.

5.1.3 Overview of Noise Reduction To Enable Hash-based Search in RawHash

To address these challenges, RawHash provides three **key mechanisms** for efficient signal encoding and similarity identification. First, RawHash encodes signal values that have a wider range of values into a smaller set of values using a quantization technique, such that signal values within a certain range are assigned to the same encoded value. This helps to alleviate the probability of having varying signal values for the same DNA content and enables RawHash to directly match these values using a hashing technique. Second, RawHash concatenates the quantized values of *multiple* consecutive signals and generates a single hash value for them. The hashing mechanism enables RawHash to efficiently identify similar signal regions of these consecutive signal values by directly matching their corresponding hash values. Representing

many consecutive signals with a single hash value increases the size of the regions examined during similarity identification without suffering from the curse of dimensionality. Using larger regions can substantially reduce the number of possible matching regions that need to be examined. RawHash is the *first* work that can accurately use hash values in the raw signal domain, which enables using efficient data structures commonly-used used in the sequence domain (e.g., hash tables in minimap2 [306]). Third, RawHash uses an existing algorithm, known as chaining [306], to find the colinear matches of hash values between signals to identify similar signal regions. These efficient and accurate mechanisms enable RawHash to perform real-time genome analysis for large genomes.

While our proposed three key mechanisms have the potential to be used for various purposes in raw signal similarity identification, we design RawHash as a tool for mapping nanopore raw signals to their corresponding reference genomes in real-time. RawHash operates the mapping in two steps 1) *indexing* and 2) *mapping*. First, in the indexing step, RawHash 1) converts the reference genome sequence into *expected* signal values by simulating the expected behavior of nanopores based on a previously-known model, 2) generates the hash values from these signals, and 3) stores the hash values in a hash table for efficient matching. Second, in the mapping step, RawHash 1) generates the hash values from the raw signals in a streaming fashion, 2) queries the hash table from the indexing step with these hash values to find the matching regions in the reference genome with the same hash value, and 3) performs chaining to find the similar region between the reference genome and the raw signal of a read.

5.1.4 The *Sequence Until* Mechanism: Dynamically Stopping the Entire Sequencing Run

We propose a novel mechanism that can stop the entire sequencing run for certain applications (e.g., relative abundance estimation) when an accurate decision can be made without sequencing the entire sample, which we call *Sequence Until*. *Sequence Until* utilizes *Run Until* to *fully* stop the *entire* sequencing of all subsequent reads after sequencing a certain amount of reads that is sufficient to make an accurate relative abundance estimation. Avoiding the redundant sequencing of further reads that are unlikely to substantially change the relative abundance estimation has the potential to significantly reduce the sequencing time and cost. To utilize *Sequence Until*, RawHash integrates a confidence calculation mechanism that evaluates the relative abundance estimations in real-time and fully stops the entire sequencing run if using more reads does not change its estimation. To stop the entire sequencing run for further reads, Run Until can be used to stop the entire sequencing run, which can enable the better utilization of nanopores. We find that *Sequence Until* can be applied to other mechanisms (e.g., UNCALLED) that can perform real-time relative abundance estimations. Prior work [790]

proposes a technique to terminate the sequencing process when species in the sample reach a certain coverage depth. The key difference of *Sequence Until* is that it reduces the cost of sequencing for relative abundance estimation and is based on our adaptive, accurate, and low-cost confidence calculation during real-time abundance estimation.

5.2 RawHash Mechanism

We propose **RawHash**, a mechanism that can efficiently and accurately identify similarities between raw nanopore signals of a read and a large reference genome in real-time (i.e., while the read is sequenced). The raw nanopore signal of each read is a series of electrical current measurements as a strand of DNA passes through a nanopore. The reference genome is a set of strings over the alphabet `A,C,G,T`. RawHash provides the mechanisms for generating hash values from both a raw nanopore signal and a reference genome such that similar regions between the two can be efficiently and accurately found by matching their hash values.

5.2.1 Overview

Figure 5.1 shows the overview of how RawHash identifies similarities between raw nanopore signals of a read and a reference genome in four steps. First, RawHash pre-processes both 1) the raw nanopore signal and 2) the reference genome into values that are comparable to each other. For raw signals, RawHash segments the raw signal into non-overlapping regions such that each region is expected to contain a certain amount of signal values that are generated from reading a fixed number k of DNA bases. Each such region is called an *event* [286]. Each event is usually represented with a value derived from the signal values in the segment. For the reference genome, RawHash translates each substring of length k (called a *k-mer*) into their *expected* event values based on the nanopore model.

The event values from the reference genome are not directly comparable to the event values from raw nanopore signals due to variability in the current measurements in nanopores generating slightly different event values for the same k-mer [286]. To generate the same values from slightly different events that may contain the same k-mer information, the second step of RawHash *quantizes* the event values from a larger set of values into a smaller set. The quantization technique ensures that the event values within a certain range are likely to be assigned to the same quantized value such that the effect of signal variation is alleviated, i.e., the same k-mer is likely assigned the same quantized value.

Due to the nature of nanopores, each event usually represents a very small k-mer of length around k=6 bases, depending on the nanopore model [291]. Such a short k-mer is likely to exist in a large number of locations in the reference genome, making it challenging to efficiently identify the correct one. To make the events more unique (i.e., such that they exist only in

a small number of locations in the reference genome), the third step of RawHash combines multiple consecutive quantized events into a single hash value. These hash values can then be used to efficiently identify similar regions between raw signals and the reference genome by matching the hash values generated from their events using efficient data structures such as hash tables.

Fourth, to map a raw nanopore signal of a read to a reference genome, RawHash uses a chaining algorithm [291, 306] that find colinear matching hash values generated from regions that are close to each other both in the reference genome and the raw nanopore signal.

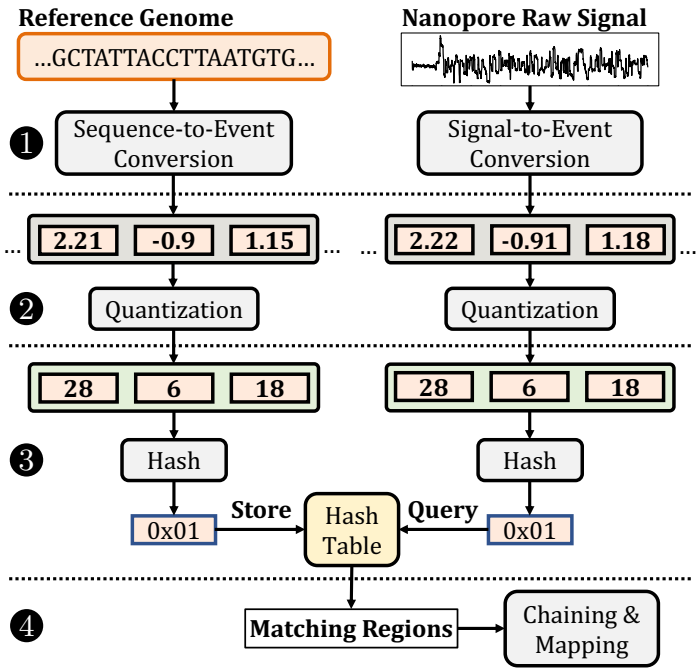

Figure 5.1: Overview of RawHash.

5.2.2 Event Generation

Our goal is to translate a reference genome sequence and a raw nanopore signal into comparable values. To this end, RawHash converts 1) each k-mer of the reference genome and 2) each segmented region of the raw signal into its corresponding event.

Sequence-to-Event Conversion. To convert a reference genome sequence into a form that can be compared with raw nanopore signals, RawHash converts the reference genome sequence into event values in three steps, as shown in Figure 5.2.

First, RawHash extracts all k-mers from the reference genome sequence, where k depends on the nanopore. The *k-mer model* of a nanopore[1] includes the information about the *expected*

[1]For many nanopore models, ONT provides the k-mer model including recent R10 and R10.4. These models can also be generated by users [835].

k-mer length of an event and the *expected* average event value for each k-mer based on certain variables affecting the signal outcome of the nanopore's current measurements.

Second, RawHash queries the k-mer model for each k-mer of the reference genome to convert k-mers into their expected event values. Although the k-mer model of a nanopore provides an extensive set of information for each possible k-mer, RawHash uses only the mean values of events that provide an average value for the signals in the same event since these mean values provide a sufficient level of meaningful information for comparison with the raw nanopore signals.

Third, RawHash normalizes the event values from the same reference genome sequence (e.g., entire chromosome sequence or a contig) by calculating the standard scores (i.e., z-scores) of these events. RawHash uses these normalized values as event values since the same normalization step is taken for raw signals to avoid certain variables that may affect the range of raw signal amplitudes during sequencing [291, 292].

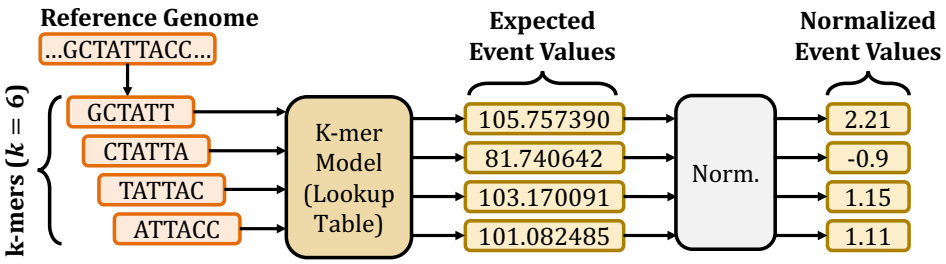

Figure 5.2: Converting sequences to event values based on the k-mer model of a nanopore.

Signal-To-Event Conversion. Our goal is to accurately convert the series of raw nanopore signals into a set of values where each value corresponds to certain DNA sequences of fixed length k, k-mers, and consecutive values differ by one base. To achieve this, RawHash converts the raw signals into their corresponding values in three steps, as shown in Figure 5.3. First, to accurately identify the distinct regions in the raw signal that correspond to a certain k-mer from DNA, RawHash performs a segmentation step as described in a basecalling tool, Scrappie, and used by earlier works UNCALLED and Sigmap. The segmentation step aims to eliminate the factors that affect the speed of the DNA molecules passing through a nanopore, as the speed affects the number of signal measurements taken for a certain amount of bases in DNA. To perform the segmentation step, RawHash identifies the boundaries in the signal where the signal value changes significantly compared to the certain amount of previously measured signal values, which indicates a base change in the nanopore. Such boundaries are computed using a statistical test, known as *Welch's t-test* [303], over a rolling window of consecutive signals. RawHash performs this t-test for multiple windows of different lengths to avoid the variables that cause a change in the number of current measurements due to the varying speed

of DNA through a nanopore, known as *skip* and *stay* errors [286]. Signals that fall within the same segment (i.e., between the same measured boundaries) are usually called *events* since each event contains the signals from a reading of a fixed amount of DNA bases, k-mers.

Second, since the number of signals that each event includes is not constant across different events due to the stay and skip errors, RawHash generates a single value for each event to quickly avoid these potential errors and other factors that cause variations from reading the same amount of DNA bases. To this end, RawHash measures the mean value of the signals that fall within the same segment and uses this mean value for an event.

Third, since the amplitudes of the signal measurements may significantly vary when reading k-mers at different times, RawHash normalizes the mean event values using the event values generated from the nanopore within the same certain time interval in a streaming fashion. Although this time interval parameter can be modified in our tool, the default configuration of RawHash processes the events of signals generated by the nanopore within one second. For normalization, RawHash uses the same z-score calculation that it uses for normalizing the event values generated from reference sequences as described earlier. RawHash uses these normalized values as event values when comparing with the event values from reference sequences.

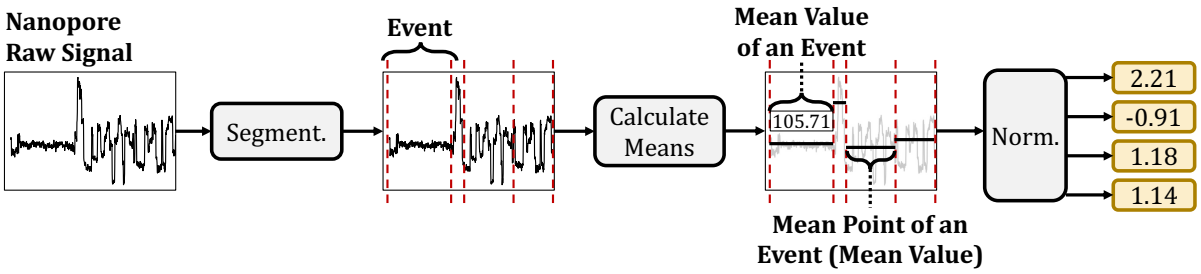

Figure 5.3: Detecting events from raw signals.

5.2.3 Quantization of Events

Our goal is to avoid the effects of generating different event values when reading the same k-mer content from nanopores so that we can identify k-mer matches by directly matching events. Although the segmentation and normalization steps explained in Section 5.2.2 can avoid the potential sequencing errors, such as stay and skip errors and significant changes in the current readings at different times, these approaches still do not guarantee to generate *exactly* the same event values when reading the *same* k-mer content. This is because *slight* changes in the normalized event values may occur when reading the same DNA content due to the high sensitivity and stochasticity of nanopores [286]. Thus, it is challenging to generate the same event value for the same k-mer content after the segmentation and normalization steps. Since these event values generated from reading the same k-mer content are expected to be close

to each other [291], we propose a quantization mechanism that encodes event values so that events with close mean values can have the same quantized value in two steps as shown in Figure 5.4.

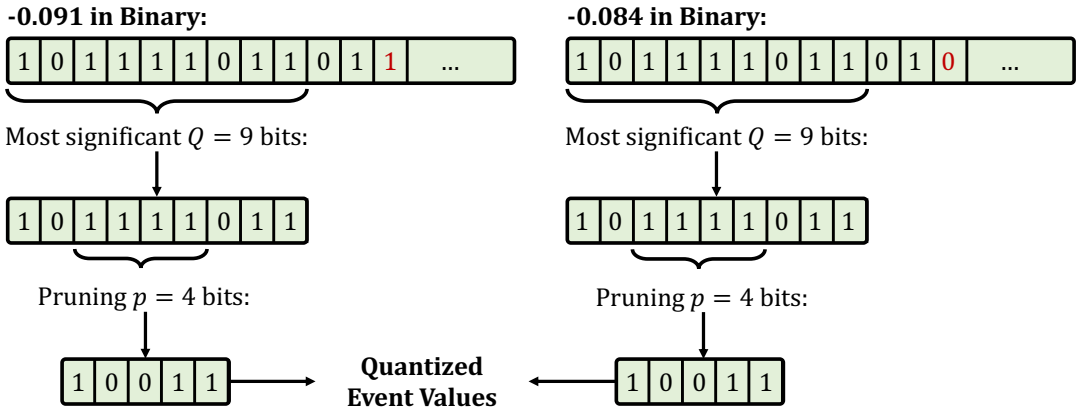

Figure 5.4: Quantization of two event values.

First, to increase the probability of assigning the same value for similar event values, RawHash trims the least significant fractional part of mean values by using only the most significant Q bits of these mean event values from their binary format, which we represent as $E[1, Q]$ for simplicity where E is the event value and $E[1, Q]$ gives the most significant Q bits of E. We assume that the mean event values are represented by the standard single-precision floating-point format with the sign, exponent, and fraction bits. This enables RawHash to reduce the wide range of floating-point numbers into a smaller range *without* significantly losing from the accuracy such that event values closer to each other can be represented by the same value in the smaller range of values. We can perform this trimming technique without significant sensitivity loss because we observe that these normalized event values mostly use at most six digits from the fractional part of their values, leaving a large number of fractional bits useless.

Second, to avoid using redundant bits that may carry little or no information in the most significant Q bits of an event value, RawHash prunes p bits after the most significant two bits of $E[1, Q]$ such that $2 + p < Q$ and the resulting *quantized value* is $E[1, 2]E[3 + p, Q]$. For simplicity, we show the quantized value of E as $E_{Q,p}$. By ignoring these p bits, we effectively pack Q bits into $Q - p$ bits without losing significant information from event values. We can perform such a pruning operation because we observe that the normalized event values are usually in the range $[-3, 3]$ such that these p bits provide little information in distinguishing different event values due to the small range of values. We note that these Q and p values are parameters to RawHash and can empirically be adjusted based on the required sensitivity and quantization efficiency. This quantization technique enables RawHash to assign the same quantized values

for a pair of *close* event values, E and F, that may be generated from reading the same k-mer such that $E_{Q,p} = F_{Q,p}$ where $|E - F| < \epsilon$ and ϵ is small enough for two events to represent the same k-mer content. RawHash always uses the most significant two bits as these two bits consistently carry the most significant information of the normalized event values, including the sign bit.

5.2.4 Generating the Hash Values

Our goal is to generate values for *large* regions of raw nanopore signals and reference sequences such that these values can be used to efficiently and accurately identify similarities between raw signals and a reference genome. To this end, RawHash generates hash values using quantized values of events in two steps, as shown in Figure 5.5. First, to avoid finding a large number of matches, RawHash uses the quantized values of n consecutive events to pack them in $n \times (Q - p)$ bits while preserving the order information of these consecutive events. RawHash uses several consecutive events in a single hash value because matching a single event is likely to generate a larger number of matches for larger genomes as a single event usually corresponds to a k-mer of 6 to 9 bases depending on the nanopore model [286]. It is essential to use several consecutive events to reduce the number of matching regions between raw signals and the reference genome by increasing the region that these consecutive events span.

Second, to efficiently and accurately find matches between large regions of raw signals and a reference genome using a constrained space, RawHash uses a low collision hash function to generate a 32-bit hash value from $n \times (Q - p)$ bits of n consecutive quantized event values. Since $n \times (Q - p)$ can be larger than 32, using such a hash function is likely to increase the collision rate for dissimilar regions. To avoid inaccurate similarity identifications due to these incorrect collisions, RawHash requires several matches of hash values within close proximity for similarity identification, which we explain next.

5.2.5 Seeding and Mapping

To efficiently identify similarities, RawHash uses hash values generated from raw nanopore signals and the reference genome in two steps. First, RawHash efficiently identifies matching regions between raw nanopore signals and a reference genome by matching their hash values. These hash values used for matching are usually known as *seeds*. Matching seeds enable efficiently finding similar regions between raw nanopore signals and a reference genome. Second, RawHash uses the chaining algorithm proposed in Sigmap [291] to identify the *best* colinear matching seeds that are close to each other in both raw nanopore signal and a reference genome. The region that the *best* chain of seed matches cover is the *mapping position* that RawHash identifies as a similar region.

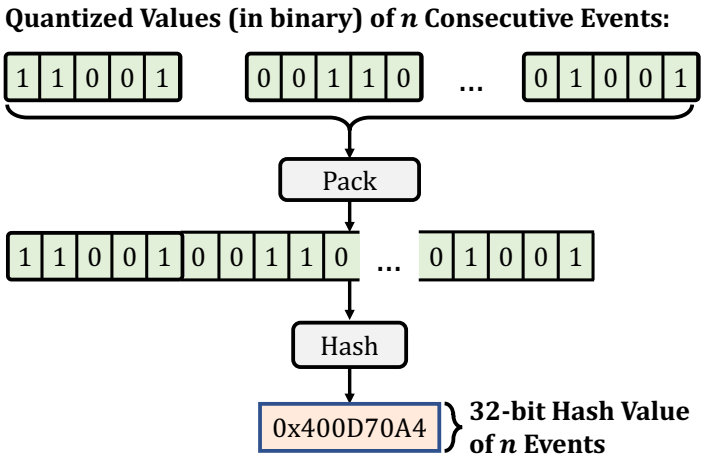

Figure 5.5: Generating a hash value from n consecutive quantized event values.

The chaining algorithm is useful for two reasons. First, the chaining algorithm can tolerate mismatches and indels as it allows gaps between seed matches, which enables finding similar regions with many seed matches without requiring the entire region to match exactly. To investigate this, we show the average gap length between a pair of anchors that RawHash finds when mapping raw nanopore signals in various datasets in Table 5.1. Read and reference anchors show the average gap length between a pair of anchors found in reads and the reference genome, respectively. We find that the chaining algorithm can tolerate a large number of mismatches and indels. Second, incorrect seed matches due to collisions or our quantization mechanism that may generate the same quantized value for distinctly dissimilar events are likely to be filtered in the chaining step due to the difficulty of finding colinear seed matches in highly dissimilar regions. We note that we modify the original chaining algorithm in Sigmap by disabling the distance coefficient as RawHash does not calculate the distance between seed matches.

Table 5.1: The average gap length between a pair of anchors in reads and a reference genome.

Tool	*SARS-CoV-2*	*E. coli*	*Yeast*	*Green Algae*	*Human*
Read Anchors	25.37	42.92	79.31	148.58	200.06
Reference Anchors	20.06	33.39	67.55	127.03	165.44

To efficiently map raw signals to a reference genome, RawHash provides efficient data structures. To this end, RawHash uses hash tables to store the hash values generated from reference genomes (i.e., the indexing step) and efficiently query the same hash table with the hash values generated from the raw signal as the read is sequenced from a nanopore to find positions in the reference genome with matching hash values. RawHash uses the events in chunks (i.e., collection of events generated within a certain time interval) to find seed matches

and perform chaining in a streaming fashion such that the chaining computation from previous chunks (i.e., seed matches) is transferred to the next chunk if the mapping is unsuccessful for the current chunk.

5.2.6 The *Sequence Until* Mechanism

Our goal with the *Sequence Until* mechanism is to reduce time and cost spent during sequencing by dynamically stopping the *entire* sequencing run if further sequencing is unnecessary for an accurate analysis. To do so, *Sequence Until* aims to continuously determine if further sequencing is unlikely to change the analysis outcome. Although *Sequence Until* is not limited to a particular analysis type, we exemplify the workflow of *Sequence Until* by focusing on the relative abundance estimation use case. Figure 5.6 shows the main steps in *Sequnce Until*.

First, *Sequence Until* continuously generates relative abundance estimations as new reads are sequenced. After every n reads ❶, *Sequence Until* performs an estimation of the relative abundance, producing a series of estimations as sequencing progresses.

Second, *Sequence Until* maintains the last t estimation results ❷ to assess the consistency of the abundance estimations. By focusing on a rolling window of the most recent estimations, *Sequence Until* can effectively monitor the ongoing trends in the relative abundance estimations.

Third, *Sequence Until* employs a cross-correlation technique to detect outliers among the last t relative abundance estimations ❸. Cross-correlation helps to identify whether recently sequenced reads significantly cause deviations from the general relative abundance estimation pattern, indicating that additional sequencing might still provide valuable information. The presence of outliers suggests that the sequencing process should continue, as further sequencing might still refine the abundance estimations.

Fourth, when no outliers are detected among the last t estimations, *Sequence Until* assumes that the abundance estimations have reached consistency. In such cases, the *Sequence Until* mechanism signals that the sequencing can be fully stopped (by using the Run Until functionality in nanopore sequencers) ❹, as further reads are unlikely to alter the relative abundance results substantially. This complete halt of sequencing helps to avoid redundant sequencing, thereby reducing the overall sequencing time and cost.

The *Sequence Until* mechanism is particularly beneficial for applications where the goal is to estimate the relative abundance of species accurately and efficiently. By dynamically adjusting the sequencing run based on real-time analysis, *Sequence Until* enables reducing unnecessary sequencing. This mechanism can be integrated into various sequencing workflows and analysis tools other than RawHash, including those utilizing other real-time relative abundance estimation tools [299].

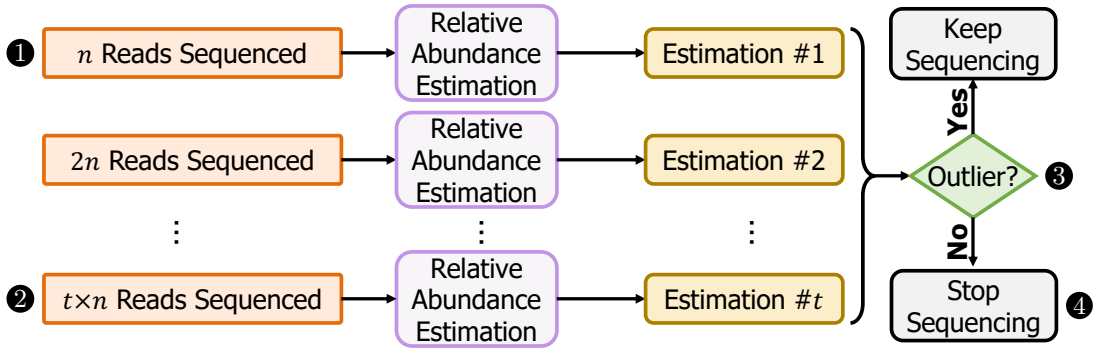

Figure 5.6: Overview of the Sequence Until mechanism.

5.3 Results

5.3.1 Evaluation Methodology

We implement RawHash as a tool for mapping raw nanopore signals to a reference genome. Similar to regular read mapping tools, RawHash has two steps to complete the mapping process: 1) indexing the reference genome and 2) mapping raw signals. Although indexing is usually a one-time task that can be performed prior to the mapping step, the indexing of RawHash can be performed relatively quickly within a few minutes for large genomes (Table 5.3). RawHash provides the mapping information using a standard pairwise mapping format (PAF). In our implementation, we provide an extensive set of parameters that allow configuring several options to fit RawHash for many other applications and nanopore models that we do not evaluate, such as configuring details about the nanopore model (e.g., number of bases per second), number of events that can be included in a single hash value, range of bits to quantize, enabling seeding techniques such as minimizers and fuzzy seed matching. We also provide a default set of parameters that we empirically choose for each common application of real-time genome analysis. These default parameters are set to accurately and efficiently analyze 1) very small (e.g., viral) genomes, 2) small and mid-sized genomes (i.e., genomes with less than a few hundred million bases), 3) large genomes (e.g., genomes with a few billion bases such as a human genome). We show the details regarding these parameter selections and the versions of tools in Supplementary Tables S5.1, S5.2, and S5.3.

We evaluate RawHash in terms of its performance, peak memory usage, accuracy, and estimated benefits in sequencing time and cost compared to two state-of-the-art tools UNCALLED and Sigmap. For performance, we evaluate the throughput and overall runtime of each tool in terms of the number of bases they can process per second. Throughput determines if the tool is at least as fast as the speed of DNA passing through a nanopore. For many nanopore models (e.g., R9.4), a DNA strand passes through a pore at around 450 bases per second [291, 292]. It

is essential to provide a throughput higher than the throughput of the nanopore to enable real-time genome analysis. To calculate the throughput, we use the tool that UNCALLED provides, `UNCALLED pafstats`, which measures the throughput of the tool from the number of bases that the tool processes and the time it takes to process those bases. Although theoretically, it is not possible to exceed the throughput of a nanopore due to the speed of raw signal generation, for comparison purposes, such a limitation is ignored by `UNCALLED pafstats`. For overall runtime, we calculate CPU time and real-time using 32 threads. CPU time shows the overall amount of CPU seconds spent running a tool, while real-time shows the overall elapsed (i.e., wall clock) time. All of these tools support multi-threading, where multiple reads can be mapped simultaneously using a single thread for each read. For all of these tools, assigning a larger number of threads enables processing a larger number of reads in parallel, similar to the behavior of nanopore sequencers with hundreds to thousands of pores (i.e., channels). We note that the throughput and mapping time per read values are not affected by the thread counts as 1) these are measured per read and 2) single thread performs the mapping of a single read.

For accuracy, we evaluate the correctness of the mapping positions that each tool provides when compared to the ground truth mapping positions. To generate the ground truth mapping, we use a read mapping tool, minimap2 [306], to map the basecalled sequences of raw nanopore signals to their corresponding *whole-genome* references. We use `UNCALLED pafstats` to compare the mapping output of a tool with the ground truth mapping to find the number of true positives or *TP* (i.e., correct mappings), false positives or *FP* (i.e., incorrect mappings), and false negatives or *FN* (i.e., unmapped reads that are mapped in ground truth). Correct and incorrect mappings are identified based on the distance of the mapping positions between ground truth and the tool. To evaluate the accuracy, we calculate the precision ($P = TP/(TP + FP)$), recall ($R = TP/(TP + FN)$) and the F_1 ($F_1 = 2 \times (P \times R)/(P + R)$) values.

For estimating the benefits in sequencing time and cost of each tool, we calculate the average length of sequenced bases per read when using UNCALLED and RawHash and the average number sequenced chunk of signals for Sigmap and RawHash. We compare RawHash with Sigmap in terms of the number of chunks because Sigmap does not provide the number of bases when a read is unmapped, while both tools provide the number of chunks used when a read is mapped or unmapped. These chunks include a portion of the signal produced by a nanopore within a certain time interval, which is by default set as one second of data for both RawHash and Sigmap. The average length of bases and the number of chunks determine the estimations of how quickly each tool can make a mapping decision to activate Read Until before sequencing the remaining portion of a read, which indicates the potential savings from overall sequencing time and cost.

We evaluate RawHash, UNCALLED, and Sigmap for three applications 1) read mapping,

2) relative abundance estimation, and 3) contamination analysis. Read mapping aims to map the raw signals to their corresponding reference genomes. Relative abundance estimation measures the abundance of each genome relative to other genomes in the same sample by mapping raw signals to a given set of reference genomes. Contamination analysis aims to identify if a sample is contaminated with a certain genome (e.g., a viral genome) by mapping raw signals to the reference genome that the sample may be contaminated with. For each tool, we use their default parameter settings in our evaluation.

To evaluate each of these applications, we use real datasets that we list in Table 5.2. These datasets include both raw nanopore signals in the FAST5 format and their corresponding basecalled sequences in the FASTA format. We note that RawHash can also use POD5 files. For relative abundance estimation, we create a mock community using all the read sets from datasets D1 to D5, and the reference genome is the combination of reference genomes used in these datasets. We slightly modify the reference genome we use in the relative abundance estimation such that the sequence IDs in the reference genome provide additional information about the species (e.g., taxonomy IDs) to enable calculating relative abundance in real-time. For contamination analysis, we combine the SARS-CoV-2 read sets (D1) with human read sets (D5) to identify if the combined sample is contaminated with the SARS-CoV-2 sample by mapping raw signals in the combined set to the SARS-CoV-2 reference genome. For all evaluations, we use the AMD EPYC 7742 processor at 2.26GHz to run the tools.

Table 5.2: Details of datasets used in our evaluation.

	Organism	Reads (#)	Bases (#)	SRA Accession	Reference Genome	Genome Size
				Read Mapping		
D1	*SARS-CoV-2*	1,382,016	594M	CADDE Centre	GCF_009858895.2	29,903
D2	*E. coli*	353,317	2,365M	ERR9127551	GCA_000007445.1	5M
D3	*Yeast*	49,989	380M	SRR8648503	GCA_000146045.2	12M
D4	*Green Algae*	29,933	609M	ERR3237140	GCF_000002595.2	111M
D5	*Human HG001*	269,507	1,584M	FAB42260 Nanopore WGS	T2T-CHM13 (v2)	3,117M
				Relative Abundance Estimation		
	D1-D5	2,084,762	5,531M	D1-D5	D1-D5	3,246M
				Contamination Analysis		
	D1 and D5	1,651,523	2,178M	D1 and D5	D1	29,903

Dataset numbers (e.g., D1-D5) show the combined datasets. Base counts in millions (M).

Evaluating *Sequence Until*. Our goal is to avoid redundant sequencing to reduce sequencing time and cost for relative abundance estimation. We find that the Run Until mechanism can

be utilized to *fully* stop the sequencing run when the real-time relative abundance estimation reaches a certain confidence level to achieve accurate estimations, which we call *Sequence Until*. While a similar mechanism is evaluated to enrich the coverage depth of low-abundance species [790] using Read Until, we evaluate the potential benefits of Run Until for low-cost relative abundance estimations. RawHash provides a set of parameters to adjust these parameters related to *Sequence Until*.

We evaluate the benefits of *Sequence Until* by comparing 1) RawHash without *Sequence Until* and 2) RawHash with *Sequence Until* in terms of 1) the difference in the relative abundance estimations and 2) the estimated benefits in sequencing time and cost. To evaluate *Sequence Until* in a realistic sequencing environment where reads from different species can be sequenced in a random order, we randomly shuffle the reads in the relative abundance dataset and generate a set of 50,000 reads with a random order of species so that we can simulate this random behavior. We also find that *Sequence Until* can be applied to other mechanisms. To evaluate the potential benefits of *Sequence Until*, we simulate the benefits when using UNCALLED with *Sequence Until* and compare it with RawHash.

5.3.2 Performance and Peak Memory

Figure 5.7 shows the throughput of regular nanopores that we use as a baseline and the throughput of the tools when mapping raw nanopore signals to each dataset for read mapping, contamination analysis, and relative abundance estimation. Figure 5.8 and Tables 5.3 and 5.4 show the mapping time per read, and the computational resources required for indexing and mapping, respectively. We make six key observations. First, RawHash and UNCALLED are the *only* tools that can perform real-time genome analysis for large genomes, as they can provide higher throughputs than nanopores for all datasets. Sigmap *cannot* perform real-time genome analysis for large genomes as it can provide 0.7× and 0.6× throughput of a nanopore for human genome mapping and relative abundance estimations, respectively. RawHash can achieve high throughput as its seeding mechanism is based on efficiently matching hash values compared to the costly distance calculations that Sigmap performs for matching seeds, which shows poor scalability for larger genomes. Second, the throughput of UNCALLED is not affected by the genome size as it provides a near-constant throughput of around 16× for all applications. This is because UNCALLED uses FM-index [366] and a branching algorithm that provides robust scaling with respect to the reference genome size [292]. Third, the throughput of RawHash decreases with larger genomes as the seeding and chaining steps start taking up a larger fraction of the entire runtime of RawHash. Fourth, RawHash provides an average throughput 25.8× and 3.4× better than UNCALLED and Sigmap, while providing an average mapping speedup of 32.1× and 2.1× per read, respectively. Higher throughput with faster

mapping times suggests that the mapping time improvements of RawHash are mainly due to its computational efficiency rather than the ability to sequence shorter prefixes of reads than UNCALLED and Sigmap. Fifth, for indexing, Sigmap usually requires a larger amount of computational resources in terms of both runtime and peak memory usage. Sixth, for mapping, UNCALLED is the most efficient tool in terms of the peak memory usage as it requires at most 10GB of peak memory while 1) RawHash requires less than 12GB of memory for almost all the datasets and 2) Sigmap requires significantly larger memory space than both tools. RawHash has a larger memory footprint, ~ 52GB, than UNCALLED for large genomes. Although such large memory requirements for larger genomes can lead to challenges in using RawHash for mobile devices with limited computational resources, such a requirement can be mitigated by using more efficient seeding techniques such as minimizers, which we leave as future work. We conclude that RawHash provides significant benefits in improving the throughput and performance for the real-time analysis of large genomes while matching the throughput of nanopores.

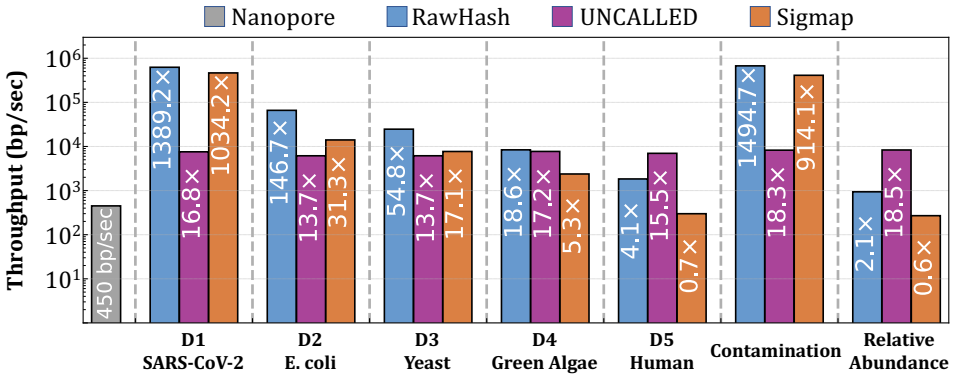

Figure 5.7: Throughput of each tool. Values inside the bars show the throughput ratio between each tool and a nanopore.

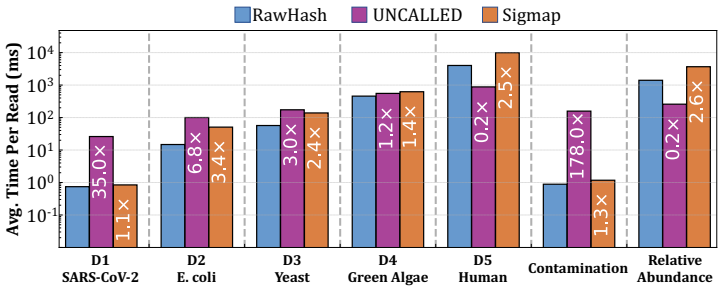

Figure 5.8: Average time spent per read by each tool in real-time. Values inside the bars show the speedups that RawHash provides over other tools in each dataset.

Table 5.3: Computational resources required in the indexing step of each tool.

Tool	*Contamination*	*SARS-CoV-2*	*E. coli*	*Yeast*	*Green Algae*	*Human*	*Relative Abundance*
				CPU Time (sec)			
UNCALLED	8.72	9.00	11.08	18.62	285.88	4,148.10	4,382.38
Sigmap	0.02	0.04	8.66	24.57	449.29	36,765.24	40,926.76
RawHash	0.18	0.13	2.62	4.48	34.18	1,184.42	788.88
				Real time (sec)			
UNCALLED	1.01	1.04	2.67	7.79	280.27	4,190.00	4,471.82
Sigmap	0.13	0.25	9.31	25.86	458.46	37,136.61	41,340.16
RawHash	0.14	0.10	1.70	2.06	15.82	278.69	154.68
				Peak memory (GB)			
UNCALLED	0.07	0.07	0.13	0.31	11.96	48.44	47.81
Sigmap	0.01	0.01	0.40	1.04	8.63	227.77	238.32
RawHash	0.01	0.01	0.35	0.76	5.33	83.09	152.80

Table 5.4: Computational resources required in the mapping step of each tool.

Tool	*Contamination*	*SARS-CoV-2*	*E. coli*	*Yeast*	*Green Algae*	*Human*	*Relative Abundance*
				CPU Time (sec)			
UNCALLED	265,902.26	36,667.26	35,821.14	8,933.52	16,769.09	262,597.83	586,561.54
Sigmap	4,573.18	1,997.84	23,894.70	11,168.96	31,544.55	4,837,058.90	11,027,652.91
RawHash	3,721.62	1,832.56	8,212.17	4,906.70	25,215.23	2,022,521.48	4,738,961.77
				Real time (sec)			
UNCALLED	20,628.57	2,794.76	1,544.68	285.42	2,138.91	8,794.30	19,409.71
Sigmap	6,725.26	3,222.32	2,067.02	1,167.08	2,398.83	158,904.69	361,443.88
RawHash	3,917.49	1,949.53	957.13	215.68	1,804.96	65,411.43	152,280.26
				Peak memory (GB)			
UNCALLED	0.65	0.19	0.52	0.37	0.81	9.46	9.10
Sigmap	111.69	28.26	111.11	14.65	29.18	311.89	489.89
RawHash	4.13	4.20	4.16	4.37	11.75	52.21	55.31

5.3.3 Accuracy

Table 5.5 shows the accuracy results of tools for each dataset and application. We make four key observations. First, RawHash provides the best accuracy in terms of precision, recall, and F_1 values compared to UNCALLED and Sigmap when mapping reads to large genomes (i.e., the human genome and the relative abundance estimation). RawHash can efficiently match several events using hash values, which is specifically beneficial in reducing the number of matching regions in large genomes and increasing the specificity due to finding longer matches compared to UNCALLED and Sigmap.

Second, RawHash and UNCALLED can accurately perform contamination analysis while Sigmap suffers from significantly lower precision and recall values. Due to the nature of a contamination analysis, it is essential to correctly eliminate the genomes other than the contaminating genome (precision) without missing the correct mappings of reads from the contaminating genome (recall). Unfortunately, Sigmap cannot provide high values in any of these categories, making it significantly unsafe for contamination detection.

Third, the precision of RawHash does not drop with the increased length in the reference

Table 5.5: Mapping accuracy.

Dataset		UNCALLED	Sigmap	RawHash
Read Mapping				
D1	Precision	0.9547	**0.9929**	0.9868
SARS-CoV-2	Recall	**0.9910**	0.5540	0.8735
	F_1	**0.9725**	0.7112	0.9267
D2	Precision	0.9816	**0.9842**	0.9573
E. coli	Recall	**0.9647**	0.9504	0.9009
	F_1	**0.9731**	0.9670	0.9282
D3	Precision	0.9459	0.9856	**0.9862**
Yeast	Recall	**0.9366**	0.9123	0.8412
	F_1	0.9412	**0.9475**	0.9079
D4	Precision	0.8836	**0.9741**	0.9691
Green Algae	Recall	0.7778	**0.8987**	0.7015
	F_1	0.8273	**0.9349**	0.8139
D5	Precision	0.4867	0.4287	**0.8959**
Human HG001	Recall	0.2379	0.2641	**0.4054**
	F_1	0.3196	0.3268	**0.5582**
Relative Abundance Estimation				
	Precision	0.7683	0.7928	**0.9484**
D1-D5	Recall	0.1273	0.2739	**0.3076**
	F_1	0.2184	0.4072	**0.4645**
Contamination Analysis				
	Precision	**0.9378**	0.7856	0.8733
D1, D5	Recall	**0.9910**	0.5540	0.8735
	F_1	**0.9637**	0.6498	0.8734

Best results are highlighted with **bold** text.

genome due to the benefits of finding long matches, which provides a higher confidence in read mapping.

Fourth, although RawHash does not provide the best accuracy when mapping reads to genomes smaller than the human genome, its accuracy is on par with UNCALLED and Sigmap for these genomes. UNCALLED and Sigmap can achieve high recall values as their mechanisms are best optimized for accurately handling matches in relatively smaller genomes with fewer repeats and ambiguous mappings [291, 292]. We conclude that RawHash is the *only* tool that can accurately scale to performing real-time genome analysis for large genomes, especially with significantly high precision rates.

Relative Abundance Estimations. Table 5.6 shows the relative abundance estimations that each tool makes and the Euclidean distance of their estimation to the ground truth estimation. We make two key observations. First, we find that RawHash provides the most accurate relative abundance estimations in terms of the estimation distance to the ground truth compared to

UNCALLED and Sigmap. This observation correlates with the accuracy results we show in Table 5.5 where RawHash provides the best overall accuracy for relative estimation, which results in generating the most accurate relative abundance estimations. Second, although Sigmap cannot perform real-time relative abundance estimation due to its throughput being lower than a nanopore (Figure 5.7), Sigmap provides accurate estimations that are on par with RawHash. This observation shows that while Sigmap provides mappings with more incorrect positions due to lower precision than RawHash (Table 5.5), these reads with incorrect mapping positions are mostly mapped to their correct species. We conclude that RawHash is the *only* tool that can *accurately* be applied to analyze relative abundance estimations while matching the throughput of nanopores at a large-scale based on the prior knowledge of the set of reference genomes to map the reads.

Table 5.6: Relative abundance estimations.

	Estimated Relative Abundance Ratios					
Tool	***SARS-CoV-2***	***E. coli***	***Yeast***	***Green Algae***	***Human***	**Distance**
Ground Truth	0.0929	0.4365	0.0698	0.1179	0.2828	N/A
UNCALLED	0.0026	0.5884	0.0615	0.1313	0.2161	0.1895
Sigmap	0.0419	0.4191	0.1038	0.0962	0.3390	0.0877
RawHash	0.1249	0.4701	0.0957	0.0629	0.2464	**0.0847**

Best results are highlighted with **bold** text.

5.3.4 Sequencing Time and Cost

Our goal is to estimate the benefits that each tool provides in reducing the sequencing time and cost. To this end, we measure the average length of sequenced bases and the average number of sequenced chunks per read. We make two key observations. First, RawHash provides significant benefits in reducing the sequencing time and cost for large genomes (e.g., Green Algae and Human) compared to UNCALLED, as RawHash can complete the mapping process per read by using smaller prefixes of reads. Second, RawHash uses on average 1.58× more chunks compared to Sigmap when mapping reads, which can proportionally lead to worse sequencing time and cost for RawHash compared to Sigmap. We conclude that although UNCALLED and Sigmap provide better advantages in reducing sequencing time and cost for smaller genomes, RawHash can provide significant reductions in sequencing time and cost for larger genomes compared to UNCALLED.

Table 5.7: The average sequenced length of bases and the number of chunks.

Tool	*SARS-CoV-2*	*E. coli*	*Yeast*	*Green Algae*	*Human*
Average sequenced base length per read					
UNCALLED	**184.51**	**580.52**	**1,233.20**	5,300.15	6,060.23
RawHash	513.95	1,376.14	2,565.09	**4,760.59**	**4,773.58**
Average sequenced number of chunks per read					
Sigmap	**1.01**	**2.11**	**4.14**	**5.76**	**10.40**
RawHash	1.24	3.20	5.83	10.72	10.70

Best results are highlighted with **bold** text.

5.3.5 Benefits of *Sequence Until*

Simulated *Sequence Until*. Our goal is to estimate the benefits of implementing the *Sequence Until* mechanism in UNCALLED and compare it with RawHash when they both use *Sequence Until* under the same conditions. To this end, we use `shuf` in Linux to randomly shuffle the mapping files that both RawHash and UNCALLED generate for relative abundance and extract a certain portion of the randomly shuffled file to identify their relative abundance estimations after 0.01%, 0.1%, 1%, 10%, and 25% of the overall reads in the sample are randomly sequenced from nanopores.

Table 5.8 shows the distance of relative abundance estimations after a certain portion of the read is randomly sequenced from nanopores. We make two key observations. First, *both* RawHash and UNCALLED can significantly benefit from *Sequence Until* by stopping sequencing after processing a smaller portion of the entire sample since their estimations using smaller portions are close to those using the entire set of reads (Table 5.6) in terms of their distance to the ground truth. This suggests that many other tools can benefit from *Sequence Until* as their sensitivity to relative abundance estimations may not significantly change while providing opportunities for reducing the sequencing time and cost up to a certain threshold based on the tool.

Second, RawHash can provide more accurate relative abundance estimations when using only 0.1% of the reads than the estimation that UNCALLED provides using the entire set of reads (Table 5.6). We conclude that *Sequence Until* provides significant opportunities in reducing sequencing time and cost while more accurate tools such as RawHash can benefit further from *Sequence Until* by using fewer portions of the entire read set than the portions that less accurate tools would need to achieve similar accuracy.

***Sequence Until* with RawHash.** Our goal is to evaluate *Sequence Until* when used in real-time with RawHash for relative abundance estimation. Table 5.9 shows the relative abundance

Table 5.8: Relative abundance with simulated *Sequence Until*.

	Estimated Relative Abundance Ratios					
Tool	***SARS-CoV-2***	***E. coli***	***Yeast***	***Green Algae***	***Human***	**Distance**
Ground Truth	0.0929	0.4365	0.0698	0.1179	0.2828	N/A
UNCALLED (25%)	0.0026	0.5890	0.0613	0.1332	0.2139	0.1910
RawHash (25%)	0.0271	0.4853	0.0920	0.0786	0.3170	**0.0995**
UNCALLED (10%)	0.0026	0.5906	0.0611	0.1316	0.2141	0.1920
RawHash (10%)	0.0273	0.4869	0.0963	0.0772	0.3124	**0.1004**
UNCALLED (1%)	0.0026	0.5750	0.0616	0.1506	0.2103	0.1836
RawHash (1%)	0.0259	0.4783	0.0987	0.0882	0.3088	**0.0928**
UNCALLED (0.1%)	0.0040	0.4565	0.0380	0.1910	0.3105	0.1242
RawHash (0.1%)	0.0212	0.5045	0.1120	0.0810	0.2814	**0.1136**
UNCALLED (0.01%)	0.0000	0.5551	0.0000	0.0000	0.4449	0.2602
RawHash (0.01%)	0.0906	0.6122	0.0000	0.0000	0.2972	**0.2232**

Percentages show the portion of the overall reads used. Best results are highlighted with **bold** text.

estimations that RawHash makes with and without *Sequence Until*. We note that the estimations we show for RawHash in Table 5.9 are different than the estimations in Table 5.6 since we randomly subsample the reads in the relative abundance estimation dataset, as explained in Section 5.3.1. We make two key observations. First, we observe that the distance between the relative abundance estimations between these two configurations of RawHash is substantially low. This indicates that our outlier detection mechanism can accurately detect the convergence to the relative abundance estimations without using a full set of reads. Second, *Sequence Until* enables accurately stopping the entire sequencing after processing 7% of the reads in the entire set without substantially sacrificing accuracy. We conclude that *Sequence Until* has the potential to significantly reduce the sequencing time and cost by using only fewer reads from a sample while producing accurate results.

Table 5.9: Relative abundance with *Sequence Until*.

	Estimated Relative Abundance Ratios in 50,000 Random Reads					
Tool	***SARS-CoV-2***	***E. coli***	***Yeast***	***Green Algae***	***Human***	**Distance**
RawHash (100%)	0.0270	0.3636	0.3062	0.1951	0.1081	N/A
RawHash + *Sequence Until* (7%)	0.0283	0.3539	0.3100	0.1946	0.1133	0.0118

Percentages show the portion of the overall reads used.

5.4 Discussion

We discuss the benefits we expect RawHash can immediately make, the limitations of RawHash, and future work. We envision that RawHash can be useful mainly for two directions. First, RawHash provides a low-cost solution for analyzing large genomes in real-time. Such an analysis can be significantly useful when using nanopore sequencers with limited computational resources to enable portable real-time genome analysis at a large scale.

Second, we expect that RawHash can also be useful for genome analysis that does not require real-time solutions by reducing the time and energy that further steps in genome analysis may require. One of the immediate steps after generating raw nanopore signals is their translation to their corresponding DNA bases as sequences of characters with a computationally intensive step, *basecalling*. Basecalling approaches are usually computationally costly and consume significant energy as they use complex deep learning models [275, 684]. Although we do not evaluate in this work, we expect that RawHash can be used as a low-cost filter [267] to eliminate the reads that are unlikely to be useful in downstream analysis, which can reduce the overall workload of basecallers and downstream analysis.

Future work. We find three key directions for future work. First, we find that our efficient hash-based similarity identification mechanism can be used to efficiently find overlaps between signals as the reads are sequenced in real-time. Although we observe that our indexing technique is efficient in terms of the amount it requires to construct an index even for large genomes, such an overlapping technique requires substantially more optimized indexing methods and techniques that can efficiently find overlaps as more reads are sequenced and *evolves* the index. Finding overlaps between signals can be beneficial in 1) providing enriched information to basecallers to increase their accuracy and 2) identifying *redundant* signals that fully overlap with already sequenced reads in an effort to generate assemblies from signals.

Second, since RawHash generates hash values for matching similar regions, it provides opportunities to use the hash-based seeding techniques that are optimized for identifying sequence similarities accurately without requiring large memory space, such as minimizers [306, 415], spaced seeds [315], syncmers [418], strobemers [310], and fuzzy seed matching as in BLEND [334]. Although we do not evaluate in this work, we implement the minimizer seeding technique in RawHash. Our initial observation motivates us that future work can exploit these seeding techniques with slight modifications in their seeding mechanisms to significantly improve the performance of certain applications without reducing the accuracy.

Third, as advancements in sequencing technologies continue at a rapid pace [1], raw data generated by sequencers is rapidly increasing in terms of volume and throughput, placing increasing pressure on computational pipelines and raw nanopore signal analysis. To bridge

the gap between analysis and sequencing, many acceleration efforts have targeted the computational bottlenecks of raw signal analysis by using GPUs [290, 643] and co-designing algorithm and hardware [138, 297]. RawHash can benefit from a hardware acceleration in two ways: 1) GPU acceleration and 2) acceleration with a a memory- or a storage-centric design. For a GPU acceleration, RawHash can benefit from a GPU implementation as its low-cost and accurate implementation can effectively be scaled to nanopore sequencers that include thousands of nanopores such that these pores can be analyzed in parallel with an efficient GPU implementation, which we leave as future work. However, as hardware acceleration increases and minimizes the computational bottleneck, the adverse effect of the data movement is likely to dominate the accelerated end-to-end execution latency. In such cases, to enable a substantial end-to-end acceleration, a memory-centric or a storage-centric design can 1) eliminate the overhead of data movement, 2) provide high parallelism capabilities, and 3) support the vast volume of genomic data [689, 710].

Limitations. We find four limitations of RawHash, which we believe can be improved with further optimizations and better solutions. First, RawHash depends on previously generated k-mer models to generate events from reference genomes. Although these k-mer models can be trained and generated [835, 1049], this makes it challenging to adapt the most accurate parameters for each k-mer model based on the nanopore model used for sequencing. A more generic k-mer model that can accurately represent all nanopores is needed to easily adapt RawHash to all possible nanopore models that may be released in the future.

Second, RawHash starts providing lower recall values as the genome size increases, which indicates that a larger portion of correct reads cannot be mapped by RawHash due to the increase in the number of false negatives. Although such an increase in false negatives does not substantially affect some applications, such as contamination analysis, where providing higher precision is more critical to correctly identify the contaminated sample, improving it is useful to provide more accurate genome analysis overall.

Third, we perform our relative abundance estimations based on a priori knowledge of reference genomes. While such an experiment can still be useful in practical scenarios, this is not the common case in metagenomic analysis, where a sample is searched against a significantly larger set of species. We expect that our mechanism can still scale to such metagenomic analyses given that many metagenomic databases are efficiently constructed by including fewer and useful information for each species [590], as opposed to our analysis, where we include whole-genome references.

Fourth, we observe that the throughput of RawHash is expected to reach the throughput of a nanopore when analyzing reference genomes slightly larger than a human genome. Such a limitation can be alleviated by applying 1) seeding techniques that provide faster and more

space-efficient searches in large spaces and 2) chaining algorithms that are optimized for hash-based seed matches without the notion of distance between seeds, unlike the chaining algorithm used in Sigmap, and 3) hardware acceleration, such as GPUs or memory-centric designs, to enable parallel processing and reduce data movement overhead, thereby ensuring RawHash can efficiently handle even larger genomes.

5.5 Summary

We propose RawHash, a novel mechanism that provides a low-cost and accurate approach for real-time genome analysis for large genomes. RawHash can efficiently and accurately perform real-time analysis of raw nanopore signals to identify similarities between the signals and a reference genome in real-time at a large scale (e.g., whole-genome analysis for human or communities with multiple samples). To efficiently and accurately identify similarities, RawHash 1) generates events from both raw signals and the reference genome, 2) quantizes the events into values such that slightly different events that correspond to the same DNA content can have the same value, and 3) generates hash values from multiple events to efficiently find matching regions between raw signals and a reference genome using hash values with efficient data structures such as hash tables. We compare RawHash with the state-of-the-art approaches, UNCALLED and Sigmap, on three important applications in terms of their performance, accuracy, and estimated benefits in reducing sequencing time and cost. Our results show that 1) RawHash is the *only* tool that can be accurately applied to analyze raw nanopore signals at large scale, 2) provides 25.8× and 3.4× better average throughput, and 3) can map reads 32.1× and 2.1× faster than UNCALLED and Sigmap, respectively. We hope that our findings will inspire future research to integrate real-time raw signal analysis in the genome analysis pipeline, especially when analyzing larger genomes, to significantly reduce the genome analysis latency as well as sequencing time and costs.

S5 Supplementary Materials

S5.1 Configuration

S5.1.1 Parameters

In Supplementary Table S5.1, we show the parameters of each tool for each dataset. In Supplementary Table S5.2, we show the details of the preset values that RawHash sets in Supplementary Table S5.1. For UNCALLED, Sigmap, and minimap2, we use the same parameter setting for all datasets. For the sake of simplicity, we only show the parameters that we explicitly set in each tool. For the descriptions of all the other parameters, we refer to the help message that each tool generates, including RawHash.

We note that the parameter names shown in Supplementary Table S5.3 are different from the parameters explained in Sections 5.2.3 and 5.2.4, although these parameters essentially perform in the same way as explained in these sections, which we describe next. First, the quantization parameter, Q in Section 5.2.3, is set using the `-q` parameter. Second, the value of p in Section 5.2.3 (i.e., pruned bits) can be calculated as $p = Q - l - 3$ where l is the least significant l bits of Q. We use l instead of q due to its easier programmability in our tool. This l value is set using the `-l` parameter in RawHash. Third, the number of events packed together, n in Section 5.2.4, is set using the `-e` parameter.

We set these `-q`, `-l`, and `-e` parameters empirically for three types of datasets: 1) viral genomes, 2) small genomes (i.e., $< 50M$ bases, and 3) large genomes (i.e., $> 50M$ bases) using the preset values `-x viral`, `-x sensitive`, and `-x fast`, respectively. In our empirical analysis, we identify that accuracy and performance are significantly impacted by the values we set for `-e`, `-q` and `-l` for three reasons. First, `e` determines the number of quantized event values packed in a single hash value. Packing a larger number of events improves the sensitivity as it becomes more challenging to find larger consecutive matches of quantized event values than finding a smaller number of consecutive matches. Finding a smaller set of matches can decrease the time spent in seeding and chaining. Second, these values determine the level of quantization of actual event values. Smaller `-q` and `-l` values can lead to loss of information due to storing only fewer bits that cannot be useful for identifying significantly different events. Larger `-q` and `-l` values can generate different quantized values for highly similar event values that may be corresponding to the same DNA content. Third, the number of bits that we store for each event, which are determined by `-q` and `-l`, impacts the number of events that can be packed in a single 32-bit or 64-bit value. Packing a larger number of events in a single hash value directly impacts sensitivity as discussed earlier in this paragraph (first point).

Table S5.1: Parameters we use in our evaluation for each tool and dataset in mapping.

Tool	*Contamination*	*SARS-CoV-2*	*E. coli*	*Yeast*	*Green Algae*	*Human*	*Relative Abundance*
RawHash	-x viral -t 32	-x viral -t 32	-x sensitive -t 32	-x sensitive -t 32	-x fast -t 32	-x fast -t 32	-x fast -t 32
UNCALLED	map -t 32						
Sigmap	-m -t 32						
Minimap2	-x map-ont -t 32						

Table S5.2: Corresponding parameters of presets (-x) in RawHash.

Preset (-x)	**Corresponding parameters**	Usage
viral	-e 5 -q 9 -l 3	Viral genomes
sensitive	-e 6 -q 9 -l 3	Small genomes (i.e., < 50M bases)
fast	-e 7 -q 9 -l 3	Large genomes (i.e., > 50M bases)

S5.1.2 Versions

Supplementary Table S5.3 shows the version and the link to these corresponding versions of each tool that we use in our experiments.

Table S5.3: Versions of each tool.

Tool	**Version**	**Link to the Source Code**
RawHash	0.9	https://github.com/CMU-SAFARI/RawHash/tree/8042b1728e352a28fcc79c2efd80c8b631fe7bac
UNCALLED	2.2	https://github.com/skovaka/UNCALLED/tree/74a5d4e5b5d02fb31d6e88926e8a0896dc3475cb
Sigmap	0.1	https://github.com/haowenz/sigmap/tree/c9a40483264c9514587a36555b5af48d3f054f6f
Minimap2	2.24	https://github.com/lh3/minimap2/releases/tag/v2.24

Chapter 6

Better Noise Reduction for and Robust Real-Time Mapping of Raw Nanopore Signals

The previous chapter develops a noise reduction mechanism for matching the hash values between raw nanopore signals. This chapter takes the next step and introduces 1) a new noise reduction technique by building a better understanding of the characteristics of noise in raw nanopore signals, 2) a more robust decision-making mechanism for read mapping, and 3) more sensitive computations for chaining seed matches between raw nanopore signals.

6.1 Background and Motivation

Nanopore technology can sequence long nucleic acid molecules up to more than two million bases at high throughput [1050]. As a molecule moves through a tiny pore, called a *nanopore*, ionic current measurements are generated at a certain throughput (e.g., around 450 bases per second for DNA [291, 292]). These electrical measurements, known as *raw signals*, can be used to 1) identify individual bases in the molecule with computational techniques such as *basecalling* [255] and 2) analyze raw signals directly *without* translating them to bases [266].

Computational techniques that can analyze the raw signals while they are generated at a speed that matches the throughput of nanopore sequencing are called *real-time analysis*. There are several mechanisms that can perform real-time analysis of raw nanopore signals to achieve accurate and fast genome analysis [138, 266, 290–292, 295, 297–299, 786–788]. Most of these solutions have three main limitations. First, many mechanisms offer limited scalability or support on resource-constrained devices due to their reliance on either 1) deep neural networks (DNNs) for real-time base translation, which are usually computationally intensive and power-

hungry [680, 786], or 2) specialized hardware such as ASICs or FPGAs [138, 290, 297]. Second, while some mechanisms can directly analyze raw signals without base translation, offering an efficient alternative for real-time analysis [291, 292], they often compromise accuracy or performance when applied to larger genomes. Third, methods based on machine learning often require retraining or reconfiguration [290, 293, 298], adding a layer of complexity and reducing their flexibility for general use cases, such as read mapping to any genome.

Among the existing works, RawHash [266] is the state-of-the-art mechanism that can accurately perform real-time mapping of raw nanopore signals for large genomes without translating them to bases with a hash-based seed-and-extend mechanism [316]. Despite its strengths in accuracy and performance, particularly for large genomes like the human genome, RawHash exhibits several limitations that require further improvements. First, RawHash utilizes a simple quantization algorithm that assumes the raw signals are distributed uniformly across their normalized value range, which limits its efficiency and accuracy. Second, RawHash uses a chaining algorithm similar to that used in Sigmap [291] without incorporating penalty scores used in minimap2 [306], which constrains its ability for more sensitive mapping. Third, RawHash performs chaining on all seed hits without filtering any of these seed hits, which substantially increases the workload of the chaining algorithm due to a large number of seed hits to chain. Fourth, the decision-making mechanism in RawHash for mapping reads to a reference genome in real-time relies on one of the mapping conditions being true (e.g., the ratio between the best and second-best chain scores), which makes it more prone to the outliers that can satisfy one of these conditions. A more robust and statistical approach that incorporates features beyond chaining scores can provide additional insights for making more sensitive and quick mapping decisions. Fifth, while the hash-based mechanism in RawHash is compatible with existing sketching techniques such as minimizers [306, 415], strobemers [310], and fuzzy seed matching as in BLEND [334], the benefits of these techniques are unknown for raw signal analysis as they are not used in RawHash. Such evaluations could potentially provide additional insights on how to use the existing hash-based sketching techniques and reduce storage requirements while maintaining high accuracy. Sixth, RawHash lacks the support for recent advancements, such as the newer R10.4 flow cell version. The integration of these features can accelerate the adoption of both real-time and offline analysis.

In this work, our goal is to address the aforementioned limitations of RawHash by improving its mechanism. To this end, we propose RawHash2 to improve RawHash in six directions. First, to generate more accurate and unique hash values, we introduce a new quantization technique, *adaptive quantization*. Second, to improve the accuracy of chaining and subsequently read mapping, we implement a more sophisticated chaining algorithm that incorporates penalty scores (as in minimap2). Third, to improve the performance of chaining by reducing its workload,

RawHash2 provides a filter that removes seeds frequently appearing in the reference genome, known as a *frequency filter*. Fourth, we introduce a statistical method that utilizes multiple features for making mapping decisions based on their weighted scores to eliminate the need for manual and fixed conditions to make decisions. Fifth, we extend the hash-based mechanism to incorporate and evaluate the minimizer sketching technique, aiming to reduce storage requirements without significantly compromising accuracy. Sixth, we integrate support for R10.4 flow cells and more recent file formats, POD5 and S/BLOW5 [1051].

Compared to RawHash, our extensive evaluations on five genomes of varying sizes and six different real datasets show that RawHash2 provides higher accuracy (by 10.57% on average and 20.25% at maximum) and better read mapping throughput (by 4.0× on average and 9.9× at maximum).

6.2 RawHash2 Mechanism

RawHash is a mechanism to perform mapping between raw signals by quickly matching their hash values. RawHash2 provides substantial improvements over RawHash in six key directions. First, to generate more accurate and distinct hash values from raw signals, RawHash2 improves the quantization mechanism with an *adaptive* approach such that signal values are quantized non-uniformly based on the characteristics of a nanopore model. Second, to provide more accurate mapping, RawHash2 improves the chaining algorithm in RawHash with more accurate penalty scores. Third, to reduce the workload in chaining for improved performance, we integrate a frequency filter to quickly eliminate the seed hits that occur too frequently. Fourth, to make more accurate and quick mapping decisions, RawHash2 determines whether a read should be mapped at a specific point during sequencing by using a weighted sum of multiple features. Fifth, to reduce the storage requirements of seeds, RawHash2 incorporates and evaluates the benefits of minimizer sketching technique. Sixth, RawHash2 includes support for the latest features introduced by ONT, such as new file formats and flow cells.

6.2.1 Adaptive Quantization

To improve the accuracy and uniqueness of hash values generated from raw nanopore signals, RawHash2 introduces a new *adaptive* quantization technique that we explain in four steps.

First, to enable a more balanced and accurate assignment of normalized signal values into quantized values (i.e., buckets), RawHash2 performs a bifurcated approach to define two different ranges: 1) fine range and 2) coarse range. These ranges are useful for fine-tuning the boundaries of normalized signal values, s falling into a certain quantized value, $q(s)$ within the integer value range $[0, n]$, as the normalized distribution of signal values is not uniform across

all ranges. Second, within the *fine range*, normalized signal values are quantized into smaller intervals to enable a high resolution, f_r, for quantization due to the larger number of normalized signal values that can be observed within this range. The boundaries of the fine range, (f_{min} and f_{max}), are empirically defined to enable robustness and high accuracy applicable given a flow cell (e.g., R9.4) and parameters to RawHash2 to enable flexibility. Third, the normalized signal values outside the fine range (i.e., the *coarse range*) are quantized into larger intervals with low resolution, $c_r = (1 - f_r) \times 0.5$, to enable a more balanced load of quantized values across all ranges by assigning more signal values within this range into the same quantized value. Fourth, depending on the range that a normalized signal is in, its corresponding quantized value is assigned as shown in Equation 6.1. The adaptive quantization approach can enable a more balanced and accurate distribution of quantized values by better distinguishing closeby signal values with high resolution and grouping signals more efficiently in the coarser range.

$$q(s) = \begin{cases} \lfloor n \times (f_r \times \frac{(s - f_{min})}{f_{max} - f_{min}}) & \text{if } f_{min} \leq s \leq f_{max} \\ \lfloor n \times (f_r + c_r \times s) & \text{if } s < f_{min} \\ \lfloor n \times (f_r + c_r + c_r \times s) & \text{if } s > f_{max} \end{cases} \tag{6.1}$$

6.2.2 Chaining with Penalty Scores

To identify the similarities between a reference genome (i.e., target sequence) and a raw signal (i.e., query sequence), the series of seed hits within close proximity in terms of their matching positions are identified using a dynamic programming (DP) algorithm, known as *chaining*. Using a chaining terminology similar to that of minimap2 [306], a seed hit between a reference genome and a raw signal is usually represented by a 3-tuple (x, y, w) value, known as *anchor*, where w represents the length of the region that a seed spans, the start and end positions of a matching interval in a reference genome and a raw signal is represented by $[x - w + 1, x]$ and $[y - w + 1, y]$, respectively. The chain of anchors within close proximity is identified by calculating the optimal chain score $f(i)$ of each anchor i, where $f(i)$ is calculated based on predecessors of anchor i when anchors are sorted by their reference positions. To calculate the chain score, $f(i)$, with dynamic programming, RawHash performs the following computation as used in Sigmap [291].

$$f(i) = \max \left\{ \max_{i > j \geq 1} \{f(j) + \alpha(j, i)\}, w_i \right\} \tag{6.2}$$

where $\alpha(j, i) = \min \left\{ \min\{y_i - y_j, x_i - x_j\}, w_i \right\}$ is the length of the matching region between the two anchors. Although such a design is useful when identifying substantially fewer seed matches using a seeding technique based on distance calculation as used in Sigmap, RawHash

identifies a larger number of seed matches as it uses hash values to identify the matching region, which is usually faster than a distance calculation with the cost of reduced sensitivity.

To identify the correct mapping regions among such a large number of seed matches, RawHash2 uses a more sensitive chaining technique as used in minimap2 by integrating the gap penalty scores such that the chain score of an anchor i is calculated as shown in Equation 6.3:

$$f(i) = \max\left\{ \max_{i>j\geq 1}\{f(j) + \alpha(j,i) - \beta(j,i)\}, w_i \right\} \tag{6.3}$$

where $\beta(j,i) = \gamma_c\big((y_i - y_j) - (x_i - x_j)\big)$ is the penalty score calculated based on the gap distance, l, between a pair of anchors i and j where $\gamma_c(l) = 0.01 \cdot w \cdot |l| + 0.5\log_2|l|$. Based on the chain score calculation with gap costs, RawHash2 integrates similar heuristics, mapping quality calculation, and the same complexity when calculating the chaining scores with the gap penalty as described in minimap2 [306].

6.2.3 Frequency Filters

RawHash2 introduces a two-step frequency filtering mechanism to 1) reduce the computational workload of the chaining process by limiting the number of anchors it processes and 2) focus on more unique and potentially meaningful seed hits. First, to reduce the number of queries made to the hash table for identifying seed hits, RawHash2 eliminates non-unique hash values generated from raw signals that appear more frequently than a specified threshold. Second, RawHash2 evaluates the frequency of each seed hit within the reference genome and removes those that surpass a predefined frequency threshold, which reduces the overall workload of the chaining algorithm by providing a reduced set of more unique seed hits.

6.2.4 Weighted Mapping Decision

RawHash performs mapping while receiving chunks of signals in real-time, as provided by nanopore sequencers. It is essential to decide if a read maps to a reference genome as quickly as possible to avoid unnecessary sequencing. The decision-making process in RawHash is based on a series of conditional checks involving chain scores. These checks are performed in a certain order and against fixed ratios and mean values, making the decision mainly rigid and less adaptive to variations.

To employ a more statistical approach that can generalize various variations between different data sets and genomes, RawHash2 calculates a weighted sum of multiple features that can impact the mapping decision. To achieve this, RawHash2 calculates normalized ratios of several metrics based on mapping quality and chain scores. These metrics are 1) the ratio of the mapping quality to a sufficiently high mapping quality (i.e., 30), 2) mapping quality ratio between the best chain and the mean quality of all chains, and 3) the ratio of the chain

score between the best and the mean score of all chains. These ratios are combined into a weighted sum as follows: $w_{\text{sum}} = \sum_{i=1} r_i \times w_i$, where r_i is a ratio of a particular metric, and w_i is the weight assigned for that particular metric. The weighted sum, w_{sum}, is compared against a predefined threshold value to decide if a read is considered to be mapped. RawHash2 maps a read if the weighted sum exceeds the threshold. Such a weighted sum approach allows RawHash2 to adaptively consider multiple aspects of the data and eliminates the potential effect of the ordering of these checks to achieve improved mapping accuracy while maintaining computational efficiency.

6.2.5 Minimizer Sketching

RawHash provides the opportunity to integrate the existing hash-based sketching techniques such as minimizers [306, 415] for 1) reduced storage requirements of index in disk and memory and 2) faster mapping due to fewer seed queries and hits.

To reduce the storage requirements of storing seeds in raw signals and due to their widespread application, RawHash2 integrates minimizers in two steps. First, RawHash2 generates hash values for seeds in both the reference genome and the raw signal. Second, within each window comprising w hash values, the minimum hash value is selected as the minimizer. These minimizer hash values can be used to find similarities using hash tables (similar to RawHash that uses hash values of all k-mers) while significantly reducing the number of hash values that need to be stored and queried during the mapping process as opposed to storing all k-mers.

6.2.6 Support for New Data Formats and Flow Cells

To enable better and faster adoption, RawHash2 incorporates support for 1) recent data formats for storing raw signals, namely POD5 and SLOW5 [1051] as well as the existing FAST5 format, and 2) the latest flow cell versions due to two main reasons. First, transitioning from the FAST5 to the POD5 file format is crucial for broad adoption, as POD5 is the new standard file format introduced by Oxford Nanopore Technologies (ONT). Second, integrating the newer flow cell versions is challenging as it requires optimization of parameters involved in mapping decisions as well as segmentation. RawHash2 enables mapping the raw signals from R10.4 flow cells by optimizing the segmentation parameters for R10.4 and adjusting the scoring parameters involved in chaining settings to enable accurate mapping for R10.4 flow cells.

6.3 Results

6.3.1 Evaluation Methodology

We implement the improvements we propose in RawHash2 directly on the RawHash implementation. Similar to RawHash, RawHash2 provides the mapping information using a standard pairwise mapping format (PAF).

We compare RawHash2 with the state-of-the-art works UNCALLED [292], Sigmap [291], RawHash [266] in terms of throughput, accuracy, and the number of bases that need to be processed before stopping the sequencing of a read to estimate the benefits in sequencing time and cost. We provide the release versions of these tools in Supplementary Table S6.6. For throughput, we calculate the number of bases that each tool can process per second per CPU thread, which is essential to determine if a calculation in a single thread is at least as fast as the speed of sequencing from a single nanopore (i.e., single pore). In many commonly used nanopore sequencers, a nucleic acid molecule passes through a pore at around 450 bases and 400 per second with sampling rates of 4 KHz and 5 KHz for DNA in R9.4.1 and R10.4.1, respectively [292, 294]. Since each read is mapped using a single thread for all tools, the throughput calculation is not affected by the number of threads available to these tools. Rather, this throughput calculation shows how many pores a single thread can process and how many CPU threads are needed to process the entire flow cell with many pores (e.g., 512 pores in a MinION flow cell). To show these results, we calculate 1) the number of pores that a single thread can process by dividing throughput by the number of bases sequenced per second per single pore and 2) the number of threads needed to cover the entire flow cell.

For accuracy, we analyze three use cases: 1) read mapping, 2) contamination analysis, and 3) relative abundance estimation. To identify the correct mappings, we generate the ground truth mapping output in PAF by mapping the basecalled sequences of corresponding raw signals to their reference genomes using minimap2 [306]. We use `UNCALLED pafstats` to compare the mapping output from each tool with their corresponding ground truth mapping output to calculate precision ($P = TP/(TP + FP)$), recall ($R = TP/(TP + FN)$), and F1 ($F1 = 2 \times (P \times R)/(P + R)$) values, similar to RawHash [266]. For read mapping, we compare the tools in terms of their precision, recall, and F-1 scores. For contamination analysis, the goal is to identify if a particular sample is contaminated with a certain genome (or set of genomes), which makes the precision metric more important for such a use case. For this use case, we compare the tools in terms of their precision in the main paper and show the full results (i.e., precision, recall, and F1) in Supplementary Table S6.1. For relative abundance estimation, we calculate the abundance ratio of each genome based on the ratio of reads mapped to a particular genome compared to all read mappings. We calculate the Euclidean distance of each estimation to the ground truth estimations generated based on minimap2 mappings of corresponding basecalled reads. We estimate the relative abundances based on the number of mapped reads rather than the number of mapped bases as we identify that larger genomes usually require sequencing a larger number of bases to map a read, which can lead to skewed estimations towards larger genomes.

To estimate the benefits in sequencing time and the cost per read, we identify the average sequencing length before making the mapping decision for a read. For all of our analyses, we use

the default parameters of each tool as we show in Supplementary Table S6.4. Table 6.1 shows the details of the datasets used in our evaluation and their corresponding sequencing run settings. The *Basecaller Model* column shows the details about the basecaller model and the version we use. Except for the D7 dataset, all other datasets include the basecalled sequences within their corresponding FAST5 files or the corresponding accession numbers available at NCBI. We provide the scripts to extract these basecalled sequences on the GitHub page of RawHash2. For the D7 dataset, we provide the necessary commands to run Dorado for basecalling on the GitHub page. Although RawHash2 does not use the minimizer sketching technique by default to achieve the maximum accuracy, we evaluate the benefits of minimizers in RawHash2, which we refer to as `RawHash2-Minimizer`. Since the evaluated versions of UNCALLED, Sigmap, and RawHash do not provide the support for the R10.4 dataset, we show the corresponding results with the R10.4 dataset without comparing to these tools. When comparing RawHash2 to other tools, we use FAST5 files containing raw signals from R9.4 flow cells on an isolated machine and SSD. We use the AMD EPYC 7742 processor at 2.26GHz to run the tools. We use 32 threads for all the tools.

Table 6.1: Details of datasets used in our evaluation.

	Organism	Device Type	Flow Cell Type	Transloc. Speed	Sampling Frequency	Basecaller Model	Reads (#)	Bases (#)	SRA Accession	Reference Genome	Genome Size
Read Mapping											
D1	*SARS-CoV-2*	MinION	R9.4.1 e8 (FLO-MIN106)	450	4000	Guppy HAC v3.2.6	1,382,016	594M	CADDE Centre	GCF_009858895.2	29,903
D2	*E. coli*	GridION	R9.4.1 e8 (FLO-MIN106)	450	4000	Guppy HAC v5.0.12	353,317	2,365M	ERR9127551	GCA_000007445.1	5M
D3	*Yeast*	MinION	R9.4.1 e8 (FLO-MIN106)	450	4000	Albacore v2.1.7	49,989	380M	SRR8648503	GCA_000146045.2	12M
D4	*Green Algae*	PromethION	R9.4.1 e8 (FLO-PRO002)	450	4000	Albacore v2.3.1	29,933	609M	ERR3237140	GCF_000002595.2	111M
D5	*Human*	MinION	R9.4.1 e8 (FLO-MIN106)	450	4000	Guppy Flip-Flop v2.3.8	269,507	1,584M	FAB42260	T2T-CHM13 (v2)	3,117M
D6	*E. coli*	GridION	R10.4 e8.1 (FLO-MIN112)	450	4000	Guppy HAC v5.0.16	1,172,775	6,123M	ERR9127552	GCA_000007445.1	5M
D7	*S. aureus*	GridION	R10.4 e8.1 (FLO-MIN112)	450	4000	Dorado SUP v0.5.3	407,727	1,281M	SRR21386013	GCF_000144955.2	2.8M
Contamination Analysis											
D1 and D5							1,651,523	2,178M	D1 and D5	D1	29,903
Relative Abundance Estimation											
D1-D5							2,084,762	5,531M	D1-D5	D1-D5	3,246M

Multiple dataset numbers in contamination analysis and relative abundance estimation show the combined datasets.
D1-D5 datasets are from R9.4, and D6 and D7 are from R10.4. Human reads are from Nanopore WGS.
Base counts in millions (M).

6.3.2 Throughput

Figure 6.1 shows the results for 1) throughput per single CPU thread and 2) number of pores that a single CPU thread can analyze as annotated by the values inside the bars. We make three key observations. First, we find that RawHash2 provides average throughput 26.5×, 19.2×, and 4.0× better than UNCALLED, Sigmap, and RawHash, respectively. Such a speedup, specifically over the earlier work RawHash, is achieved by reducing the workload of chaining with the unique and accurate hash values using the new quantization mechanism and the filtering technique (see the filtering ratios in Supplementary Table S6.3). Second, we find that RawHash2-Minimizer enables reducing the computational requirements for mapping

raw signals and enables improving the average throughput by 2.5× compared to RawHash2, while the other computational resources, such as the peak memory usage and CPU time in both indexing and mapping, and the mean time spent per read are also significantly reduced as shown in Supplementary Tables S6.2 and Supplementary Figure S6.1. Third, RawHash2-Minimizer requires *at most* 7 threads for analyzing the entire flowcell for any evaluated dataset, while RawHash2 requires at most 2 threads for smaller genomes and 9 to 26 threads for Green Algae and human. This shows that RawHash2 and RawHash2-Minimizer can reduce computational requirements and energy consumption significantly compared to 28 threads required, on average, regardless of the genome size for UNCALLED, which is critical for portable sequencing. We conclude that RawHash2 and RawHash2-Minimizer significantly reduce the computational overhead of mapping raw signals to reference genomes, enabling better scalability to even larger genomes.

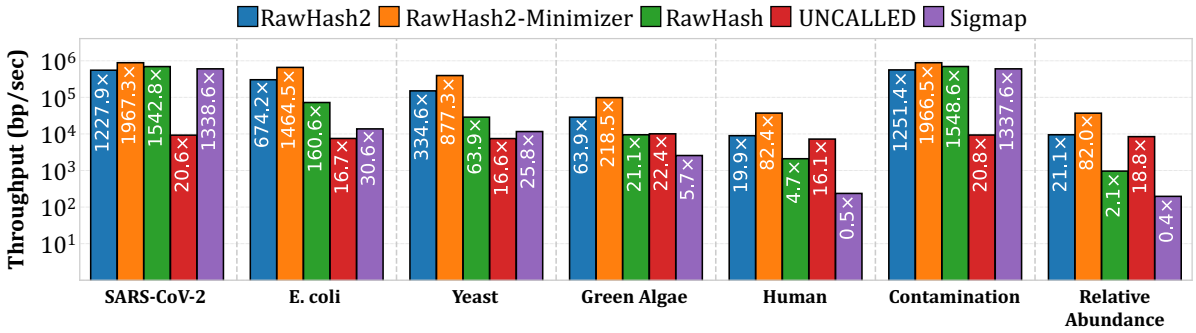

Figure 6.1: Throughput of each tool. Values inside the bars show how many nanopores (i.e., pores) that a single CPU thread can process.

6.3.3 Accuracy

Table 6.2 shows the accuracy results for read mapping, contamination analysis, and relative abundance estimation based on their corresponding most relevant accuracy metrics (results with all metrics are shown in Supplementary Table S6.1 and Figure 6.2). We make two key observations. First, we find that RawHash2 provides 1) the best accuracy in terms of the F1 score in all datasets for read mapping, 2) the best precision for contamination analysis, and 3) the most accurate relative abundance estimation. This is mainly achieved because 1) the adaptive quantization enables finding more accurate mapping positions while substantially reducing the false seed hits due to less precise quantization in RawHash, and 2) the more sensitive chaining implementation with penalty scores can identify the correct mappings more accurately.

Second, RawHash2-Minimizer provides mapping accuracy similar to that of RawHash2 with an exception for the human genome and better accuracy than RawHash, providing substantially

Table 6.2: Accuracy.

Dataset	Metric	RH2	RH2-Min.	RH	UNCALLED	Sigmap
SARS-CoV-2	F1	**0.9867**	0.9691	0.9252	0.9725	0.7112
E. coli	F1	**0.9748**	0.9631	0.9280	0.9731	0.9670
Yeast	F1	**0.9602**	0.9472	0.9060	0.9407	0.9469
Green Algae	F1	**0.9351**	0.9191	0.8114	0.8277	0.9350
Human	F1	**0.7599**	0.6699	0.5574	0.3197	0.3269
Contamination	Precision	**0.9595**	0.9424	0.8702	0.9378	0.7856
Rel. Abundance	Distance	**0.2678**	0.4243	0.4385	0.6812	0.5430

Best results are **highlighted**.

better performance results as discussed in Section 6.3.2. Such an accuracy-performance trade-off puts RawHash2-Minimizer in an important position when a slight drop in accuracy can be tolerated for a particular use case when a substantially better throughput is needed. For the relatively lower accuracy that RawHash2 and RawHash2-Minimizer achieve compared to minimap2, we believe the accuracy gap is due to the increased difficulty in distinguishing the chain with the correct mapping position among many chains with similar quality scores, potentially due to the false seed matches in repetitive regions. Although our in-house evaluation shows that accuracy can substantially be improved further by enabling the correct chains to be distinguished more accurately than the incorrect chains with more sensitive quantization parameters, this comes with increased performance costs due to increased seed matches and chaining calculations. Future work can focus on designing more sensitive filters to improve the accuracy for larger and repetitive genomes by eliminating seed matches from such false regions. We conclude that RawHash2 is the most accurate tool regardless of the genome size, while the minimizer sketching technique in RawHash2-Minimizer can provide better accuracy than RawHash and on-par accuracy to all other tools while providing the best overall performance.

6.3.4 Sequencing Time and Cost

Table 6.3 shows the average sequencing lengths in terms of bases and chunks that each tool needs to process before stopping the sequencing process of a read. Processing fewer bases can significantly help reduce the overall sequencing time and potentially the cost spent for each read by enabling better utilization of nanopores without sequencing the reads unnecessarily. Figure 6.3 shows the combined results of each tool in terms of throughput, F-1 Score (i.e., accuracy), and average sequencing length for each dataset. The dotted lines in each triangle show the ideal combined result. Each edge of the triangle shows the best result for the corresponding metric, as shown in the figure. For the edge that shows the F-1 score, the best point is 1.0. All tools have F-1 scores between 0 and 1, as shown in Table 6.2. For the other two edges, which

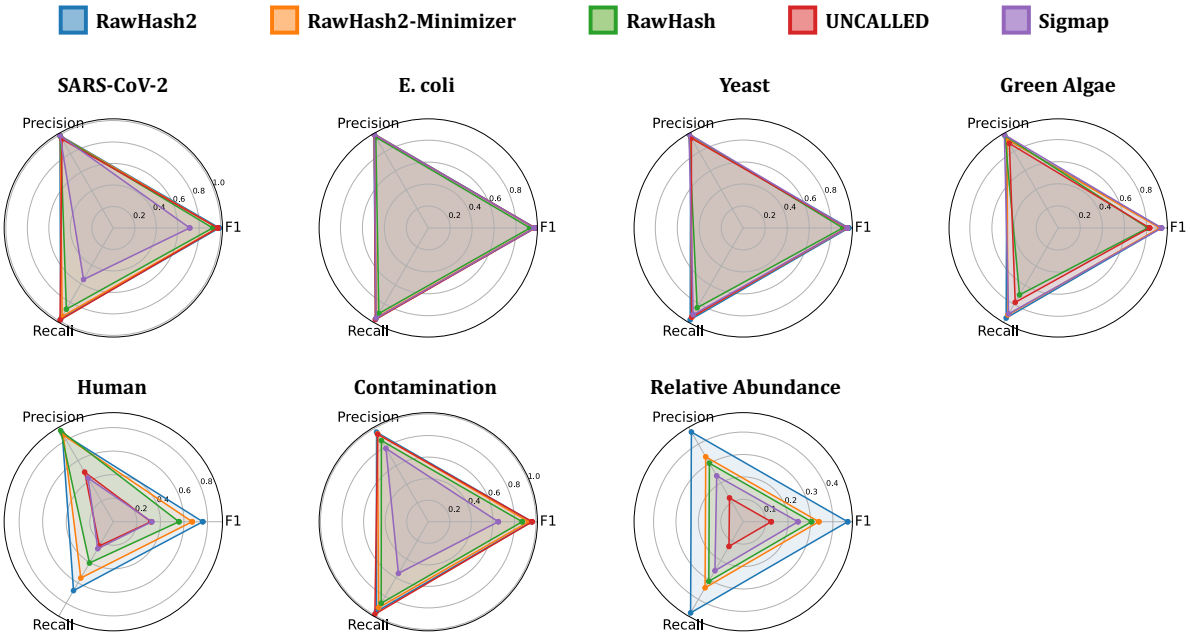

Figure 6.2: Read mapping accuracy results in terms of F1 score, precision, and recall across different datasets. The dotted triangles show the best possible results, where each edge shows the best result for its corresponding metric.

show throughput and average sequencing length, the best result is determined based on the highest result we observe for that dataset. We adjust all other results using these highest results so that the adjusted throughput and average sequencing length values are always between 0 and 1. We make three key observations.

Table 6.3: Average length of sequencing per read.

Dataset	RH2	RH2-Min.	RH	UNCALLED	Sigmap
SARS-CoV-2	443.92	460.85	513.95	**184.51**	452.38
E. coli	851.31	1,030.74	1,376.14	**580.52**	950.03
Yeast	**1,147.66**	1,395.87	2,565.09	1,233.20	1,862.69
Green Algae	**1,385.59**	1,713.46	4,760.59	5,300.15	2,591.16
Human	**2,130.59**	2,455.99	4,773.58	6,060.23	4,680.50
Contamination	670.69	**667.89**	742.56	1,582.63	927.82
Rel. Abundance	**1,024.28**	1,182.04	1,669.46	2,158.50	1,533.04

Best results are **highlighted**.

First, RawHash2 reduces the average sequencing length by 1.9× compared to RawHash mainly due to the improvements in mapping accuracy, which enables making quick decisions without using longer sequences. Second, as the genome size increases, RawHash2 provides the smallest average sequencing lengths compared to all tools. Third, when the average length of

sequencing is combined with other important metrics such as mapping accuracy in terms of F1 score and throughput, RawHash2 provides the best trade-off in terms of all these three metrics for all datasets as shown in Figure 6.3. We conclude that RawHash2 is the best tool for longer genomes to reduce the sequencing time and cost per read as it provides the smallest average sequencing lengths, while UNCALLED is the best tool for shorter genomes.

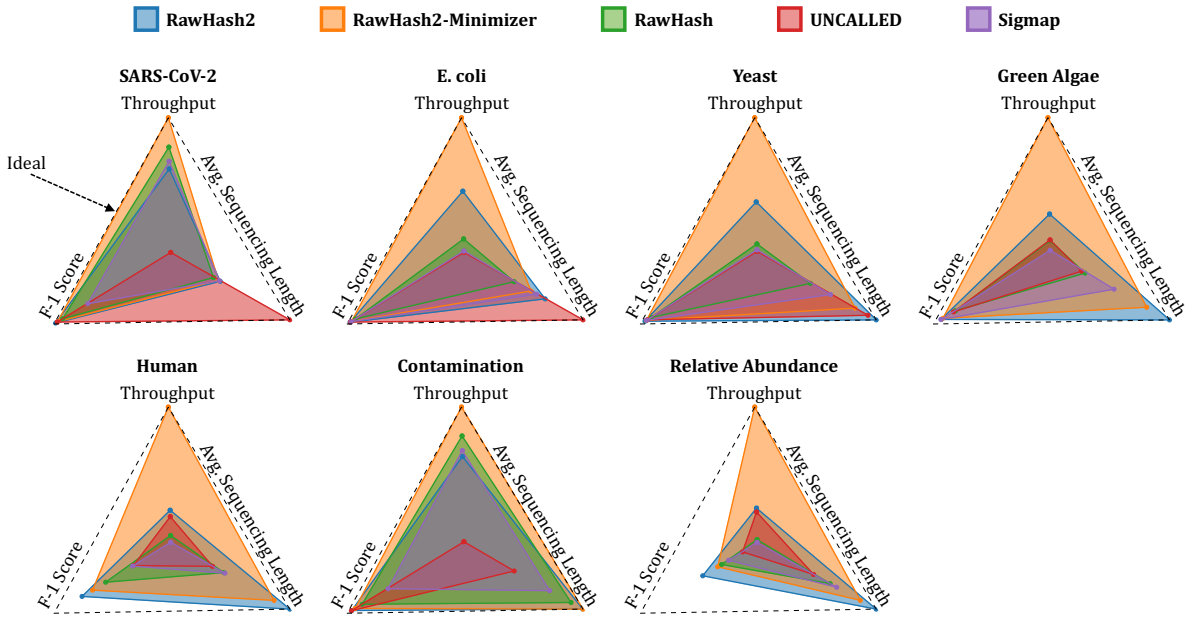

Figure 6.3: Combined results in terms of throughput, F-1 score (i.e., accuracy), and average sequencing length across different datasets. The dotted triangles show the best possible results, where each edge shows the best result for its corresponding metric.

6.3.5 Evaluating New File Formats and R10.4

Impact of Different File Formats on Performance. Table 6.4 shows the overall execution time when using different raw signal file formats: FAST5, POD5, and BLOW5 [1051]. To evaluate the direct impact of these formats, we run RawHash2 (RH2) and RawHash2-Minimizer (RH2-Min.) 1) using a single thread (i.e., single thread for the entire execution including *both* file IO and mapping), 2) using an isolated SSD on a PCI-e interface, 3) using the same compression type (i.e., zstd) for all file formats, and 4) clearing the disk cache before each execution. When using a single thread, we confirm that the underlying libraries for FAST5, POD5, and BLOW5 are not aggressively using more threads than what is allocated to them, as the thread utilization is reported as 0.99 (i.e., 99%) by the `time -v` command for the entire execution.

We note that even if we use multiple threads when running RawHash2, the file IO step (i.e., reading from or writing to a file) always uses a single thread and is overlapped with the mapping step (i.e., either the read or write operation is run in parallel together with the mapping step by using one thread where the mapping step takes rest of the allocated threads).

The design is due to the pipelining implementation strategy we adopt, similar to the minimap2 implementation [306]. We note that if RawHash2 is run using a single thread, none of these steps overlap with each other, and they run sequentially using only one thread, which is our evaluation setting we show in Table 6.4.

We find that POD5 and SLOW5 significantly speed up total elapsed time compared to FAST5. These results indicate that a large portion of the overhead spent for reading from a file can be mitigated with approaches that can perform faster compression and decompression, as these signal files are mostly stored in a compressed form.

Table 6.4: Comparison of overall execution time when using different file formats in RawHash2 in a single-threaded mode.

Tool	*E. coli*	*Yeast*
Elapsed Time (mm:ss)		
RH2-FAST5	19:27	08:35
RH2-POD5	16:55	07:33
RH2-BLOW5	17:32	07:38
RH2-Min.-FAST5	12:13	03:56
RH2-Min.-POD5	09:42	02:56
RH2-Min.-BLOW5	10:16	03:02

R10.4 Accuracy and Performance. In Table 6.5, we show the accuracy and performance results in terms of throughput and mean time spent per read when using R10.4 flow cells. For comparison purposes between R10.4 and R9.4, we include the results from R9.4 flow cells for *E. coli*. We do not show the R9.4 results for *S. aureus*, since we do not have raw signals from the same sample for this dataset.

We find that RawHash2 can perform fast analysis with reasonable accuracy that can be useful for certain use cases (e.g., contamination analysis) when using raw signals from R10.4, although RawHash2 achieves lower accuracy with R10.4 than using R9.4. This is likely because 1) we use a k-mer model optimized for the R10.4.1 flow cell version rather than R10.4, and 2) minimap2 can provide more accurate mapping due to improved accuracy of these basecalled reads. Future work can focus on generating a k-mer model specifically designed for R10.4 to generate more accurate results. We exclude the accuracy results for R10.4.1 as the number of events found for R10.4.1 is around 35% larger than that of R10.4, which leads to inaccurate mapping. We suspect that our segmentation algorithm and parameters are not optimized for R10.4.1. Our future work will focus on improving these segmentation parameters and techniques to achieve higher accuracy with R10.4.1 as well as RNA sequencing data. We believe this can be achieved because RawHash2 1) is highly flexible to change all the parameters corresponding to segmentation and 2) can map accurately without requiring long sequencing

lengths (Table 6.3), which can mainly be useful for RNA read sets. We conclude that RawHash2 can provide accurate and fast analysis when using the recent features released by ONT.

Table 6.5: Accuracy and performance results when using R10.4 and R9.4 datasets

Flow Cell		RH2	RH2-Min.
Read Mapping Accuracy (E. coli)			
R9.4	F1	0.9748	0.9631
	Precision	0.9904	0.9865
	Recall	0.9597	0.9408
R10.4	F1	0.8960	0.8389
	Precision	0.9506	0.9325
	Recall	0.8473	0.7623
Read Mapping Accuracy (S. aureus)			
R10.4	F1	0.7749	0.6778
	Precision	0.8649	0.8167
	Recall	0.7018	0.5793
Performance (E. coli)			
R9.4	Throughput [bp/sec]	303,382.45	659,013.57
	Mean time per read [ms]	2.161	1.099
R10.4	Throughput [bp/sec]	175,351.94	480,471.75
	Mean time per read [ms]	6.598	2.505
Performance (S. aureus)			
R10.4	Throughput [bp/sec]	256,680.4	617,308.7
	Mean time per read [ms]	5.478	2.243

6.4 Summary

We introduce RawHash2, a tool that provides substantial improvements over the previous state-of-the-art mechanism RawHash. We make five key improvements over RawHash: 1) more sensitive quantization and chaining, 2) reduced seed hits with filtering mechanisms, 3) more accurate mapping decisions with weighted decisions, 4) the first minimizer sketching technique for raw signals, and 5) integration of the recent features from ONT. We find the RawHash2 provides substantial improvements in throughput and accuracy over RawHash. We conclude that RawHash2, overall, is the best tool for mapping raw signals due to its combined benefits in throughput, accuracy, and reduced sequencing time and cost per read compared to the existing mechanisms, especially for longer genomes. We hope that, with the substantial improvements in performance and accuracy RawHash2 provides, it leads to 1) wider adoption of real-time and raw nanopore signal analysis and 2) more accurate techniques for reducing the noise in signals to achieve faster and more accurate analysis.

S6 Supplementary Materials

S6.1 Accuracy

S6.1.1 Read Mapping Accuracy

In Supplementary Table S6.1, we show the read mapping accuracy in all metrics (i.e., F1, Precision, and Recall) for all datasets.

Table S6.1: Read mapping accuracy in all metrics: F1, Precision, and Recall.

Dataset	Metric	RH2	RH2-Min.	RH	UNCALLED	Sigmap
SARS-CoV-2	F1	**0.9867**	0.9691	0.9252	0.9725	0.7112
	Precision	**0.9939**	0.9868	0.9832	0.9547	0.9929
	Recall	0.9796	0.9521	0.8736	**0.9910**	0.5540
E. coli	F1	**0.9748**	0.9631	0.9280	0.9731	0.9670
	Precision	**0.9904**	0.9865	0.9563	0.9817	0.9842
	Recall	0.9597	0.9408	0.9014	**0.9647**	0.9504
Yeast	F1	**0.9602**	0.9472	0.9060	0.9407	0.9469
	Precision	0.9553	0.9561	0.9852	0.9442	**0.9857**
	Recall	**0.9652**	0.9385	0.8387	0.9372	0.9111
Green Algae	F1	**0.9351**	0.9191	0.8114	0.8277	0.9350
	Precision	0.9284	0.9280	0.9652	0.8843	**0.9743**
	Recall	**0.9418**	0.9104	0.6999	0.7779	0.8987
Human	F1	**0.7599**	0.6699	0.5574	0.3197	0.3269
	Precision	0.8675	0.8511	**0.8943**	0.4868	0.4288
	Recall	**0.6760**	0.5523	0.4049	0.2380	0.2642
Contamination	F1	0.9614	0.9317	0.8718	**0.9637**	0.6498
	Precision	**0.9595**	0.9424	0.8702	0.9378	0.7856
	Recall	0.9632	0.9212	0.8736	**0.9910**	0.5540
Rel. Abundance	F1	**0.4659**	0.3375	0.3045	0.1249	0.2443
	Precision	**0.4623**	0.3347	0.3018	0.1226	0.2366
	Recall	**0.4695**	0.3404	0.3071	0.1273	0.2525

Best results are **highlighted**.

S6.2 Performance

S6.2.1 Runtime, Peak Memory Usage, and Throughput

Supplementary Table S6.2 shows the computational resources required by each tool during the indexing and mapping steps. To measure the required computational resources, we collect CPU time and peak memory usage of each tool for all the datasets. To collect these results, we use `time -v` command in Linux. CPU time shows the total user and system time. Peak memory usage shows the maximum resident set size in the main memory that the application requires to complete its task. To measure the CPU threads needed for analyzing the entire MinION Flowcell with 512 pores, we divide 512 with the number of pores that a single thread can process (as shown with the values inside the bars in Figure 6.1) and round up the values to provide the maximum number of threads needed.

Table S6.2: Computational resources required in the indexing and mapping steps.

Dataset	RH2	RH2-Min.	RH	UNCALLED	Sigmap
Indexing CPU Time (sec)					
SARS-CoV-2	0.12	0.06	0.16	8.40	**0.02**
E. coli	2.48	**1.61**	2.56	10.57	8.86
Yeast	4.56	**3.02**	4.44	16.40	25.29
Green Algae	27.60	**17.73**	24.51	213.13	420.25
Human	1,093.56	**588.30**	809.08	3,496.76	41,993.26
Contamination	0.13	0.06	0.15	8.38	**0.03**
Rel. Abundance	747.74	**468.14**	751.67	3,666.14	36,216.87
Indexing Peak Memory (GB)					
SARS-CoV-2	**0.01**	**0.01**	**0.01**	0.06	**0.01**
E. coli	0.35	0.19	0.35	**0.11**	0.40
Yeast	0.75	0.39	0.76	**0.30**	1.04
Green Algae	5.11	**2.60**	5.33	11.94	8.63
Human	80.75	**40.59**	83.09	48.43	227.77
Contamination	**0.01**	**0.01**	**0.01**	0.06	**0.01**
Rel. Abundance	152.59	75.62	152.84	**47.80**	238.32
Mapping CPU Time (sec)					
SARS-CoV-2	1,705.43	**1,227.05**	1,539.64	29,282.90	1,413.32
E. coli	1,296.34	**787.49**	7,453.21	28,767.58	22,923.09
Yeast	545.77	**246.37**	4,145.38	7,181.44	7,146.32
Green Algae	2,135.83	**657.63**	22,103.03	12,593.01	26,778.44
Human	100,947.58	**21,860.05**	1,825,061.23	245,128.15	6,101,179.89
Contamination	3,783.69	**2,332.28**	3,480.43	234,199.60	3,011.78
Rel. Abundance	250,076.90	**62,477.76**	4,551,349.79	569,824.13	15,178,633.11
Mapping Peak Memory (GB)					
SARS-CoV-2	4.15	4.16	4.20	**0.17**	28.26
E. coli	4.13	4.03	4.18	**0.50**	111.12
Yeast	4.38	4.12	4.37	**0.36**	14.66
Green Algae	6.11	4.98	11.77	**0.78**	29.18
Human	48.75	25.04	52.43	**10.62**	311.94
Contamination	4.16	4.14	4.17	**0.62**	111.70
Rel. Abundance	49.14	25.82	54.89	**8.99**	486.63
Mapping Throughput (bp/sec)					
SARS-CoV-2	552,561.25	**885,263.48**	694,274.92	9,260.31	602,380.96
E. coli	303,382.45	**659,013.57**	72,281.32	7,515.76	13,750.97
Yeast	150,547.61	**394,766.80**	28,757.15	7,471.48	11,624.82
Green Algae	28,742.46	**98,323.70**	9,488.79	10,069.41	2,569.89
Human	8,968.78	**37,086.38**	2,099.35	7,225.67	236.45
Contamination	563,129.81	**884,929.30**	696,873.20	9,343.95	601,936.49
Rel. Abundance	9,501.37	**36,919.79**	962.79	8,437.70	196.48
CPU Threads Needed for the entire MinION Flowcell (512 pores)					
SARS-CoV-2	**1**	**1**	**1**	25	**1**
E. coli	**1**	**1**	4	31	17
Yeast	2	**1**	9	31	20
Green Algae	9	**3**	25	23	90
Human	26	**7**	110	32	975
Contamination	**1**	**1**	**1**	25	**1**
Rel. Abundance	25	**7**	240	28	1173

Best results are **highlighted**.

S6.2.2 Mapping Time per Read

Supplementary Figure S6.1 shows the average mapping time that each tool spends per read for all the datasets we evaluate. The mapping times spent per read are provided by each tool as PAF output with the `mt` tag. We use these reported values to calculate the average mapping time across all reads reported in their corresponding PAF files.

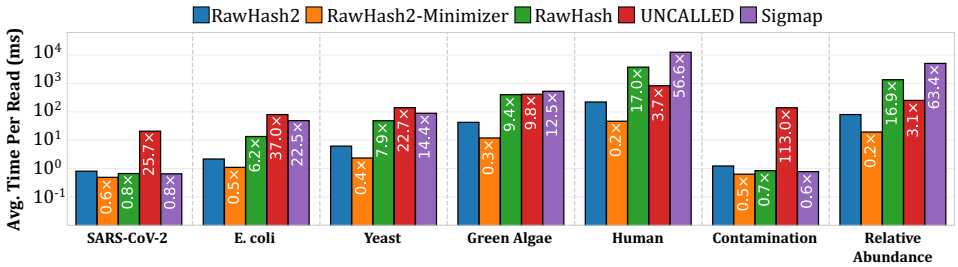

Figure S6.1: Average time spent per read by each tool in real-time. Values inside the bars show the speedups that RawHash2 provides over other tools in each dataset.

S6.2.3 Ratio of Filtered Seeds from Frequency Filter

Supplementary Table S6.3 shows the ratio of seed hits filtered out by the frequency filtered in RawHash2. We calculate these ratios in three steps. First, for each seed (i.e., a hash value that RawHash2 constructs from raw signals), we perform a query to the hash table that is used as an index. If the hash value exists, the table returns a list of genomic regions that share the same hash value. Each region counts as a seed hit, and the list length indicates the number of seed hits. Second, for all seeds generated from raw signals, we count 1) the overall number of seed hits and 2) the number of seed hits filtered out by frequency filter. We note that if the list length (i.e., number of seed hits) returned after querying a particular seed is above a certain threshold (defined by our frequency filter), all seed hits within the same list are filtered out. Third, we calculate the ratio of filtered seed hits to the total seed hits and report these ratios in Supplementary Table S6.3.

Table S6.3: Ratio of filtered seed hits from frequency filter.

Dataset	Average Filtered Ratio
SARS-CoV-2	0.0627
E. coli	0.5505
Yeast	0.5356
Green Algae	0.8106
Human	0.5104
E. coli (R10.4)	0.6895
S. aureus (R10.4)	0.6003

S6.3 Configuration

S6.3.1 Parameters

In Supplementary Table S6.4, we show the parameters of each tool for each dataset. In Supplementary Table S6.5, we show the details of the preset values that RawHash2 sets in Supplementary Table S6.4. For UNCALLED [292], Sigmap [291], and minimap2 [306], we use the same parameter setting for all datasets. For the sake of simplicity, we only show the parameters we explicitly set in each tool. For the descriptions of all the other parameters, we refer to the help message that each tool generates, including RawHash2.

Table S6.4: Parameters we use in our evaluation for each tool and dataset in mapping.

Tool	*Contamination*	*SARS-CoV-2*	*E. coli* (R9.4)	*Yeast*	*Green Algae*	*Human*	*Rel. Abundance*	*E. coli* (R10.4)	*S. aureus* (R10.4)
RawHash2	-x viral –depletion -t 32	-x viral -t 32	-x sensitive -t 32	-x sensitive -t 32	-x sensitive -t 32	-x fast -t 32	-x fast -t 32	-x sensitive –r10 -t 32	-x sensitive –r10 -t 32
RawHash2-Minimizer	-x viral -w3 –depletion -t 32	-x viral -w3 -t 32	-x sensitive -w3 -t 32	-x sensitive -w3 -t 32	-x sensitive -w3 -t 32	-x fast -w3 -t 32	-x fast -w3 -t 32	-x sensitive –r10 -w3 -t 32	-x sensitive –r10 -w3 -t 32
RawHash	-x viral -t 32	-x viral -t 32	-x sensitive -t 32	-x sensitive -t 32	-x fast -t 32	-x fast -t 32	-x fast -t 32	NA	NA
UNCALLED	map -t 32							NA	NA
Sigmap	-m -t 32							NA	NA
Minimap2	-x map-ont -t 32								

Table S6.5: Corresponding parameters of presets (-x) in RawHash2.

Preset	Corresponding parameters	Usage
viral	-e 6 -q 4 –max-chunks 5 –bw 100 –max-target-gap 500 –max-target-gap 500 –min-score 10 –chain-gap-scale 1.2 –chain-skip-scale 0.3	Viral genomes
sensitive	-e 8 -q 4 –fine-range 0.4	Small genomes (i.e., < 500M bases)
fast	-e 8 -q 4 –max-chunks 20	Large genomes (i.e., > 500M bases)
	Other helper parameters	
depletion	–best-chains 5 –min-mapq 10 –w-threshold 0.5 –min-anchors 2 –min-score 15 –chain-skip-scale 0	Contamination analysis
r10	-k9 –seg-window-length1 3 –seg-window-length2 6 –seg-threshold1 6.5 –seg-threshold2 4 –seg-peak-height 0.2 –chain-gap-scale 1.2	For R10.4 Flow Cells

S6.3.2 Versions

Supplementary Table S6.6 shows the version and the link to these corresponding versions of each tool and library we use in our experiments and in RawHash2, respectively.

Table S6.6: Versions of each tool and library.

Tool	Version	Link to the Source Code
RawHash2	2.1	https://github.com/CMU-SAFARI/RawHash/releases/tag/v2.1
RawHash	1.0	https://github.com/CMU-SAFARI/RawHash/releases/tag/v1.0
UNCALLED	2.3	https://github.com/skovaka/UNCALLED/releases/tag/v2.3
Sigmap	0.1	https://github.com/haowenz/sigmap/releases/tag/v0.1
Minimap2	2.24	https://github.com/lh3/minimap2/releases/tag/v2.24
		Library versions
FAST5 (HDF5)	1.10	https://github.com/HDFGroup/hdf5/tree/db30c2d
POD5	0.2.2	https://github.com/nanoporetech/pod5-file-format/releases/tag/0.3.10
S/BLOW5	1.2.0-beta	https://github.com/hasindu2008/slow5lib/tree/e0d0d0f

Chapter 7

Enabling New Applications by Overlapping and Assembling Raw Nanopore Signals

Throughout the previous three chapters, we built a detailed understanding of the effects of noise and the limitations this noise puts on the heuristics and algorithms designed to handle noise. In this chapter, after better handling and understanding of this noise for raw nanopore signals, we enable a new application in raw nanopore signals by introducing a new technique, Rawsamble, that can find all-vs-all overlapping of reads to build *de novo* assemblies from raw signals. We discuss new directions that can be enabled by Rawsamble.

7.1 Background and Motivation

Nanopore sequencing technology [225–247] can sequence long nucleic acid molecules of up to a few million bases at high throughput [255, 820, 1050]. As a molecule moves through a tiny pore, called a *nanopore*, ionic current measurements, called *raw signals*, are generated [229]. Nanopore sequencing provides three unique key benefits.

First, nanopore sequencing enables stopping the sequencing of single reads or the entire sequencing run early, known as *adaptive sampling* or *selective sequencing* [264], while raw signals are generated and analyzed during sequencing, called *real-time analysis*. Adaptive sampling can substantially reduce the sequencing time and cost by avoiding unnecessary sequencing. Second, raw nanopore signals retain more information than nucleotides, such as methylation, and can be useful for many applications [1052]. Third, compact nanopore sequencing devices enable on-site portable sequencing and analysis, which can be coupled with real-time analysis [1053].

Existing works that analyze raw nanopore signals [138, 264–288, 290–302] mainly utilize deep learning mechanisms [267–284] to translate these signals into nucleotides, a process called *basecalling*. Basecalling mechanisms usually 1) are designed to use large chunks of raw signal data for accurate analysis [291] and 2) have high computational requirements [255, 297]. This can impose limitations to enable 1) accurate real-time analysis [291] and 2) portable sequencing with constrained resources [297].

To fully utilize the unique benefits of nanopore sequencing, it is necessary to analyze raw signals with 1) high accuracy and low latency for adaptive sampling and 2) low resource usage for portability and efficiency. To achieve this, several mechanisms focus on analyzing raw nanopore signals *without* basecalling [138, 264–266, 290–302].[1] Although raw nanopore signal mapping to a reference genome is widely studied to achieve relatively accurate and fast mapping of raw signals [138, 264–266, 291, 292, 295, 297, 299–302], *none* of these works *without* a reference genome.

When a reference genome is not available, a genome can be constructed from scratch, called *de novo* assembly construction [250]. To construct a *de novo* assembly, similar regions between all read pairs are identified, called *all-vs-all overlapping, instead of identifying similarities between reads and a reference genome.* [255, 305, 334]. However, an all-vs-all overlapping between raw nanopore signals remains unsolved due to several unique challenges [229, 822–824].

First, it is challenging to identify similarities between a pair of noisy raw signals as compared to a noisy raw signal and an accurate signal generated from a reference genome, an approach commonly used in earlier works for read mapping [265, 266, 291, 292, 295]. This is because converting reference genomes to their expected raw signal values is free from certain types of noise that raw nanopore signals contain (e.g., stochastic signal fluctuations [229] and variable speed of DNA molecules moving through nanopores [823, 824]), while none of the raw signals are free from such noise in all-vs-all overlapping.

Second, existing raw signal analysis works lack the mapping strategies typically used in all-vs-all overlapping, such as reporting multiple mappings (i.e., overlaps) of a read to many reads. This is because these works are mainly designed to stop the mapping process as soon as there is an accurate mapping for a read to minimize the unnecessary sequencing [292]. For all-vs-all overlapping, pairwise mappings of a read to many reads (instead of a single mapping) must be reported while avoiding certain trivial cyclic pairwise mappings to construct an assembly [305].

Third, read overlapping can increase the space requirements for storing and using indexing. This is because a read set is likely to be sequenced such that its overall number of bases is larger than the number of bases in its corresponding genome. Such an increased index size

[1] We use the *raw signal analysis* term specifically for these mechanisms in the remainder of the paper.

can raise the computational and space demands for read overlapping. It is essential to provide high-throughput and scalable computation to enable real-time downstream analysis for future work (as discussed in Section 7.4).

Our goal is to enable 1) raw signal analysis *without* a reference genome and 2) new use cases with raw nanopore signal analysis, such as *assembly from overlapping* raw signals, by addressing the challenges of accurate and fast all-vs-all overlapping of raw nanopore signals. To this end, we propose *Rawsamble*, the first mechanism that enables fast and accurate overlap finding between raw nanopore signals. The key idea in Rawsamble is to re-design the existing state-of-the-art hash-based seeding mechanism [265, 266] for raw signals with more effective noise reduction techniques and useful outputting strategies to find all overlapping pairs accurately, which we explain in three key steps.

First, to enable identifying similarities between a pair of noisy raw signals accurately, Rawsamble filters raw signals to select those substantially distinct from their surrounding signals. Such non-distinct and adjacent signals are usually the result of a certain error type in the analysis, known as *stay errors* [266, 291, 299]. Although similar filtering strategies [266, 291, 299] are exploited when mapping raw signals to reference genomes, Rawsamble performs a more aggressive filtering to avoid storing the erroneous portions of signals in the index to enable accurate similarity identification from the sufficiently distinct regions of signals. Second, to find multiple overlaps for a read, Rawsamble identifies highly accurate chains from seed matches based on their chaining scores and reports *all* of these chains as mappings, as opposed to choosing solely the best mapping as determined by weighted decisions among all such chains [265]. Third, to prevent trivial cycles between a pair of overlapping reads, Rawsamble ensures that only one of the overlapping reads in each pair is always chosen as a query sequence, and the other is always chosen as a target sequence based on a deterministic ordering mechanism between these reads. These steps enable Rawsamble to find overlaps between raw signals accurately and quickly.

7.2 Methods

7.2.1 Overview

Rawsamble is a mechanism to find overlapping pairs between raw nanopore signals (i.e., *all-vs-all overlapping*) for constructing *de novo* assemblies without basecalling, as shown in Figure 7.1. To achieve this, Rawsamble builds on the state-of-the-art raw signal mapper, RawHash2 [265, 266]. Rawsamble extends RawHash2 to support all-vs-all raw signal overlapping in four key steps: First, to enable efficient and accurate indexing from the noisy raw signals (❶), Rawsamble aggressively filters the raw input after the signal-to-event conversion to avoid nanopore-related errors. Second, to improve the accuracy of overlapping (❷), Rawsamble

adjusts the minimum chaining score to avoid false chains, ensuring that only high-confidence overlaps are considered. Third, to enable finding useful and long connections from many overlaps, Rawsamble adjusts the output strategy such that 1) *all* chains rather than only the best chain are reported and 2) cyclic overlaps are avoided. Fourth, we introduce a new assembler to output the assembled sequences after performing *de novo* assembly from raw signal overlaps (❸). Rawsamble enables the use of existing *de novo* assemblers such as miniasm [305] as it provides the overlap information in a standardized format that these assemblers use.

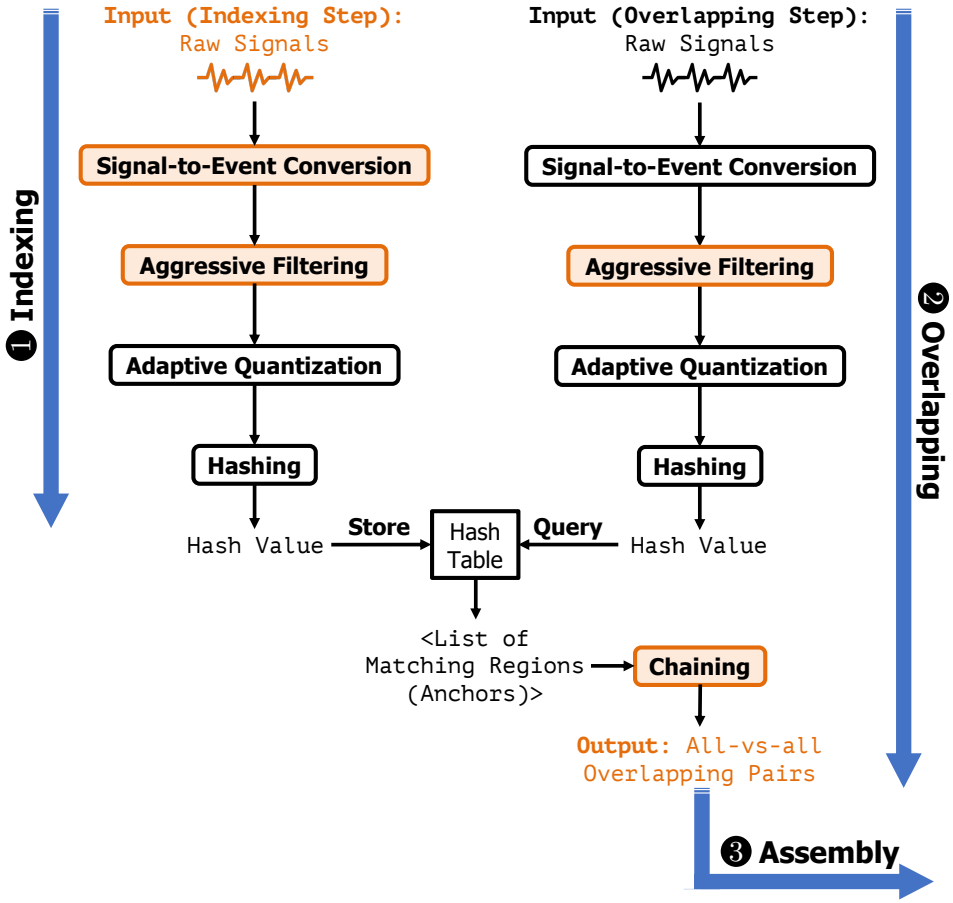

Figure 7.1: Overview of Rawsamble. We use colors for the inputs, steps, and outputs to highlight the parts that Rawsamble modifies over RawHash2.

7.2.2 Constructing an Index from Noisy Raw Signals via Aggressive Filtering

Rawsamble identifies overlapping regions between raw nanopore signals using a hash-based seeding mechanism that operates in two steps. First, to enable quick matching between raw signals, Rawsamble enables constructing an index directly from raw nanopore signals instead of using an existing reference genome sequence. While converting reference genomes to their expected raw signal values to store them in an index is mainly free from certain

types of noise that raw nanopore signals contain (e.g., stochastic signal fluctuations [229] and variable speed of DNA molecules moving through nanopores [823, 824]), noise in raw nanopore signals can cause challenges to find accurate matches between raw signals when these signals are stored in an index. Second, to reduce noise stored in the hash tables and enable accurate similarity identification when both signals are noisy, Rawsamble aggressively filters raw signals as shown in Figure 7.2. The filtering mechanism iteratively compares two adjacent signals, s_i and s_j, and removes the second signal, s_j, if the absolute difference between the two adjacent signals is below a certain threshold T. This filtering generates a list of filtered signals that aggressively aims to reduce the impact of stay errors during sequencing. Although similar filtering approaches are used in prior works [265, 266, 291] to reduce the stay errors (e.g., by aiming to perform homopolymer compression of raw signals [299]), Rawsamble employs a substantially larger threshold (i.e., aggressive) for filtering to minimize noise both in the index and during mapping. By applying the aggressive filtering technique, Rawsamble ensures that only high-quality, informative events are used in indexing to improve the accuracy and efficiency of the overall overlapping mechanism when both signals are noisy.

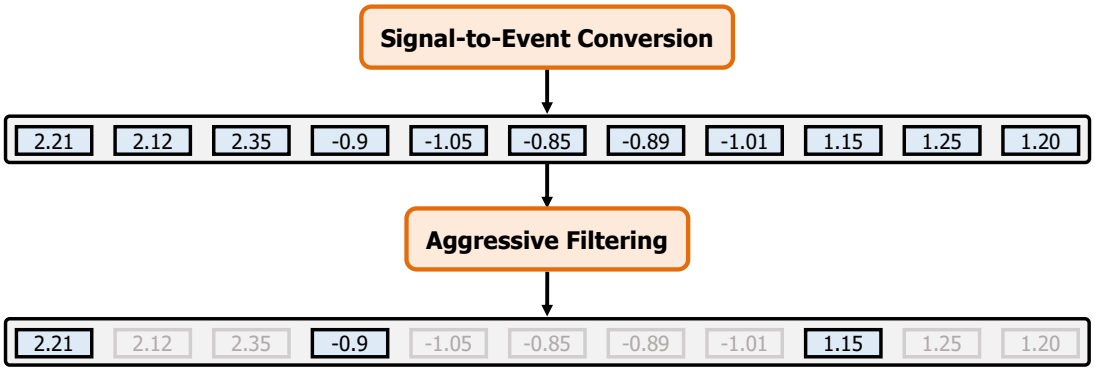

Figure 7.2: Filtering in Rawsamble. Values in gray boxes show the filtered signals as their values are close to the previous signal (in a blue box) that is not filtered out.

7.2.3 Adjusting the Chaining Mechanism for Overlapping

To reduce the number of chains that do not result in mapping (i.e., false chains) and construct longer chains, Rawsamble adjusts the chaining mechanism [265, 306] in two ways. First, Rawsamble constructs chains between seed matches (i.e., anchors) with longer gaps by adjusting the maximum gap length between anchors. This adjustment is needed because the filtering mechanism results in a sparser list with potentially long gaps between signals. Second, to ensure that only high-confidence chains are considered as overlaps among many pairwise overlapping regions, Rawsamble sets a higher minimum chaining score for overlapping than mapping, which effectively filters out spurious matches. These adjustments to the chaining mechanism enable Rawsamble to construct long and accurate chains between noisy and sparse

raw signals, improving the overall sensitivity of the overlapping process and further downstream analysis such as *de novo* assembly construction.

7.2.4 Adjusting the Mapping Strategy

Rawsamble adjusts the mapping strategy used in RawHash2 in two ways. First, Rawsamble generates pairwise mappings of a read to many reads (instead of a single mapping per read). To achieve this, Rawsamble identifies all valid chains based on their chaining scores and reports all such chains between raw signals. This enables overlapping a single raw signal with multiple other raw signals. Second, Rawsamble filters out *trivial* cyclic overlaps between two reads. To do so, Rawsamble avoids reporting overlapping signal pairs both as *query* and *target* in the mapping output, which can complicate the *de novo* assembly construction process [305, 440, 1054]. To avoid these trivial cyclic overlaps, Rawsamble uses a pre-defined and deterministic ordering between raw signals (e.g., based on the lexicographic ordering of read names). By comparing raw signals deterministically, Rawsamble guarantees that only one of each pair of overlapping reads is processed as a query sequence while the other is treated as a target sequence. Reporting all chains while avoiding cyclic overlaps between raw signals allows for a comprehensive representation of the overlapping regions, which is useful for constructing accurate and long assemblies.

7.2.5 Signal Assembly from Overlaps

Rawsamble provides the overlapping information between raw signals using the Pairwise mApping Format (PAF) [305], which can be used by existing assemblers like miniasm [305] to construct an assembly as an assembly graph for output in Graphical Fragment Assembly (GFA) [305, 1055] format.

To output the sequence of assembled signals directly from identified overlaps along with the assembly graph in GFA format, Rawsamble provides a new assembler, Rawasm [1056]. This assembler can be useful for facilitating complete *de novo* assembly without requiring basecalling or to aid the basecalling process in several ways, as we discuss in Section 7.4.

Rawasm is implemented as a small extension to miniasm, introducing three key modifications to enable outputting the assembled signals. First, to read raw signals and store the assembled signals in standard raw signal file formats, Rawasm supports all major raw signal file formats: FAST5, POD5, S/BLOW5 [1051]. Second, to improve memory efficiency and reduce the significant I/O burden associated with reading and writing raw signal files, which are typically larger and stored in complex compressed formats compared to basecalled sequences, Rawasm implements several optimizations. Unlike miniasm, which keeps all unitigs in main memory until the GFA file is written, Rawasm overlaps the graph building and writing processes. When the signals for a unitig are trimmed for outputting, and their positions within the unitig are

calculated, Rawasm updates the output and frees up the corresponding memory. This approach minimizes memory usage and I/O delays to improve efficiency and reduce I/O overhead. Third, to output the assembled signals without losing the additional important metadata associated with raw signals, Rawasm generates a new raw signal file for each unitig in the assembly graph by concatenating the original signals that construct the corresponding unitig. The concatenation mechanism trims and arranges the signals based on the start and end positions used for constructing the unitig as provided by the assembly graph, similar to miniasm. Since each raw signal may contain different metadata (e.g., signal variables determined based on the characteristics of a nanopore channel), Rawasm keeps this metadata associated with each raw signal.

The overlapping information that Rawsamble generates can be used by any assembler that takes Pairwise mApping Format (PAF) [305] as input, including Rawasm and miniasm [305]. Although any such assembler can construct assemblies from PAF and output the resulting assembly information in GFA, Rawasm is the first implementation to enable outputting the resulting assembly signals directly after from the resulting GFA.

7.3 Results

7.3.1 Evaluation Methodology

We implement the improvements we propose in Rawsamble on the RawHash2 implementation [265]. Similar to RawHash2 and minimap2 [306], Rawsamble provides the mapping information in the standard Pairwise mApping Format (PAF) [305]. We basecall the signals on two different hardware setups using Dorado's 1) high accuracy (HAC) and super high accuracy (SUP) models with an NVIDIA RTX A6000 GPU [1057], and 2) fast (Fast) and high accuracy (HAC) models with an Intel Xeon Gold 6226R CPU [1058]. We use Dorado's 1) Fast model on a CPU to show the best performance (i.e., speed) that a CPU-based basecalling can provide and 2) SUP model on a GPU to show the best accuracy that basecalling provides along with its associated increased computational cost. We use POD5 files for all datasets, as suggested by Dorado [272] for optimal performance. Since no prior works can overlap reads using raw signals, we compare Rawsamble to minimap2, the current state-of-the-art read overlapper for basecalled sequences using the datasets shown in Table 7.1. We use datasets both with high sequencing depth of coverage (D1 to D3) and low coverage (D4 and D5) to evaluate the capability to construct assemblies in both scenarios.

We evaluate computational requirements in terms of overall run time with 32 CPU threads and peak memory usage. We use 32 CPU threads since the average thread utilization of CPU-based basecalling is around 32. We report the elapsed time, CPU time (when using only CPUs without GPUs), and peak memory usage of 1) minimap2 with and without basecalling and

Table 7.1: Details of datasets used in our evaluation.

	Organism	Device Type	Reads (#)	Bases (#)	Avg. Read Length	Estimated Coverage (×)	SRA Accession
D1	*SARS-CoV-2*	MinION	10,001	4.02M	402	135×	CADDE Centre
D2	*E. coli*	GridION	353,948	2,332M	6,588	445×	ERR9127551
D3	*Yeast*	MinION	50,023	385M	7,698	32×	SRR8648503
D4	*Green Algae*	PromethION	30,012	622M	20,731	5.6×	ERR3237140
D5	*Human*	MinION	270,006	1,773M	6,567	0.6×	FAB42260

Base counts in millions (M). Coverage is estimated using corresponding reference genomes.
All datasets are generated using ONT's R9.4.1 e8 flow cells (450 bp/sec translocation speed with 4000 signals/sec sampling frequency).
All of the datasets are basecalled using Dorado (HAC) v3.3.

2) Rawsamble. For Rawsamble, we additionally report throughput (average number of signals analyzed per second per single CPU thread) to provide insights about its capabilities for real-time analysis. To measure performance and peak memory usage, we use the `time -v` command in Linux when running Rawsamble, minimap2, and Dorado. For average speedup and memory comparisons of Rawsamble against other methods, we use the geometric mean to reduce the impact of outlier data points on the average calculation.

We evaluate Rawsamble based on two use cases: 1) read overlapping and 2) *de novo* assembly. We generate read overlaps from 1) raw signals using Rawsamble and 2) basecalled sequences (with Dorado's HAC model) of corresponding strands of raw signals using minimap2. To evaluate all-vs-all overlapping, we calculate the ratio of overlapping pairs 1) shared by both Rawsamble and minimap2, 2) unique to Rawsamble, and 3) unique to minimap2. For *de novo* assembly, we use miniasm [305] to assemble the read overlaps that Rawsamble and minimap2 generate in a PAF file. We use miniasm because it enables us to evaluate the impact of overlaps that Rawsamble and minimap2 finds by *using the same assembler* for both of them. To identify the roofline in terms of the assembly contiguity given a dataset, we use a highly accurate assembler as *gold standard* for our evaluations. To do this, we use Dorado's most accurate model (i.e., SUP) to construct highly accurate reads and use Flye [514] to construct assemblies from these reads, as suggested in the guidelines by ONT [1059]. This approach enables us to evaluate the gap between the assemblies that Rawsamble generates and the golden standard assembly that can be constructed from the same datasets.

To evaluate the assemblies constructed from the overlaps that Rawsamble and minimap2 generate, we use several metrics that are mainly related to the contiguity of the assemblies. These metrics are 1) the total length of unitigs (Total Length), 2) the number of nucleotides in the largest assembly graph component (Largest Comp.), 3) unitig length at which 50% of the assembly is covered by unitigs of equal or greater length (N50), 4) area under the Nx curve

to generate a more robust evaluation of contiguity (auN) [1060], 5) longest unitig length, and 6) the number of unitigs. We use Bandage [1061] to visualize these assemblies.

We provide the parameter settings and versions for each tool as well as the details of the preset parameters in Rawsamble in Supplementary Tables S7.2 (parameters), S7.3 (details of presets), and S7.4 (versions). We provide the scripts to fully reproduce our results on the GitHub repository at https://github.com/CMU-SAFARI/RawHash, which contains the corresponding release version of Rawsamble we show in Supplementary Table S7.4. We provide the detailed reasoning behind the choice of Flye in order to generate the gold standard in Supplementary Material S7.

7.3.2 Performance and Memory

Throughput. Table 7.2 shows the throughput (i.e., signals processed per second per CPU thread) that Rawsamble reports. Our goal is to estimate the number of CPU threads needed to achieve a throughput faster than the throughput of a single sequencer. Using as few CPU threads as possible is useful to 1) provide better scalability (i.e., analyzing a larger amount of data with the same computation capabilities) and 2) reduce the overall computational requirements and corresponding energy consumption (i.e., analyzing the same amount of data with less computational capabilities). The latter is especially essential for resource-constrained devices (e.g., devices with external and limited batteries). The throughput of a single nanopore is around 5,000 signals per second [272], and the entire sequencer is usually equipped with 512 nanopores. This means a single sequencer's throughput is around 2,560,000 signals per second (5,000×512) [1062]. We find that Rawsamble provides an average throughput of 2,432,561 signals per second **per CPU thread**. This means Rawsamble can achieve an analysis throughput faster than a single sequencer's throughput by using an average of two CPU threads. This shows that the overlapping mechanism of Rawsamble is fast enough for performing real-time overlapping tasks using very few CPU threads.

Table 7.2: Rawsamble Throughput (signals processed per second per CPU thread).

	D1 *SARS-CoV-2*	D2 *E. coli*	D3 *Yeast*	D4 *Green Algae*	D5 *Human*
Throughput	2,065,764	2,720,702	2,128,800	1,668,065	3,579,472

Computational resources. Table 7.3 shows the computational resources of the different approaches. Inside the parentheses provided with the minimap2 results, we provide the ratio between the reported result and the corresponding Rawsamble result (if higher than 1×, Rawsamble is better).

Table 7.3: Comparison of various tools across different organisms in terms of elapsed time, CPU time, and peak memory usage. The values in parentheses represent the ratio of the result shown in the cell compared to the corresponding result of Rawsamble (values higher than 1× indicate that Rawsamble performs better). We highlight the cells that Rawsamble provide a better and worse results with colors.

Organism	Tool	Elapsed time (hh:mm:ss)	CPU time (sec)	Peak Mem. (GB)
D1	**Rawsamble**	0:00:03	33	1.07
SARS-CoV-2	**Minimap2**	0:00:01 (0.33×)	19 (0.58×)	0.16 (0.15×)
	Minimap2 + Dorado CPU (Fast)	0:01:45 (35.00×)	3,227 (97.79×)	44.93 (41.99×)
	Minimap2 + Dorado CPU (HAC)	0:05:45 (115.00×)	5,457 (165.36×)	57.98 (54.19×)
	Minimap2 + Dorado GPU (HAC)	0:01:41 (33.67×)	NA	0.8 (0.75×)
	Minimap2 + Dorado GPU (SUP)	0:25:47 (515.67×)	NA	1.23 (1.15×)
D2	**Rawsamble**	1:12:44	132,758	6.72
E. coli	**Minimap2**	0:14:25 (0.20×)	25,721 (0.19×)	26.73 (3.98×)
	Minimap2 + Dorado CPU (Fast)	7:17:05 (6.01×)	583,358 (4.39×)	50.43 (7.50×)
	Minimap2 + Dorado CPU (HAC)	32:26:12 (26.76×)	1,335,697 (10.06×)	38.0 (5.65×)
	Minimap2 + Dorado GPU (HAC)	0:36:14 (0.50×)	NA	26.73 (3.98×)
	Minimap2 + Dorado GPU (SUP)	1:30:30 (1.24×)	NA	26.73 (3.98×)
D3	**Rawsamble**	0:01:18	2,241	6.39
Yeast	**Minimap2**	0:00:21 (0.27×)	290 (0.13×)	5.25 (0.82×)
	Minimap2 + Dorado CPU (Fast)	0:54:04 (41.59×)	71,796 (32.04×)	56.13 (8.78×)
	Minimap2 + Dorado CPU (HAC)	3:13:56 (149.18×)	193,640 (86.41×)	65.43 (10.24×)
	Minimap2 + Dorado GPU (HAC)	0:04:33 (3.50×)	NA	5.25 (0.82×)
	Minimap2 + Dorado GPU (SUP)	0:10:33 (8.12×)	NA	5.92 (0.93×)
D4	**Rawsamble**	0:07:57	14,064	8.67
Green Algae	**Minimap2**	0:00:47 (0.10×)	882 (0.06×)	8.7 (1.00×)
	Minimap2 + Dorado CPU (Fast)	1:16:35 (9.63×)	79,606 (5.66×)	50.88 (5.87×)
	Minimap2 + Dorado CPU (HAC)	4:30:07 (33.98×)	286,362 (20.36×)	64.07 (7.39×)
	Minimap2 + Dorado GPU (HAC)	0:06:01 (0.76×)	NA	8.7 (1.00×)
	Minimap2 + Dorado GPU (SUP)	0:14:54 (1.87×)	NA	8.7 (1.00×)
D5	**Rawsamble**	0:28:56	51,975	6.0
Human	**Minimap2**	0:01:52 (0.06×)	1,372 (0.03×)	20.21 (3.37×)
	Minimap2 + Dorado CPU (Fast)	6:42:24 (13.91×)	802,983 (15.45×)	81.98 (13.66×)
	Minimap2 + Dorado CPU (HAC)	23:27:18 (48.64×)	1,219,043 (23.45×)	46.12 (7.69×)
	Minimap2 + Dorado GPU (HAC)	0:20:24 (0.71×)	NA	20.31 (3.38×)
	Minimap2 + Dorado GPU (SUP)	1:05:48 (2.27×)	NA	20.21 (3.37×)

We make three key observations. First, Rawsamble provides a substantial speedup and lower peak memory usage compared to minimap2 when combined with Dorado's fast model on a CPU by, on average, 16.36× (elapsed time), 16.45× (CPU time), and 11.73× (peak memory usage). When using Dorado's HAC model, Rawsamble provides even better results on average by 59.70× (elapsed time), 36.93× (CPU time), and 11.23× (peak memory usage), since running the HAC model is computationally more costly than running the fast model.

Second, Rawsamble achieves an average: 1) speedup of 1.99× (HAC) and 7.40× (SUP), and 2) reduced peak memory usage of 1.53× (HAC) and 1.70× (SUP), compared to GPU-based basecalling followed by minimap2. Although basecalling with GPUs followed by overlapping is usually faster in most cases when using the HAC model (**Minimap2 + Dorado GPU (HAC)**

in Table 7.3), Rawsamble still achieves better average speedup due to the substantial speedup it provides for the D1 dataset. Although comparing the results between CPUs and GPUs is not ideal due to the massive parallelism that GPUs provide compared to CPUs, Rawsamble still provides comparable and even better performance even with the limited parallelism (i.e., 32 threads) it uses on CPUs compared to the parallelism that GPUs provide (e.g., a few tens of thousands of threads). This shows that Rawsamble can provide even faster results when accelerated with GPUs (outside the scope of this work) based on its comparison when basecalling is done using CPUs and GPUs.

Third, minimap2, without considering the resources that basecalling requires, is more resource-efficient than Rawsamble as it takes on average 0.16× (elapsed time), 0.12× (CPU time), and 1.11× (peak memory usage) of those Rawsamble takes. This is mainly expected as analyzing raw signals requires additional steps (e.g., signal-to-event conversion), more memory due to sequencing depth (as opposed to a reference genome), and incurs noise that requires less efficient parameter settings for Rawsamble (e.g., the window parameter in minimizers) when analyzing raw signals compared to minimap2. Future work can focus on processing these raw signals more accurately to reduce the limits for adjusting parameters that can impact performance and memory usage.

We conclude that Rawsamble can perform read overlapping with higher throughput that can be useful when focusing on real-time analysis and better computational resources than CPU-based basecalling followed by minimap2. We find that Rawsamble's speed is mainly dependent on the amount of bases stored in the hash table, as the speed decreases with increasing the number of locations that need to be analyzed in the index per read. To resolve this scalability issue, future work should focus on designing indexing and filtering methods that provide a limitation on the volume of signals stored in the index and processed during the overlapping step. These results can also be useful when basecalling raw signals using GPUs to reduce the computational overhead that GPU-based basecalling requires, which we discuss in Section 7.4.

7.3.3 All-vs-All Overlapping Statistics

Table 7.4 shows the all-vs-all overlapping statistics between Rawsamble and minimap2, where each row sums to 100%. We make two key observations. First, on average, 36.57% of overlap pairs generated by Rawsamble are shared with the overlap pairs that minimap2 generates. This shows that a large fraction of overlapping pairs does not require basecalling to generate identical overlapping information identified by minimap2 after basecalling. Second, although Rawsamble can find a substantial amount of overlap pairs identical to the overlaps minimap2 finds, specifically for the D1 dataset, there are still overlapping pairs unique to Rawsamble (16.25% on average) and minimap2 (47.17% on average), mainly due to the decreased

shared overlaps as the genome size increases. These differences are likely due to differences in 1) certain parameters (e.g., chaining scores) and 2) increased noise inherent in raw signals compared to basecalled sequences, which can become more pronounced in genomes with greater size, complexity, or repetitiveness. Additionally, as observed in an earlier work [523], constructing contiguous assemblies is not dependent on finding as many overlapping pairs as possible. Instead, contiguous assemblies can still be constructed using a smaller but useful portion of overlapping pairs. We conclude that Rawsamble provides a mechanism that shares a large portion of overlaps with minimap2, while the shared portions decrease as the genome size increases.

Table 7.4: All-vs-all Overlapping Statistics. Percentages show the overlapping pairs that are 1) unique to Rawsamble, 2) unique to minimap2, and 3) reported in both tools (Shared Overlaps).

	Organism	Unique to Rawsamble (%)	Unique to Minimap2 (%)	Shared Overlaps (%)
D1	*SARS-CoV-2*	11.55	15.27	73.18
D2	*E. coli*	8.33	50.62	41.05
D3	*Yeast*	24.94	35.17	39.89
D4	*Green Algae*	3.76	78.64	17.61
D5	*Human*	32.69	56.18	11.13

7.3.4 *De novo* Assembly Evaluations

To evaluate the impact of differing reported overlaps between Rawsamble and minimap2, we construct *de novo* assemblies from these overlaps. Table 7.5 shows the statistics of these *de novo* assemblies. We do not provide the assembly statistics for the D1 dataset as miniasm cannot generate an assembly using the overlaps from Rawsamble and minimap2, potentially due to the small size of the genome. We make three observations.

First, we find that the overlaps found using Rawsamble can form long paths during assembly that lead to assembly constructions and large connected components. **Rawsamble is the first work that enables *de novo* assembly construction directly from raw signal overlaps without basecalling**, which has several implications and can enable future work, as we discuss in Section 7.4.

Second, we observe that the unitigs generated from Rawsamble overlaps are usually less contiguous compared to those generated from minimap2 overlaps, based on all the metrics we show in Table 7.5 and Supplementary Figures S7.1–S7.4. Compared to the gold standard assemblies generated using highly accurate basecalled reads and a state-of-the-art assembler, we find that Rawsamble can still achieve a significant portion of the assembly contiguity, especially

for the D2 dataset (*E.coli*). For example, Rawsamble constructs unitigs with the longest unitig length of 2.7 million bases, which is over half the length of the *E. coli* genome, whereas the gold standard assembly produces a single unitig covering the entire genome. **This indicates that Rawsamble can achieve substantial contiguity relative to the gold standard without basecalling.**

Third, for the larger genomes in the D4 and D5 datasets, we find that Rawsamble can construct longer unitigs compared to those generated by minimap2 and, notably, achieves better contiguity than the gold standard assemblies in the D5 dataset. The coverage of these datasets is substantially lower than that of other datasets. This suggests that Rawsamble can find more useful overlaps leading to longer paths in the assembly for the cases when coverage is relatively lower (e.g., coverage of less than or around 5× in Table 7.1). Basecalled sequences may generate more contiguous assemblies when the coverage is higher. We conclude that Rawsamble generates useful overlapping information, enabling the construction of assemblies directly from raw signals. This approach achieves substantial contiguity and, in some cases (e.g., at low coverage), provides superior contiguity compared to minimap2 and gold standard assemblies.

Table 7.5: Assembly Statistics.

Dataset	Tool	Total Length (bp)	Largest Comp. (bp)	N50 (bp)	auN (bp)	Longest Unitig (bp)	Unitig Count
D2	Rawsamble	14,525,505	4,841,669	1,535,079	1,309,738	2,722,499	31
E. coli	minimap2	10,434,542	5,207,206	5,204,754	5,194,738	5,207,206	4
	Gold standard	5,235,343	5,235,343	5,235,343	5,235,343	5,235,343	1
D3	Rawsamble	13,898,208	362,050	41,118	48,106	161,883	396
Yeast	minimap2	23,755,455	1,611,876	134,050	150,908	464,054	282
	Gold standard	11,963,521	11,835,059	640,934	623,210	1,073,346	68
D4	Rawsamble	3,448,899	448,422	93,111	108,818	252,038	50
Green Algae	minimap2	2,117,190	198,709	63,310	88,906	198,709	55
	Gold standard	106,479,288	2,255,807	452,774	538,136	1,667,975	420
D5	Rawsamble	1,850,419	493,004	51,300	116,049	364,113	48
Human	minimap2	747,607	65,951	19,476	22,103	48,424	61
	Gold standard	8,365,210	367,305	19,329	29,697	150,470	592

7.4 Discussion and Future Work

Limitations. Our evaluation demonstrates that Rawsamble achieves high throughput, making it a viable candidate for real-time analysis while constructing *de novo* assemblies. However, there are still two main challenges to fully utilize real-time *de novo* assembly construction during sequencing. First, the hash-based index should be constructed and updated dynamically in real-time while sequencing is in progress. Although Rawsamble provides the mechanisms for

storing multiple hash tables that can be constructed for each chunk of raw signals generated in real-time, it is not computationally feasible to dynamically update the index for all sequenced signals as the memory and computational burden increases with each hash table generated. Such an approach requires a decision-making mechanism to dynamically stop updating the index after sequencing a certain amount of signals. The stopping mechanism should be accurate enough to ensure that the current state of the index provides sufficient information to find the overlaps between the already sequenced signals and the new signals generated after the index construction. Second, the assembly graph should be constructed and updated dynamically while the new overlap information is generated in real time. This is challenging as the intermediate steps for generating the unitigs (e.g., the transitive reduction step [532]) in assembly graphs may not work optimally without the full overlap information, as it is likely to remove graph connections that can be useful with new overlap information. It might be feasible to use graph structures that are more suitable for streaming data generation, such as de Bruijn graphs, for assembly construction purposes [1063–1065].

Rawsamble can find overlaps only between reads coming from the same strand, as it lacks the capability to construct the reverse complemented version of the signals to identify the matches on the other strands. This potentially leads to gaps in the assembly and construction of the unitigs from both strands. Designing a mechanism that can reverse complement raw signals without basecalling is future work.

We evaluate Rawsamble using raw signals generated from ONT's R9.4 flowcells to show that overlapping information between raw signals can be used to construct *de novo* assemblies. We leave optimizations for newer flow cell versions (e.g., R10.4.1) for future work.

New Directions for Future Work. We identify several new directions for future work. First, integrating the overlapping information with basecalling can provide accuracy benefits for basecallers. Existing basecallers are designed to basecall individual reads without such overlapping information. Such a combined approach can provide additional useful features for the underlying machine learning models in basecallers to improve the basecalling accuracy. Second, *de novo* assembly construction enables identifying the reads that do not provide useful information for assembly if they are fully contained by other overlapping read pairs [305]. Identifying these potentially useless reads early on provides the opportunity to avoid basecalling them, which can improve the overall execution time of basecalling by filtering out such reads. Third, *de novo* assembly construction from raw signals provides new opportunities to design new basecallers that can basecall the assembled signals. Such a basecaller has the potential to substantially improve the performance, as it enables basecalling fewer long unitigs instead of many shorter reads.

While Rawsamble enables new directions in raw signal analysis, particularly for *de novo*

assembly, several challenges and opportunities for future work remain. Addressing these challenges has the potential to enable new use cases and applications in raw signal and basecalled sequence analyses.

7.5 Summary

We introduce Rawsamble, the *first* mechanism that can find overlaps between two sets of raw nanopore signals without translating them to bases. We find that Rawsamble can 1) find overlaps while meeting the real-time requirements with an average throughput of 2,432,561 signals/sec per CPU thread, 2) reduce the overall time needed for finding overlaps (on average by 16.36× and up to by 41.59×) and peak memory usage (on average by 11.73× and up to by 41.99×) compared to the time and memory needed to run state-of-the-art read mapper (minimap2) combined with the Dorado basecaller running on a CPU with its fastest model, 3) share a large portion of overlapping pairs with minimap2 (36.57% on average), and 4) construct long assemblies from these useful overlaps. We find that we can construct assemblies of half the length of the entire *E. coli* genome (i.e., of length 2.7 million bases) directly from raw signal overlaps without basecalling. Finding overlapping pairs from raw signals is critical for enabling new directions that have not been explored before for raw signal analysis, such as *de novo* assembly construction from overlaps that we explore in this work. We discuss many other new directions that can be enabled by finding overlaps and constructing *de novo* assemblies.

We hope and believe that Rawsamble enables future work in at least two key directions. First, we aim to fully perform end-to-end genome analysis without basecalling. This can be achieved by further improving tools such as Rawsamble to construct *de novo* assemblies directly from raw signals such that these assemblies are consistently better than those generated from basecalled sequences in all cases. Second, we should rethink how we train and use modern neural network-based basecallers by integrating additional useful information (e.g., overlaps or assemblies of signals) generated by Rawsamble into these basecallers.

S7 Supplementary Materials

S7.1 Visualizing Assemblies

Supplementary Figures S7.1, S7.2, S7.3, and S7.4 show the assembly graphs created with miniasm in GFA formats. To construct these assembly graphs with miniasm, we provide the overlap information that Rawsamble and minimap2 provide as a PAF file [305]. We use Bandage [1061] to visualize these assembly graphs. To provide the scale between unitigs generated using the same dataset, we annotate the longest contig in each assembly graph with their lengths.

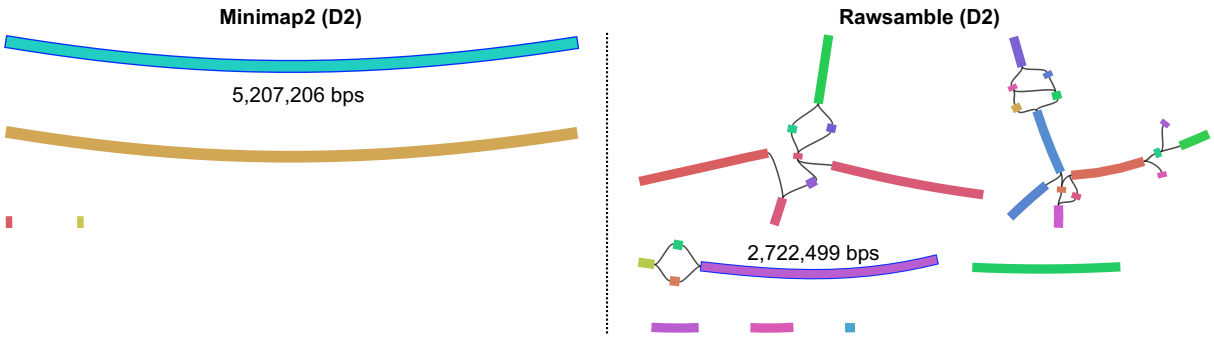

Figure S7.1: Visualization of the assembly graphs generated using Rawsamble and minimap2 overlaps from the D2 (*E. coli*) dataset.

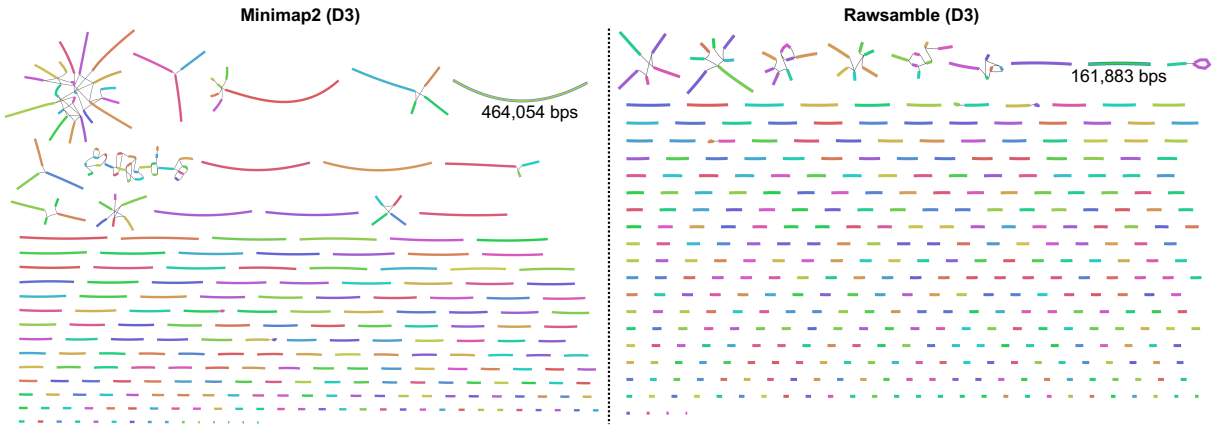

Figure S7.2: Visualization of the assembly graphs generated using Rawsamble and minimap2 overlaps from the D3 (*Yeast*) dataset.

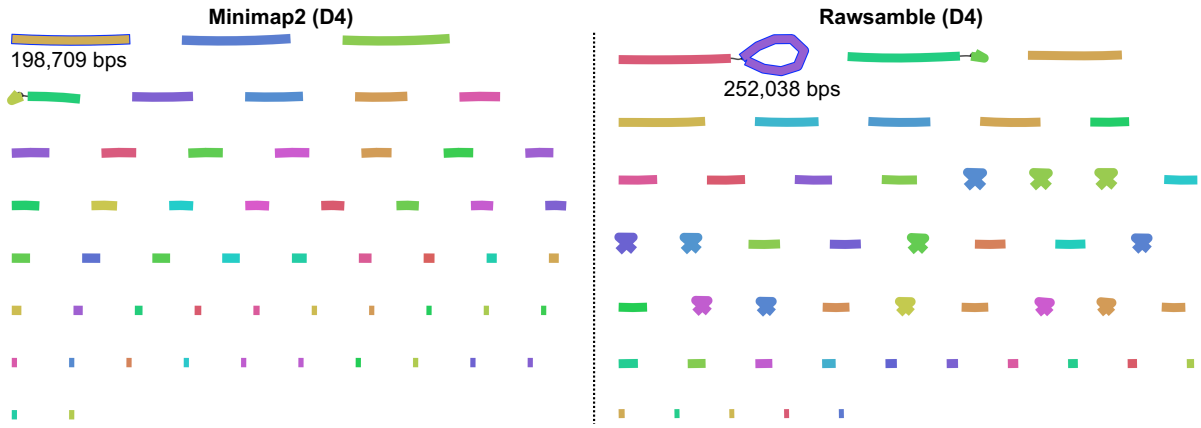

Figure S7.3: Visualization of the assembly graphs generated using Rawsamble and minimap2 overlaps from the D4 (*Green Algae*) dataset.

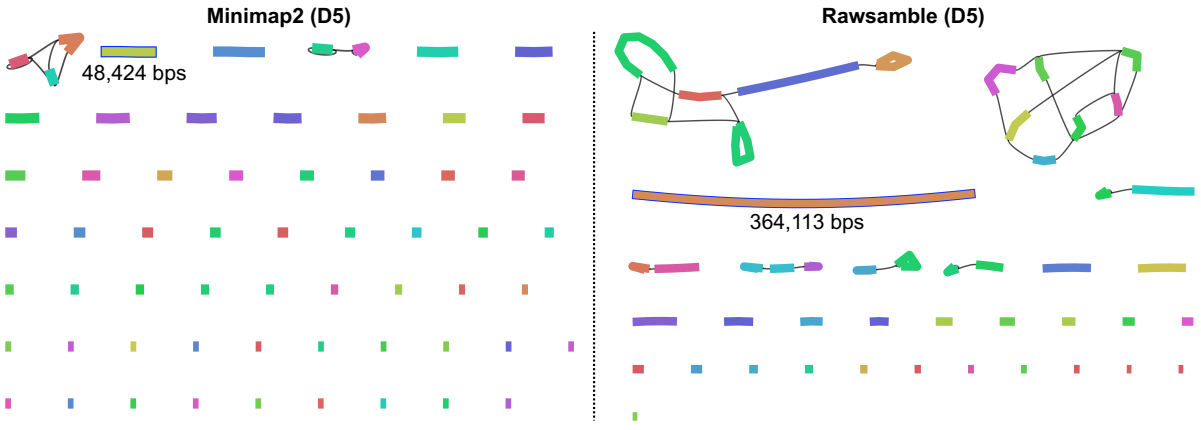

Figure S7.4: Visualization of the assembly graphs generated using Rawsamble and minimap2 overlaps from the D5 (*Human*) dataset.

S7 Generating a Gold Standard Assembly

To generate a gold standard assembly using R9.4 Simplex reads, we explore two approaches.

First, we use the raw basecalled sequences directly with a set of state-of-the-art assemblers identified from recent benchmarks [1066–1069], namely Hifiasm [505], Verkko [519], LJA [517], HiCanu [507], and Flye [514].

Second, we apply error correction to the reads using the HERRO tool [866] developed by Oxford Nanopore Technologies (ONT), which is also integrated into the latest versions of Dorado as *dorado correct*. HERRO corrects erroneous R9.4 and R10.4 data, enabling their use as a replacement for accurate PacBio HiFi reads required by state-of-the-art hybrid assembly approaches. Supplementary Table S7.1 shows the estimated sequencing depth of coverage before and after the HERRO correction for each dataset.

For the high-coverage D2 *E.coli* dataset, Hifiasm outputs a fragmented assembly, while Verkko requires extensive parameter tuning (specifically in terms of coverage, `–unitig-abundance`, and `–base-k`) to achieve desirable contiguity and completeness. LJA produces an almost perfect assembly when a minimum length filter of 30kbp is applied to the corrected reads, as suggested by the HERRO paper. Flye performs comparably to LJA without the need for error correction.

For the other datasets, with or without error correction, most assemblers either fail to produce assemblies or generate suboptimal results, except for Flye. We attribute this to their sensitivity to inaccuracies in the uncorrected reads and the reduced coverage after correction. Supplementary Table S7.1 shows how the coverage levels change after correction for each dataset. Notably, Flye works well (and sometimes even slightly better) with uncorrected reads, highlighting its robustness to noisy ONT reads.

Based on these observations, we select Flye as the assembler to generate the gold standard assemblies in our evaluations, given its ability to handle noisy ONT reads without the need for error correction and its consistent performance across different datasets. Therefore, we use Flye to construct the gold standard assemblies from the basecalled reads in our study.

Table S7.1: Coverage levels before and after error correction for each dataset.

Dataset	Coverage Before Correction	Coverage After Correction
D2 *E. coli*	445×	240×
D3 *Yeast*	32×	12×
D4 *Green algae*	5.6×	3.7×
D5 *Human*	0.6×	0.002×

S7.1 Configuration

S7.1.1 Parameters

In Supplementary Table S7.2, we show the parameters of each tool for each dataset. In Supplementary Table S7.3, we show the details of the preset values that Rawsamble sets in Supplementary Table S7.2. For minimap2 [306], we use the same parameter setting for all datasets. For miniasm [305], we use the default parameter settings for all datasets.

Table S7.2: Parameters we use in our evaluation for each tool and dataset in mapping.

Tool	D1 *SARS-CoV-2*	D2 *E. coli*	D3 *Yeast*	D4 *Green Algae*	D5 *Human*
Rawsamble	-x ava-viral -t 32	-x ava -t 32	-x ava -t 32	-x ava -t 32	-x ava –chain-gap-scale 0.6 -t 32
Minimap2	-x ava-ont –for-only -t 32				
Dorado CPU (Fast)	basecaller -x cpu dna_r9.4.1_e8_fast@v3.4				
Dorado CPU (HAC)	basecaller -x cpu dna_r9.4.1_e8_hac@v3.3				
Dorado GPU (HAC)	basecaller dna_r9.4.1_e8_hac@v3.3				
Dorado GPU (SUP)	basecaller dna_r9.4.1_e8_sup@v3.3				
Miniasm					

Table S7.3: Corresponding parameters of presets (-x) in Rawsamble.

Preset	Corresponding parameters	Usage
ava-viral	-e 6 -q 4 -w 0 –sig-diff 0.45 –fine-range 0.4 –min-score 20 –min-score2 30 –min-anchors 5 –min-mapq 5 –bw 1000 –max-target-gap 2500 –max-query-gap 2500 –chain-gap-scale 1.2 –chain-skip-scale 0.3	Viral genomes
ava	-e 8 -q 4 -w 3 –sig-diff 0.45 –fine-range 0.4 –min-score 40 –min-score2 75 –min-anchors 5 –min-mapq 5 –bw 5000 –max-target-gap 2500 –max-query-gap 2500	Default case

S7.1.2 Versions

Supplementary Table S7.4 shows the version and the link to these corresponding versions of each tool.

Table S7.4: Versions of each tool and library.

Tool	Version	Link to the Source Code
Rawsamble	2.1	https://github.com/CMU-SAFARI/RawHash/releases/tag/v2.1
Minimap2	2.24	https://github.com/lh3/minimap2/releases/tag/v2.24
Dorado	0.7.3	https://github.com/nanoporetech/dorado/releases/tag/v0.7.3
Miniasm	0.3-r179	https://github.com/lh3/miniasm/releases/tag/v0.3
Rawasm	main	https://github.com/CMU-SAFARI/rawasm
Flye	2.9.5	https://github.com/mikolmogorov/Flye/releases/tag/2.9.5
HERRO	0.1	https://github.com/lbcb-sci/herro

Chapter 8

Conclusions and Future Directions

In summary, the goal of this dissertation is twofold: 1) understand how various sources of noise in sequencing data and its analysis impact the performance, accuracy, efficiency and scalability of genome analysis; and 2) develop new techniques to tolerate or reduce noise for faster, more accurate, and real-time analysis of genomic sequencing data. To this end, we propose a series of novel algorithms and methods that improve the efficiency and effectiveness of key steps in the genome analysis pipeline.

First, we introduce BLEND (Chapter 4), a novel noise-tolerant hashing mechanism for fuzzy seed matching, enabling quick identification of highly similar genomic sequences. Our evaluations show that BLEND significantly improves the performance and memory usage of read overlapping and mapping, providing substantial benefits for assembling and analyzing high-coverage genomic data.

Second, we develop RawHash (Chapter 5), the first mechanism that performs accurate and scalable real-time analysis of raw nanopore signals for large genomes. RawHash uses a unique quantization technique that effectively reduces noise in raw signals, enabling a hash-based search mechanism that bypasses the computationally intensive basecalling step. Our evaluations demonstrate that RawHash matches the throughput of nanopore sequencers while providing high accuracy and scalability for large genomic datasets.

Third, we propose RawHash2 (Chapter 6), an improved version of RawHash that provides better noise reduction in raw nanopore signals and increases the robustness of mapping decisions. RawHash2 incorporates more sensitive quantization and chaining algorithms, weighted mapping decisions, and frequency filters, which collectively improve both the accuracy and throughput of real-time mapping of raw nanopore signals.

Fourth, we introduce Rawsamble (Chapter 7), a novel mechanism that enables all-vs-all overlapping of raw nanopore signals, facilitating the construction of *de novo* assemblies directly from raw signals without basecalling. Rawsamble achieves significant speedups and reductions in memory usage compared to traditional basecalling and overlapping pipelines, paving the way for new applications in raw signal analysis.

Finally, based on the insights developed throughout this work, we emphasize the importance of integrating noise-tolerant techniques and real-time analysis mechanisms in future genome analysis pipelines. We argue that these advancements can 1) improve the efficiency and accuracy of current methodologies and 2) enable new applications and directions in the field of genomics.

8.1 Future Research Directions

This dissertation opens up several promising avenues for future research. In this section, we review potential directions for extending and enhancing the work presented in this dissertation.

8.1.1 Integration of Overlapping and Assembly Information into Basecalling

Integrating overlapping and assembly information with basecalling is a potential direction for future work. By combining the overlapping and assembly information that Rawsamble generates with existing basecalling methods, it is possible to provide additional features to the machine learning models used in basecallers. This integration can improve the accuracy and performance of basecalling, especially in noisy datasets, by leveraging the contextual information provided by overlapping and assembly of raw signals. Enhancing basecallers with this additional context may lead to more accurate sequence interpretations and better overall genomic analyses.

8.1.2 Real-Time *De Novo* Assembly Construction

Building upon the capabilities of Rawsamble, future work can explore the *real-time* construction of *de novo* assemblies during sequencing. A real-time *de novo* assembly construction involves dynamically updating 1) the hash-based index and 2) assembly graph as new raw signals are sequenced. Enabling immediate identification and assembly of genomic sequences without waiting for the completion of sequencing can 1) reduce the overall time and computational resources required for assembly and 2) enable new directions for adaptive sampling. However, implementing real-time assembly poses challenges, such as managing data structures efficiently in real-time and handling the continuous stream of incoming data. Addressing these challenges can significantly accelerate genomic assembly processes and benefit time-sensitive applications.

8.1.3 Performing Further Steps Fully Using Raw Nanopore Signals Without Basecalling

Developing new algorithms and techniques for raw signal analysis is crucial to include many downstream steps of genome analysis without relying on basecalling. Developing techniques to directly process and interpret raw signals for applications such as variant calling, metagenomics, and functional genomics can reduce computational overhead and improve the speed of these analyses. By bypassing the basecalling step, it is possible to leverage the rich information contained in raw signals, potentially increasing accuracy and sensitivity in detecting genomic features. This approach is particularly beneficial in time-sensitive applications where rapid results are crucial.

8.1.4 Exploring Hardware Acceleration for Raw Signal Analysis

Given the large data volumes and massive parallelism in genome analysis, exploring hardware acceleration for raw signal analysis is an important direction. Implementing the algorithms used in raw signal analysis on suitable hardware, such as GPUs [300,301], FPGAs [138,297,302], or processing-in-memory (PIM) architectures [890,908,910,982], could significantly improve the performance of analysis while reducing the overall energy consumption. By leveraging specialized hardware, it is possible to achieve higher throughput and lower latency, which are critical for processing large volumes of genomic data efficiently, especially for portable sequencing devices where computational resources are limited.

8.1.5 Integrating BLEND in RawHash

Integrating the BLEND mechanism in RawHash is a potential avenue for improving the sensitivity and accuracy of raw nanopore signal analysis. By incorporating BLEND's fuzzy seed matching capabilities, RawHash can improve its ability to identify similar regions in noisy datasets. This integration may lead to better handling of the variability and noise inherent in raw signals. Additionally, it can reduce the computational cost associated with processing noisy data, making the analysis more efficient.

8.2 Concluding Remarks

In this dissertation, we address critical challenges in genome analysis by developing new techniques to tolerate and reduce noise in sequencing data. We build a detailed understanding of how noise affects the computational mechanisms in genome analysis and propose solutions that improve the speed, accuracy, and scalability of key steps in the genome analysis pipeline. We introduce four novel mechanisms—BLEND, RawHash, RawHash2, and Rawsamble —that collectively enable faster, more accurate, and scalable genome analysis. We hope that the

methods and insights presented in this dissertation will inspire future research and lead to the development of more efficient and innovative approaches to genome analysis.

Appendix A

Other Works of the Author

During my Ph.D. studies, I was involved in various other projects across different fields, focusing mainly on genome analysis acceleration, metagenomics, basecalling, and genome assembly.

As a first and co-first author, I contributed to several fields in genome analysis and its acceleration. I was a co-first author of *AirLift* [502], a fast and comprehensive technique that significantly reduces the computational overhead of remapping genomic data between reference genomes. I also co-led the paper entitled *Accelerating Genome Analysis via Algorithm-Architecture Co-Design* [262], which reviews recent advancements in genome analysis acceleration and highlights the integration of multiple steps using suitable architectures to improve performance and efficiency. In genome assembly and polishing, I was the first author of *Apollo* [257], an assembly polishing algorithm that is both sequencing-technology-independent and scalable to large genomes. Apollo was designed to improve the accuracy of genome assembly polishing by utilizing reads from all available sequencing technologies within a single run. I was also the first author of *ApHMM* [593], a flexible acceleration framework that reduces computational and energy overheads associated with profile hidden Markov models (pHMMs) used in bioinformatics applications, including assembly polishing we developed in Apollo.

As a co-author, I contributed to several papers in acceleration of genome analysis with hardware and software co-design. I was involved in *GenASM* [490], a framework that accelerates approximate string matching, a key step in genome sequence analysis, providing significant performance and power benefits. I contributed to *GenStore* [689], an in-storage processing system that improves read mapping performance by reducing data movement and computational overheads, and *SeGraM* [598], a universal hardware accelerator designed for both sequence-to-graph and sequence-to-sequence mapping, addressing critical bottlenecks in genomic mapping pipelines. In the metagenomics field, I contributed to *MegIS* [710], an in-storage processing system designed to reduce data movement overhead in metagenomic

analysis. I was involved in *Demeter* [195], a fast and energy-efficient food profiler using hyperdimensional computing in memory, which significantly improves throughput and reduces memory usage while maintaining accuracy.

My work in basecalling and nanopore sequencing includes contributions to 1) *RUBICON* [275], a framework for designing efficient deep learning-based basecallers, 2) *TargetCall* [267], a pre-basecalling filter that eliminates wasted computation by discarding off-target reads before the basecalling process, and 3) *RawAlign* [295], a tool designed for accurate, fast, and scalable raw nanopore signal mapping by combining seeding and alignment. I also contributed to *Swordfish* [711], a framework for evaluating deep neural network-based basecalling using computation-in-memory with non-ideal memristors, which explores hardware/software co-design solutions to address accuracy loss due to hardware limitations. We also proposed *GenPIP* [684], an in-memory genome analysis accelerator that tightly integrates basecalling and read mapping to reduce inefficient computation and data movement.

I worked on *FastRemap* [503], a tool for quickly remapping reads between genome assemblies, which significantly improves the efficiency of genome assembly processes. Moreover, I contributed to *Molecules to Genomic Variations* [261], a comprehensive overview of intelligent algorithms and architectures designed to accelerate genome analysis at the population level. In the area of computational biology tools and infrastructure, I worked on a systematic review titled *Packaging and Containerization of Computational Methods* [1070], which describes and compares software packaging and containerization platforms used in omics research, highlighting the need for easier installation and greater usability of biomedical research tools.

I also worked on the acceleration of virtual-to-physical address mapping with *Utopia* [1071], which proposes a hybrid virtual-to-physical address mapping scheme that enhances address translation performance.

I am currently involved in several works that are available as preprint. I am involved in *SequenceLab* [1072], a comprehensive benchmark of computational methods for comparing genomic sequences, which provides an in-depth analysis of the heterogeneity among widely-used methods, and *MetaFast* [1073], which enables fast metagenomic classification through seed counting and edit distance approximation to improve accuracy-runtime tradeoffs in metagenomic analysis.

Appendix B

Complete List of the Author's Contributions

This section lists the author's contributions to the literature in reverse chronological order under four categories: 1) first author (Section B.1), 2) co-first authored (Section B.2), 3) co-supervised contributions (Section B.3), and 4) other contributions (Section B.4).

B.1 Major Contributions Led by the Author

1. Can Firtina, Maximilian Mordig, Harun Mustafa, Sayan Goswami, Nika Mansouri Ghiasi, Stefano Mercogliano, Furkan Eris, Joël Lindegger, Andre Kahles, and Onur Mutlu, **"Rawsamble: Overlapping and Assembling Raw Nanopore Signals using a Hash-based Seeding Mechanism,"** *arXiv* [Under Submission], 2024. Source code available: https://github.com/CMU-SAFARI/RawHash

2. Can Firtina, Melina Soysal, Joël Lindegger, and Onur Mutlu, **"RawHash2: Mapping Raw Nanopore Signals Using Hash-Based Seeding and Adaptive Quantization,"** *Bioinformatics*, 2024. Source code available: https://github.com/CMU-SAFARI/RawHash

3. Can Firtina, Kamlesh Pillai, Gurpreet S. Kalsi, Bharathwaj Suresh, Damla Senol Cali, Jeremie S. Kim, Taha Shahroodi, Meryem Banu Cavlak, Joël Lindegger, Mohammed Alser, Juan Gómez Luna, Sreenivas Subramoney, and Onur Mutlu, **"ApHMM: Accelerating Profile Hidden Markov Models for Fast and Energy-efficient Genome Analysis,"** *ACM TACO*, 2024. Source code available: https://github.com/CMU-SAFARI/ApHMM-GPU

4. Can Firtina, Nika Mansouri Ghiasi, Joël Lindegger, Gagandeep Singh, Meryem Banu

Cavlak, Haiyu Mao, and Onur Mutlu, **"RawHash: enabling fast and accurate real-time analysis of raw nanopore signals for large genomes,"** in *Proceedings of the 31st Annual Conference on Intelligent Systems for Molecular Biology and the 22nd European Conference on Computational Biology (ISMB/ECCB)*, Lyon, France, 2023. Source code available: https://github.com/CMU-SAFARI/RawHash

5. Can Firtina, Jisung Park, Mohammed Alser, Jeremie S. Kim, Damla Senol Cali, Taha Shahroodi, Nika Mansouri Ghiasi, Gagandeep Singh, Konstantinos Kanellopoulos, Can Alkan, and Onur Mutlu, **"BLEND: a fast, memory-efficient and accurate mechanism to find fuzzy seed matches in genome analysis,"** *NAR Genomics and Bioinformatics*, 2023. Source code available: https://github.com/CMU-SAFARI/BLEND

6. Can Firtina, Jeremie S. Kim, Mohammed Alser, Damla Senol Cali, A. Ercument Cicek, Can Alkan, and Onur Mutlu, **"Apollo: a sequencing-technology-independent, scalable and accurate assembly polishing algorithm,"** *Bioinformatics*, 2020. Source code available: https://github.com/CMU-SAFARI/Apollo

B.2 Major Contributions Co-Led by the Author

1. Jeremie S. Kim, Can Firtina, Meryem Banu Cavlak, Damla Senol Cali, Nastaran Hajinazar, Mohammed Alser, Can Alkan, and Onur Mutlu, **"AirLift: A Fast and Comprehensive Technique for Remapping Alignments between Reference Genomes,"** *IEEE/ACM TCBB*, 2024. Source code available: https://github.com/CMU-SAFARI/AirLift

2. Onur Mutlu and Can Firtina, **"Accelerating Genome Analysis via Algorithm-Architecture Co-Design,"** in *DAC*, 2023.

B.3 Co-supervised Contributions

1. Meryem Banu Cavlak, Gagandeep Singh, Mohammed Alser, Can Firtina, Joel Lindegger, Mohammad Sadrosadati, Nika Mansouri Ghiasi, Can Alkan, and Onur Mutlu, **"TargetCall: Eliminating the Wasted Computation in Basecalling via Pre-Basecalling Filtering,"** *Frontiers in Genetics*, 2024. Source code available: https://github.com/CMU-SAFARI/TargetCall

2. Joël Lindegger, Can Firtina, Nika Mansouri Ghiasi, Mohammad Sadrosadati, Mohammed Alser, and Onur Mutlu, **"RawAlign: Accurate, Fast, and Scalable Raw Nanopore Signal Mapping via Combining Seeding and Alignment,"** *arXiv* [Under Submission], 2023. Source code available: https://github.com/CMU-SAFARI/RawAlign

B.4 Other Contributions

1. Nika Mansouri Ghiasi, Mohammad Sadrosadati, Harun Mustafa, Arvid Gollwitzer, Can Firtina, Julien Eudine, Haiyu Mao, Joël Lindegger, Meryem Banu Cavlak, Mohammed Alser, Jisung Park, and Onur Mutlu, **"MegIS: High-Performance, Energy-Efficient, and Low-Cost Metagenomic Analysis with In-Storage Processing,"** in *ISCA*, 2024.

2. Mohammed Alser, Brendan Lawlor, Richard J. Abdill, Sharon Waymost, Ram Ayyala, Neha Rajkumar, Nathan LaPierre, Jaqueline Brito, André M. Ribeiro-dos-Santos, Nour Almadhoun, Varuni Sarwal, Can Firtina, Tomasz Osinski, Eleazar Eskin, Qiyang Hu, Derek Strong, Byoung-Do (B.D) Kim, Malak S. Abedalthagafi, Onur Mutlu, and Serghei Mangul, **"Packaging and containerization of computational methods,"** *Nature Protocols*, 2024.

3. Gagandeep Singh, Mohammed Alser, Kristof Denolf, Can Firtina, Alireza Khodamoradi, Meryem Banu Cavlak, Henk Corporaal, and Onur Mutlu, **"RUBICON: a framework for designing efficient deep learning-based genomic basecallers,"** *Genome Biology*, 2024. Source code available: https://github.com/CMU-SAFARI/Rubicon

4. Arvid E. Gollwitzer, Mohammed Alser, Joel Bergtholdt, Joel Lindegger, Maximilian-David Rumpf, Can Firtina, Serghei Mangul, and Onur Mutlu, **"MetaFast: Enabling Fast Metagenomic Classification via Seed Counting and Edit Distance Approximation,"** *arXiv* [Under Submission], 2023. Source code available: https://github.com/CMU-SAFARI/MetaTrinity

5. Maximilian-David Rumpf, Mohammed Alser, Arvid E. Gollwitzer, Joel Lindegger, Nour Almadhoun, Can Firtina, Serghei Mangul, and Onur Mutlu, **"SequenceLab: A Comprehensive Benchmark of Computational Methods for Comparing Genomic Sequences,"** *arXiv* [Under Submission], 2023. Source code available: https://github.com/CMU-SAFARI/SequenceLab

6. Taha Shahroodi, Gagandeep Singh, Mahdi Zahedi, Haiyu Mao, Joel Lindegger, Can Firtina, Stephan Wong, Onur Mutlu, and Said Hamdioui, **"Swordfish: A Framework for Evaluating Deep Neural Network-Based Basecalling Using Computation-In-Memory with Non-Ideal Memristors,"** in *MICRO*, 2023.

7. Konstantinos Kanellopoulos, Rahul Bera, Kosta Stojiljkovic, F. Nisa Bostanci, Can Firtina, Rachata Ausavarungnirun, Rakesh Kumar, Nastaran Hajinazar, Mohammad Sadrosadati,

Nandita Vijaykumar, and Onur Mutlu, **"Utopia: Fast and Efficient Address Translation via Hybrid Restrictive & Flexible Virtual-to-Physical Address Mappings,"** in *MICRO*, 2023. Artifact available: https://github.com/CMU-SAFARI/Utopia

8. Haiyu Mao, Mohammed Alser, Mohammad Sadrosadati, Can Firtina, Akanksha Baranwal, Damla Senol Cali, Aditya Manglik, Nour Almadhoun Alserr, and Onur Mutlu, **"GenPIP: In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping,"** in *MICRO*, 2022.

9. Jeremie S. Kim, Can Firtina, Meryem Banu Cavlak, Damla Senol Cali, Can Alkan, and Onur Mutlu, **"FastRemap: a tool for quickly remapping reads between genome assemblies,"** *Bioinformatics*, 2022. Source code available: https://github.com/CMU-SAFARI/FastRemap

10. Mohammed Alser, Joel Lindegger, Can Firtina, Nour Almadhoun, Haiyu Mao, Gagandeep Singh, Juan Gomez-Luna, and Onur Mutlu, **"From molecules to genomic variations: Accelerating genome analysis via intelligent algorithms and architectures,"** *CSBJ*, 2022.

11. Taha Shahroodi, Mahdi Zahedi, Can Firtina, Mohammed Alser, Stephan Wong, Onur Mutlu, and Said Hamdioui, **"Demeter: A fast and energy-efficient food profiler using hyperdimensional computing in memory,"** *IEEE Access*, 2022.

12. Damla Senol Cali, Konstantinos Kanellopoulos, Joël Lindegger, Zülal Bingöl, Gurpreet S. Kalsi, Ziyi Zuo, Can Firtina, Meryem Banu Cavlak, Jeremie Kim, Nika Mansouri Ghiasi, Gagandeep Singh, Juan Gómez-Luna, Nour Almadhoun Alserr, Mohammed Alser, Sreenivas Subramoney, Can Alkan, Saugata Ghose, and Onur Mutlu, **"SeGraM: A universal hardware accelerator for genomic sequence-to-graph and sequence-to-sequence mapping,"** in *ISCA*, 2022. Artifact available: https://github.com/CMU-SAFARI/SeGraM

13. Nika Mansouri Ghiasi, Jisung Park, Harun Mustafa, Jeremie Kim, Ataberk Olgun, Arvid Gollwitzer, Damla Senol Cali, Can Firtina, Haiyu Mao, Nour Almadhoun Alserr, Rachata Ausavarungnirun, Nandita Vijaykumar, Mohammed Alser, and Onur Mutlu, **"GenStore: A high-performance in-storage processing system for genome sequence analysis,"** in *ASPLOS*, 2022. Artifact available: https://github.com/CMU-SAFARI/GenStore

14. Damla Senol Cali, Gurpreet S. Kalsi, Zülal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S. Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, Anant Norion, Allison Scibisz, Sreenivas Subramoneyon, Can Alkan, Saugata

Ghose, and Onur Mutlu, **"GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis,"** in *MICRO*, 2020. Artifact available: https://github.com/CMU-SAFARI/GenASM

Bibliography

[1] Y. Wang, Y. Zhao, A. Bollas, Y. Wang, and K. F. Au, "Nanopore sequencing technology, bioinformatics and applications," *Nature Biotechnology*, vol. 39, no. 11, pp. 1348–1365, Nov. 2021.

[2] C. Alkan, J. M. Kidd, T. Marques-Bonet, G. Aksay, F. Antonacci, F. Hormozdiari, J. O. Kitzman, C. Baker, M. Malig, O. Mutlu, S. C. Sahinalp, R. A. Gibbs, and E. E. Eichler, "Personalized copy number and segmental duplication maps using next-generation sequencing," *Nature Genetics*, vol. 41, no. 10, pp. 1061–1067, Oct. 2009.

[3] G. Lightbody, V. Haberland, F. Browne, L. Taggart, H. Zheng, E. Parkes, and J. K. Blayney, "Review of applications of high-throughput sequencing in personalized medicine: barriers and facilitators of future progress in research and clinical application," *Briefings Bioinform.*, 2019.

[4] S. Morganti, P. Tarantino, E. Ferraro, P. D'Amico, B. A. Duso, and G. Curigliano, "Next Generation Sequencing (NGS): A Revolutionary Technology in Pharmacogenomics and Personalized Medicine in Cancer," in *Adv. Exp. Med. Biol.*, Cham, 2019.

[5] I. Branco and A. Choupina, "Bioinformatics: new tools and applications in life science and personalized medicine," *Applied Microbiology and Biotechnology*, vol. 105, no. 3, pp. 937–951, Feb. 2021.

[6] S. Quazi, "Artificial intelligence and machine learning in precision and genomic medicine," *Medical Oncology*, vol. 39, no. 8, p. 120, Jun. 2022.

[7] S. J. Aronson and H. L. Rehm, "Building the foundation for genomics in precision medicine," *Nature*, vol. 526, no. 7573, pp. 336–342, Oct. 2015.

[8] H. F. Löchel and D. Heider, "Comparative analyses of error handling strategies for next-generation sequencing in precision medicine," *Scientific Reports*, vol. 10, no. 1, p. 5750, Apr. 2020.

[9] E. Papadopoulou, D. Bouzarelou, G. Tsaousis, A. Papathanasiou, G. Vogiatzi, C. Vlachopoulos, A. Miliou, P. Papachristou, E. Prappa, G. Servos, K. Ritsatos, A. Seretis, A. Frogoudaki, and G. Nasioulas, "Application of next generation sequencing in cardiology: current and future precision medicine implications," *Frontiers in Cardiovascular Medicine*, vol. 10, 2023.

[10] A. Tafazoli, H.-J. Guchelaar, W. Miltyk, A. J. Kretowski, and J. J. Swen, "Applying Next-Generation Sequencing Platforms for Pharmacogenomic Testing in Clinical Practice," *Frontiers in Pharmacology*, vol. 12, 2021.

[11] V. Gambardella, N. Tarazona, J. M. Cejalvo, P. Lombardi, M. Huerta, S. Roselló, T. Fleitas, D. Roda, and A. Cervantes, "Personalized Medicine: Recent Progress in Cancer Therapy," *Cancers*, vol. 12, no. 4, 2020.

[12] R. J. Leary, I. Kinde, F. Diehl, K. Schmidt, C. Clouser, C. Duncan, A. Antipova, C. Lee, K. McKernan, F. M. De La Vega, K. W. Kinzler, B. Vogelstein, L. A. Diaz, and V. E. Velculescu, "Development of Personalized Tumor Biomarkers Using Massively Parallel Sequencing," *Science Translational Medicine*, vol. 2, no. 20, pp. 20ra14–20ra14, Feb. 2010.

[13] Hamburg Margaret A. and Collins Francis S., "The Path to Personalized Medicine," *New England Journal of Medicine*, vol. 363, no. 4, pp. 301–304, 2010.

[14] M. van der Lee, M. Kriek, H.-J. Guchelaar, and J. J. Swen, "Technologies for Pharmacogenomics: A Review," *Genes*, vol. 11, no. 12, 2020.

[15] D. Moon, H. W. Park, D. Surl, D. Won, S.-T. Lee, S. Shin, J. R. Choi, and J. Han, "Precision Medicine through Next-Generation Sequencing in Inherited Eye Diseases in a Korean Cohort," *Genes*, vol. 13, no. 1, 2022.

[16] S. Mohan, V. Foy, M. Ayub, H. S. Leong, P. Schofield, S. Sahoo, T. Descamps, B. Kilerci, N. K. Smith, M. Carter, L. Priest, C. Zhou, T. H. Carr, C. Miller, C. Faivre-Finn, F. Blackhall, D. G. Rothwell, C. Dive, and G. Brady, "Profiling of Circulating Free DNA Using Targeted and Genome-wide Sequencing in Patients with SCLC," *Journal of Thoracic Oncology*, vol. 15, no. 2, pp. 216–230, Feb. 2020.

[17] C. C. Chung, G. K. Leung, C. C. Mak, J. L. Fung, M. Lee, S. L. Pei, M. H. Yu, V. C. Hui, J. C. Chan, J. F. Chau, M. C. Chan, M. H. Tsang, W. H. Wong, J. Y. Tung, K. S. Lun, Y. K. Ng, C. W. Fung, M. S. Wong, R. M. Wong, Y. L. Lau, G. C. Chan, S. L. Lee, K. S. Yeung, and B. H. Chung, "Rapid whole-exome sequencing facilitates precision medicine in paediatric rare disease patients and reduces healthcare costs," *The Lancet Regional Health – Western Pacific*, vol. 1, Aug. 2020.

[18] S. J. Bielinski, J. E. Olson, J. Pathak, R. M. Weinshilboum, L. Wang, K. J. Lyke, E. Ryu, P. V. Targonski, M. D. Van Norstrand, M. A. Hathcock, P. Y. Takahashi, J. B. McCormick, K. J. Johnson, K. J. Maschke, C. R. Rohrer Vitek, M. S. Ellingson, E. D. Wieben, G. Farrugia, J. A. Morrisette, K. J. Kruckeberg, J. K. Bruflat, L. M. Peterson, J. H. Blommel, J. M. Skierka, M. J. Ferber, J. L. Black, L. M. Baudhuin, E. W. Klee, J. L. Ross, T. L. Veldhuizen, C. G. Schultz, P. J. Caraballo, R. R. Freimuth, C. G. Chute, and I. J. Kullo, "Preemptive Genotyping for Personalized Medicine: Design of the Right Drug, Right Dose, Right Time—Using Genomic Data to Individualize Treatment Protocol," *Mayo Clinic Proceedings*, vol. 89, no. 1, pp. 25–33, Jan. 2014.

[19] D. Ho, S. R. Quake, E. R. McCabe, W. J. Chng, E. K. Chow, X. Ding, B. D. Gelb, G. S. Ginsburg, J. Hassenstab, C.-M. Ho, W. C. Mobley, G. P. Nolan, S. T. Rosen, P. Tan, Y. Yen, and A. Zarrinpar, "Enabling Technologies for Personalized and Precision Medicine," *Trends in Biotechnology*, vol. 38, no. 5, pp. 497–518, May 2020.

[20] B. M. Hussen, S. T. Abdullah, A. Salihi, D. K. Sabir, K. R. Sidiq, M. F. Rasul, H. J. Hidayat, S. Ghafouri-Fard, M. Taheri, and E. Jamali, "The emerging roles of NGS in clinical oncology and personalized medicine," *Pathology - Research and Practice*, vol. 230, p. 153760, Feb. 2022.

[21] L. E. Russell, Y. Zhou, A. A. Almousa, J. K. Sodhi, C. K. Nwabufo, and V. M. Lauschke, "Pharmacogenomics in the era of next generation sequencing – from byte to bedside," *Drug Metabolism Reviews*, vol. 53, no. 2, pp. 253–278, Apr. 2021.

[22] R. Verma, K. E. da Silva, N. Rockwood, R. E. Wasmann, N. Yende, T. Song, E. Kim, P. Denti, R. J. Wilkinson, and J. R. Andrews, "A Nanopore Sequencing-based Pharmacogenomic Panel to Personalize Tuberculosis Drug Dosing," *American Journal of Respiratory and Critical Care Medicine*, vol. 209, no. 12, pp. 1486–1496, Jun. 2024.

[23] M. Jinek, K. Chylinski, I. Fonfara, M. Hauer, J. A. Doudna, and E. Charpentier, "A Programmable Dual-RNA–Guided DNA Endonuclease in Adaptive Bacterial Immunity," *Science*, vol. 337, no. 6096, pp. 816–821, Aug. 2012.

[24] J. A. Doudna, "The promise and challenge of therapeutic genome editing," *Nature*, 2020.

[25] A. Pickar-Oliver and C. A. Gersbach, "The next generation of CRISPR–Cas technologies and applications," *Nat. Rev. Mol. Cell Biol.*, 2019.

[26] S. C. Selvakumar, K. A. Preethi, K. Ross, D. Tusubira, M. W. A. Khan, P. Mani, T. N. Rao, and D. Sekar, "CRISPR/Cas9 and next generation sequencing in the personalized treatment of Cancer," *Molecular Cancer*, vol. 21, no. 1, p. 83, Mar. 2022.

[27] H. Puchta, J. Jiang, K. Wang, and Y. Zhao, "Updates on gene editing and its applications," *Plant Physiology*, vol. 188, no. 4, pp. 1725–1730, Apr. 2022.

[28] M. Luo, J. Wang, Z. Dong, C. Wang, and G. Lu, "CRISPR-Cas9 sgRNA design and outcome assessment: Bioinformatics tools and aquaculture applications," *SI : Emerging and disruptive technologies for aquaculture*, vol. 7, no. 2, pp. 121–130, Mar. 2022.

[29] C. Li, W. Chu, R. A. Gill, S. Sang, Y. Shi, X. Hu, Y. Yang, Q. U. Zaman, and B. Zhang, "Computational tools and resources for CRISPR/Cas genome editing," *Genomics, Proteomics & Bioinformatics*, Mar. 2022.

[30] A. Katti, B. J. Diaz, C. M. Caragine, N. E. Sanjana, and L. E. Dow, "CRISPR in cancer biology and therapy," *Nature Reviews Cancer*, vol. 22, no. 5, pp. 259–279, May 2022.

[31] Y. Huang, M. Shang, T. Liu, and K. Wang, "High-throughput methods for genome editing: the more the better," *Plant Physiology*, vol. 188, no. 4, pp. 1731–1745, Apr. 2022.

[32] C. J. Braun, A. C. Adames, D. Saur, and R. Rad, "Tutorial: design and execution of CRISPR in vivo screens," *Nature Protocols*, Jul. 2022.

[33] G.-H. Hwang, B. Song, and S. Bae, "Current widely-used web-based tools for CRISPR nucleases, base editors, and prime editors," *Gene and Genome Editing*, vol. 1, p. 100004, Jun. 2021.

[34] C. Brakebusch, "CRISPR Genome Editing: How to Make a Fantastic Method Even Better," *Cells*, vol. 10, no. 2, 2021.

[35] X. R. Bao, Y. Pan, C. M. Lee, T. H. Davis, and G. Bao, "Tools for experimental and computational analyses of off-target editing by programmable nucleases," *Nature Protocols*, vol. 16, no. 1, pp. 10–26, Jan. 2021.

[36] Y. Zhang, G. Zhao, F. Y. H. Ahmed, T. Yi, S. Hu, T. Cai, and Q. Liao, "In silico Method in CRISPR/Cas System: An Expedite and Powerful Booster," *Frontiers in Oncology*, vol. 10, 2020.

[37] J. Yan, D. Xue, G. Chuai, Y. Gao, G. Zhang, and Q. Liu, "Benchmarking and integrating genome-wide CRISPR off-target detection and prediction," *Nucleic Acids Research*, vol. 48, no. 20, pp. 11 370–11 379, Nov. 2020.

[38] P. Sledzinski, M. Nowaczyk, and M. Olejniczak, "Computational Tools and Resources Supporting CRISPR-Cas Experiments," *Cells*, vol. 9, no. 5, p. 1288, May 2020.

[39] H. Manghwar, B. Li, X. Ding, A. Hussain, K. Lindsey, X. Zhang, and S. Jin, "CRISPR/Cas Systems in Genome Editing: Methodologies and Tools for sgRNA Design, Off-Target Evaluation, and Strategies to Mitigate Off-Target Effects," *Advanced Science*, vol. 7, no. 6, p. 1902312, 2020.

[40] G. Liu, Y. Zhang, and T. Zhang, "Computational approaches for effective CRISPR guide RNA design and evaluation," *Computational and Structural Biotechnology Journal*, vol. 18, pp. 35–44, 2020.

[41] X. Lin, A. Chemparathy, M. La Russa, T. Daley, and L. S. Qi, "Computational Methods for Analysis of Large-Scale CRISPR Screens," *Annual Review of Biomedical Data Science*, vol. 3, no. 1, pp. 137–162, Jul. 2020.

[42] Y. K. Jeong, B. Song, and S. Bae, "Current Status and Challenges of DNA Base Editing Tools," *Molecular Therapy*, vol. 28, no. 9, pp. 1938–1952, Sep. 2020.

[43] R. E. Hanna and J. G. Doench, "Design and analysis of CRISPR–Cas experiments," *Nature Biotechnology*, vol. 38, no. 7, pp. 813–823, Jul. 2020.

[44] K. Clement, J. Y. Hsu, M. C. Canver, J. K. Joung, and L. Pinello, "Technologies and Computational Analysis Strategies for CRISPR Applications," *Molecular Cell*, vol. 79, no. 1, pp. 11–29, Jul. 2020.

[45] H. G. Chaudhari, J. Penterman, H. J. Whitton, S. J. Spencer, N. Flanagan, M. C. Lei Zhang, E. Huang, A. S. Khedkar, J. M. Toomey, C. A. Shearer, A. W. Needham, T. W. Ho, J. D. Kulman, T. Cradick, and A. Kernytsky, "Evaluation of Homology-Independent CRISPR-Cas9 Off-Target Assessment Methods," *The CRISPR Journal*, vol. 3, no. 6, pp. 440–453, Dec. 2020.

[46] S. Bodapati, T. P. Daley, X. Lin, J. Zou, and L. S. Qi, "A benchmark of algorithms for the analysis of pooled CRISPR screens," *Genome Biology*, vol. 21, no. 1, p. 62, Mar. 2020.

[47] J. Bradford and D. Perrin, "A benchmark of computational CRISPR-Cas9 guide design methods," *PLOS Computational Biology*, vol. 15, no. 8, p. e1007274, Aug. 2019.

[48] J. Yan, G. Chuai, C. Zhou, C. Zhu, J. Yang, C. Zhang, F. Gu, H. Xu, J. Wei, and Q. Liu, "Benchmarking CRISPR on-target sgRNA design," *Briefings in Bioinformatics*, vol. 19, no. 4, pp. 721–724, Jul. 2018.

[49] M. F. Sentmanat, S. T. Peters, C. P. Florian, J. P. Connelly, and S. M. Pruett-Miller, "A Survey of Validation Strategies for CRISPR-Cas9 Editing," *Scientific Reports*, vol. 8, no. 1, p. 888, Jan. 2018.

[50] G.-h. Chuai, Q.-L. Wang, and Q. Liu, "In Silico Meets In Vivo: Towards Computational CRISPR-Based sgRNA Design," *Trends in Biotechnology*, vol. 35, no. 1, pp. 12–21, Jan. 2017.

[51] M. Kanehisa, "Toward understanding the origin and evolution of cellular organisms," *Protein Science*, 2019.

[52] J. Qing, Y.-D. Meng, F. He, Q.-X. Du, J. Zhong, H.-Y. Du, P.-F. Liu, L.-Y. Du, and L. Wang, "Whole genome re-sequencing reveals the genetic diversity and evolutionary patterns of Eucommia ulmoides," *Molecular Genetics and Genomics*, vol. 297, no. 2, pp. 485–494, Mar. 2022.

[53] P. J. Wittkopp and G. Kalay, "Cis-regulatory elements: molecular mechanisms and evolutionary processes underlying divergence," *Nature Reviews Genetics*, vol. 13, no. 1, pp. 59–69, Jan. 2012.

[54] I. G. Romero, I. Ruvinsky, and Y. Gilad, "Comparative studies of gene expression and the evolution of gene regulation," *Nature Reviews Genetics*, vol. 13, no. 7, pp. 505–516, Jul. 2012.

[55] X. Wang, H. Feng, Y. Chang, C. Ma, L. Wang, X. Hao, A. Li, H. Cheng, L. Wang, P. Cui, J. Jin, X. Wang, K. Wei, C. Ai, S. Zhao, Z. Wu, Y. Li, B. Liu, G.-D. Wang, L. Chen, J. Ruan, and Y. Yang, "Population sequencing enhances understanding of tea plant evolution," *Nature Communications*, vol. 11, no. 1, p. 4447, Sep. 2020.

[56] M. S. Hill, P. Vande Zande, and P. J. Wittkopp, "Molecular and evolutionary processes generating variation in gene expression," *Nature Reviews Genetics*, vol. 22, no. 4, pp. 203–215, Apr. 2021.

[57] E. D. Vaishnav, C. G. de Boer, J. Molinet, M. Yassour, L. Fan, X. Adiconis, D. A. Thompson, J. Z. Levin, F. A. Cubillos, and A. Regev, "The evolution, evolvability and engineering of gene regulatory DNA," *Nature*, vol. 603, no. 7901, pp. 455–463, Mar. 2022.

[58] X. Zhang, S. Chen, L. Shi, D. Gong, S. Zhang, Q. Zhao, D. Zhan, L. Vasseur, Y. Wang, J. Yu, Z. Liao, X. Xu, R. Qi, W. Wang, Y. Ma, P. Wang, N. Ye, D. Ma, Y. Shi, H. Wang, X. Ma, X. Kong, J. Lin, L. Wei, Y. Ma, R. Li, G. Hu, H. He, L. Zhang, R. Ming, G. Wang, H. Tang, and M. You, "Haplotype-resolved genome assembly provides insights into evolutionary history of the tea plant Camellia sinensis," *Nature Genetics*, vol. 53, no. 8, pp. 1250–1259, Aug. 2021.

[59] J. C. Fay and P. J. Wittkopp, "Evaluating the role of natural selection in the evolution of gene regulation," *Heredity*, vol. 100, no. 2, pp. 191–199, Feb. 2008.

[60] A. M. Kanzi, J. E. San, B. Chimukangara, E. Wilkinson, M. Fish, V. Ramsuran, and T. de Oliveira, "Next Generation Sequencing and Bioinformatics Analysis of Family Genetic Inheritance," *Frontiers in Genetics*, vol. 11, 2020.

[61] G. A. Wray, M. W. Hahn, E. Abouheif, J. P. Balhoff, M. Pizer, M. V. Rockman, and L. A. Romano, "The Evolution of Transcriptional Regulation in Eukaryotes," *Molecular Biology and Evolution*, vol. 20, no. 9, pp. 1377–1419, Sep. 2003.

[62] A. Wu, L. Wang, H.-Y. Zhou, C.-Y. Ji, S. Z. Xia, Y. Cao, J. Meng, X. Ding, S. Gold, T. Jiang, and G. Cheng, "One year of SARS-CoV-2 evolution," *Cell Host & Microbe*, vol. 29, no. 4, pp. 503–507, Apr. 2021.

[63] S. A. Signor and S. V. Nuzhdin, "The Evolution of Gene Expression in cis and trans," *Trends in Genetics*, vol. 34, no. 7, pp. 532–544, Jul. 2018.

[64] A. Whitehead and D. L. Crawford, "Variation within and among species in gene expression: raw material for evolution," *Molecular Ecology*, vol. 15, no. 5, pp. 1197–1211, Apr. 2006.

[65] J. D. Coolon, C. J. McManus, K. R. Stevenson, B. R. Graveley, and P. J. Wittkopp, "Tempo and mode of regulatory evolution in Drosophila," *Genome Research*, vol. 24, no. 5, pp. 797–808, May 2014.

[66] M. S. Lawrence, P. Stojanov, P. Polak, G. V. Kryukov, K. Cibulskis, A. Sivachenko, S. L. Carter, C. Stewart, C. H. Mermel, S. A. Roberts, A. Kiezun, P. S. Hammerman, A. McKenna, Y. Drier, L. Zou, A. H. Ramos, T. J. Pugh, N. Stransky, E. Helman, J. Kim, C. Sougnez, L. Ambrogio, E. Nickerson, E. Shefler, M. L. Cortés, D. Auclair, G. Saksena, D. Voet, M. Noble, D. DiCara, P. Lin, L. Lichtenstein, D. I. Heiman, T. Fennell, M. Imielinski, B. Hernandez, E. Hodis, S. Baca, A. M. Dulak, J. Lohr, D.-A. Landau, C. J. Wu, J. Melendez-Zajgla, A. Hidalgo-Miranda, A. Koren, S. A. McCarroll, J. Mora, R. S. Lee, B. Crompton, R. Onofrio, M. Parkin, W. Winckler, K. Ardlie, S. B. Gabriel, C. W. M. Roberts, J. A. Biegel, K. Stegmaier, A. J. Bass, L. A. Garraway, M. Meyerson, T. R. Golub, D. A. Gordenin, S. Sunyaev, E. S. Lander, and G. Getz, "Mutational heterogeneity in cancer and the search for new cancer-associated genes," *Nature*, 2013.

[67] B. Vogelstein, N. Papadopoulos, V. E. Velculescu, S. Zhou, L. A. Diaz, and K. W. Kinzler, "Cancer Genome Landscapes," *Science*, 2013.

[68] D. Ramsköld, S. Luo, Y.-C. Wang, R. Li, Q. Deng, O. R. Faridani, G. A. Daniels, I. Khrebtukova, J. F. Loring, L. C. Laurent, G. P. Schroth, and R. Sandberg, "Full-length mRNA-Seq from single-cell levels of RNA and individual circulating tumor cells," *Nature Biotechnology*, vol. 30, no. 8, pp. 777–782, Aug. 2012.

[69] T. Baslan and J. Hicks, "Unravelling biology and shifting paradigms in cancer with single-cell sequencing," *Nature Reviews Cancer*, vol. 17, no. 9, pp. 557–569, Sep. 2017.

[70] E. Shapiro, T. Biezuner, and S. Linnarsson, "Single-cell sequencing-based technologies will revolutionize whole-organism science," *Nature Reviews Genetics*, vol. 14, no. 9, pp. 618–630, Sep. 2013.

[71] Y. Sakamoto, S. Sereewattanawoot, and A. Suzuki, "A new era of long-read sequencing for cancer genomics," *Journal of Human Genetics*, vol. 65, no. 1, pp. 3–10, Jan. 2020.

[72] Q. Jia, H. Chu, Z. Jin, H. Long, and B. Zhu, "High-throughput single-cell sequencing in cancer research," *Signal Transduction and Targeted Therapy*, vol. 7, no. 1, p. 145, May 2022.

[73] D. A. Lawson, K. Kessenbrock, R. T. Davis, N. Pervolarakis, and Z. Werb, "Tumour heterogeneity and metastasis at single-cell resolution," *Nature Cell Biology*, vol. 20, no. 12, pp. 1349–1360, Dec. 2018.

[74] C. Liu, Q. Shi, X. Huang, S. Koo, N. Kong, and W. Tao, "mRNA-based cancer therapeutics," *Nature Reviews Cancer*, vol. 23, no. 8, pp. 526–543, Aug. 2023.

[75] B. Van de Sande, J. S. Lee, E. Mutasa-Gottgens, B. Naughton, W. Bacon, J. Manning, Y. Wang, J. Pollard, M. Mendez, J. Hill, N. Kumar, X. Cao, X. Chen, M. Khaladkar, J. Wen, A. Leach, and E. Ferran, "Applications of single-cell RNA sequencing in drug discovery and development," *Nature Reviews Drug Discovery*, vol. 22, no. 6, pp. 496–520, Jun. 2023.

[76] D. Chakravarty and D. B. Solit, "Clinical cancer genomic profiling," *Nature Reviews Genetics*, vol. 22, no. 8, pp. 483–501, Aug. 2021.

[77] I. Cortés-Ciriano, D. C. Gulhan, J. J.-K. Lee, G. E. M. Melloni, and P. J. Park, "Computational analysis of cancer genome sequencing data," *Nature Reviews Genetics*, vol. 23, no. 5, pp. 298–314, May 2022.

[78] I. W. Deveson, B. Gong, K. Lai, J. S. LoCoco, T. A. Richmond, J. Schageman, Z. Zhang, N. Novoradovskaya, J. C. Willey, W. Jones, R. Kusko, G. Chen, B. S. Madala, J. Blackburn, I. Stevanovski, A. Bhandari, D. Close, J. Conroy, M. Hubank, N. Marella, P. A. Mieczkowski, F. Qiu, R. Sebra, D. Stetson, L. Sun, P. Szankasi, H. Tan, L.-y. Tang, H. Arib, H. Best, B. Burgher, P. R. Bushel, F. Casey, S. Cawley, C.-J. Chang, J. Choi, J. Dinis, D. Duncan, A. K. Eterovic, L. Feng, A. Ghosal, K. Giorda, S. Glenn, S. Happe, N. Haseley, K. Horvath, L.-Y. Hung, M. Jarosz, G. Kushwaha, D. Li, Q.-Z. Li, Z. Li, L.-C. Liu, Z. Liu, C. Ma, C. E. Mason, D. B. Megherbi, T. Morrison, C. Pabón-Peña, M. Pirooznia, P. Z. Proszek, A. Raymond, P. Rindler, R. Ringler, A. Scherer, R. Shaknovich, T. Shi, M. Smith, P. Song, M. Strahl, V. J. Thodima, N. Tom, S. Verma, J. Wang, L. Wu, W. Xiao, C. Xu, M. Yang, G. Zhang, S. Zhang, Y. Zhang, L. Shi, W. Tong, D. J. Johann, T. R. Mercer, J. Xu, and SEQC2 Oncopanel Sequencing Working Group, "Evaluating the analytical validity of circulating tumor DNA sequencing assays for precision oncology," *Nature Biotechnology*, vol. 39, no. 9, pp. 1115–1128, Sep. 2021.

[79] W. Xiao, L. Ren, Z. Chen, L. T. Fang, Y. Zhao, J. Lack, M. Guan, B. Zhu, E. Jaeger, L. Kerrigan, T. M. Blomquist, T. Hung, M. Sultan, K. Idler, C. Lu, A. Scherer, R. Kusko, M. Moos, C. Xiao, S. T. Sherry, O. D. Abaan, W. Chen, X. Chen, J. Nordlund, U. Liljedahl, R. Maestro, M. Polano, J. Drabek, P. Vojta, S. Kõks, E. Reimann, B. S. Madala, T. Mercer, C. Miller, H. Jacob, T. Truong, A. Moshrefi, A. Natarajan, A. Granat, G. P. Schroth, R. Kalamegham, E. Peters, V. Petitjean, A. Walton, T.-W. Shen, K. Talsania, C. J. Vera, K. Langenbach, M. de Mars, J. A. Hipp, J. C. Willey, J. Wang, J. Shetty, Y. Kriga, A. Raziuddin, B. Tran, Y. Zheng, Y. Yu, M. Cam, P. Jailwala, C. Nguyen, D. Meerzaman, Q. Chen, C. Yan, B. Ernest, U. Mehra, R. V. Jensen, W. Jones, J.-L. Li, B. N. Papas, M. Pirooznia, Y.-C. Chen, F. Seifuddin, Z. Li, X. Liu, W. Resch, J. Wang, L. Wu, G. Yavas, C. Miles, B. Ning, W. Tong, C. E. Mason, E. Donaldson, S. Lababidi, L. M. Staudt, Z. Tezak, H. Hong, C. Wang, and L. Shi, "Toward best practice in cancer mutation detection with whole-genome and whole-exome sequencing," *Nature Biotechnology*, vol. 39, no. 9, pp. 1141–1150, Sep. 2021.

[80] K. L. Bolton, R. N. Ptashkin, T. Gao, L. Braunstein, S. M. Devlin, D. Kelly, M. Patel, A. Berthon, A. Syed, M. Yabe, C. C. Coombs, N. M. Caltabellotta, M. Walsh, K. Offit, Z. Stadler, D. Mandelker, J. Schulman, A. Patel, J. Philip, E. Bernard, G. Gundem, J. E. A. Ossa, M. Levine, J. S. M. Martinez, N. Farnoud, D. Glodzik, S. Li, M. E. Robson, C. Lee, P. D. P. Pharoah, K. H. Stopsack, B. Spitzer, S. Mantha, J. Fagin, L. Boucai, C. J. Gibson, B. L. Ebert, A. L. Young, T. Druley, K. Takahashi, N. Gillis, M. Ball, E. Padron, D. M. Hyman, J. Baselga, L. Norton, S. Gardos, V. M. Klimek, H. Scher, D. Bajorin, E. Paraiso, R. Benayed, M. E. Arcila, M. Ladanyi, D. B. Solit, M. F. Berger, M. Tallman, M. Garcia-Closas, N. Chatterjee, L. A. Diaz, R. L. Levine, L. M. Morton, A. Zehir, and E. Papaemmanuil, "Cancer therapy shapes the fitness landscape of clonal hematopoiesis," *Nature Genetics*, vol. 52, no. 11, pp. 1219–1226, Nov. 2020.

[81] J. D. Szustakowski, S. Balasubramanian, E. Kvikstad, S. Khalid, P. G. Bronson, A. Sasson, E. Wong, D. Liu, J. Wade Davis, C. Haefliger, A. Katrina Loomis, R. Mikkilineni, H. J. Noh, S. Wadhawan, X. Bai, A. Hawes, O. Krasheninina, R. Ulloa, A. E. Lopez, E. N. Smith, J. F. Waring, C. D. Whelan, E. A. Tsai, J. D. Overton, W. J. Salerno, H. Jacob, S. Szalma, H. Runz, G. Hinkle, P. Nioi, S. Petrovski,

M. R. Miller, A. Baras, L. J. Mitnaul, J. G. Reid, O. Moiseyenko, C. Rios, S. Saha, G. Abecasis, N. Banerjee, C. Beechert, B. Boutkov, M. Cantor, G. Coppola, A. Economides, G. Eom, C. Forsythe, E. D. Fuller, Z. Gu, L. Habegger, M. B. Jones, R. Lanche, M. Lattari, M. LeBlanc, D. Li, L. A. Lotta, K. Manoochehri, A. J. Mansfield, E. K. Maxwell, J. Mighty, M. Nafde, S. O'Keeffe, M. Orelus, M. S. Padilla, R. Panea, T. Polanco, M. Pradhan, A. Rasool, T. D. Schleicher, D. Sharma, A. Shuldiner, J. C. Staples, C. V. Van Hout, L. Widom, S. E. Wolf, S. John, C.-Y. Chen, D. Sexton, V. Kupelian, E. Marshall, T. Swan, S. Eaton, J. Z. Liu, S. Loomis, M. Jensen, S. Duraisamy, J. Tetrault, D. Merberg, S. Badola, M. Reppell, J. Grundstad, X. Zheng, A. M. Deaton, M. M. Parker, L. D. Ward, A. O. Flynn-Carroll, C. Austin, R. March, M. N. Pangalos, A. Platt, M. Snowden, A. Matakidou, S. Wasilewski, Q. Wang, S. Deevi, K. Carss, K. Smith, M. Sogaard, X. Hu, X. Chen, Z. Ye, and UKB-ESC Research Team, "Advancing human genetics research and drug discovery through exome sequencing of the UK Biobank," *Nature Genetics*, vol. 53, no. 7, pp. 942–948, Jul. 2021.

[82] N. Navin and J. Hicks, "Future medical applications of single-cell sequencing in cancer," *Genome Medicine*, vol. 3, no. 5, p. 31, May 2011.

[83] M. Hong, S. Tao, L. Zhang, L.-T. Diao, X. Huang, S. Huang, S.-J. Xie, Z.-D. Xiao, and H. Zhang, "RNA sequencing: new technologies and applications in cancer research," *Journal of Hematology & Oncology*, vol. 13, no. 1, p. 166, Dec. 2020.

[84] Y. Lei, R. Tang, J. Xu, W. Wang, B. Zhang, J. Liu, X. Yu, and S. Shi, "Applications of single-cell sequencing in cancer research: progress and perspectives," *Journal of Hematology & Oncology*, vol. 14, no. 1, p. 91, Jun. 2021.

[85] Y. Han, D. Wang, L. Peng, T. Huang, X. He, J. Wang, and C. Ou, "Single-cell sequencing: a promising approach for uncovering the mechanisms of tumor metastasis," *Journal of Hematology & Oncology*, vol. 15, no. 1, p. 59, May 2022.

[86] G. Federici and S. Soddu, "Variants of uncertain significance in the era of high-throughput genome sequencing: a lesson from breast and ovary cancers," *Journal of Experimental & Clinical Cancer Research*, vol. 39, no. 1, p. 46, Mar. 2020.

[87] Y. Zhang, D. Wang, M. Peng, L. Tang, J. Ouyang, F. Xiong, C. Guo, Y. Tang, Y. Zhou, Q. Liao, X. Wu, H. Wang, J. Yu, Y. Li, X. Li, G. Li, Z. Zeng, Y. Tan, and W. Xiong, "Single-cell RNA sequencing in cancer research," *Journal of Experimental & Clinical Cancer Research*, vol. 40, no. 1, p. 81, Mar. 2021.

[88] X. Ren, B. Kang, and Z. Zhang, "Understanding tumor ecosystems by single-cell sequencing: promises and limitations," *Genome Biology*, vol. 19, no. 1, p. 211, Dec. 2018.

[89] L. Tian, Y. Li, M. N. Edmonson, X. Zhou, S. Newman, C. McLeod, A. Thrasher, Y. Liu, B. Tang, M. C. Rusch, J. Easton, J. Ma, E. Davis, A. Trull, J. R. Michael, K. Szlachta, C. Mullighan, S. J. Baker, J. R. Downing, D. W. Ellison, and J. Zhang, "CICERO: a versatile method for detecting complex and diverse driver fusions using cancer RNA sequencing data," *Genome Biology*, vol. 21, no. 1, p. 126, May 2020.

[90] E. R. Malone, M. Oliva, P. J. B. Sabatini, T. L. Stockley, and L. L. Siu, "Molecular profiling for precision cancer therapies," *Genome Medicine*, vol. 12, no. 1, p. 8, Jan. 2020.

[91] X. Tang, Y. Huang, J. Lei, H. Luo, and X. Zhu, "The single-cell sequencing: new developments and medical applications," *Cell & Bioscience*, vol. 9, no. 1, p. 53, Jun. 2019.

[92] D. L. Ellsworth, H. L. Blackburn, C. D. Shriver, S. Rabizadeh, P. Soon-Shiong, and R. E. Ellsworth, "Single-cell sequencing and tumorigenesis: improved understanding of tumor evolution and metastasis," *Clinical and Translational Medicine*, vol. 6, no. 1, p. 15, Apr. 2017.

[93] Y. Zhong, F. Xu, J. Wu, J. Schubert, and M. M. Li, "Application of Next Generation Sequencing in Laboratory Medicine," *alm*, vol. 41, no. 1, pp. 25–43, Jan. 2021.

[94] Z. K. Stadler, A. Maio, D. Chakravarty, Y. Kemel, M. Sheehan, E. Salo-Mullen, K. Tkachuk, C. J. Fong, B. Nguyen, A. Erakky, K. Cadoo, Y. Liu, M. I. Carlo, A. Latham, H. Zhang, R. Kundra, S. Smith, J. Galle, C. Aghajanian, N. Abu-Rustum, A. Varghese, E. M. O'Reilly, M. Morris, W. Abida, M. Walsh, A. Drilon, G. Jayakumaran, A. Zehir, M. Ladanyi, O. Ceyhan-Birsoy, D. B. Solit, N. Schultz, M. F. Berger, D. Mandelker, L. A. Diaz, K. Offit, and M. E. Robson, "Therapeutic Implications of Germline Testing in Patients With Advanced Cancers," *Journal of Clinical Oncology*, vol. 39, no. 24, pp. 2698–2709, Aug. 2021.

[95] A. C. Tan and D. S. Tan, "Targeted Therapies for Lung Cancer Patients With Oncogenic Driver Molecular Alterations," *Journal of Clinical Oncology*, vol. 40, no. 6, pp. 611–625, Feb. 2022.

[96] A. Degasperi, X. Zou, T. Dias Amarante, A. Martinez-Martinez, G. C. C. Koh, J. M. L. Dias, L. Heskin, L. Chmelova, G. Rinaldi, V. Y. W. Wang, A. S. Nanda, A. Bernstein, S. E. Momen, J. Young, D. Perez-Gil, Y. Memari, C. Badja, S. Shooter, J. Czarnecki, M. A. Brown, H. R. Davies, Genomics England Research Consortium3†, S. Nik-Zainal, J. C. Ambrose, P. Arumugam, R. Bevers, M. Bleda, F. Boardman-Pretty, C. R. Boustred, H. Brittain, M. J. Caulfield, G. C. Chan, T. Fowler, A. Giess, A. Hamblin, S. Henderson, T. J. P. Hubbard, R. Jackson, L. J. Jones, D. Kasperaviciute, M. Kayikci, A. Kousathanas, L. Lahnstein, S. E. A. Leigh, I. U. S. Leong, F. J. Lopez, F. Maleady-Crowe, M. McEntagart, F. Minneci, L. Moutsianas, M. Mueller, N. Murugaesu, A. C. Need, P. O'Donovan, C. A. Odhams, C. Patch, D. Perez-Gil, M. B. Pereira, J. Pullinger, T. Rahim, A. Rendon, T. Rogers, K. Savage, K. Sawant, R. H. Scott, A. Siddiq, A. Sieghart, S. C. Smith, A. Sosinsky, A. Stuckey, M. Tanguy, A. L. Taylor Tavares, E. R. A. Thomas, S. R. Thompson, A. Tucci, M. J. Welland, E. Williams, K. Witkowska, and S. M. Wood, "Substitution mutational signatures in whole-genome–sequenced cancers in the UK population," *Science*, vol. 376, no. 6591, p. abl9283, 2022.

[97] J. Xu, Y. Fang, K. Chen, S. Li, S. Tang, Y. Ren, Y. Cen, W. Fei, B. Zhang, Y. Shen, and W. Lu, "Single-Cell RNA Sequencing Reveals the Tissue Architecture in Human High-Grade Serous Ovarian Cancer," *Clinical Cancer Research*, vol. 28, no. 16, pp. 3590–3602, Aug. 2022.

[98] P. Horak, C. Heining, S. Kreutzfeldt, B. Hutter, A. Mock, J. Hüllein, M. Fröhlich, S. Uhrig, A. Jahn, A. Rump, L. Gieldon, L. Möhrmann, D. Hanf, V. Teleanu, C. E. Heilig, D. B. Lipka, M. Allgäuer, L. Ruhnke, A. Laßmann, V. Endris, O. Neumann, R. Penzel, K. Beck, D. Richter, U. Winter, S. Wolf, K. Pfütze, C. Geörg, B. Meißburger, I. Buchhalter, M. Augustin, W. E. Aulitzky, P. Hohenberger, M. Kroiss, P. Schirmacher, R. F. Schlenk, U. Keilholz, F. Klauschen, G. Folprecht, S. Bauer, J. T. Siveke, C. H. Brandts, T. Kindler, M. Boerries, A. L. Illert, N. von Bubnoff, P. J. Jost, K. Spiekermann, M. Bitzer, K. Schulze-Osthoff, C. von Kalle, B. Klink, B. Brors, A. Stenzinger, E. Schröck, D. Hübschmann, W. Weichert, H. Glimm, and S. Fröhling, "Comprehensive Genomic and Transcriptomic Analysis for Guiding Therapeutic Decisions in Patients with Rare Cancers," *Cancer Discovery*, vol. 11, no. 11, pp. 2780–2795, Nov. 2021.

[99] X. Zhang, S. L. Marjani, Z. Hu, S. M. Weissman, X. Pan, and S. Wu, "Single-Cell Sequencing for Precise Cancer Research: Progress and Prospects," *Cancer Research*, vol. 76, no. 6, pp. 1305–1312, Mar. 2016.

[100] R. Bruno and G. Fontanini, "Next Generation Sequencing for Gene Fusion Analysis in Lung Cancer: A Literature Review," *Diagnostics*, vol. 10, no. 8, 2020.

[101] A. De Luca, R. Esposito Abate, A. M. Rachiglio, M. R. Maiello, C. Esposito, C. Schettino, F. Izzo, G. Nasti, and N. Normanno, "FGFR Fusions in Cancer: From Diagnostic Approaches to Therapeutic Intervention," *International Journal of Molecular Sciences*, vol. 21, no. 18, 2020.

[102] M. R. Waarts, A. J. Stonestrom, Y. C. Park, and R. L. Levine, "Targeting mutations in cancer," *The Journal of Clinical Investigation*, vol. 132, no. 8, Apr. 2022.

[103] B. Lim, Y. Lin, and N. Navin, "Advancing Cancer Research and Medicine with Single-Cell Genomics," *Cancer Cell*, vol. 37, no. 4, pp. 456–470, Apr. 2020.

[104] R. Colomer, R. Mondejar, N. Romero-Laorden, A. Alfranca, F. Sanchez-Madrid, and M. Quintela-Fandino, "When should we order a next generation sequencing test in a patient with cancer?" *eClinicalMedicine*, vol. 25, Aug. 2020.

[105] A. Saadatpour, S. Lai, G. Guo, and G.-C. Yuan, "Single-Cell Analysis in Cancer Genomics," *Trends in Genetics*, vol. 31, no. 10, pp. 576–586, Oct. 2015.

[106] N. Dizman, Z. E. Arslan, M. Feng, and S. K. Pal, "Sequencing Therapies for Metastatic Renal Cell Carcinoma," *Urologic Clinics*, vol. 47, no. 3, pp. 305–318, Aug. 2020.

[107] A. Buzdin, M. Sorokin, A. Garazha, A. Glusker, A. Aleshin, E. Poddubskaya, M. Sekacheva, E. Kim, N. Gaifullin, A. Giese, A. Seryakov, P. Rumiantsev, S. Moshkovskii, and A. Moiseev, "RNA sequencing for research and diagnostics in clinical oncology," *Enigmatic tumor properties associated with metastatic spread*, vol. 60, pp. 311–323, Feb. 2020.

[108] Y. Xiao and D. Yu, "Tumor microenvironment as a therapeutic target in cancer," *Pharmacology & Therapeutics*, vol. 221, p. 107753, May 2021.

[109] A. Nandwani, S. Rathore, and M. Datta, "LncRNAs in cancer: Regulatory and therapeutic implications," *Cancer Letters*, vol. 501, pp. 162–171, Mar. 2021.

[110] F. Marchetti, R. Cardoso, C. L. Chen, G. R. Douglas, J. Elloway, P. A. Escobar, T. Harper, R. H. Heflich, D. Kidd, A. M. Lynch, M. B. Myers, B. L. Parsons, J. J. Salk, R. S. Settivari, S. L. Smith-Roe, K. L. Witt, C. L. Yauk, R. Young, S. Zhang, and S. Minocherhomji, "Error-corrected next generation sequencing – Promises and challenges for genotoxicity and cancer risk assessment," *Mutation Research/Reviews in Mutation Research*, vol. 792, p. 108466, Jul. 2023.

[111] X. Chen, H. Zhu, C. Qiao, S. Zhao, L. Liu, Y. Wang, H. Jin, S. Qian, and Y. Wu, "Next-generation sequencing reveals gene mutations landscape and clonal evolution in patients with acute myeloid leukemia," *Hematology*, vol. 26, no. 1, pp. 111–122, Jan. 2021.

[112] N. E. Navin, "The first five years of single-cell cancer genomics and beyond," *Genome Research*, vol. 25, no. 10, pp. 1499–1507, Oct. 2015.

[113] N. A. Miller, E. G. Farrow, M. Gibson, L. K. Willig, G. Twist, B. Yoo, T. Marrs, S. Corder, L. Krivohlavek, A. Walter, J. E. Petrikin, C. J. Saunders, I. Thiffault, S. E. Soden, L. D. Smith, D. L. Dinwiddie, S. Herd, J. A. Cakici, S. Catreux, M. Ruehle, and S. F. Kingsmore, "A 26-hour system of highly sensitive whole genome sequencing for emergency management of genetic diseases," *Genome Medicine*, vol. 7, no. 1, p. 100, Sep. 2015.

[114] C. J. Saunders, N. A. Miller, S. E. Soden, D. L. Dinwiddie, A. Noll, N. A. Alnadi, N. Andraws, M. L. Patterson, L. A. Krivohlavek, J. Fellis, S. Humphray, P. Saffrey, Z. Kingsbury, J. C. Weir, J. Betley, R. J. Grocock, E. H. Margulies, E. G. Farrow, M. Artman, N. P. Safina, J. E. Petrikin, K. P. Hall, and S. F. Kingsmore, "Rapid Whole-Genome Sequencing for Genetic Disease Diagnosis in Neonatal Intensive Care Units," *Science Translational Medicine*, vol. 4, no. 154, pp. 154ra135–154ra135, Oct. 2012.

[115] S. E. Soden, C. J. Saunders, L. K. Willig, E. G. Farrow, L. D. Smith, J. E. Petrikin, J.-B. LePichon, N. A. Miller, I. Thiffault, D. L. Dinwiddie, G. Twist, A. Noll, B. A. Heese, L. Zellmer, A. M. Atherton, A. T. Abdelmoity, N. Safina, S. S. Nyp, B. Zuccarelli, I. A. Larson, A. Modrcin, S. Herd, M. Creed, Z. Ye, X. Yuan, R. A. Brodsky, and S. F. Kingsmore, "Effectiveness of exome and genome sequencing guided by acuity of illness for diagnosis of neurodevelopmental disorders," *Science Translational Medicine*, vol. 6, no. 265, pp. 265ra168–265ra168, Dec. 2014.

[116] J. R. Priest, S. R. Ceresnak, F. E. Dewey, L. E. Malloy-Walton, K. Dunn, M. E. Grove, M. V. Perez, K. Maeda, A. M. Dubin, and E. A. Ashley, "Molecular diagnosis of long QT syndrome at 10 days of life by rapid whole genome sequencing," *Heart Rhythm*, vol. 11, no. 10, pp. 1707–1713, Oct. 2014.

[117] L. K. Willig, J. E. Petrikin, L. D. Smith, C. J. Saunders, I. Thiffault, N. A. Miller, S. E. Soden, J. A. Cakici, S. M. Herd, G. Twist, A. Noll, M. Creed, P. M. Alba, S. L. Carpenter, M. A. Clements, R. T. Fischer, J. A. Hays, H. Kilbride, R. J. McDonough, J. L. Rosterman, S. L. Tsai, L. Zellmer, E. G. Farrow, and S. F. Kingsmore, "Whole-genome sequencing for identification of Mendelian disorders in critically ill infants: a retrospective analysis of diagnostic and clinical findings," *The Lancet Respiratory Medicine*, vol. 3, no. 5, pp. 377–387, May 2015.

[118] R. Mellis, K. Oprych, E. Scotchman, M. Hill, and L. S. Chitty, "Diagnostic yield of exome sequencing for prenatal diagnosis of fetal structural anomalies: A systematic review and meta-analysis," *Prenatal Diagnosis*, vol. 42, no. 6, pp. 662–685, May 2022.

[119] S. Best, K. Wou, N. Vora, I. B. Van der Veyver, R. Wapner, and L. S. Chitty, "Promises, pitfalls and practicalities of prenatal whole exome sequencing," *Prenatal Diagnosis*, vol. 38, no. 1, pp. 10–19, Jan. 2018.

[120] R. Sabbagh and I. B. Van den Veyver, "The current and future impact of genome-wide sequencing on fetal precision medicine," *Human Genetics*, vol. 139, no. 9, pp. 1121–1130, Sep. 2020.

[121] K. G. Monaghan, N. T. Leach, D. Pekarek, P. Prasad, N. C. Rose, and on behalf of the ACMG Professional Practice and Guidelines Committee, "The use of fetal exome sequencing in prenatal diagnosis: a points to consider document of the American College of Medical Genetics and Genomics (ACMG)," *Genetics in Medicine*, vol. 22, no. 4, pp. 675–680, Apr. 2020.

[122] T. Wang, K. Hoekzema, D. Vecchio, H. Wu, A. Sulovari, B. P. Coe, M. A. Gillentine, A. B. Wilfert, L. A. Perez-Jurado, M. Kvarnung, Y. Sleyp, R. K. Earl, J. A. Rosenfeld, M. R. Geisheker, L. Han, B. Du, C. Barnett, E. Thompson, M. Shaw, R. Carroll, K. Friend, R. Catford, E. E. Palmer, X. Zou, J. Ou, H. Li, H. Guo, J. Gerdts, E. Avola, G. Calabrese, M. Elia, D. Greco, A. Lindstrand, A. Nordgren, B.-M. Anderlid, G. Vandeweyer, A. Van Dijck, N. Van der Aa, B. McKenna, M. Hancarova, S. Bendova, M. Havlovicova, G. Malerba, B. D. Bernardina, P. Muglia, A. van Haeringen, M. J. V. Hoffer, B. Franke, G. Cappuccio, M. Delatycki, P. J. Lockhart, M. A. Manning, P. Liu, I. E. Scheffer, N. Brunetti-Pierri, N. Rommelse, D. G. Amaral, G. W. E. Santen, E. Trabetti, Z. Sedláček, J. J. Michaelson, K. Pierce, E. Courchesne, R. F. Kooy, J. Acampado, A. J. Ace, A. Amatya, I. Astrovskaya, A. Bashar, E. Brooks, M. E. Butler, L. A. Cartner, W. Chin, W. K. Chung, A. M. Daniels, P. Feliciano, C. Fleisch, S. Ganesan, W. Jensen, A. E. Lash, R. Marini, V. J. Myers, E. O'Connor, C. Rigby, B. E. Robertson, N. Shah, S. Shah, E. Singer, L. G. Snyder, A. N. Stephens, J. Tjernagel, B. M. Vernoia, N. Volfovsky, L. C. White, A. Hsieh, Y. Shen, X. Zhou, T. N. Turner, E. Bahl, T. R. Thomas, L. Brueggeman, T. Koomar, J. J. Michaelson, B. J. O'Roak, R. A. Barnard, R. A. Gibbs, D. Muzny, A. Sabo, K. L. B. Ahmed, E. E. Eichler, M. Siegel, L. Abbeduto, D. G. Amaral, B. A. Hilscher, D. Li, K. Smith, S. Thompson, C. Albright, E. M. Butter, S. Eldred, N. Hanna, M. Jones, D. L. Coury, J. Scherr, T. Pifher, E. Roby, B. Dennis, L. Higgins, M. Brown, M. Alessandri, A. Gutierrez, M. N. Hale, L. M. Herbert, H. L. Schneider, G. David, R. D. Annett, D. E. Sarver, I. Arriaga, A. Camba, A. C. Gulsrud, M. Haley, J. T. McCracken, S. Sandhu, M. Tafolla, W. S. Yang, L. A. Carpenter, C. C. Bradley, F. Gwynette, P. Manning, R. Shaffer, C. Thomas, R. A. Bernier, E. A. Fox, J. A. Gerdts, M. Pepper, T. Ho, D. Cho, J. Piven, H. Lechniak, L. V. Soorya, R. Gordon, A. Wainer, L. Yeh, C. Ochoa-Lubinoff, N. Russo, E. Berry-Kravis, S. Booker, C. A. Erickson, L. M. Prock, K. G. Pawlowski, E. T. Matthews, S. J. Brewster, M. A. Hojlo, E. Abada, E. Lamarche, T. Wang, S. C. Murali, W. T. Harvey, H. E. Kaplan, K. L. Pierce, L. DeMarco, S. Horner, J. Pandey, S. Plate, M. Sahin, K. D. Riley, E. Carmody, J. Constantini, A. Esler, A. Fatemi, H. Hutter, R. J. Landa, A. P. McKenzie, J. Neely, V. Singh, B. Van Metre, E. L. Wodka, E. J. Fombonne, L. Y. Huang-Storms, L. D. Pacheco, S. A. Mastel, L. A. Coppola, S. Francis, A. Jarrett, S. Jacob, N. Lillie, J. Gunderson, D. Istephanous, L. Simon, O. Wasserberg, A. L. Rachubinski, C. R. Rosenberg, S. M. Kanne, A. D. Shocklee, N. Takahashi, S. L. Bridwell, R. L. Klimczac, M. A. Mahurin, H. E. Cotrell, C. A. Grant, S. G. Hunter, C. L. Martin, C. M. Taylor, L. K. Walsh, K. A. Dent, A. Mason, A. Sziklay, C. J. Smith, M. Nordenskjöld, C. Romano, H. Peeters, R. A. Bernier, J. Gecz, K. Xia, E. E. Eichler, and The SPARK Consortium, "Large-scale targeted sequencing identifies risk genes for neurodevelopmental disorders," *Nature Communications*, vol. 11, no. 1, p. 4932, Oct. 2020.

[123] D. Guadagnolo, G. Mastromoro, F. Di Palma, A. Pizzuti, and E. Marchionni, "Prenatal Exome Sequencing: Background, Current Practice and Future Perspectives—A Systematic Review," *Diagnostics*, vol. 11, no. 2, 2021.

[124] A. Emms, J. Castleman, S. Allen, D. Williams, E. Kinning, and M. Kilby, "Next Generation Sequencing after Invasive Prenatal Testing in Fetuses with Congenital Malformations: Prenatal or Neonatal Investigation," *Genes*, vol. 13, no. 9, 2022.

[125] N. J. Chandler, E. Scotchman, R. Mellis, V. Ramachandran, R. Roberts, and L. S. Chitty, "Lessons learnt from prenatal exome sequencing," *Prenatal Diagnosis*, vol. 42, no. 7, pp. 831–844, Jun. 2022.

[126] H. Wang, F. Xiao, X. Dong, Y. Lu, G. Cheng, L. Wang, W. Lu, L. Yang, L. Chen, W. Kang, L. Li, X. Pan, Q. Wei, D. Zhuang, D. Chen, Z. Yin, L. Yang, Q. Ni, R. Liu, G. Li, P. Zhang, Y. Qian, X. Li, X. Peng, Y. Wang, F. Liu, D. Wang, H. Li, C. Shen, L. Qian, Y. Cao, B. Wu, and W. Zhou, "Diagnostic and clinical utility of next-generation sequencing in children born with multiple congenital anomalies in the China neonatal genomes project," *Human Mutation*, vol. 42, no. 4, pp. 434–444, Apr. 2021.

[127] R. Li, F. Fu, Q. Yu, D. Wang, X. Jing, Y. Zhang, F. Li, F. Li, J. Han, M. Pan, L. Zhen, D. Li, and C. Liao, "Prenatal exome sequencing in fetuses with congenital heart defects," *Clinical Genetics*, vol. 98, no. 3, pp. 215–230, Sep. 2020.

[128] F. Mone, H. Abu Subieh, S. Doyle, S. Hamilton, D. J. Mcmullan, S. Allen, T. Marton, D. Williams, and M. D. Kilby, "Evolving fetal phenotypes and clinical impact of progressive prenatal exome sequencing pathways: cohort study," *Ultrasound in Obstetrics & Gynecology*, vol. 59, no. 6, pp. 723–730, Jun. 2022.

[129] B. Poljak, U. Agarwal, Z. Alfirevic, S. Allen, N. Canham, J. Higgs, A. Kaelin Agten, A. Khalil, D. Roberts, F. Mone, and K. Navaratnam, "Prenatal exome sequencing and impact on perinatal outcome: cohort study," *Ultrasound in Obstetrics & Gynecology*, vol. 61, no. 3, pp. 339–345, Mar. 2023.

[130] M. A. de Koning, M. J. V. Hoffer, E. A. R. Nibbeling, E. K. Bijlsma, M. J. P. Toirkens, P. N. Adama-Scheltema, E. J. Verweij, M. B. Veenhof, G. W. E. Santen, and C. M. P. C. D. Peeters-Scholte, "Prenatal exome sequencing: A useful tool for the fetal neurologist," *Clinical Genetics*, vol. 101, no. 1, pp. 65–77, Jan. 2022.

[131] M. Kilby, "The role of next-generation sequencing in the investigation of ultrasound-identified fetal structural anomalies," *BJOG: An International Journal of Obstetrics & Gynaecology*, vol. 128, no. 2, pp. 420–429, Jan. 2021, publisher: John Wiley & Sons, Ltd.

[132] N. L. Vora and M. E. Norton, "Prenatal exome and genome sequencing for fetal structural abnormalities," *American Journal of Obstetrics and Gynecology*, vol. 228, no. 2, pp. 140–149, Feb. 2023.

[133] L. K. Tolusso, P. Hazelton, B. Wong, and D. T. Swarr, "Beyond diagnostic yield: prenatal exome sequencing results in maternal, neonatal, and familial clinical management changes," *Genetics in Medicine*, vol. 23, no. 5, pp. 909–917, May 2021.

[134] K. M. Bowling, M. L. Thompson, C. R. Finnila, S. M. Hiatt, D. R. Latner, M. D. Amaral, J. M. Lawlor, K. M. East, M. E. Cochran, V. Greve, W. V. Kelley, D. E. Gray, S. A. Felker, H. Meddaugh, A. Cannon, A. Luedecke, K. E. Jackson, L. G. Hendon, H. M. Janani, M. Johnston, L. A. Merin, S. L. Deans, C. Tuura, H. Williams, K. Laborde, M. B. Neu, J. Patrick-Esteve, A. C. Hurst, J. Kandasamy, W. Carlo, K. B. Brothers, B. M. Kirmse, R. Savich, D. Superneau, S. B. Spedale, S. J. Knight, G. S. Barsh, B. R. Korf, and G. M. Cooper, "Genome sequencing as a first-line diagnostic test for hospitalized infants," *Genetics in Medicine*, vol. 24, no. 4, pp. 851–861, Apr. 2022.

[135] B. Rinaldi, V. Race, A. Corveleyn, E. Van Hoof, M. Bauters, K. Van Den Bogaert, E. Denayer, T. de Ravel, E. Legius, M. Baldewijns, M. Aertsen, L. Lewi, L. De Catte, J. Breckpot, and K. Devriendt, "Next-generation sequencing in prenatal setting: Some examples of unexpected variant association," *European Journal of Medical Genetics*, vol. 63, no. 5, p. 103875, May 2020.

[136] L. Lei, L. Zhou, and J.-j. Xiong, "Whole-exome sequencing increases the diagnostic rate for prenatal fetal structural anomalies," *European Journal of Medical Genetics*, vol. 64, no. 9, p. 104288, Sep. 2021.

[137] M. P. Zalusky, J. A. Gustafson, S. C. Bohaczuk, B. Mallory, P. Reed, T. Wenger, E. Beckman, I. J. Chang, C. R. Paschal, J. G. Buchan, C. M. Lockwood, M. Puia-Dumitrescu, D. R. Garalde, J. Guillory, A. J. Markham, M. J. Bamshad, E. E. Eichler, A. B. Stergachis, and D. E. Miller, "3-hour genome sequencing and targeted analysis to rapidly assess genetic risk," *Genetics in Medicine Open*, vol. 2, p. 101833, Jan. 2024.

[138] T. Dunn, H. Sadasivan, J. Wadden, K. Goliya, K.-Y. Chen, D. Blaauw, R. Das, and S. Narayanasamy, "SquiggleFilter: An accelerator for portable virus detection," in *MICRO*, 2021.

[139] E. R. Robinson, T. M. Walker, and M. J. Pallen, "Genomics and outbreak investigation: from sequence to consequence," *Genome Medicine*, vol. 5, no. 4, p. 36, Apr. 2013.

[140] P.-E. Fournier, G. Dubourg, and D. Raoult, "Clinical detection and characterization of bacterial pathogens in the genomics era," *Genome Medicine*, vol. 6, no. 11, p. 114, Nov. 2014.

[141] C. U. Köser, M. J. Ellington, E. J. P. Cartwright, S. H. Gillespie, N. M. Brown, M. Farrington, M. T. G. Holden, G. Dougan, S. D. Bentley, J. Parkhill, and S. J. Peacock, "Routine Use of Microbial Whole Genome Sequencing in Diagnostic and Public Health Microbiology," *PLOS Pathogens*, vol. 8, no. 8, p. e1002824, Aug. 2012.

[142] M. Eloit and m. lecuit, "The diagnosis of infectious diseases by whole genome next generation sequencing: a new era is opening," *Frontiers in Cellular and Infection Microbiology*, vol. 4, 2014.

[143] Gardy Jennifer L., Johnston James C., Sui Shannan J. Ho, Cook Victoria J., Shah Lena, Brodkin Elizabeth, Rempel Shirley, Moore Richard, Zhao Yongjun, Holt Robert, Varhol Richard, Birol Inanc, Lem Marcus, Sharma Meenu K., Elwood Kevin, Jones Steven J.M., Brinkman Fiona S.L., Brunham Robert C., and Tang Patrick, "Whole-Genome Sequencing and Social-Network Analysis of a Tuberculosis Outbreak," *New England Journal of Medicine*, vol. 364, no. 8, pp. 730–739, 2011.

[144] Taylor Angela J., Lappi Victoria, Wolfgang William J., Lapierre Pascal, Palumbo Michael J., Medus Carlota, and Boxrud David, "Characterization of Foodborne Outbreaks of Salmonella enterica Serovar Enteritidis with Whole-Genome Sequencing Single Nucleotide Polymorphism-Based Analysis for Surveillance and Outbreak Detection," *Journal of Clinical Microbiology*, vol. 53, no. 10, pp. 3334–3340, Sep. 2015.

[145] Quainoo Scott, Coolen Jordy P. M., van Hijum Sacha A. F. T., Huynen Martijn A., Melchers Willem J. G., van Schaik Willem, and Wertheim Heiman F. L., "Whole-Genome Sequencing of Bacterial Pathogens: the Future of Nosocomial Outbreak Analysis," *Clinical Microbiology Reviews*, vol. 30, no. 4, pp. 1015–1063, Aug. 2017.

[146] Goldberg Brittany, Sichtig Heike, Geyer Chelsie, Ledeboer Nathan, and Weinstock George M., "Making the Leap from Research Laboratory to Clinic: Challenges and Opportunities for Next-Generation Sequencing in Infectious Disease Diagnostics," *mBio*, vol. 6, no. 6, pp. 10.1128/mbio.01 888–15, Dec. 2015.

[147] Gilchrist Carol A., Turner Stephen D., Riley Margaret F., Petri William A., and Hewlett Erik L., "Whole-Genome Sequencing in Outbreak Analysis," *Clinical Microbiology Reviews*, vol. 28, no. 3, pp. 541–563, Apr. 2015.

[148] A. Arias, S. J. Watson, D. Asogun, E. A. Tobin, J. Lu, M. V. T. Phan, U. Jah, R. E. G. Wadoum, L. Meredith, L. Thorne, S. Caddy, A. Tarawalie, P. Langat, G. Dudas, N. R. Faria, S. Dellicour, A. Kamara, B. Kargbo, B. O. Kamara, S. Gevao, D. Cooper, M. Newport, P. Horby, J. Dunning, F. Sahr, T. Brooks, A. J. Simpson, E. Groppelli, G. Liu, N. Mulakken, K. Rhodes, J. Akpablie, Z. Yoti, M. Lamunu, E. Vitto, P. Otim, C. Owilli, I. Boateng, L. Okoror, E. Omomoh, J. Oyakhilome, R. Omiunu, I. Yemisis, D. Adomeh, S. Ehikhiametalor, P. Akhilomen, C. Aire, A. Kurth, N. Cook, J. Baumann, M. Gabriel, R. Wölfel, A. Di Caro, M. W. Carroll, S. Günther, J. Redd, D. Naidoo, O. G. Pybus, A. Rambaut, P. Kellam, I. Goodfellow, and M. Cotten, "Rapid outbreak sequencing of Ebola virus in Sierra Leone identifies transmission chains linked to sporadic cases," *Virus Evolution*, vol. 2, no. 1, p. vew016, Jan. 2016.

[149] J. M. Besser, H. A. Carleton, E. Trees, S. G. Stroika, K. Hise, M. Wise, and P. Gerner-Smidt, "Interpretation of Whole-Genome Sequencing for Enteric Disease Surveillance and Outbreak Investigation," *Foodborne Pathogens and Disease*, vol. 16, no. 7, pp. 504–512, Jul. 2019.

[150] W. Li, Q. Cui, L. Bai, P. Fu, H. Han, J. Liu, and Y. Guo, "Application of Whole-Genome Sequencing in the National Molecular Tracing Network for Foodborne Disease Surveillance in China," *Foodborne Pathogens and Disease*, vol. 18, no. 8, pp. 538–546, Aug. 2021.

[151] Y. Deng, M. Jiang, P. S. Kwan, C. Yang, Q. Chen, Y. Lin, Y. Qiu, Y. Li, X. Shi, L. Li, Y. Cui, Q. Sun, and Q. Hu, "Integrated Whole-Genome Sequencing Infrastructure for Outbreak Detection and Source Tracing of Salmonella enterica Serotype Enteritidis," *Foodborne Pathogens and Disease*, vol. 18, no. 8, pp. 582–589, Aug. 2021.

[152] C. Bertelli and G. Greub, "Rapid bacterial genome sequencing: methods and applications in clinical microbiology," *Clinical Microbiology and Infection*, vol. 19, no. 9, pp. 803–813, Sep. 2013.

[153] J. Kwong, N. Mccallum, V. Sintchenko, and B. Howden, "Whole genome sequencing in clinical and public health microbiology," *Pathology*, vol. 47, no. 3, pp. 199–210, Apr. 2015.

[154] R. H. Deurenberg, E. Bathoorn, M. A. Chlebowicz, N. Couto, M. Ferdous, S. García-Cobos, A. M. Kooistra-Smid, E. C. Raangs, S. Rosema, A. C. Veloo, K. Zhou, A. W. Friedrich, and J. W. Rossen, "Application of next generation sequencing in clinical microbiology and infection prevention," *Journal of Biotechnology*, vol. 243, pp. 16–24, Feb. 2017.

[155] P. Tang, M. A. Croxen, M. R. Hasan, W. W. Hsiao, and L. M. Hoang, "Infection control in the new age of genomic epidemiology," *American Journal of Infection Control*, vol. 45, no. 2, pp. 170–179, Feb. 2017.

[156] N. J. Croucher and X. Didelot, "The application of genomics to tracing bacterial pathogen transmission," *Host–microbe interactions: bacteria • Genomics*, vol. 23, pp. 62–67, Feb. 2015.

[157] J. Comin, A. Chaure, A. Cebollada, D. Ibarz, J. Viñuelas, M. A. Vitoria, M. J. Iglesias, and S. Samper, "Investigation of a rapidly spreading tuberculosis outbreak using whole-genome sequencing," *Infection, Genetics and Evolution*, vol. 81, p. 104184, Jul. 2020.

[158] J. S. Johnson, D. J. Spakowicz, B.-Y. Hong, L. M. Petersen, P. Demkowicz, L. Chen, S. R. Leopold, B. M. Hanson, H. O. Agresta, M. Gerstein, E. Sodergren, and G. M. Weinstock, "Evaluation of 16S rRNA gene sequencing for species and strain-level microbiome analysis," *Nat. Comm.*, 2019.

[159] M. Land, L. Hauser, S.-R. Jun, I. Nookaew, M. R. Leuze, T.-H. Ahn, T. Karpinets, O. Lund, G. Kora, T. Wassenaar, S. Poudel, and D. W. Ussery, "Insights from 20 years of bacterial genome sequencing," *Functional & Integrative Genomics*, vol. 15, no. 2, pp. 141–161, Mar. 2015.

[160] J. Galloway-Peña and B. Hanson, "Tools for Analysis of the Microbiome," *Digestive Diseases and Sciences*, vol. 65, no. 3, pp. 674–685, Mar. 2020.

[161] E.-J. Song, E.-S. Lee, and Y.-D. Nam, "Progress of analytical tools and techniques for human gut microbiome research," *Journal of Microbiology*, vol. 56, no. 10, pp. 693–705, Oct. 2018.

[162] C. Y. Chiu and S. A. Miller, "Clinical metagenomics," *Nature Reviews Genetics*, vol. 20, no. 6, pp. 341–355, Jun. 2019.

[163] B. Lötstedt, M. Stražar, R. Xavier, A. Regev, and S. Vickovic, "Spatial host–microbiome sequencing reveals niches in the mouse gut," *Nature Biotechnology*, Nov. 2023.

[164] M. J. Strong, G. Xu, L. Morici, S. Splinter Bon-Durant, M. Baddoo, Z. Lin, C. Fewell, C. M. Taylor, and E. K. Flemington, "Microbial Contamination in Next Generation Sequencing: Implications for Sequence-Based Analysis of Clinical Samples," *PLOS Pathogens*, vol. 10, no. 11, p. e1004437, Nov. 2014.

[165] G. Galazzo, N. van Best, B. J. Benedikter, K. Janssen, L. Bervoets, C. Driessen, M. Oomen, M. Lucchesi, P. H. van Eijck, H. E. F. Becker, M. W. Hornef, P. H. Savelkoul, F. R. M. Stassen, P. F. Wolffs, and J. Penders, "How to Count Our Microbes? The Effect of Different Quantitative Microbiome Profiling Approaches," *Frontiers in Cellular and Infection Microbiology*, vol. 10, 2020.

[166] M. A. Malla, A. Dubey, A. Kumar, S. Yadav, A. Hashem, and E. F. Abd_Allah, "Exploring the Human Microbiome: The Potential Future Role of Next-Generation Sequencing in Disease Diagnosis and Treatment," *Frontiers in Immunology*, vol. 9, 2019.

[167] J. Jovel, J. Patterson, W. Wang, N. Hotte, S. O'Keefe, T. Mitchel, T. Perry, D. Kao, A. L. Mason, K. L. Madsen, and G. K.-S. Wong, "Characterization of the Gut Microbiome Using 16S or Shotgun Metagenomics," *Frontiers in Microbiology*, vol. 7, 2016.

[168] P. Lepage, M. C. Leclerc, M. Joossens, S. Mondot, H. M. Blottière, J. Raes, D. Ehrlich, and J. Doré, "A metagenomic insight into our gut's microbiome," *Gut*, vol. 62, no. 1, p. 146, Jan. 2013.

[169] G. Bhagwat, Q. Zhu, W. O'Connor, S. Subashchandrabose, I. Grainge, R. Knight, and T. Palanisami, "Exploring the Composition and Functions of Plastic Microbiome Using Whole-Genome Sequencing," *Environmental Science & Technology*, vol. 55, no. 8, pp. 4899–4913, Apr. 2021.

[170] B. Gao, L. Chi, Y. Zhu, X. Shi, P. Tu, B. Li, J. Yin, N. Gao, W. Shen, and B. Schnabl, "An Introduction to Next Generation Sequencing Bioinformatic Analysis in Gut Microbiome Studies," *Biomolecules*, vol. 11, no. 4, 2021.

[171] Comeau André M., Douglas Gavin M., and Langille Morgan G. I., "Microbiome Helper: a Custom and Streamlined Workflow for Microbiome Research," *mSystems*, vol. 2, no. 1, pp. 10.1128/msystems.00 127–16, Jan. 2017.

[172] L. A. Cummings, K. Kurosawa, D. R. Hoogestraat, D. J. SenGupta, F. Candra, M. Doyle, S. Thielges, T. A. Land, C. A. Rosenthal, N. G. Hoffman, S. J. Salipante, and B. T. Cookson, "Clinical Next Generation Sequencing Outperforms Standard Microbiological Culture for Characterizing Polymicrobial Samples," *Clinical Chemistry*, vol. 62, no. 11, pp. 1465–1473, Nov. 2016.

[173] M. J. Cox, W. O. Cookson, and M. F. Moffatt, "Sequencing the human microbiome in health and disease," *Human Molecular Genetics*, vol. 22, no. R1, pp. R88–R94, Oct. 2013.

[174] R. Bharti and D. G. Grimm, "Current challenges and best-practice protocols for microbiome analysis," *Briefings in Bioinformatics*, vol. 22, no. 1, pp. 178–193, Jan. 2021.

[175] S. Bashiardes, G. Zilberman-Schapira, and E. Elinav, "Use of Metatranscriptomics in Microbiome Research," *Bioinformatics and Biology Insights*, vol. 10, p. BBI.S34610, Jan. 2016.

[176] I. Laudadio, V. Fulci, F. Palone, L. Stronati, S. Cucchiara, and C. Carissimi, "Quantitative Assessment of Shotgun Metagenomics and 16S rDNA Amplicon Sequencing in the Study of Human Gut Microbiome," *OMICS: A Journal of Integrative Biology*, vol. 22, no. 4, pp. 248–254, Apr. 2018.

[177] C. R. Wensel, J. L. Pluznick, S. L. Salzberg, and C. L. Sears, "Next-generation sequencing: insights to advance clinical investigations of the microbiome," *The Journal of Clinical Investigation*, vol. 132, no. 7, Apr. 2022.

[178] S. L. Mitchell and P. J. Simner, "Next-Generation Sequencing in Clinical Microbiology: Are We There Yet?" *Clinics in Laboratory Medicine*, vol. 39, no. 3, pp. 405–418, Sep. 2019.

[179] T. Ojala, A.-E. Häkkinen, E. Kankuri, and M. Kankainen, "Current concepts, advances, and challenges in deciphering the human microbiota with metatranscriptomics," *Trends in Genetics*, vol. 39, no. 9, pp. 686–702, Sep. 2023.

[180] R. Ranjan, A. Rani, A. Metwally, H. S. McGee, and D. L. Perkins, "Analysis of the microbiome: Advantages of whole genome shotgun versus 16S amplicon sequencing," *Biochemical and Biophysical Research Communications*, vol. 469, no. 4, pp. 967–977, Jan. 2016.

[181] P. C. Kashyap, N. Chia, H. Nelson, E. Segal, and E. Elinav, "Microbiome at the Frontier of Personalized Medicine," *Mayo Clinic Proceedings*, vol. 92, no. 12, pp. 1855–1864, Dec. 2017.

[182] A. M. Fricker, D. Podlesny, and W. F. Fricke, "What is new and relevant for sequencing-based microbiome research? A mini-review," *Special Issue on Plant Microbiome*, vol. 19, pp. 105–112, Sep. 2019.

[183] The Arabidopsis Genome Initiative, "Analysis of the genome sequence of the flowering plant Arabidopsis thaliana," *Nature*, vol. 408, no. 6814, pp. 796–815, Dec. 2000.

[184] H. Zhu, C. Li, and C. Gao, "Applications of CRISPR–Cas in agriculture and plant biotechnology," *Nature Reviews Molecular Cell Biology*, vol. 21, no. 11, pp. 661–677, Nov. 2020.

[185] J. Y. Choi, Z. N. Lye, S. C. Groen, X. Dai, P. Rughani, S. Zaaijer, E. D. Harrington, S. Juul, and M. D. Purugganan, "Nanopore sequencing-based genome assembly and evolutionary genomics of circum-basmati rice," *Genome Biology*, vol. 21, no. 1, p. 21, Feb. 2020.

[186] K. A. Stevens, J. L. Wegrzyn, A. Zimin, D. Puiu, M. Crepeau, C. Cardeno, R. Paul, D. Gonzalez-Ibeas, M. Koriabine, A. E. Holtz-Morris, P. J. Martínez-García, U. U. Sezen, G. Marçais, K. Jermstad, P. E. McGuire, C. A. Loopstra, J. M. Davis, A. Eckert, P. de Jong, J. A. Yorke, S. L. Salzberg, D. B. Neale, and C. H. Langley, "Sequence of the Sugar Pine Megagenome," *Genetics*, vol. 204, no. 4, pp. 1613–1626, Dec. 2016.

[187] M. D. Campos, M. d. R. Félix, M. Patanita, P. Materatski, and C. Varanda, "High throughput sequencing unravels tomato-pathogen interactions towards a sustainable plant breeding," *Horticulture Research*, vol. 8, p. 171, Jan. 2021.

[188] C. Gao, "Genome engineering for crop improvement and future agriculture," *Cell*, vol. 184, no. 6, pp. 1621–1635, Mar. 2021, publisher: Elsevier. [Online]. Available: https://doi.org/10.1016/j.cell.2021.01.005

[189] A. D. J. van Dijk, G. Kootstra, W. Kruijer, and D. de Ridder, "Machine learning in plant science and plant breeding," *iScience*, vol. 24, no. 1, Jan. 2021.

[190] Y. Sun, L. Shang, Q.-H. Zhu, L. Fan, and L. Guo, "Twenty years of plant genome sequencing: achievements and challenges," *Trends in Plant Science*, vol. 27, no. 4, pp. 391–401, Apr. 2022.

[191] K. D. Kim, Y. Kang, and C. Kim, "Application of Genomic Big Data in Plant Breeding: Past, Present, and Future," *Plants*, vol. 9, no. 11, 2020.

[192] M. Thudi, R. Palakurthi, J. C. Schnable, A. Chitikineni, S. Dreisigacker, E. Mace, R. K. Srivastava, C. T. Satyavathi, D. Odeny, V. K. Tiwari, H.-M. Lam, Y. B. Hong, V. K. Singh, G. Li, Y. Xu, X. Chen, S. Kaila, H. Nguyen, S. Sivasankar, S. A. Jackson, T. J. Close, W. Shubo, and R. K. Varshney, "Genomic resources in plant breeding for sustainable agriculture," *Journal of Plant Physiology*, vol. 257, p. 153351, Feb. 2021.

[193] T. P. Michael and R. VanBuren, "Building near-complete plant genomes," *Genome studies and molecular genetics*, vol. 54, pp. 26–33, Apr. 2020.

[194] Y. Shen, G. Zhou, C. Liang, and Z. Tian, "Omics-based interdisciplinarity is accelerating plant breeding," *Current Opinion in Plant Biology*, vol. 66, p. 102167, Apr. 2022.

[195] T. Shahroodi, M. Zahedi, C. Firtina, M. Alser, S. Wong, O. Mutlu, and S. Hamdioui, "Demeter: A fast and energy-efficient food profiler using hyperdimensional computing in memory," *IEEE Access*, 2022.

[196] B. Bruijns, R. Tiggelaar, and H. Gardeniers, "Massively parallel sequencing techniques for forensics: A review," *Electrophoresis*, 2018.

[197] D. Ballard, J. Winkler-Galicki, and J. Wesoły, "Massive parallel sequencing in forensics: advantages, issues, technicalities, and prospects," *International Journal of Legal Medicine*, vol. 134, no. 4, pp. 1291–1303, Jul. 2020.

[198] M. Smith and S. Miller, "The Evolution of Forensic Genomics: Regulating Massively Parallel Sequencing," *Journal of Bioethical Inquiry*, vol. 21, no. 2, pp. 365–372, Jun. 2024.

[199] R. Ogden, N. Vasiljevic, and S. Prost, "Nanopore sequencing in non-human forensic genetics," *Emerging Topics in Life Sciences*, vol. 5, no. 3, pp. 465–473, May 2021.

[200] Y. Yang, B. Xie, and J. Yan, "Application of Next-Generation Sequencing Technology in Forensic Science," *Genomics, Proteomics & Bioinformatics*, vol. 12, no. 5, pp. 190–197, Oct. 2014.

[201] O. Tytgat, Y. Gansemans, J. Weymaere, K. Rubben, D. Deforce, and F. Van Nieuwerburgh, "Nanopore Sequencing of a Forensic STR Multiplex Reveals Loci Suitable for Single-Contributor STR Profiling," *Genes*, vol. 11, no. 4, 2020.

[202] M. Diepenbroek, B. Bayer, and K. Anslinger, "Pushing the Boundaries: Forensic DNA Phenotyping Challenged by Single-Cell Sequencing," *Genes*, vol. 12, no. 9, 2021.

[203] A. Michaleas, P. Fremont-Smith, C. Lennartz, and D. O. Ricke, "Parallel Computing with DNA Forensics Data," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, 2022, pp. 1–7.

[204] R. Daniel, C. Santos, C. Phillips, M. Fondevila, R. van Oorschot, A. Carracedo, M. Lareu, and D. McNevin, "A SNaPshot of next generation sequencing for forensic SNP analysis," *Forensic Science International: Genetics*, vol. 14, pp. 50–60, Jan. 2015.

[205] C. Børsting and N. Morling, "Next generation sequencing and its applications in forensic genetics," *New Trends in Forensic Science Genetics*, vol. 18, pp. 78–89, Sep. 2015.

[206] Y.-Y. Liu and S. Harbison, "A review of bioinformatic methods for forensic DNA analyses," *Forensic Science International: Genetics*, vol. 33, pp. 117–128, Mar. 2018.

[207] S. Köcher, P. Müller, B. Berger, M. Bodner, W. Parson, L. Roewer, and S. Willuweit, "Interlaboratory validation study of the ForenSeq™ DNA Signature Prep Kit," *Forensic Science International: Genetics*, vol. 36, pp. 77–85, Sep. 2018.

[208] P. de Knijff, "From next generation sequencing to now generation sequencing in forensics," *Forensic Science International: Genetics*, vol. 38, pp. 175–180, Jan. 2019.

[209] N. Gandotra, W. C. Speed, W. Qin, Y. Tang, A. J. Pakstis, K. K. Kidd, and C. Scharfe, "Validation of novel forensic DNA markers using multiplex microhaplotype sequencing," *Forensic Science International: Genetics*, vol. 47, p. 102275, Jul. 2020.

[210] C. Haas, J. Neubauer, A. P. Salzmann, E. Hanson, and J. Ballantyne, "Forensic transcriptome analysis using massively parallel sequencing," *Forensic Science International: Genetics*, vol. 52, p. 102486, May 2021.

[211] M. M. Foley and F. Oldoni, "A global snapshot of current opinions of next-generation sequencing technologies usage in forensics," *Forensic Science International: Genetics*, vol. 63, p. 102819, Mar. 2023.

[212] J. A. Ariza, A. H. M. Rocha, R. C. Pérez, and C. I. Bermudez-Santana, "Next-generation sequencing of postmortem molecular markers to support for medicolegal autopsy," *Forensic Science International: Reports*, vol. 6, p. 100300, Dec. 2022.

[213] K. A. Meiklejohn, M. K. Burnham-Curtis, D. J. Straughan, J. Giles, and M. K. Moore, "Current methods, future directions and considerations of DNA-based taxonomic identification in wildlife forensics," *Forensic Science International: Animals and Environments*, vol. 1, p. 100030, Nov. 2021.

[214] M. J. Alvarez-Cubero, M. Saiz, B. Martínez-García, S. M. Sayalero, C. Entrala, J. A. Lorente, and L. J. Martinez-Gonzalez, "Next generation sequencing: an application in forensic sciences?" *Annals of Human Biology*, vol. 44, no. 7, pp. 581–592, Oct. 2017.

[215] D. R. Bentley, S. Balasubramanian, H. P. Swerdlow, G. P. Smith, J. Milton, C. G. Brown, K. P. Hall, D. J. Evers, C. L. Barnes, H. R. Bignell, J. M. Boutell, J. Bryant, R. J. Carter, R. Keira Cheetham, A. J. Cox, D. J. Ellis, M. R. Flatbush, N. A. Gormley, S. J. Humphray, L. J. Irving, M. S. Karbelashvili, S. M. Kirk, H. Li, X. Liu, K. S. Maisinger, L. J. Murray, B. Obradovic, T. Ost, M. L. Parkinson, M. R. Pratt, I. M. J. Rasolonjatovo, M. T. Reed, R. Rigatti, C. Rodighiero, M. T. Ross, A. Sabot, S. V. Sankar, A. Scally, G. P. Schroth, M. E. Smith, V. P. Smith, A. Spiridou, P. E. Torrance, S. S. Tzonev, E. H. Vermaas, K. Walter, X. Wu, L. Zhang, M. D. Alam, C. Anastasi, I. C. Aniebo, D. M. D. Bailey, I. R. Bancarz, S. Banerjee, S. G. Barbour, P. A. Baybayan, V. A. Benoit, K. F. Benson, C. Bevis, P. J. Black, A. Boodhun, J. S. Brennan, J. A. Bridgham, R. C. Brown, A. A. Brown, D. H. Buermann, A. A. Bundu, J. C. Burrows, N. P. Carter, N. Castillo, M. Chiara E. Catenazzi, S. Chang, R. Neil Cooley, N. R. Crake, O. O. Dada, K. D. Diakoumakos, B. Dominguez-Fernandez, D. J. Earnshaw, U. C. Egbujor, D. W. Elmore, S. S. Etchin, M. R. Ewan, M. Fedurco, L. J. Fraser, K. V. Fuentes Fajardo, W. Scott Furey, D. George, K. J. Gietzen, C. P. Goddard, G. S. Golda, P. A. Granieri, D. E. Green, D. L. Gustafson, N. F. Hansen, K. Harnish, C. D. Haudenschild, N. I. Heyer, M. M. Hims, J. T. Ho, A. M. Horgan, K. Hoschler, S. Hurwitz, D. V. Ivanov, M. Q. Johnson, T. James, T. A. Huw Jones, G.-D. Kang, T. H. Kerelska, A. D. Kersey, I. Khrebtukova, A. P. Kindwall, Z. Kingsbury, P. I. Kokko-Gonzales, A. Kumar, M. A. Laurent, C. T. Lawley, S. E. Lee, X. Lee, A. K. Liao, J. A. Loch, M. Lok, S. Luo, R. M. Mammen, J. W. Martin, P. G. McCauley, P. McNitt, P. Mehta, K. W. Moon, J. W. Mullens, T. Newington, Z. Ning, B. Ling Ng, S. M. Novo, M. J. O'Neill, M. A. Osborne, A. Osnowski, O. Ostadan, L. L. Paraschos, L. Pickering, A. C. Pike, A. C. Pike, D. Chris Pinkard, D. P. Pliskin, J. Podhasky, V. J. Quijano, C. Raczy, V. H. Rae, S. R. Rawlings, A. Chiva Rodriguez, P. M. Roe, J. Rogers, M. C. Rogert Bacigalupo, N. Romanov, A. Romieu, R. K. Roth, N. J. Rourke, S. T. Ruediger, E. Rusman, R. M. Sanches-Kuiper, M. R. Schenker, J. M. Seoane, R. J. Shaw, M. K. Shiver, S. W. Short, N. L. Sizto, J. P. Sluis, M. A. Smith, J. Ernest Sohna Sohna, E. J. Spence, K. Stevens, N. Sutton, L. Szajkowski, C. L. Tregidgo, G. Turcatti, S. vandeVondele, Y. Verhovsky, S. M. Virk, S. Wakelin, G. C. Walcott, J. Wang, G. J. Worsley, J. Yan, L. Yau, M. Zuerlein, J. Rogers, J. C. Mullikin, M. E. Hurles, N. J. McCooke, J. S. West, F. L. Oaks, P. L. Lundberg, D. Klenerman, R. Durbin, and A. J. Smith, "Accurate whole human genome sequencing using reversible terminator chemistry," *Nature*, 2008.

[216] M. Margulies, M. Egholm, W. E. Altman, S. Attiya, J. S. Bader, L. A. Bemben, J. Berka, M. S. Braverman, Y.-J. Chen, Z. Chen, S. B. Dewell, L. Du, J. M. Fierro, X. V. Gomes, B. C. Godwin, W. He, S. Helgesen, C. H. Ho, G. P. Irzyk, S. C. Jando, M. L. I. Alenquer, T. P. Jarvie, K. B. Jirage, J.-B. Kim, J. R. Knight, J. R. Lanza, J. H. Leamon, S. M. Lefkowitz, M. Lei, J. Li, K. L. Lohman, H. Lu, V. B. Makhijani, K. E. McDade, M. P. McKenna, E. W. Myers, E. Nickerson, J. R. Nobile, R. Plant, B. P. Puc, M. T. Ronan, G. T. Roth, G. J. Sarkis, J. F. Simons, J. W. Simpson, M. Srinivasan, K. R. Tartaro, A. Tomasz, K. A. Vogt, G. A. Volkmer, S. H. Wang, Y. Wang, M. P. Weiner, P. Yu, R. F. Begley, and J. M. Rothberg, "Genome sequencing in microfabricated high-density picolitre reactors," *Nature*, vol. 437, no. 7057, pp. 376–380, Sep. 2005.

[217] J. Shendure, G. J. Porreca, N. B. Reppas, X. Lin, J. P. McCutcheon, A. M. Rosenbaum, M. D. Wang, K. Zhang, R. D. Mitra, and G. M. Church, "Accurate Multiplex Polony Sequencing of an Evolved Bacterial Genome," *Science*, vol. 309, no. 5741, pp. 1728–1732, Sep. 2005.

[218] T. D. Harris, P. R. Buzby, H. Babcock, E. Beer, J. Bowers, I. Braslavsky, M. Causey, J. Colonell, J. DiMeo, J. W. Efcavitch, E. Giladi, J. Gill, J. Healy, M. Jarosz, D. Lapen, K. Moulton, S. R. Quake, K. Steinmann, E. Thayer, A. Tyurina, R. Ward, H. Weiss, and Z. Xie, "Single-Molecule DNA Sequencing of a Viral Genome," *Science*, vol. 320, no. 5872, pp. 106–109, Apr. 2008.

[219] G. Turcatti, A. Romieu, M. Fedurco, and A.-P. Tairi, "A new class of cleavable fluorescent nucleotides: synthesis and optimization as reversible terminators for DNA sequencing by synthesis †," *Nucleic Acids Research*, vol. 36, no. 4, pp. e25–e25, Mar. 2008.

[220] W. Wu, B. P. Stupi, V. A. Litosh, D. Mansouri, D. Farley, S. Morris, S. Metzker, and M. L. Metzker, "Termination of DNA synthesis by N6 -alkylated, not 3'- O -alkylated, photocleavable 2'-deoxyadenosine triphosphates," *Nucleic Acids Research*, vol. 35, no. 19, pp. 6339–6349, Oct. 2007.

[221] C. W. Fuller, "Rapid parallel nucleic acid analysis," Patent, Sep., 2007.

[222] K. McKernan, A. Blanchard, L. Kotler, and G. Costa, "Reagents, methods, and libraries for bead-based sequencing," Patent, Jan., 2008.

[223] C. W. Fuller and J. R. Nelson, "Method for nucleic acid analysis," Patent, Jan., 2011.

[224] J. Eid, A. Fehr, J. Gray, K. Luong, J. Lyle, G. Otto, P. Peluso, D. Rank, P. Baybayan, B. Bettman, A. Bibillo, K. Bjornson, B. Chaudhuri, F. Christians, R. Cicero, S. Clark, R. Dalal, A. deWinter, J. Dixon, M. Foquet, A. Gaertner, P. Hardenbol, C. Heiner, K. Hester, D. Holden, G. Kearns, X. Kong, R. Kuse, Y. Lacroix, S. Lin, P. Lundquist, C. Ma, P. Marks, M. Maxham, D. Murphy, I. Park, T. Pham, M. Phillips, J. Roy, R. Sebra, G. Shen, J. Sorenson, A. Tomaney, K. Travers, M. Trulson, J. Vieceli, J. Wegener, D. Wu, A. Yang, D. Zaccarin, P. Zhao, F. Zhong, J. Korlach, and S. Turner, "Real-Time DNA Sequencing from Single Polymerase Molecules," *Science*, 2009.

[225] G. Menestrina, "Ionic channels formed byStaphylococcus aureus alpha-toxin: Voltage-dependent inhibition by divalent and trivalent cations," *The Journal of Membrane Biology*, vol. 90, no. 2, pp. 177–190, Jun. 1986.

[226] G. M. Cherf, K. R. Lieberman, H. Rashid, C. E. Lam, K. Karplus, and M. Akeson, "Automated forward and reverse ratcheting of DNA in a nanopore at 5-Å precision," *Nature Biotechnology*, vol. 30, no. 4, pp. 344–348, Apr. 2012.

[227] E. A. Manrao, I. M. Derrington, A. H. Laszlo, K. W. Langford, M. K. Hopper, N. Gillgren, M. Pavlenok, M. Niederweis, and J. H. Gundlach, "Reading DNA at single-nucleotide resolution with a mutant MspA nanopore and phi29 DNA polymerase," *Nature Biotechnology*, vol. 30, no. 4, pp. 349–353, Apr. 2012.

[228] A. H. Laszlo, I. M. Derrington, B. C. Ross, H. Brinkerhoff, A. Adey, I. C. Nova, J. M. Craig, K. W. Langford, J. M. Samson, R. Daza, K. Doering, J. Shendure, and J. H. Gundlach, "Decoding long nanopore sequencing reads of natural DNA," *Nature Biotechnology*, vol. 32, no. 8, pp. 829–833, Aug. 2014.

[229] D. Deamer, M. Akeson, and D. Branton, "Three decades of nanopore sequencing," *Nature Biotechnology*, vol. 34, no. 5, pp. 518–524, May 2016.

[230] J. J. Kasianowicz, E. Brandin, D. Branton, and D. W. Deamer, "Characterization of individual polynucleotide molecules using a membrane channel," *Proceedings of the National Academy of Sciences*, vol. 93, no. 24, pp. 13 770–13 773, Nov. 1996.

[231] A. Meller, L. Nivon, E. Brandin, J. Golovchenko, and D. Branton, "Rapid nanopore discrimination between single polynucleotide molecules," *Proceedings of the National Academy of Sciences*, vol. 97, no. 3, pp. 1079–1084, Feb. 2000.

[232] D. Stoddart, A. J. Heron, E. Mikhailova, G. Maglia, and H. Bayley, "Single-nucleotide discrimination in immobilized DNA oligonucleotides with a biological nanopore," *Proceedings of the National Academy of Sciences*, vol. 106, no. 19, pp. 7702–7707, May 2009.

[233] A. H. Laszlo, I. M. Derrington, H. Brinkerhoff, K. W. Langford, I. C. Nova, J. M. Samson, J. J. Bartlett, M. Pavlenok, and J. H. Gundlach, "Detection and mapping of 5-methylcytosine and 5-hydroxymethylcytosine with nanopore MspA," *Proceedings of the National Academy of Sciences*, vol. 110, no. 47, pp. 18 904–18 909, Nov. 2013.

[234] J. Schreiber, Z. L. Wescoe, R. Abu-Shumays, J. T. Vivian, B. Baatar, K. Karplus, and M. Akeson, "Error rates for nanopore discrimination among cytosine, methylcytosine, and hydroxymethylcytosine along individual DNA strands," *Proceedings of the National Academy of Sciences*, vol. 110, no. 47, pp. 18 910–18 915, Nov. 2013.

[235] T. Z. Butler, M. Pavlenok, I. M. Derrington, M. Niederweis, and J. H. Gundlach, "Single-molecule DNA detection with an engineered MspA protein nanopore," *Proceedings of the National Academy of Sciences*, vol. 105, no. 52, pp. 20 647–20 652, Dec. 2008.

[236] I. M. Derrington, T. Z. Butler, M. D. Collins, E. Manrao, M. Pavlenok, M. Niederweis, and J. H. Gundlach, "Nanopore DNA sequencing with MspA," *Proceedings of the National Academy of Sciences*, vol. 107, no. 37, pp. 16 060–16 065, Sep. 2010.

[237] L. Song, M. R. Hobaugh, C. Shustak, S. Cheley, H. Bayley, and J. E. Gouaux, "Structure of Staphylococcal a-Hemolysin, a Heptameric Transmembrane Pore," *Science*, vol. 274, no. 5294, pp. 1859–1865, Dec. 1996.

[238] B. Walker, J. Kasianowicz, M. Krishnasastry, and H. Bayley, "A pore-forming protein with a metal-actuated switch," *Protein Engineering, Design and Selection*, vol. 7, no. 5, pp. 655–662, May 1994.

[239] Z. L. Wescoe, J. Schreiber, and M. Akeson, "Nanopores Discriminate among Five C5-Cytosine Variants in DNA," *Journal of the American Chemical Society*, vol. 136, no. 47, pp. 16 582–16 587, Nov. 2014.

[240] K. R. Lieberman, G. M. Cherf, M. J. Doody, F. Olasagasti, Y. Kolodji, and M. Akeson, "Processive Replication of Single DNA Molecules in a Nanopore Catalyzed by phi29 DNA Polymerase," *Journal of the American Chemical Society*, vol. 132, no. 50, pp. 17 961–17 972, Dec. 2010.

[241] S. M. Bezrukov, I. Vodyanoy, R. A. Brutyan, and J. J. Kasianowicz, "Dynamics and Free Energy of Polymers Partitioning into a Nanoscale Pore," *Macromolecules*, vol. 29, no. 26, pp. 8517–8522, Jan. 1996.

[242] M. Akeson, D. Branton, J. J. Kasianowicz, E. Brandin, and D. W. Deamer, "Microsecond Time-Scale Discrimination Among Polycytidylic Acid, Polyadenylic Acid, and Polyuridylic Acid as Homopolymers or as Segments Within Single RNA Molecules," *Biophysical Journal*, vol. 77, no. 6, pp. 3227–3233, Dec. 1999.

[243] D. Stoddart, A. J. Heron, J. Klingelhoefer, E. Mikhailova, G. Maglia, and H. Bayley, "Nucleobase Recognition in ssDNA at the Central Constriction of the a-Hemolysin Pore," *Nano Letters*, vol. 10, no. 9, pp. 3633–3637, Sep. 2010.

[244] N. Ashkenasy, J. Sánchez-Quesada, H. Bayley, and M. R. Ghadiri, "Recognizing a Single Base in an Individual DNA Strand: A Step Toward DNA Sequencing in Nanopores," *Angewandte Chemie International Edition*, vol. 44, no. 9, pp. 1401–1404, Feb. 2005.

[245] D. Stoddart, G. Maglia, E. Mikhailova, A. J. Heron, and H. Bayley, "Multiple Base-Recognition Sites in a Biological Nanopore: Two Heads are Better than One," *Angewandte Chemie International Edition*, vol. 49, no. 3, pp. 556–559, Jan. 2010.

[246] S. M. Bezrukov and J. J. Kasianowicz, "Current noise reveals protonation kinetics and number of ionizable sites in an open protein ion channel," *Physical Review Letters*, vol. 70, no. 15, pp. 2352–2355, Apr. 1993.

[247] J.-Y. Zhang, Y. Zhang, L. Wang, F. Guo, Q. Yun, T. Zeng, X. Yan, L. Yu, L. Cheng, W. Wu, X. Shi, J. Chen, Y. Sun, J. Yang, R. Guo, X. Zhang, L. Kong, Z. Wang, J. Yao, Y. Tan, L. Shi, Z. Zhao, Z. Feng, X. Yu, C. Li, W. Zhan, Y. Ren, F. Yang, Z. Liu, G. Fan, W. Zhong, D. Li, L. He, Y. Qi, M. Zhang, Y. Zhu, H. Chi, Z. Zhao, Z. Wei, Z. Song, Y. Ju, R. Guo, L. Xiao, X. Lin, L. Chen, C. Yang, Q. Li, O. Wang, X. Jin, M. Ni, W. Zhang, L. Liu, Y. Gu, J. Wang, Y. Li, X. Xu, and Y. Dong, "A single-molecule nanopore sequencing platform," *bioRxiv*, p. 2024.08.19.608720, 2024.

[248] J. Shendure, S. Balasubramanian, G. M. Church, W. Gilbert, J. Rogers, J. A. Schloss, and R. H. Waterston, "DNA sequencing at 40: past, present and future," *Nature*, vol. 550, no. 7676, pp. 345–353, Oct. 2017.

[249] F. Sanger, S. Nicklen, and A. R. Coulson, "DNA sequencing with chain-terminating inhibitors," *Proceedings of the National Academy of Sciences*, vol. 74, no. 12, pp. 5463–5467, Dec. 1977.

[250] R. D. Fleischmann, M. D. Adams, O. White, R. A. Clayton, E. F. Kirkness, A. R. Kerlavage, C. J. Bult, J.-F. Tomb, B. A. Dougherty, J. M. Merrick, K. McKenney, G. Sutton, W. FitzHugh, C. Fields, J. D. Gocayne, J. Scott, R. Shirley, L.-l. Liu, A. Glodek, J. M. Kelley, J. F. Weidman, C. A. Phillips, T. Spriggs, E. Hedblom, M. D. Cotton, T. R. Utterback, M. C. Hanna, D. T. Nguyen, D. M. Saudek, R. C. Brandon, L. D. Fine, J. L. Fritchman, J. L. Fuhrmann, N. S. M. Geoghagen, C. L. Gnehm, L. A. McDonald, K. V. Small, C. M. Fraser, H. O. Smith, and J. C. Venter, "Whole-Genome Random Sequencing and Assembly of Haemophilus influenzae Rd," *Science*, vol. 269, no. 5223, pp. 496–512, Jul. 1995.

[251] E. W. Myers, G. G. Sutton, A. L. Delcher, I. M. Dew, D. P. Fasulo, M. J. Flanigan, S. A. Kravitz, C. M. Mobarry, K. H. J. Reinert, K. A. Remington, E. L. Anson, R. A. Bolanos, H.-H. Chou, C. M. Jordan, A. L. Halpern, S. Lonardi, E. M. Beasley, R. C. Brandon, L. Chen, P. J. Dunn, Z. Lai, Y. Liang, D. R. Nusskern, M. Zhan, Q. Zhang, X. Zheng, G. M. Rubin, M. D. Adams, and J. C. Venter, "A Whole-Genome Assembly of Drosophila," *Science*, vol. 287, no. 5461, pp. 2196–2204, Mar. 2000.

[252] C. Alkan, S. Sajjadian, and E. E. Eichler, "Limitations of next-generation genome sequence assembly," *Nature Methods*, vol. 8, no. 1, pp. 61–65, Jan. 2011.

[253] N. Nagarajan and M. Pop, "Sequence assembly demystified," *Nature Reviews Genetics*, vol. 14, no. 3, pp. 157–167, Mar. 2013.

[254] J. M. Heather and B. Chain, "The sequence of sequencers: The history of sequencing DNA," *Genomics*, vol. 107, no. 1, pp. 1–8, Jan. 2016.

[255] D. Senol Cali, J. S. Kim, S. Ghose, C. Alkan, and O. Mutlu, "Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions," *Briefings in Bioinformatics*, vol. 20, no. 4, pp. 1542–1559, Jul. 2019.

[256] C. Firtina, Z. Bar-Joseph, C. Alkan, and A. Cicek, "Hercules: a profile HMM-based hybrid error correction algorithm for long reads," *Nucleic Acids Research*, vol. 46, no. 21, pp. e125–e125, Nov. 2018.

[257] C. Firtina, J. S. Kim, M. Alser, D. Senol Cali, A. E. Cicek, C. Alkan, and O. Mutlu, "Apollo: a sequencing-technology-independent, scalable and accurate assembly polishing algorithm," *Bioinformatics*, vol. 36, no. 12, pp. 3669–3679, Jun. 2020.

[258] M. Alser, J. Rotman, D. Deshpande, K. Taraszka, H. Shi, P. I. Baykal, H. T. Yang, V. Xue, S. Knyazev, B. D. Singer, B. Balliu, D. Koslicki, P. Skums, A. Zelikovsky, C. Alkan, O. Mutlu, and S. Mangul, "Technology dictates algorithms: recent developments in read alignment," *Genome Biology*, vol. 22, no. 1, p. 249, Aug. 2021.

[259] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson, "Big Data: Astronomical or Genomical?" *PLOS Biology*, 2015.

[260] M. Alser, Z. Bingöl, D. S. Cali, J. S. Kim, S. Ghose, C. Alkan, and O. Mutlu, "Accelerating Genome Analysis: A Primer on an Ongoing Journey," *IEEE Micro*, vol. 40, no. 5, pp. 65–75, Oct. 2020.

[261] M. Alser, J. Lindegger, C. Firtina, N. Almadhoun, H. Mao, G. Singh, J. Gomez-Luna, and O. Mutlu, "From molecules to genomic variations: Accelerating genome analysis via intelligent algorithms and architectures," *Computational and Structural Biotechnology Journal*, vol. 20, pp. 4579–4599, Jan. 2022.

[262] O. Mutlu and C. Firtina, "Accelerating genome analysis via algorithm-architecture co-design," in *DAC*, 2023.

[263] J. D. Watson and F. H. C. Crick, "Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid," *Nature*, vol. 171, no. 4356, pp. 737–738, Apr. 1953.

[264] M. Loose, S. Malla, and M. Stout, "Real-time selective sequencing using nanopore technology," *Nature Methods*, vol. 13, no. 9, pp. 751–754, Sep. 2016.

[265] C. Firtina, M. Soysal, J. Lindegger, and O. Mutlu, "RawHash2: Mapping Raw Nanopore Signals Using Hash-Based Seeding and Adaptive Quantization," *Bioinform.*, 2024.

[266] C. Firtina, N. Mansouri Ghiasi, J. Lindegger, G. Singh, M. B. Cavlak, H. Mao, and O. Mutlu, "RawHash: enabling fast and accurate real-time analysis of raw nanopore signals for large genomes," *Bioinform.*, 2023.

[267] M. B. Cavlak, G. Singh, M. Alser, C. Firtina, J. Lindegger, M. Sadrosadati, N. M. Ghiasi, C. Alkan, and O. Mutlu, "TargetCall: Eliminating the Wasted Computation in Basecalling via Pre-Basecalling Filtering," *Frontiers in Genetics*, Sep. 2024.

[268] Z. Xu, Y. Mai, D. Liu, W. He, X. Lin, C. Xu, L. Zhang, X. Meng, J. Mafofo, W. A. Zaher, and others, "Fast-bonito: A Faster Deep Learning Based Basecaller for Nanopore Sequencing," *Artificial Intelligence in the Life Sciences*, vol. 1, p. 100011, 2021, publisher: Elsevier.

[269] P. Perešíni, V. Boža, B. Brejová, and T. Vinař, "Nanopore base calling on the edge," *Bioinformatics*, 2021.

[270] V. Boža, B. Brejová, and T. Vinař, "DeepNano: Deep recurrent neural networks for base calling in MinION nanopore reads," *PLOS One*, 2017.

[271] V. Boža, P. Perešíni, B. Brejová, and T. Vinař, "DeepNano-blitz: a fast base caller for MinION nanopore sequencers," *Bioinformatics*, vol. 36, no. 14, pp. 4191–4192, Jul. 2020.

[272] Oxford Nanopore Technologies, "Dorado," 2024. [Online]. Available: https://github.com/nanoporetech/dorado

[273] Oxford Nanopore Technologies, "Guppy," 2017.

[274] X. Lv, Z. Chen, Y. Lu, and Y. Yang, "An end-to-end Oxford nanopore basecaller using convolution-augmented transformer," in *BIBM*, 2020.

[275] G. Singh, M. Alser, K. Denolf, C. Firtina, A. Khodamoradi, M. B. Cavlak, H. Corporaal, and O. Mutlu, "RUBICON: a framework for designing efficient deep learning-based genomic basecallers," *Genome Biology*, 2024.

[276] Y.-z. Zhang, A. Akdemir, G. Tremmel, S. Imoto, S. Miyano, T. Shibuya, and R. Yamaguchi, "Nanopore basecalling from a perspective of instance segmentation," *BMC Bioinformatics*, vol. 21, no. 3, p. 136, Apr. 2020.

[277] X. Xu, N. Bhalla, P. Ståhl, and J. Jaldén, "Lokatt: a hybrid DNA nanopore basecaller with an explicit duration hidden Markov model and a residual LSTM network," *BMC Bioinformatics*, vol. 24, no. 1, p. 461, Dec. 2023.

[278] J. Zeng, H. Cai, H. Peng, H. Wang, Y. Zhang, and T. Akutsu, "Causalcall: Nanopore Basecalling Using a Temporal Convolutional Network," *Frontiers in Genetics*, vol. 10, 2020.

[279] H. Teng, M. D. Cao, M. B. Hall, T. Duarte, S. Wang, and L. J. M. Coin, "Chiron: translating nanopore raw signal directly into nucleotide sequence using deep learning," *GigaScience*, vol. 7, no. 5, p. giy037, May 2018.

[280] H. Konishi, R. Yamaguchi, K. Yamaguchi, Y. Furukawa, and S. Imoto, "Halcyon: an accurate basecaller exploiting an encoder–decoder model with monotonic attention," *Bioinformatics*, vol. 37, no. 9, pp. 1211–1217, Jun. 2021.

[281] Y.-M. Yeh and Y.-C. Lu, "MSRCall: a multi-scale deep neural network to basecall Oxford Nanopore sequences," *Bioinformatics*, vol. 38, no. 16, pp. 3877–3884, Aug. 2022.

[282] B. Noordijk, R. Nijland, V. J. Carrion, J. M. Raaijmakers, D. de Ridder, and C. de Lannoy, "baseLess: lightweight detection of sequences in raw MinION data," *Bioinformatics Advances*, vol. 3, no. 1, p. vbad017, Jan. 2023.

[283] N. Huang, F. Nie, P. Ni, F. Luo, and J. Wang, "SACall: A Neural Network Basecaller for Oxford Nanopore Sequencing Data Based on Self-Attention Mechanism," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 19, no. 1, pp. 614–623, 2022.

[284] N. Miculinic, M. Ratkovic, and M. Sikic, "MinCall - MinION end2end convolutional deep learning basecaller," *arXiv*, 2019.

[285] N. J. Loman, J. Quick, and J. T. Simpson, "A complete bacterial genome assembled de novo using only nanopore sequencing data," *Nature Methods*, vol. 12, no. 8, pp. 733–735, Aug. 2015.

[286] M. David, L. J. Dursi, D. Yao, P. C. Boutros, and J. T. Simpson, "Nanocall: an open source basecaller for Oxford Nanopore sequencing data," *Bioinformatics*, vol. 33, no. 1, pp. 49–55, Jan. 2017.

[287] W. Timp, J. Comer, and A. Aksimentiev, "DNA Base-Calling from a Nanopore Using a Viterbi Algorithm," *Biophysical Journal*, 2012.

[288] J. Schreiber and K. Karplus, "Analysis of nanopore data using hidden Markov models," *Bioinformatics*, vol. 31, no. 12, pp. 1897–1903, Jun. 2015.

[289] R. F. Purnell, K. K. Mehta, and J. J. Schmidt, "Nucleotide Identification and Orientation Discrimination of DNA Homopolymers Immobilized in a Protein Nanopore," *Nano Letters*, 2008.

[290] Y. Bao, J. Wadden, J. R. Erb-Downward, P. Ranjan, W. Zhou, T. L. McDonald, R. E. Mills, A. P. Boyle, R. P. Dickson, D. Blaauw, and J. D. Welch, "SquiggleNet: real-time, direct classification of nanopore signals," *Genome Biology*, vol. 22, no. 1, p. 298, Oct. 2021.

[291] H. Zhang, H. Li, C. Jain, H. Cheng, K. F. Au, H. Li, and S. Aluru, "Real-time mapping of nanopore raw signals," *Bioinformatics*, vol. 37, no. Supplement_1, pp. i477–i483, Jul. 2021.

[292] S. Kovaka, Y. Fan, B. Ni, W. Timp, and M. C. Schatz, "Targeted nanopore sequencing by real-time mapping of raw electrical signal with UNCALLED," *Nature Biotechnology*, vol. 39, no. 4, pp. 431–441, Apr. 2021.

[293] A. Senanayake, H. Gamaarachchi, D. Herath, and R. Ragel, "DeepSelectNet: deep neural network based selective sequencing for oxford nanopore sequencing," *BMC Bioinformatics*, vol. 24, no. 1, p. 31, Jan. 2023.

[294] Sam Kovaka, Paul W. Hook, Katharine M. Jenike, Vikram Shivakumar, Luke B. Morina, Roham Razaghi, Winston Timp, and Michael C. Schatz, "Uncalled4 improves nanopore DNA and RNA modification detection via fast and accurate signal alignment," *bioRxiv*, 2024.

[295] J. Lindegger, C. Firtina, N. M. Ghiasi, M. Sadrosadati, M. Alser, and O. Mutlu, "RawAlign: Accurate, Fast, and Scalable Raw Nanopore Signal Mapping via Combining Seeding and Alignment," *IEEE Access*, 2024.

[296] C. Firtina, M. Mordig, H. Mustafa, S. Goswami, N. M. Ghiasi, S. Mercogliano, F. Eris, J. Lindegger, A. Kahles, and O. Mutlu, "Rawsamble: Overlapping and Assembling Raw Nanopore Signals using a Hash-based Seeding Mechanism," *arXiv*, Oct. 2024.

[297] P. J. Shih, H. Saadat, S. Parameswaran, and H. Gamaarachchi, "Efficient real-time selective genome sequencing on resource-constrained devices," *GigaScience*, 2023.

[298] H. Sadasivan, J. Wadden, K. Goliya, P. Ranjan, R. P. Dickson, D. Blaauw, R. Das, and S. Narayanasamy, "Rapid Real-time Squiggle Classification for Read Until Using RawMap," *Arch. Clin. Biomed. Res.*, 2023.

[299] V. S. Shivakumar, O. Y. Ahmed, S. Kovaka, M. Zakeri, and B. Langmead, "Sigmoni: classification of nanopore signal with a compressed pangenome index," *Bioinform.*, 2024.

[300] H. Sadasivan, D. Stiffler, A. Tirumala, J. Israeli, and Satish Narayanasamy, "Accelerated Dynamic Time Warping on GPU for Selective Nanopore Sequencing," *Journal of Biotechnology and Biomedicine*, vol. 7, pp. 137–148, 2024.

[301] H. Gamaarachchi, C. W. Lam, G. Jayatilaka, H. Samarakoon, J. T. Simpson, M. A. Smith, and S. Parameswaran, "GPU accelerated adaptive banded event alignment for rapid comparative nanopore signal analysis," *BMC Bioinformatics*, vol. 21, no. 1, p. 343, Aug. 2020.

[302] S. Samarasinghe, P. Premathilaka, W. Herath, H. Gamaarachchi, and R. Ragel, "Energy Efficient Adaptive Banded Event Alignment using OpenCL on FPGAs," in *ICIAfS*, 2021, pp. 369–374.

[303] G. D. Ruxton, "The unequal variance t-test is an underused alternative to Student's t-test and the Mann–Whitney U test," *Behavioral Ecology*, vol. 17, no. 4, pp. 688–690, Jul. 2006.

[304] A. Chakraborty, B. Morgenstern, and S. Bandyopadhyay, "S-conLSH: alignment-free gapped mapping of noisy long reads," *BMC Bioinformatics*, vol. 22, no. 1, p. 64, Feb. 2021.

[305] H. Li, "Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences," *Bioinformatics*, vol. 32, no. 14, pp. 2103–2110, Jul. 2016.

[306] H. Li, "Minimap2: pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, Sep. 2018.

[307] A. Chakraborty and S. Bandyopadhyay, "conLSH: Context based Locality Sensitive Hashing for mapping of noisy SMRT reads," *Computational Biology and Chemistry*, vol. 85, p. 107206, Apr. 2020.

[308] H. Lin, Z. Zhang, M. Q. Zhang, B. Ma, and M. Li, "ZOOM! Zillions of oligos mapped," *Bioinformatics*, vol. 24, no. 21, pp. 2431–2437, Nov. 2008.

[309] M. David, M. Dzamba, D. Lister, L. Ilie, and M. Brudno, "SHRiMP2: Sensitive yet Practical Short Read Mapping," *Bioinformatics*, vol. 27, no. 7, pp. 1011–1012, Apr. 2011.

[310] K. Sahlin, "Effective sequence similarity detection with strobemers," *Genome Research*, vol. 31, no. 11, pp. 2080–2094, Nov. 2021.

[311] J. Ren and M. J. P. Chaisson, "lra: A long read aligner for sequences and contigs," *PLOS Computational Biology*, vol. 17, no. 6, p. e1009078, Jun. 2021.

[312] H. Jiang and W. H. Wong, "SeqMap: mapping massive amount of oligonucleotides to the genome," *Bioinformatics*, vol. 24, no. 20, pp. 2395–2396, Oct. 2008.

[313] R. Li, Y. Li, K. Kristiansen, and J. Wang, "SOAP: short oligonucleotide alignment program," *Bioinformatics*, vol. 24, no. 5, pp. 713–714, Mar. 2008.

[314] A. D. Smith, Z. Xuan, and M. Q. Zhang, "Using quality scores and longer reads improves accuracy of Solexa read mapping," *BMC Bioinformatics*, vol. 9, no. 1, p. 128, Feb. 2008.

[315] B. Ma, J. Tromp, and M. Li, "PatternHunter: faster and more sensitive homology search," *Bioinformatics*, vol. 18, no. 3, pp. 440–445, Mar. 2002.

[316] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, Oct. 1990.

[317] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, Sep. 1997.

[318] W. J. Kent, "BLAT—The BLAST-Like Alignment Tool," *Genome Research*, vol. 12, no. 4, pp. 656–664, Apr. 2002.

[319] Z. Ning, A. J. Cox, and J. C. Mullikin, "SSAHA: A Fast Search Method for Large DNA Databases," *Genome Research*, vol. 11, no. 10, pp. 1725–1729, Oct. 2001.

[320] L. Egidi and G. Manzini, "Better spaced seeds using Quadratic Residues," *Journal of Computer and System Sciences*, vol. 79, no. 7, pp. 1144–1155, Nov. 2013.

[321] G. Rizk and D. Lavenier, "GASSST: global alignment short sequence search tool," *Bioinformatics*, vol. 26, no. 20, pp. 2534–2540, Oct. 2010.

[322] T. D. Wu and S. Nacu, "Fast and SNP-tolerant detection of complex variants and splicing in short reads," *Bioinformatics*, vol. 26, no. 7, pp. 873–881, Apr. 2010.

[323] B. Liu, D. Guan, M. Teng, and Y. Wang, "rHAT: fast alignment of noisy long reads with regional hashing," *Bioinformatics*, vol. 32, no. 11, pp. 1625–1631, Jun. 2016.

[324] F. Hach, F. Hormozdiari, C. Alkan, F. Hormozdiari, I. Birol, E. E. Eichler, and S. C. Sahinalp, "mrsFAST: a cache-oblivious algorithm for short-read mapping," *Nature Methods*, vol. 7, no. 8, pp. 576–577, Aug. 2010.

[325] S. M. Rumble, P. Lacroute, A. V. Dalca, M. Fiume, A. Sidow, and M. Brudno, "SHRiMP: Accurate Mapping of Short Color-space Reads," *PLOS Computational Biology*, vol. 5, no. 5, p. e1000386, May 2009.

[326] D. Weese, A.-K. Emde, T. Rausch, A. Döring, and K. Reinert, "RazerS—fast read mapping with sensitivity control," *Genome Research*, vol. 19, no. 9, pp. 1646–1654, Sep. 2009.

[327] K. Schneeberger, J. Hagmann, S. Ossowski, N. Warthmann, S. Gesing, O. Kohlbacher, and D. Weigel, "Simultaneous alignment of short reads against multiple genomes," *Genome Biology*, vol. 10, no. 9, p. R98, Sep. 2009.

[328] N. Homer, B. Merriman, and S. F. Nelson, "BFAST: An Alignment Tool for Large Scale Genome Resequencing," *PLOS ONE*, vol. 4, no. 11, p. e7767, Nov. 2009.

[329] B. D. Ondov, A. Varadarajan, K. D. Passalacqua, and N. H. Bergman, "Efficient mapping of Applied Biosystems SOLiD sequence data to a reference genome for functional genomic applications," *Bioinformatics*, vol. 24, no. 23, pp. 2776–2777, Dec. 2008.

[330] T. D. Wu and C. K. Watanabe, "GMAP: a genomic mapping and alignment program for mRNA and EST sequences," *Bioinformatics*, vol. 21, no. 9, pp. 1859–1875, May 2005.

[331] G. S. C. Slater and E. Birney, "Automated generation of heuristics for biological sequence comparison," *BMC Bioinformatics*, vol. 6, no. 1, p. 31, Feb. 2005.

[332] S. Schwartz, W. J. Kent, A. Smit, Z. Zhang, R. Baertsch, R. C. Hardison, D. Haussler, and W. Miller, "Human–Mouse Alignments with BLASTZ," *Genome Research*, vol. 13, no. 1, pp. 103–107, Jan. 2003.

[333] K. Sahlin, "Strobealign: flexible seed size enables ultra-fast and accurate read alignment," *Genome Biology*, vol. 23, no. 1, p. 260, Dec. 2022.

[334] C. Firtina, J. Park, M. Alser, J. S. Kim, D. S. Cali, T. Shahroodi, N. M. Ghiasi, G. Singh, K. Kanellopoulos, C. Alkan, and O. Mutlu, "BLEND: a fast, memory-efficient and accurate mechanism to find fuzzy seed matches in genome analysis," *NAR Genomics and Bioinformatics*, vol. 5, no. 1, p. lqad004, Mar. 2023.

[335] H. Li, J. Ruan, and R. Durbin, "Mapping short DNA sequencing reads and calling variants using mapping quality scores," *Genome Res.*, 2008.

[336] I. Sović, M. Šikić, A. Wilm, S. N. Fenlon, S. Chen, and N. Nagarajan, "Fast and sensitive mapping of nanopore sequencing reads with GraphMap," *Nature Communications*, vol. 7, no. 1, p. 11307, Apr. 2016.

[337] J. Shaw and Y. W. Yu, "Fast and robust metagenomic sequence comparison through sparse chaining with skani," *Nature Methods*, vol. 20, no. 11, pp. 1661–1665, Nov. 2023.

[338] S. A. Shiryev and R. Agarwala, "Indexing and searching petabase-scale nucleotide resources," *Nature Methods*, vol. 21, no. 6, pp. 994–1002, Jun. 2024.

[339] Y. Xi and W. Li, "BSMAP: whole genome bisulfite sequence MAPping program," *BMC Bioinformatics*, vol. 10, no. 1, p. 232, Jul. 2009.

[340] J. Kim, C. Li, and X. Xie, "Improving read mapping using additional prefix grams," *BMC Bioinformatics*, vol. 15, no. 1, p. 42, Feb. 2014.

[341] B. Liu, Y. Liu, J. Li, H. Guo, T. Zang, and Y. Wang, "deSALT: fast and accurate long transcriptomic read alignment with de Bruijn graph-based index," *Genome Biology*, vol. 20, no. 1, p. 274, Dec. 2019.

[342] W.-P. Lee, M. P. Stromberg, A. Ward, C. Stewart, E. P. Garrison, and G. T. Marth, "MOSAIK: A Hash-Based Algorithm for Accurate Next-Generation Sequencing Short-Read Mapping," *PLOS ONE*, vol. 9, no. 3, p. e90581, Mar. 2014.

[343] Y. Li, J. M. Patel, and A. Terrell, "WHAM: A High-Throughput Sequence Alignment Method," *ACM Trans. Database Syst.*, vol. 37, no. 4, Dec. 2012.

[344] M. C. Schatz, "CloudBurst: highly sensitive read mapping with MapReduce," *Bioinformatics*, vol. 25, no. 11, pp. 1363–1369, Jun. 2009.

[345] N. L. Clement, Q. Snell, M. J. Clement, P. C. Hollenhorst, J. Purwar, B. J. Graves, B. R. Cairns, and W. E. Johnson, "The GNUMAP algorithm: unbiased probabilistic mapping of oligonucleotides from next-generation sequencing," *Bioinformatics*, vol. 26, no. 1, pp. 38–45, Jan. 2010.

[346] H. L. Eaves and Y. Gao, "MOM: maximum oligonucleotide mapping," *Bioinformatics*, vol. 25, no. 7, pp. 969–970, Apr. 2009.

[347] D. Campagna, A. Albiero, A. Bilardi, E. Caniato, C. Forcato, S. Manavski, N. Vitulo, and G. Valle, "PASS: a program to align short sequences," *Bioinformatics*, vol. 25, no. 7, pp. 967–968, Apr. 2009.

[348] Y. Chen, T. Souaiaia, and T. Chen, "PerM: efficient mapping of short sequencing reads with periodic full sensitive spaced seeds," *Bioinformatics*, vol. 25, no. 19, pp. 2514–2521, Oct. 2009.

[349] N. Malhis, Y. S. N. Butterfield, M. Ester, and S. J. M. Jones, "Slider—maximum use of probability information for alignment of short sequence reads and SNP detection," *Bioinformatics*, vol. 25, no. 1, pp. 6–13, Jan. 2009.

[350] F. J. Sedlazeck, P. Rescheneder, and A. von Haeseler, "NextGenMap: fast and accurate read mapping in highly polymorphic genomes," *Bioinformatics*, vol. 29, no. 21, pp. 2790–2791, Nov. 2013.

[351] K. F. Au, H. Jiang, L. Lin, Y. Xing, and W. H. Wong, "Detection of splice junctions from paired-end RNA-seq data by SpliceMap," *Nucleic Acids Research*, vol. 38, no. 14, pp. 4570–4578, Aug. 2010.

[352] D. W. Bryant, Jr, R. Shen, H. D. Priest, W.-K. Wong, and T. C. Mockler, "Supersplat—spliced RNA-seq alignment," *Bioinformatics*, vol. 26, no. 12, pp. 1500–1505, Jun. 2010.

[353] P. Wang, B. Cao, T. Ma, B. Wang, Q. Zhang, and P. Zheng, "DUHI: Dynamically updated hash index clustering method for DNA storage," *Computers in Biology and Medicine*, vol. 164, p. 107244, Sep. 2023.

[354] J. C. Mu, H. Jiang, A. Kiani, M. Mohiyuddin, N. Bani Asadi, and W. H. Wong, "Fast and accurate read alignment for resequencing," *Bioinformatics*, vol. 28, no. 18, pp. 2366–2373, Sep. 2012.

[355] G. G. Faust and I. M. Hall, "YAHA: fast and flexible long-read alignment with optimal breakpoint detection," *Bioinformatics*, vol. 28, no. 19, pp. 2417–2424, Oct. 2012.

[356] J. Hu, H. Ge, M. Newman, and K. Liu, "OSA: a fast and accurate alignment tool for RNA-Seq," *Bioinformatics*, vol. 28, no. 14, pp. 1933–1934, Jul. 2012.

[357] Y. Liao, G. K. Smyth, and W. Shi, "The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote," *Nucleic Acids Research*, vol. 41, no. 10, pp. e108–e108, May 2013.

[358] P. M. Gontarz, J. Berger, and C. F. Wong, "SRmapper: a fast and sensitive genome-hashing alignment tool," *Bioinformatics*, vol. 29, no. 3, pp. 316–321, Feb. 2013.

[359] F. Hach, I. Sarrafi, F. Hormozdiari, C. Alkan, E. E. Eichler, and S. C. Sahinalp, "mrsFAST-Ultra: a compact, SNP-aware mapper for high performance sequencing applications," *Nucleic Acids Research*, vol. 42, no. W1, pp. W494–W500, Jul. 2014.

[360] M. Karami, A. Soltani Mohammadi, M. Martin, B. Ekim, W. Shen, L. Guo, M. Xu, G. E. Pibiri, R. Patro, and K. Sahlin, "Designing efficient randstrobes for sequence similarity analyses," *Bioinformatics*, vol. 40, no. 4, p. btae187, Apr. 2024.

[361] B. Ekim, K. Sahlin, P. Medvedev, B. Berger, and R. Chikhi, "Efficient mapping of accurate long reads in minimizer space with mapquik," *Genome Research*, vol. 33, no. 7, pp. 1188–1197, Jul. 2023.

[362] G. Marçais, A. L. Delcher, A. M. Phillippy, R. Coston, S. L. Salzberg, and A. Zimin, "MUMmer4: A fast and versatile genome alignment system," *PLOS Computational Biology*, vol. 14, no. 1, p. e1005944, Jan. 2018.

[363] M. J. Chaisson and G. Tesler, "Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory," *BMC Bioinformatics*, vol. 13, no. 1, p. 238, Sep. 2012.

[364] R. Li, C. Yu, Y. Li, T.-W. Lam, S.-M. Yiu, K. Kristiansen, and J. Wang, "SOAP2: an improved ultrafast tool for short read alignment," *Bioinformatics*, vol. 25, no. 15, pp. 1966–1967, Aug. 2009.

[365] B. Langmead, "Aligning Short Sequencing Reads with Bowtie," *Current Protocols in Bioinformatics*, vol. 32, no. 1, pp. 11.7.1–11.7.14, Dec. 2010.

[366] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *FOCS*, 2000.

[367] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biol.*, 2009.

[368] M. Vasimuddin, S. Misra, H. Li, and S. Aluru, "Efficient architecture-aware acceleration of BWA-MEM for multicore systems," in *IPDPS*, 2019.

[369] O. Ahmed, M. Rossi, S. Kovaka, M. C. Schatz, T. Gagie, C. Boucher, and B. Langmead, "Pan-genomic matching statistics for targeted nanopore sequencing," *iScience*, vol. 24, no. 6, Jun. 2021.

[370] O. Y. Ahmed, M. Rossi, T. Gagie, C. Boucher, and B. Langmead, "SPUMONI 2: improved classification using a pangenome index of minimizer digests," *Genome Biology*, vol. 24, no. 1, p. 122, May 2023.

[371] T. W. Lam, W. K. Sung, S. L. Tam, C. K. Wong, and S. M. Yiu, "Compressed indexing and local alignment of DNA," *Bioinformatics*, vol. 24, no. 6, pp. 791–797, Mar. 2008.

[372] M. Wheeler and M. Burrows, "A block-sorting lossless data compression algorithm," *SRC Res. Rep*, vol. 124, 1994.

[373] C. Boucher, D. Cenzato, Z. Lipták, M. Rossi, and M. Sciortino, "r-Indexing the eBWT," in *String Processing and Information Retrieval*, T. Lecroq and H. Touzet, Eds. Cham: Springer International Publishing, 2021, pp. 3–12.

[374] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nature Methods*, vol. 9, no. 4, pp. 357–359, Apr. 2012.

[375] S. Marco-Sola, M. Sammeth, R. Guigó, and P. Ribeca, "The GEM mapper: fast, accurate and versatile alignment by filtration," *Nature Methods*, vol. 9, no. 12, pp. 1185–1188, Dec. 2012.

[376] D. Kim, B. Langmead, and S. L. Salzberg, "HISAT: a fast spliced aligner with low memory requirements," *Nature Methods*, vol. 12, no. 4, pp. 357–360, Apr. 2015.

[377] D. Kim, J. M. Paggi, C. Park, C. Bennett, and S. L. Salzberg, "Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype," *Nature Biotechnology*, vol. 37, no. 8, pp. 907–915, Aug. 2019.

[378] P.-Y. Chen, S. J. Cokus, and M. Pellegrini, "BS Seeker: precise mapping for bisulfite sequencing," *BMC Bioinformatics*, vol. 11, no. 1, p. 203, Apr. 2010.

[379] C. Boucher, T. Gagie, A. Kuhnle, B. Langmead, G. Manzini, and T. Mun, "Prefix-free parsing for building big BWTs," *Algorithms for Molecular Biology*, vol. 14, no. 1, p. 13, May 2019.

[380] A. Hong, M. Oliva, D. Köppl, H. Bannai, C. Boucher, and T. Gagie, "Pfp-fm: an accelerated FM-index," *Algorithms for Molecular Biology*, vol. 19, no. 1, p. 15, Apr. 2024.

[381] S. Hoffmann, C. Otto, S. Kurtz, C. M. Sharma, P. Khaitovich, J. Vogel, P. F. Stadler, and J. Hackermüller, "Fast Mapping of Short Sequences with Mismatches, Insertions and Deletions Using Index Structures," *PLOS Computational Biology*, vol. 5, no. 9, p. e1000502, Sep. 2009.

[382] S. Burkhardt, A. Crauser, P. Ferragina, H.-P. Lenhof, E. Rivals, and M. Vingron, "q-gram based database searching using a suffix array (QUASAR)," in *Proceedings of the Third Annual International Conference on Computational Molecular Biology*, ser. RECOMB '99. New York, NY, USA: Association for Computing Machinery, 1999, pp. 77–83.

[383] E. M. McCreight, "A Space-Economical Suffix Tree Construction Algorithm," *J. ACM*, vol. 23, no. 2, pp. 262–272, Apr. 1976.

[384] R. Grossi and J. S. Vitter, "Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract)," in *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, ser. STOC '00. New York, NY, USA: Association for Computing Machinery, 2000, pp. 397–406.

[385] T. Gagie, G. Navarro, and N. Prezza, "Optimal-time text indexing in BWT-runs bounded space," in *SODA*, 2018.

[386] T. Gagie, G. Navarro, and N. Prezza, "Fully Functional Suffix Trees and Optimal Text Searching in BWT-Runs Bounded Space," *J. ACM*, vol. 67, no. 1, Jan. 2020.

[387] D. Kempa and T. Kociumaka, "Dynamic suffix array with polylogarithmic queries and updates," in *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2022. New York, NY, USA: Association for Computing Machinery, 2022, pp. 1657–1670.

[388] C. Meek, J. M. Patel, and S. Kasetty, "- OASIS: An Online and Accurate Technique for Local-alignment Searches on Biological Sequences," in *Proceedings 2003 VLDB Conference*, J.-C. Freytag, P. Lockemann, S. Abiteboul, M. Carey, P. Selinger, and A. Heuer, Eds. San Francisco: Morgan Kaufmann, Jan. 2003, pp. 910–921.

[389] F. De Bona, S. Ossowski, K. Schneeberger, and G. Rätsch, "Optimal spliced alignments of short sequence reads," *Bioinformatics*, vol. 24, no. 16, pp. i174–i180, Aug. 2008.

[390] C. Trapnell, L. Pachter, and S. L. Salzberg, "TopHat: discovering splice junctions with RNA-Seq," *Bioinformatics*, vol. 25, no. 9, pp. 1105–1111, May 2009.

[391] E. Y. Harris, N. Ponts, K. G. Le Roch, and S. Lonardi, "BRAT-BW: efficient and accurate mapping of bisulfite-treated reads," *Bioinformatics*, vol. 28, no. 13, pp. 1795–1796, Jul. 2012.

[392] C. Tennakoon, R. W. Purbojati, and W.-K. Sung, "BatMis: a fast algorithm for k-mismatch mapping," *Bioinformatics*, vol. 28, no. 16, pp. 2122–2128, Aug. 2012.

[393] E. Siragusa, D. Weese, and K. Reinert, "Fast and accurate read mapping with approximate seeds and multiple backtracking," *Nucleic Acids Research*, vol. 41, no. 7, pp. e78–e78, Apr. 2013.

[394] J. Tárraga, V. Arnau, H. Martínez, R. Moreno, D. Cazorla, J. Salavert-Torres, I. Blanquer-Espert, J. Dopazo, and I. Medina, "Acceleration of short and long DNA read mapping without loss of accuracy using suffix array," *Bioinformatics*, vol. 30, no. 23, pp. 3396–3398, Dec. 2014.

[395] H.-N. Lin and W.-L. Hsu, "Kart: a divide-and-conquer algorithm for NGS read alignment," *Bioinformatics*, vol. 33, no. 15, pp. 2281–2287, Aug. 2017.

[396] H.-N. Lin and W.-L. Hsu, "DART: a fast and accurate RNA-seq mapper with a partitioning strategy," *Bioinformatics*, vol. 34, no. 2, pp. 190–197, Jan. 2018.

[397] E. Haghshenas, S. C. Sahinalp, and F. Hach, "lordFAST: sensitive and Fast Alignment Search Tool for LOng noisy Read sequencing Data," *Bioinformatics*, vol. 35, no. 1, pp. 20–27, Jan. 2019.

[398] Y. Jung and D. Han, "BWA-MEME: BWA-MEM emulated with a machine learning approach," *Bioinformatics*, vol. 38, no. 9, pp. 2404–2413, Apr. 2022.

[399] F. Ji, Q. Zhou, J. Ruan, Z. Zhu, and X. Liu, "A compressive seeding algorithm in conjunction with reordering-based compression," *Bioinformatics*, vol. 40, no. 3, p. btae100, Mar. 2024.

[400] R. A. Lippert, "Space-Efficient Whole Genome Comparisons with Burrows–Wheeler Transforms," *Journal of Computational Biology*, vol. 12, no. 4, pp. 407–415, May 2005.

[401] T. Mun, A. Kuhnle, C. Boucher, T. Gagie, B. Langmead, and G. Manzini, "Matching Reads to Many Genomes with the r-Index," *Journal of Computational Biology*, vol. 27, no. 4, pp. 514–518, Apr. 2020.

[402] M. Rossi, M. Oliva, B. Langmead, T. Gagie, and C. Boucher, "MONI: A Pangenomic Index for Finding Maximal Exact Matches," *Journal of Computational Biology*, vol. 29, no. 2, pp. 169–187, Feb. 2022.

[403] M. Rossi, M. Oliva, P. Bonizzoni, B. Langmead, T. Gagie, and C. Boucher, "Finding Maximal Exact Matches Using the r-Index," *Journal of Computational Biology*, vol. 29, no. 2, pp. 188–194, Feb. 2022.

[404] A. Subramaniyan, J. Wadden, K. Goliya, N. Ozog, X. Wu, S. Narayanasamy, D. Blaauw, and R. Das, "Accelerated Seeding for Genome Sequence Alignment with Enumerated Radix Trees," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 388–401.

[405] K. Sadakane, "New text indexing functionalities of the compressed suffix arrays," *Journal of Algorithms*, vol. 48, no. 2, pp. 294–313, Sep. 2003.

[406] C. Boucher, D. Cenzato, Z. Lipták, M. Rossi, and M. Sciortino, "r-indexing the eBWT," *Information and Computation*, vol. 298, p. 105155, Jun. 2024.

[407] L. Guo and H. Huo, "An efficient Burrows–Wheeler transform-based aligner for short read mapping," *Computational Biology and Chemistry*, vol. 110, p. 108050, Jun. 2024.

[408] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," *arXiv preprint arXiv:1303.3997*, 2013.

[409] S. M. Kiełbasa, R. Wan, K. Sato, P. Horton, and M. C. Frith, "Adaptive seeds tame genomic sequence comparison," *Genome Research*, vol. 21, no. 3, pp. 487–493, Mar. 2011.

[410] N. Bertram, J. Fischer, and L. Nalbach, "Move-r: Optimizing the r-index," in *22nd International Symposium on Experimental Algorithms (SEA 2024)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), L. Liberti, Ed., vol. 301. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, pp. 1:1–1:19.

[411] H. Li, "BWT construction and search at the terabase scale," *arXiv*, 2024.

[412] H. Zheng, C. Kingsford, and G. Marçais, "Improved design and analysis of practical minimizers," *Bioinformatics*, vol. 36, no. Supplement_1, pp. i119–i127, Jul. 2020.

[413] C. Jain, A. Rhie, H. Zhang, C. Chu, B. P. Walenz, S. Koren, and A. M. Phillippy, "Weighted minimizer sampling improves long read mapping," *Bioinformatics*, vol. 36, no. Supplement_1, pp. i111–i118, Jul. 2020.

[414] B. D. Ondov, T. J. Treangen, P. Melsted, A. B. Mallonee, N. H. Bergman, S. Koren, and A. M. Phillippy, "Mash: fast genome and metagenome distance estimation using MinHash," *Genome Biology*, vol. 17, no. 1, p. 132, Jun. 2016.

[415] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount, and J. A. Yorke, "Reducing storage requirements for biological sequence comparison," *Bioinformatics*, vol. 20, no. 18, pp. 3363–3369, Dec. 2004.

[416] K. Berlin, S. Koren, C.-S. Chin, J. P. Drake, J. M. Landolin, and A. M. Phillippy, "Assembling large genomes with single-molecule sequencing and locality-sensitive hashing," *Nature Biotechnology*, vol. 33, no. 6, pp. 623–630, Jun. 2015.

[417] D. N. Baker and B. Langmead, "Dashing: fast and accurate genomic distances with HyperLogLog," *Genome Biology*, vol. 20, no. 1, p. 265, Dec. 2019.

[418] R. Edgar, "Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences," *PeerJ*, vol. 9, pp. e10 805–e10 805, Feb. 2021.

[419] C.-S. Chin and A. Khalak, "Human Genome Assembly in 100 Minutes," *bioRxiv*, p. 705616, 2019.

[420] A. Broder, "On the resemblance and containment of documents," in *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, 1997, pp. 21–29.

[421] D. N. Baker and B. Langmead, "Dashing 2: genomic sketching with multiplicities and locality-sensitive hashing," *bioRxiv*, 2023.

[422] A. Joudaki, A. Meterez, H. Mustafa, R. Groot Koerkamp, A. Kahles, and G. Rätsch, "Aligning distant sequences to graphs using long seed sketches," *Genome Res.*, 2023.

[423] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: Local Algorithms for Document Fingerprinting," in *SIGMOD*, New York, NY, USA, 2003.

[424] A. Joudaki, G. Rätsch, and A. Kahles, "Fast Alignment-Free Similarity Estimation By Tensor Sketching," *bioRxiv*, p. 2020.11.13.381814, 2021.

[425] L. Irber, P. T. Brooks, T. Reiter, N. T. Pierce-Ward, M. R. Hera, D. Koslicki, and C. T. Brown, "Lightweight compositional analysis of metagenomes with FracMinHash and minimum metagenome covers," *bioRxiv*, p. 2022.01.11.475838, 2022.

[426] Y. Orenstein, D. Pellow, G. Marçais, R. Shamir, and C. Kingsford, "Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing," *PLOS Computational Biology*, vol. 13, no. 10, p. e1005777, Oct. 2017.

[427] A. Dutta, D. Pellow, and R. Shamir, "Parameterized syncmer schemes improve long-read mapping," *PLOS Computational Biology*, vol. 18, no. 10, p. e1010638, Oct. 2022.

[428] U. Manber, "Finding similar files in a large file system," in *Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference*, ser. WTEC'94. USA: USENIX Association, 1994, p. 2.

[429] G. Marçais, D. DeBlasio, and C. Kingsford, "Asymptotically optimal minimizers schemes," *Bioinformatics*, vol. 34, no. 13, pp. i13–i22, Jul. 2018.

[430] M. C. Frith, L. Noé, and G. Kucherov, "Minimally overlapping words for sequence similarity search," *Bioinformatics*, vol. 36, no. 22-23, pp. 5344–5350, Apr. 2021.

[431] M. C. Frith, J. Shaw, and J. L. Spouge, "How to optimally sample a sequence for rapid analysis," *Bioinformatics*, vol. 39, no. 2, p. btad057, Feb. 2023.

[432] H. Xin, M. Shao, and C. Kingsford, "Context-aware seeds for read mapping," *Algorithms for Molecular Biology*, vol. 15, no. 1, p. 10, May 2020.

[433] M. S. Charikar, "Similarity Estimation Techniques from Rounding Algorithms," in *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '02. New York, NY, USA: Association for Computing Machinery, 2002, pp. 380–388.

[434] G. S. Manku, A. Jain, and A. Das Sarma, "Detecting Near-Duplicates for Web Crawling," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07. New York, NY, USA: Association for Computing Machinery, 2007, pp. 141–150.

[435] E. Petrucci, L. Noé, C. Pizzi, and M. Comin, "Iterative Spaced Seed Hashing: Closing the Gap Between Spaced Seed Hashing and k-mer Hashing," *Journal of Computational Biology*, vol. 27, no. 2, pp. 223–233, Feb. 2020.

[436] A. Mallik and L. Ilie, "ALeS: adaptive-length spaced-seed design," *Bioinformatics*, vol. 37, no. 9, pp. 1206–1210, May 2021.

[437] K. Sinha and P. Ram, "Fruit-Fly Inspired Neighborhood Encoding for Classification," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1470–1480.

[438] Y. Chen, S. Chen, and X. Zhang, "Using DenseFly algorithm for cell searching on massive scRNA-seq datasets," *BMC Genomics*, vol. 21, no. 5, p. 222, Dec. 2020.

[439] J. Sharma and S. Navlakha, "Improving Similarity Search with High-dimensional Locality-sensitive Hashing," *arXiv*, 2018.

[440] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. Jones, and I. Birol, "ABySS: A parallel assembler for short read sequence data," *Genome Research*, 2009.

[441] G. Greenberg, A. N. Ravi, and I. Shomorony, "LexicHash: sequence similarity estimation via lexicographic comparison of hashes," *Bioinformatics*, vol. 39, no. 11, p. btad652, Nov. 2023.

[442] H. Mohamadi, J. Chu, B. P. Vandervalk, and I. Birol, "ntHash: recursive nucleotide hashing," *Bioinformatics*, vol. 32, no. 22, pp. 3492–3494, Nov. 2016.

[443] P. Kazemi, J. Wong, V. Nikolić, H. Mohamadi, R. L. Warren, and I. Birol, "ntHash2: recursive spaced seed hashing for nucleotide sequences," *Bioinformatics*, vol. 38, no. 20, pp. 4812–4813, Oct. 2022.

[444] G. H. Gonnet and R. A. Baeza-Yates, "An analysis of the Karp-Rabin string matching algorithm," *Information Processing Letters*, vol. 34, no. 5, pp. 271–274, May 1990.

[445] J. D. Cohen, "Recursive hashing functions for n-grams," *ACM Trans. Inf. Syst.*, vol. 15, no. 3, pp. 291–320, Jul. 1997.

[446] G. E. Pibiri, Y. Shibuya, and A. Limasset, "Locality-preserving minimal perfect hashing of k-mers," *Bioinformatics*, vol. 39, no. Supplement_1, pp. i534–i543, Jun. 2023.

[447] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM Journal of Research and Development*, vol. 31, no. 2, pp. 249–260, 1987.

[448] D. Lemire and O. Kaser, "Recursive n-gram hashing is pairwise independent, at best," *Computer Speech & Language*, vol. 24, no. 4, pp. 698–710, Oct. 2010.

[449] J. Wong, P. Kazemi, L. Coombe, R. L. Warren, and I. Birol, "aaHash: recursive amino acid sequence hashing," *Bioinformatics Advances*, vol. 3, no. 1, p. vbad162, Jan. 2023.

[450] M. Steinegger and J. Söding, "Clustering huge protein sequence sets in linear time," *Nature Communications*, vol. 9, no. 1, p. 2542, Jun. 2018.

[451] M. Farach and S. Muthukrishnan, "Perfect hashing for strings: Formalization and algorithms," in *Combinatorial Pattern Matching*, D. Hirschberg and G. Myers, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 130–140.

[452] G. E. Pibiri, "On weighted k-mer dictionaries," *Algorithms for Molecular Biology*, vol. 18, no. 1, p. 3, Jun. 2023.

[453] G. E. Pibiri, "Sparse and skew hashing of K-mers," *Bioinformatics*, vol. 38, no. Supplement_1, pp. i185–i194, Jun. 2022.

[454] A. A. Karcioglu and H. Bulut, "Improving hash-q exact string matching algorithm with perfect hashing for DNA sequences," *Computers in Biology and Medicine*, vol. 131, p. 104292, Apr. 2021.

[455] S. Girotto, M. Comin, and C. Pizzi, "Fast Spaced Seed Hashing," in *WABI*, R. Schwartz and K. Reinert, Eds., vol. 88, Dagstuhl, Germany, 2017, pp. 7:1–7:14.

[456] S. Girotto, M. Comin, and C. Pizzi, "Efficient computation of spaced seed hashing with block indexing," *BMC Bioinformatics*, vol. 19, no. 15, p. 441, Nov. 2018.

[457] S. Girotto, M. Comin, and C. Pizzi, "FSH: fast spaced seed hashing exploiting adjacent hashes," *Algorithms for Molecular Biology*, vol. 13, no. 1, p. 8, Mar. 2018.

[458] C. Ryali, J. Hopfield, L. Grinberg, and D. Krotov, "Bio-Inspired Hashing for Unsupervised Similarity Search," in *ICML*, H. D. III and A. Singh, Eds., vol. 119, Jul. 2020, pp. 8295–8306.

[459] G. Marçais, D. DeBlasio, P. Pandey, and C. Kingsford, "Locality-sensitive hashing for the edit distance," *Bioinformatics*, vol. 35, no. 14, pp. i127–i135, Jul. 2019.

[460] H. Xin, D. Lee, F. Hormozdiari, S. Yedkar, O. Mutlu, and C. Alkan, "Accelerating read mapping with FastHASH," *BMC Genomics*, vol. 14, no. 1, p. S13, Jan. 2013.

[461] M. Alser, H. Hassan, H. Xin, O. Ergin, O. Mutlu, and C. Alkan, "GateKeeper: a new hardware architecture for accelerating pre-alignment in DNA short read mapping," *Bioinformatics*, vol. 33, no. 21, pp. 3355–3363, Nov. 2017.

[462] H. Xin, J. Greth, J. Emmons, G. Pekhimenko, C. Kingsford, C. Alkan, and O. Mutlu, "Shifted Hamming distance: a fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping," *Bioinformatics*, vol. 31, no. 10, pp. 1553–1560, May 2015.

[463] J. S. Kim, D. Senol Cali, H. Xin, D. Lee, S. Ghose, M. Alser, H. Hassan, O. Ergin, C. Alkan, and O. Mutlu, "GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies," *BMC Genomics*, vol. 19, no. 2, p. 89, May 2018.

[464] M. Alser, H. Hassan, A. Kumar, O. Mutlu, and C. Alkan, "Shouji: a fast and efficient pre-alignment filter for sequence alignment," *Bioinformatics*, vol. 35, no. 21, pp. 4255–4263, Nov. 2019.

[465] M. Alser, T. Shahroodi, J. Gómez-Luna, C. Alkan, and O. Mutlu, "SneakySnake: a fast and accurate universal genome pre-alignment filter for CPUs, GPUs and FPGAs," *Bioinformatics*, vol. 36, no. 22-23, pp. 5282–5290, Dec. 2020.

[466] M. Alser, O. Mutlu, and C. Alkan, "MAGNET: Understanding and improving the accuracy of genome pre-Alignment filtering," *arXiv*, 2017.

[467] A. F. Laguna, H. Gamaarachchi, X. Yin, M. Niemier, S. Parameswaran, and X. S. Hu, "Seed-and-Vote based in-memory accelerator for DNA read mapping," in *ICCAD*, 2020.

[468] H. Xin, S. Nahar, R. Zhu, J. Emmons, G. Pekhimenko, C. Kingsford, C. Alkan, and O. Mutlu, "Optimal seed solver: optimizing seed selection in read mapping," *Bioinformatics*, vol. 32, no. 11, pp. 1632–1642, Jun. 2016.

[469] M. I. Abouelhoda and E. Ohlebusch, "Chaining algorithms for multiple genome comparison," *Combinatorial Pattern Matching (CPM) Special Issue*, vol. 3, no. 2, pp. 321–341, Jun. 2005.

[470] M. I. Abouelhoda and E. Ohlebusch, "Multiple Genome Alignment: Chaining Algorithms Revisited," in *Combinatorial Pattern Matching*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 1–16.

[471] C. Jain, D. Gibney, and S. V. Thankachan, "Co-linear Chaining with Overlaps and Gap Costs," in *Research in Computational Molecular Biology*. Cham: Springer International Publishing, 2022, pp. 246–262.

[472] G. Chandra and C. Jain, "Sequence to Graph Alignment Using Gap-Sensitive Co-linear Chaining," in *Research in Computational Molecular Biology*. Cham: Springer Nature Switzerland, 2023, pp. 58–73.

[473] N. Rizzo, M. Cáceres, and V. Mäkinen, "Chaining of Maximal Exact Matches in Graphs," in *String Processing and Information Retrieval*. Cham: Springer Nature Switzerland, 2023, pp. 353–366.

[474] C. Otto, S. Hoffmann, J. Gorodkin, and P. F. Stadler, "Fast local fragment chaining using sum-of-pair gap costs," *Algorithms for Molecular Biology*, vol. 6, no. 1, p. 4, Mar. 2011.

[475] J. Rajput, G. Chandra, and C. Jain, "Co-linear chaining on pangenome graphs," *Algorithms for Molecular Biology*, vol. 19, no. 1, p. 4, Jan. 2024.

[476] D. Eppstein, Z. Galil, R. Giancarlo, and G. F. Italiano, "Sparse dynamic programming I: linear cost functions," *J. ACM*, vol. 39, no. 3, pp. 519–545, Jul. 1992.

[477] D. Eppstein, Z. Galil, R. Giancarlo, and G. F. Italiano, "Sparse dynamic programming II: convex and concave cost functions," *J. ACM*, vol. 39, no. 3, pp. 546–567, Jul. 1992.

[478] G. Myers and W. Miller, "Chaining multiple-alignment fragments in sub-quadratic time," in *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '95. USA: Society for Industrial and Applied Mathematics, 1995, pp. 38–47.

[479] R. Uricaru, A. Mancheron, and E. Rivals, "Novel Definition and Algorithm for Chaining Fragments with Proportional Overlaps," *Journal of Computational Biology*, vol. 18, no. 9, pp. 1141–1154, Sep. 2011.

[480] C. Jain, D. Gibney, and S. V. Thankachan, "Algorithms for Colinear Chaining with Overlaps and Gap Costs," *Journal of Computational Biology*, vol. 29, no. 11, pp. 1237–1251, Nov. 2022.

[481] G. Chandra and C. Jain, "Gap-Sensitive Colinear Chaining Algorithms for Acyclic Pangenome Graphs," *Journal of Computational Biology*, vol. 30, no. 11, pp. 1182–1197, Nov. 2023.

[482] V. Mäkinen and K. Sahlin, "Chaining with Overlaps Revisited," in *31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 161. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020, pp. 25:1–25:12.

[483] S. Marco-Sola, J. C. Moure, M. Moreto, and A. Espinosa, "Fast gap-affine pairwise alignment using the wavefront algorithm," *Bioinformatics*, 2021.

[484] J. Lindegger, D. Senol Cali, M. Alser, J. Gómez-Luna, N. M. Ghiasi, and O. Mutlu, "Scrooge: a fast and memory-frugal genomic sequence aligner for CPUs, GPUs, and ASICs," *Bioinform.*, 2023.

[485] S. Marco-Sola, J. M. Eizenga, A. Guarracino, B. Paten, E. Garrison, and M. Moreto, "Optimal gap-affine alignment in O(s) space," *Bioinform.*, 2023.

[486] R. Baeza-Yates and G. H. Gonnet, "A New Approach to Text Searching," *Commun. ACM*, 1992.

[487] G. Myers, "A Fast Bit-Vector Algorithm for Approximate String Matching Based on Dynamic Programming," *J. ACM*, 1999.

[488] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *JMB*, 1970.

[489] T. Smith and M. Waterman, "Identification of common molecular subsequences," *JMB*, 1981.

[490] D. Senol Cali, G. S. Kalsi, Z. Bingöl, C. Firtina, L. Subramanian, J. S. Kim, R. Ausavarungnirun, M. Alser, J. Gomez-Luna, A. Boroumand, A. Norion, A. Scibisz, S. Subramoneyon, C. Alkan, S. Ghose, and O. Mutlu, "GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 951–966.

[491] D. Papamichail and G. Papamichail, "Improved algorithms for approximate string matching (extended abstract)," *BMC Bioinformatics*, vol. 10, no. 1, p. S10, Jan. 2009.

[492] H. Suzuki and M. Kasahara, "Introducing difference recurrence relations for faster semi-global alignment of long sequences," *BMC Bioinformatics*, vol. 19, no. 1, p. 45, Feb. 2018.

[493] M. Waterman, T. Smith, and W. Beyer, "Some biological sequence metrics," *Advances in Mathematics*, vol. 20, no. 3, pp. 367–387, Jun. 1976.

[494] S. Wu, U. Manber, G. Myers, and W. Miller, "An O(NP) sequence comparison algorithm," *Information Processing Letters*, vol. 35, no. 6, pp. 317–323, Sep. 1990.

[495] O. Gotoh, "An improved algorithm for matching biological sequences," *Journal of Molecular Biology*, vol. 162, no. 3, pp. 705–708, Dec. 1982.

[496] S. Wu and U. Manber, "Fast text searching: allowing errors," *Commun. ACM*, vol. 35, no. 10, pp. 83–91, Oct. 1992.

[497] R. A. Wagner and M. J. Fischer, "The String-to-String Correction Problem," *J. ACM*, vol. 21, no. 1, pp. 168–173, Jan. 1974.

[498] D. Sankoff, "Matching Sequences under Deletion/Insertion Constraints," *Proceedings of the National Academy of Sciences*, vol. 69, no. 1, pp. 4–6, Jan. 1972.

[499] R. Groot Koerkamp and P. Ivanov, "Exact global alignment using A* with chaining seed heuristic and match pruning," *Bioinformatics*, vol. 40, no. 3, p. btae032, Mar. 2024.

[500] P. H. Sellers, "On the Theory and Computation of Evolutionary Distances," *SIAM Journal on Applied Mathematics*, vol. 26, no. 4, pp. 787–793, Jun. 1974.

[501] E. Ukkonen, "Algorithms for approximate string matching," *International Conference on Foundations of Computation Theory*, vol. 64, no. 1, pp. 100–118, Jan. 1985.

[502] J. S. Kim, C. Firtina, M. B. Cavlak, D. Senol Cali, N. Hajinazar, M. Alser, C. Alkan, and O. Mutlu, "AirLift: A Fast and Comprehensive Technique for Remapping Alignments between Reference Genomes," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pp. 1–9, 2024.

[503] J. S. Kim, C. Firtina, M. B. Cavlak, D. Senol Cali, C. Alkan, and O. Mutlu, "FastRemap: a tool for quickly remapping reads between genome assemblies," *Bioinformatics*, vol. 38, no. 19, pp. 4633–4635, Sep. 2022.

[504] J. D. Kececioglu and E. W. Myers, "Combinatorial algorithms for DNA sequence assembly," *Algorithmica*, vol. 13, no. 1, pp. 7–51, Feb. 1995.

[505] H. Cheng, G. T. Concepcion, X. Feng, H. Zhang, and H. Li, "Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm," *Nature Methods*, vol. 18, no. 2, pp. 170–175, Feb. 2021.

[506] B. Ekim, B. Berger, and R. Chikhi, "Minimizer-space de Bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer," *Cell Systems*, vol. 12, no. 10, pp. 958–968.e6, Oct. 2021.

[507] S. Nurk, B. P. Walenz, A. Rhie, M. R. Vollger, G. A. Logsdon, R. Grothe, K. H. Miga, E. E. Eichler, A. M. Phillippy, and S. Koren, "HiCanu: accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads," *Genome Research*, vol. 30, no. 9, pp. 1291–1305, Sep. 2020. [Online]. Available: http://genome.cshlp.org/content/30/9/1291.abstract

[508] Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean, "De novo assembly and genotyping of variants using colored de Bruijn graphs," *Nature Genetics*, vol. 44, no. 2, pp. 226–232, Feb. 2012.

[509] C.-S. Chin, P. Peluso, F. J. Sedlazeck, M. Nattestad, G. T. Concepcion, A. Clum, C. Dunn, R. O'Malley, R. Figueroa-Balderas, A. Morales-Cruz, G. R. Cramer, M. Delledonne, C. Luo, J. R. Ecker, D. Cantu, D. R. Rank, and M. C. Schatz, "Phased diploid genome assembly with single-molecule real-time sequencing," *Nature Methods*, vol. 13, no. 12, pp. 1050–1054, Dec. 2016.

[510] C.-L. Xiao, Y. Chen, S.-Q. Xie, K.-N. Chen, Y. Wang, Y. Han, F. Luo, and Z. Xie, "MECAT: fast mapping, error correction, and de novo assembly for single-molecule sequencing reads," *Nature Methods*, vol. 14, no. 11, pp. 1072–1074, Nov. 2017.

[511] Y. Chen, F. Nie, S.-Q. Xie, Y.-F. Zheng, Q. Dai, T. Bray, Y.-X. Wang, J.-F. Xing, Z.-J. Huang, D.-P. Wang, L.-J. He, F. Luo, J.-X. Wang, Y.-Z. Liu, and C.-L. Xiao, "Efficient assembly of nanopore reads via highly accurate and intact error correction," *Nature Communications*, vol. 12, no. 1, p. 60, Jan. 2021.

[512] H. Li and R. Durbin, "Genome assembly in the telomere-to-telomere era," *Nature Reviews Genetics*, vol. 25, no. 9, pp. 658–670, Sep. 2024.

[513] E. D. Jarvis, G. Formenti, A. Rhie, A. Guarracino, C. Yang, J. Wood, A. Tracey, F. Thibaud-Nissen, M. R. Vollger, D. Porubsky, H. Cheng, M. Asri, G. A. Logsdon, P. Carnevali, M. J. P. Chaisson, C.-S. Chin, S. Cody, J. Collins, P. Ebert, M. Escalona, O. Fedrigo, R. S. Fulton, L. L. Fulton, S. Garg, J. L. Gerton, J. Ghurye, A. Granat, R. E. Green, W. Harvey, P. Hasenfeld, A. Hastie, M. Haukness, E. B. Jaeger, M. Jain, M. Kirsche, M. Kolmogorov, J. O. Korbel, S. Koren, J. Korlach, J. Lee, D. Li, T. Lindsay, J. Lucas, F. Luo, T. Marschall, M. W. Mitchell, J. McDaniel, F. Nie, H. E. Olsen, N. D. Olson, T. Pesout, T. Potapova, D. Puiu, A. Regier, J. Ruan, S. L. Salzberg, A. D. Sanders, M. C. Schatz, A. Schmitt, V. A. Schneider, S. Selvaraj, K. Shafin, A. Shumate, N. O. Stitziel, C. Stober, J. Torrance, J. Wagner, J. Wang, A. Wenger, C. Xiao, A. V. Zimin, G. Zhang, T. Wang, H. Li, E. Garrison, D. Haussler, I. Hall, J. M. Zook, E. E. Eichler, A. M. Phillippy, B. Paten, K. Howe, K. H. Miga, and Human Pangenome Reference Consortium, "Semi-automated assembly of high-quality diploid human reference genomes," *Nature*, vol. 611, no. 7936, pp. 519–531, Nov. 2022.

[514] M. Kolmogorov, J. Yuan, Y. Lin, and P. A. Pevzner, "Assembly of long, error-prone reads using repeat graphs," *Nature Biotechnology*, vol. 37, no. 5, pp. 540–546, May 2019.

[515] K. Shafin, T. Pesout, R. Lorig-Roach, M. Haukness, H. E. Olsen, C. Bosworth, J. Armstrong, K. Tigyi, N. Maurer, S. Koren, F. J. Sedlazeck, T. Marschall, S. Mayes, V. Costa, J. M. Zook, K. J. Liu, D. Kilburn, M. Sorensen, K. M. Munson, M. R. Vollger, J. Monlong, E. Garrison, E. E. Eichler, S. Salama, D. Haussler, R. E. Green, M. Akeson, A. Phillippy, K. H. Miga, P. Carnevali, M. Jain, and B. Paten, "Nanopore sequencing and the Shasta toolkit enable efficient de novo assembly of eleven human genomes," *Nature Biotechnology*, vol. 38, no. 9, pp. 1044–1053, Sep. 2020.

[516] A. Di Genova, E. Buena-Atienza, S. Ossowski, and M.-F. Sagot, "Efficient hybrid de novo assembly of human genomes with WENGAN," *Nature Biotechnology*, vol. 39, no. 4, pp. 422–430, Apr. 2021.

[517] A. Bankevich, A. V. Bzikadze, M. Kolmogorov, D. Antipov, and P. A. Pevzner, "Multiplex de Bruijn graphs enable genome assembly from long, high-fidelity reads," *Nature Biotechnology*, vol. 40, no. 7, pp. 1075–1081, Jul. 2022.

[518] H. Cheng, E. D. Jarvis, O. Fedrigo, K.-P. Koepfli, L. Urban, N. J. Gemmell, and H. Li, "Haplotype-resolved assembly of diploid genomes without parental data," *Nature Biotechnology*, vol. 40, no. 9, pp. 1332–1335, Sep. 2022.

[519] M. Rautiainen, S. Nurk, B. P. Walenz, G. A. Logsdon, D. Porubsky, A. Rhie, E. E. Eichler, A. M. Phillippy, and S. Koren, "Telomere-to-telomere assembly of diploid chromosomes with Verkko," *Nature Biotechnology*, vol. 41, no. 10, pp. 1474–1482, Oct. 2023.

[520] J. Ruan and H. Li, "Fast and accurate long-read assembly with wtdbg2," *Nature Methods*, vol. 17, no. 2, pp. 155–158, Feb. 2020.

[521] H. Cheng, M. Asri, J. Lucas, S. Koren, and H. Li, "Scalable telomere-to-telomere assembly for diploid and polyploid genomes with double graph," *Nature Methods*, vol. 21, no. 6, pp. 967–970, Jun. 2024.

[522] C. Eché, C. Iampietro, C. Birbes, A. Dréau, C. Kuchly, A. Di Franco, C. Klopp, T. Faraut, S. Djebali, A. Castinel, M. Zytnicki, E. Denis, M. Boussaha, C. Grohs, D. Boichard, C. Gaspin, D. Milan, and C. Donnadieu, "A Bos taurus sequencing methods benchmark for assembly, haplotyping, and variant calling," *Scientific Data*, vol. 10, no. 1, p. 369, Jun. 2023.

[523] R. Vaser and M. Šikić, "Time- and memory-efficient genome assembly with Raven," *Nature Computational Science*, vol. 1, no. 5, pp. 332–336, May 2021.

[524] Y. Chen, Y. Zhang, A. Y. Wang, M. Gao, and Z. Chong, "Accurate long-read de novo assembly evaluation with Inspector," *Genome Biology*, vol. 22, no. 1, p. 312, Nov. 2021.

[525] P. A. Pevzner, H. Tang, and M. S. Waterman, "An Eulerian path approach to DNA fragment assembly," *Proceedings of the National Academy of Sciences*, vol. 98, no. 17, pp. 9748–9753, Aug. 2001.

[526] Y. Lin, J. Yuan, M. Kolmogorov, M. W. Shen, M. Chaisson, and P. A. Pevzner, "Assembly of long error-prone reads using de Bruijn graphs," *Proceedings of the National Academy of Sciences*, vol. 113, no. 52, pp. E8396–E8405, Dec. 2016.

[527] J. K. Bonfield, K. F. Smith, and R. Staden, "A new DNA sequence assembly program," *Nucleic Acids Research*, vol. 23, no. 24, pp. 4992–4999, Dec. 1995.

[528] H. Peltola, H. Söderlund, and E. Ukkonen, "SEQAID: a DNA sequence assembling program based on a mathematical model," *Nucleic Acids Research*, vol. 12, no. 1Part1, pp. 307–321, Jan. 1984.

[529] G. M. Kamath, I. Shomorony, F. Xia, T. A. Courtade, and D. N. Tse, "HINGE: long-read assembly achieves optimal repeat resolution," *Genome Research*, vol. 27, no. 5, pp. 747–756, May 2017.

[530] J. Butler, I. MacCallum, M. Kleber, I. A. Shlyakhter, M. K. Belmonte, E. S. Lander, C. Nusbaum, and D. B. Jaffe, "ALLPATHS: De novo assembly of whole-genome shotgun microreads," *Genome Research*, vol. 18, no. 5, pp. 810–820, May 2008.

[531] S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and A. M. Phillippy, "Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation," *Genome Research*, vol. 27, no. 5, pp. 722–736, May 2017.

[532] E. W. Myers, "The fragment assembly string graph," *Bioinform.*, 2005.

[533] C. Alkan, B. P. Coe, and E. E. Eichler, "Genome structural variation discovery and genotyping," *Nature Reviews Genetics*, vol. 12, no. 5, pp. 363–376, May 2011.

[534] F. J. Sedlazeck, P. Rescheneder, M. Smolka, H. Fang, M. Nattestad, A. von Haeseler, and M. C. Schatz, "Accurate detection of complex structural variations using single-molecule sequencing," *Nature Methods*, vol. 15, no. 6, pp. 461–468, Jun. 2018.

[535] R. Poplin, V. Ruano-Rubio, M. A. DePristo, T. J. Fennell, M. O. Carneiro, G. A. Van der Auwera, D. E. Kling, L. D. Gauthier, A. Levy-Moonshine, D. Roazen, K. Shakir, J. Thibault, S. Chandran, C. Whelan, M. Lek, S. Gabriel, M. J. Daly, B. Neale, D. G. MacArthur, and E. Banks, "Scaling accurate genetic variant discovery to tens of thousands of samples," *bioRxiv*, 2018.

[536] S. Weckx, J. Del-Favero, R. Rademakers, L. Claes, M. Cruts, P. De Jonghe, C. Van Broeckhoven, and P. De Rijk, "novoSNP, a novel computational tool for sequence variation discovery," *Genome Res.*, 2005.

[537] P.-Y. Kwok, C. Carlson, T. D. Yager, W. Ankener, and D. A. Nickerson, "Comparative Analysis of Human DNA Variations by Fluorescence-Based Sequencing of PCR Products," *Genomics*, 1994.

[538] D. A. Nickerson, V. O. Tobe, and S. L. Taylor, "PolyPhred: automating the detection and genotyping of single nucleotide substitutions using fluorescence-based resequencing," *NAR*, 1997.

[539] G. T. Marth, I. Korf, M. D. Yandell, R. T. Yeh, Z. Gu, H. Zakeri, N. O. Stitziel, L. Hillier, P.-Y. Kwok, and W. R. Gish, "A general approach to single-nucleotide polymorphism discovery," *Nat. Genetics*, 1999.

[540] R. Poplin, P.-C. Chang, D. Alexander, S. Schwartz, T. Colthurst, A. Ku, D. Newburger, J. Dijamco, N. Nguyen, P. T. Afshar, S. S. Gross, L. Dorfman, C. Y. McLean, and M. A. DePristo, "A universal SNP and small-indel variant caller using deep neural networks," *Nat. Biotech.*, 2018.

[541] D. F. Conrad, D. Pinto, R. Redon, L. Feuk, O. Gokcumen, Y. Zhang, J. Aerts, T. D. Andrews, C. Barnes, P. Campbell, T. Fitzgerald, M. Hu, C. H. Ihm, K. Kristiansson, D. G. MacArthur, J. R. MacDonald, I. Onyiah, A. W. C. Pang, S. Robson, K. Stirrups, A. Valsesia, K. Walter, J. Wei, C. Tyler-Smith, N. P. Carter, C. Lee, S. W. Scherer, M. E. Hurles, and The Wellcome Trust Case Control Consortium, "Origins and functional impact of copy number variation in the human genome," *Nature*, vol. 464, no. 7289, pp. 704–712, Apr. 2010.

[542] R. E. Mills, K. Walter, C. Stewart, R. E. Handsaker, K. Chen, C. Alkan, A. Abyzov, S. C. Yoon, K. Ye, R. K. Cheetham, A. Chinwalla, D. F. Conrad, Y. Fu, F. Grubert, I. Hajirasouliha, F. Hormozdiari, L. M. Iakoucheva, Z. Iqbal, S. Kang, J. M. Kidd, M. K. Konkel, J. Korn, E. Khurana, D. Kural, H. Y. K. Lam, J. Leng, R. Li, Y. Li, C.-Y. Lin, R. Luo, X. J. Mu, J. Nemesh, H. E. Peckham, T. Rausch, A. Scally, X. Shi, M. P. Stromberg, A. M. Stütz, A. E. Urban, J. A. Walker, J. Wu, Y. Zhang, Z. D. Zhang, M. A. Batzer, L. Ding, G. T. Marth, G. McVean, J. Sebat, M. Snyder, J. Wang, K. Ye, E. E. Eichler, M. B. Gerstein, M. E. Hurles, C. Lee, S. A. McCarroll, J. O. Korbel, and 1000 Genomes Project, "Mapping copy number variation by population-scale genome sequencing," *Nature*, vol. 470, no. 7332, pp. 59–65, Feb. 2011.

[543] G. M. Cooper, T. Zerr, J. M. Kidd, E. E. Eichler, and D. A. Nickerson, "Systematic assessment of copy number variant detection via genome-wide SNP genotyping," *Nature Genetics*, vol. 40, no. 10, pp. 1199–1203, Oct. 2008.

[544] E. E. Eichler, "Widening the spectrum of human genetic variation," *Nature Genetics*, vol. 38, no. 1, pp. 9–11, Jan. 2006.

[545] D. L. Cameron, L. Di Stefano, and A. T. Papenfuss, "Comprehensive evaluation and characterisation of short read general-purpose structural variant calling software," *Nature Communications*, vol. 10, no. 1, p. 3240, Jul. 2019.

[546] L. Han, X. Zhao, M. L. Benton, T. Perumal, R. L. Collins, G. E. Hoffman, J. S. Johnson, L. Sloofman, H. Z. Wang, M. R. Stone, S. Akbarian, J. Bendl, M. Breen, K. J. Brennand, L. Brown, A. Browne, J. D. Buxbaum, A. Charney, A. Chess, L. Couto, G. Crawford, O. Devillers, B. Devlin, A. Dobbyn, E. Domenici, M. Filosi, E. Flatow, N. Francoeur, J. Fullard, S. E. Gil, K. Girdhar, A. Gulyás-Kovács, R. Gur, C.-G. Hahn, V. Haroutunian, M. E. Hauberg, L. Huckins, R. Jacobov, Y. Jiang, J. S. Johnson, B. Kassim, Y. Kim, L. Klei, R. Kramer, M. Lauria, T. Lehner, D. A. Lewis, B. K. Lipska, K. Montgomery, R. Park, C. Rosenbluh, P. Roussos, D. M. Ruderfer, G. Senthil, H. R. Shah, L. Sloofman, L. Song, E. Stahl, P. Sullivan, R. Visintainer, J. Wang, Y.-C. Wang, J. Wiseman, E. Xia, W. Zhang, E. Zharovsky, K. J. Brennand, H. Brand, S. K. Sieberts, S. Marenco, M. A. Peters, B. K. Lipska, P. Roussos, J. A. Capra, M. Talkowski, D. M. Ruderfer, and CommonMind Consortium, "Functional annotation of rare structural variation in the human brain," *Nature Communications*, vol. 11, no. 1, p. 2990, Jun. 2020.

[547] B. Mandiracioglu, F. Ozden, G. Kaynar, M. A. Yilmaz, C. Alkan, and A. E. Cicek, "ECOLE: Learning to call copy number variants on whole exome sequencing data," *Nature Communications*, vol. 15, no. 1, p. 132, Jan. 2024.

[548] Y. Dou, M. Kwon, R. E. Rodin, I. Cortés-Ciriano, R. Doan, L. J. Luquette, A. Galor, C. Bohrson, C. A. Walsh, and P. J. Park, "Accurate detection of mosaic variants in sequencing data without matched controls," *Nature Biotechnology*, vol. 38, no. 3, pp. 314–319, Mar. 2020.

[549] M. Smolka, L. F. Paulin, C. M. Grochowski, D. W. Horner, M. Mahmoud, S. Behera, E. Kalef-Ezra, M. Gandhi, K. Hong, D. Pehlivan, S. W. Scholz, C. M. B. Carvalho, C. Proukakis, and F. J. Sedlazeck, "Detection of mosaic and population-level structural variants with Sniffles2," *Nature Biotechnology*, Jan. 2024.

[550] J. Lin, S. Wang, P. A. Audano, D. Meng, J. I. Flores, W. Kosters, X. Yang, P. Jia, T. Marschall, C. R. Beck, and K. Ye, "SVision: a deep learning approach to resolve complex structural variants," *Nature Methods*, vol. 19, no. 10, pp. 1230–1233, Oct. 2022.

[551] V. Popic, C. Rohlicek, F. Cunial, I. Hajirasouliha, D. Meleshko, K. Garimella, and A. Maheshwari, "Cue: a deep-learning framework for structural variant discovery and genotyping," *Nature Methods*, vol. 20, no. 4, pp. 559–568, Apr. 2023.

[552] G. Narzisi, A. Corvelo, K. Arora, E. A. Bergmann, M. Shah, R. Musunuri, A.-K. Emde, N. Robine, V. Vacic, and M. C. Zody, "Genome-wide somatic variant calling using localized colored de Bruijn graphs," *Communications Biology*, vol. 1, no. 1, p. 20, Mar. 2018.

[553] Z. Zheng, S. Li, J. Su, A. W.-S. Leung, T.-W. Lam, and R. Luo, "Symphonizing pileup and full-alignment for deep learning-based long-read variant calling," *Nature Computational Science*, vol. 2, no. 12, pp. 797–803, Dec. 2022.

[554] J. Zhang, J. Wang, and Y. Wu, "An improved approach for accurate and efficient calling of structural variations with low-coverage sequence data," *BMC Bioinformatics*, vol. 13, no. 6, p. S6, Apr. 2012.

[555] R. M. Layer, C. Chiang, A. R. Quinlan, and I. M. Hall, "LUMPY: a probabilistic framework for structural variant discovery," *Genome Biology*, vol. 15, no. 6, p. R84, Jun. 2014.

[556] F. Karaoğlanoğlu, C. Ricketts, E. Ebren, M. E. Rasekh, I. Hajirasouliha, and C. Alkan, "VALOR2: characterization of large-scale structural variants using linked-reads," *Genome Biology*, vol. 21, no. 1, p. 72, Mar. 2020.

[557] T. Jiang, Y. Liu, Y. Jiang, J. Li, Y. Gao, Z. Cui, Y. Liu, B. Liu, and Y. Wang, "Long-read-based human genomic structural variation detection with cuteSV," *Genome Biology*, vol. 21, no. 1, p. 189, Aug. 2020.

[558] M. U. Ahsan, Q. Liu, L. Fang, and K. Wang, "NanoCaller for accurate detection of SNPs and indels in difficult-to-map regions from long-read sequencing by haplotype-aware deep neural networks," *Genome Biology*, vol. 22, no. 1, p. 261, Sep. 2021.

[559] A. E. Minoche, B. Lundie, G. B. Peters, T. Ohnesorg, M. Pinese, D. M. Thomas, A. Zankl, T. Roscioli, N. Schonrock, S. Kummerfeld, L. Burnett, M. E. Dinger, and M. J. Cowley, "ClinSV: clinical grade structural and copy number variant detection from whole genome sequencing data," *Genome Medicine*, vol. 13, no. 1, p. 32, Feb. 2021.

[560] N. A. Baird, P. D. Etter, T. S. Atwood, M. C. Currey, A. L. Shiver, Z. A. Lewis, E. U. Selker, W. A. Cresko, and E. A. Johnson, "Rapid SNP Discovery and Genetic Mapping Using Sequenced RAD Markers," *PLOS ONE*, vol. 3, no. 10, p. e3376, Oct. 2008.

[561] S. Zarate, A. Carroll, M. Mahmoud, O. Krasheninina, G. Jun, W. J. Salerno, M. C. Schatz, E. Boerwinkle, R. A. Gibbs, and F. J. Sedlazeck, "Parliament2: Accurate structural variant calling at scale," *GigaScience*, vol. 9, no. 12, p. giaa145, Nov. 2020.

[562] S. O'Donnell and G. Fischer, "MUM&Co: accurate detection of all SV types through whole-genome alignment," *Bioinformatics*, vol. 36, no. 10, pp. 3242–3243, May 2020.

[563] C. Xu, X. Gu, R. Padmanabhan, Z. Wu, Q. Peng, J. DiCarlo, and Y. Wang, "smCounter2: an accurate low-frequency variant caller for targeted sequencing data with unique molecular identifiers," *Bioinformatics*, vol. 35, no. 8, pp. 1299–1309, Apr. 2019.

[564] D. Heller and M. Vingron, "SVIM: structural variant identification using mapped long reads," *Bioinformatics*, vol. 35, no. 17, pp. 2907–2915, Sep. 2019.

[565] B. S. Pedersen and A. R. Quinlan, "cyvcf2: fast, flexible variant analysis with Python," *Bioinformatics*, vol. 33, no. 12, pp. 1867–1869, Jun. 2017.

[566] H. Li, "FermiKit: assembly-based variant calling for Illumina resequencing data," *Bioinformatics*, vol. 31, no. 22, pp. 3694–3696, Nov. 2015.

[567] J. Eisfeldt, F. Vezzi, P. Olason, D. Nilsson, and A. Lindstrand, "TIDDIT, an efficient and comprehensive structural variant caller for massive parallel sequencing data [version 2; peer review: 2 approved]," *F1000Research*, vol. 6, no. 664, 2017.

[568] Y. Zheng, X. Shang, and W.-K. Sung, "SVsearcher: A more accurate structural variation detection method in long read data," *Computers in Biology and Medicine*, vol. 158, p. 106843, May 2023.

[569] P. Medvedev, M. Fiume, M. Dzamba, T. Smith, and M. Brudno, "Detecting copy number variation with mated short reads," *Genome Research*, vol. 20, no. 11, pp. 1613–1622, Nov. 2010.

[570] E. Garrison and G. Marth, "Haplotype-based variant detection from short-read sequencing," *arXiv*, 2012.

[571] N. LaPierre, M. Alser, E. Eskin, D. Koslicki, and S. Mangul, "Metalign: efficient alignment-based metagenomic profiling via containment min hash," *Genome biology*, vol. 21, no. 1, pp. 1–15, 2020, publisher: BioMed Central.

[572] D. E. Wood, J. Lu, and B. Langmead, "Improved metagenomic analysis with Kraken 2," *Genome Biology*, vol. 20, no. 1, p. 257, Nov. 2019.

[573] N. Segata, L. Waldron, A. Ballarini, V. Narasimhan, O. Jousson, and C. Huttenhower, "Metagenomic microbial community profiling using unique clade-specific marker genes," *Nature Methods*, vol. 9, no. 8, pp. 811–814, Aug. 2012.

[574] J. Alneberg, B. S. Bjarnason, I. de Bruijn, M. Schirmer, J. Quick, U. Z. Ijaz, L. Lahti, N. J. Loman, A. F. Andersson, and C. Quince, "Binning metagenomic contigs by coverage and composition," *Nature Methods*, vol. 11, no. 11, pp. 1144–1146, Nov. 2014.

[575] A. Milanese, D. R. Mende, L. Paoli, G. Salazar, H.-J. Ruscheweyh, M. Cuenca, P. Hingamp, R. Alves, P. I. Costea, L. P. Coelho, T. S. B. Schmidt, A. Almeida, A. L. Mitchell, R. D. Finn, J. Huerta-Cepas, P. Bork, G. Zeller, and S. Sunagawa, "Microbial abundance, activity and population genomic profiling with mOTUs2," *Nature Communications*, vol. 10, no. 1, p. 1014, Mar. 2019.

[576] A. Blanco-Míguez, F. Beghini, F. Cumbo, L. J. McIver, K. N. Thompson, M. Zolfo, P. Manghi, L. Dubois, K. D. Huang, A. M. Thomas, W. A. Nickols, G. Piccinno, E. Piperni, M. Punčochář, M. Valles-Colomer, A. Tett, F. Giordano, R. Davies, J. Wolf, S. E. Berry, T. D. Spector, E. A. Franzosa, E. Pasolli, F. Asnicar, C. Huttenhower, and N. Segata, "Extending and improving metagenomic taxonomic profiling with uncharacterized species using MetaPhlAn 4," *Nature Biotechnology*, vol. 41, no. 11, pp. 1633–1644, Nov. 2023.

[577] Z. Zhao, A. Cristian, and G. Rosen, "Keeping up with the genomes: efficient learning of our increasing knowledge of the tree of life," *BMC Bioinformatics*, vol. 21, no. 1, p. 412, Sep. 2020.

[578] R. Ounit, S. Wanamaker, T. J. Close, and S. Lonardi, "CLARK: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers," *BMC Genomics*, vol. 16, no. 1, p. 236, Mar. 2015.

[579] V. R. Marcelino, P. T. L. C. Clausen, J. P. Buchmann, M. Wille, J. R. Iredell, W. Meyer, O. Lund, T. C. Sorrell, and E. C. Holmes, "CCMetagen: comprehensive and accurate identification of eukaryotes and prokaryotes in metagenomic data," *Genome Biology*, vol. 21, no. 1, p. 103, Apr. 2020.

[580] L. Song and B. Langmead, "Centrifuger: lossless compression of microbial genomes for efficient and accurate metagenomic sequence classification," *Genome Biology*, vol. 25, no. 1, p. 106, Apr. 2024.

[581] Koslicki David and Falush Daniel, "MetaPalette: a k-mer Painting Approach for Metagenomic Taxonomic Profiling and Quantification of Novel Strain Variation," *mSystems*, vol. 1, no. 3, pp. 10.1128/msystems.00 020–16, Jun. 2016.

[582] V. C. Piro, M. S. Lindner, and B. Y. Renard, "DUDes: a top-down taxonomic profiler for metagenomics," *Bioinformatics*, vol. 32, no. 15, pp. 2272–2280, Aug. 2016.

[583] Y.-W. Wu, B. A. Simmons, and S. W. Singer, "MaxBin 2.0: an automated binning algorithm to recover genomes from multiple metagenomic datasets," *Bioinformatics*, vol. 32, no. 4, pp. 605–607, Feb. 2016.

[584] V. C. Piro, T. H. Dadi, E. Seiler, K. Reinert, and B. Y. Renard, "ganon: precise metagenomics classification against large and up-to-date sets of reference sequences," *Bioinformatics*, vol. 36, no. Supplement_1, pp. i12–i20, Jul. 2020.

[585] W. Shen, H. Xiang, T. Huang, H. Tang, M. Peng, D. Cai, P. Hu, and H. Ren, "KMCP: accurate metagenomic profiling of both prokaryotic and viral populations by pseudo-mapping," *Bioinformatics*, vol. 39, no. 1, p. btac845, Jan. 2023.

[586] D. D. Kang, J. Froula, R. Egan, and Z. Wang, "MetaBAT, an efficient tool for accurately reconstructing single genomes from complex microbial communities," *PeerJ*, vol. 3, p. e1165, 2015.

[587] D. D. Kang, F. Li, E. Kirton, A. Thomas, R. Egan, H. An, and Z. Wang, "MetaBAT 2: an adaptive binning algorithm for robust and efficient genome reconstruction from metagenome assemblies," *PeerJ*, vol. 7, p. e7359, 2019.

[588] J. Lu, F. P. Breitwieser, P. Thielen, and S. L. Salzberg, "Bracken: estimating species abundance in metagenomics data," *PeerJ Computer Science*, vol. 3, p. e104, 2017.

[589] D. Kim, L. Song, F. P. Breitwieser, and S. L. Salzberg, "Centrifuge: rapid and sensitive classification of metagenomic sequences," *Genome Research*, vol. 26, no. 12, pp. 1721–1729, Dec. 2016.

[590] F. P. Breitwieser, D. N. Baker, and S. L. Salzberg, "KrakenUniq: confident and fast metagenomics classification using unique k-mer counts," *Genome Biology*, vol. 19, no. 1, p. 198, Nov. 2018.

[591] Z. Jahshan and L. Yavits, "ViTAL: Vision TrAnsformer based Low coverage SARS-CoV-2 lineage assignment," *Bioinformatics*, vol. 40, no. 3, p. btae093, Mar. 2024.

[592] S. Liu and D. Koslicki, "CMash: fast, multi-resolution estimation of k-mer-based Jaccard and containment indices," *Bioinform.*, 2022.

[593] C. Firtina, K. Pillai, G. S. Kalsi, B. Suresh, D. S. Cali, J. S. Kim, T. Shahroodi, M. B. Cavlak, J. Lindegger, M. Alser, J. G. Luna, S. Subramoney, and O. Mutlu, "ApHMM: Accelerating Profile Hidden Markov Models for Fast and Energy-efficient Genome Analysis," *ACM Trans. Archit. Code Optim.*, 2024.

[594] T. Pan, P. Flick, C. Jain, Y. Liu, and S. Aluru, "Kmerind: A Flexible Parallel Library for K-Mer Indexing of Biological Sequences on Distributed Memory Systems," in *BCB*, 2016.

[595] G. Guidi, M. Ellis, D. Rokhsar, K. Yelick, and A. Buluç, "BELLA: Berkeley Efficient Long-Read to Long-Read Aligner and Overlapper," in *Proceedings of the 2021 SIAM Conference on Applied and Computational Discrete Algorithms (ACDA21)*, ser. Proceedings. Society for Industrial and Applied Mathematics, Jan. 2021, pp. 123–134.

[596] M. Ellis, G. Guidi, A. Buluç, L. Oliker, and K. Yelick, "DiBELLA: Distributed Long Read to Long Read Alignment," in *ICPP*, 2019.

[597] K. Kanellopoulos, N. Vijaykumar, C. Giannoula, R. Azizi, S. Koppula, N. M. Ghiasi, T. Shahroodi, J. G. Luna, and O. Mutlu, "SMASH: Co-designing Software Compression and Hardware-Accelerated Indexing for Efficient Sparse Matrix Operations," in *MICRO*, 2019.

[598] D. Senol Cali, K. Kanellopoulos, J. Lindegger, Z. Bingöl, G. S. Kalsi, Z. Zuo, C. Firtina, M. B. Cavlak, J. Kim, N. M. Ghiasi, G. Singh, J. Gómez-Luna, N. A. Alserr, M. Alser, S. Subramoney, C. Alkan, S. Ghose, and O. Mutlu, "SeGraM: A universal hardware accelerator for genomic sequence-to-graph and sequence-to-sequence mapping," in *ISCA*, 2022.

[599] V. K. Kundeti, "PaKman+: Fast Distributed Sequence Assembly with a Concurrent K-Mer Counting Algorithm," in *BIBE*, 2023, pp. 6–13.

[600] E. Georganas, A. Buluç, J. Chapman, S. Hofmeyr, C. Aluru, R. Egan, L. Oliker, D. Rokhsar, and K. Yelick, "HipMer: an extreme-scale de novo genome assembler," in *SC*, 2015, pp. 1–11.

[601] P. Ghosh, S. Krishnamoorthy, and A. Kalyanaraman, "PaKman: Scalable Assembly of Large Genomes on Distributed Memory Machines," in *IPDPS*, 2019, pp. 578–589.

[602] P. Ghosh, S. Krishnamoorthy, and A. Kalyanaraman, "PaKman: A Scalable Algorithm for Generating Genomic Contigs on Distributed Memory Machines," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1191–1209, 2021.

[603] E. Georganas, R. Egan, S. Hofmeyr, E. Goltsman, B. Arndt, A. Tritt, A. Buluç, L. Oliker, and K. Yelick, "Extreme Scale De Novo Metagenome Assembly," in *SC*, 2018, pp. 122–134.

[604] K. Mahadik, C. Wright, M. Kulkarni, S. Bagchi, and S. Chaterji, "Scalable Genomic Assembly through Parallel de Bruijn Graph Construction for Multiple K-mers," in *ACM-BCB*, 2017, pp. 425–431.

[605] G. Guidi, O. Selvitopi, M. Ellis, L. Oliker, K. Yelick, and A. Buluç, "Parallel String Graph Construction and Transitive Reduction for De Novo Genome Assembly," in *IPDPS*, 2021, pp. 517–526.

[606] S. Goswami, K. Lee, and S.-J. Park, "Distributed de novo assembler for large-scale long-read datasets," in *Big Data*, 2020, pp. 1166–1175.

[607] T. Pan, R. Nihalani, and S. Aluru, "Fast de Bruijn Graph Compaction in Distributed Memory Environments," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 17, no. 1, pp. 136–148, 2020.

[608] S. Goswami, A. K. Das, R. Płatania, K. Lee, and S.-J. Park, "Lazer: Distributed memory-efficient assembly of large-scale genomes," in *Big Data*, 2016, pp. 1171–1181.

[609] J. Meng, B. Wang, Y. Wei, S. Feng, and P. Balaji, "SWAP-Assembler: scalable and efficient genome assembly towards thousands of cores," *BMC Bioinformatics*, vol. 15, no. 9, p. S2, Sep. 2014.

[610] E. Georganas, A. Buluç, J. Chapman, L. Oliker, D. Rokhsar, and K. Yelick, "Parallel De Bruijn Graph Construction and Traversal for De Novo Genome Assembly," in *SC*, 2014, pp. 437–448.

[611] S. Qiu and Q. Luo, "Parallelizing Big De Bruijn Graph Construction on Heterogeneous Processors," in *ICDCS*, 2017, pp. 1431–1441.

[612] A. Sinha, H.-C. Yang, P.-Y. Liu, Y.-S. Kuo, Y. Fang, T.-S. Chang, K.-H. Li, and B.-C. Lai, "DSIM: Distributed Sequence Matching on Near-DRAM Accelerator for Genome Assembly," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 12, no. 2, pp. 486–499, 2022.

[613] K. Poje, M. Brcic, J. Knezovic, and M. Kovac, "First Steps towards Efficient Genome Assembly on ARM-Based HPC," *Electronics*, vol. 13, no. 1, 2024.

[614] A. Kalyanaraman, S. Emrich, P. Schnable, and S. Aluru, "Assembling genomes on large-scale parallel computers," in *IPDPS*, 2006.

[615] G. Guidi, G. Raulet, D. Rokhsar, L. Oliker, K. Yelick, and A. Buluç, "Distributed-Memory Parallel Contig Generation for De Novo Long-Read Genome Assembly," in *ICPP*, 2023.

[616] S. Goswami, K. Lee, S. Shams, and S.-J. Park, "GPU-Accelerated Large-Scale Genome Assembly," in *IPDPS*, 2018, pp. 814–824.

[617] G. Jain, L. Rathore, and K. Paul, "GAMS: Genome Assembly on Multi-GPU Using String Graph," in *HPCC/SmartCity/DSS*, 2016, pp. 348–355.

[618] A. Garg, A. Jain, and K. Paul, "GGAKE: GPU Based Genome Assembly Using K-Mer Extension," in *HPC and EUC*, 2013, pp. 1105–1112.

[619] A. Jain, A. Garg, and K. Paul, "GAGM: Genome assembly on GPU using mate pairs," in *HiPC*, 2013, pp. 176–185.

[620] S. F. Mahmood and H. Rangwala, "GPU-Euler: Sequence Assembly Using GPGPU," in *HPCC*, 2011, pp. 153–160.

[621] P. Meng, M. Jacobsen, M. Kimura, V. Dergachev, T. Anantharaman, M. Requa, and R. Kastner, "Hardware accelerated novel optical de novo assembly for large-scale genomes," in *FPL*, 2014, pp. 1–8.

[622] S. Qiu, Z. Feng, and Q. Luo, "Parallelizing Big De Bruijn Graph Traversal for Genome Assembly on GPU Clusters," in *DASFAA*, G. Li, J. Yang, J. Gama, J. Natwichai, and Y. Tong, Eds., 2019, pp. 466–470.

[623] M. Lu, Q. Luo, B. Wang, J. Wu, and J. Zhao, "GPU-Accelerated Bidirected De Bruijn Graph Construction for Genome Assembly," in *APWeb*, Y. Ishikawa, J. Li, W. Wang, R. Zhang, and W. Zhang, Eds., 2013, pp. 51–62.

[624] N. Ahmed, T. D. Qiu, K. Bertels, and Z. Al-Ars, "GPU acceleration of Darwin read overlapper for de novo assembly of long DNA reads," *BMC Bioinformatics*, vol. 21, no. 13, p. 388, Sep. 2020.

[625] Q. Aguado-Puig, S. Marco-Sola, J. C. Moure, D. Castells-Rufas, L. Alvarez, A. Espinosa, and M. Moreto, "Accelerating edit-distance sequence alignment on GPU using the wavefront algorithm," *IEEE Access*, 2022.

[626] Q. Aguado-Puig, S. Marco-Sola, J. C. Moure, C. Matzoros, D. Castells-Rufas, A. Espinosa, and M. Moreto, "WFA-GPU: Gap-affine pairwise alignment using GPUs," *bioRxiv*, 2022.

[627] E. J. Houtgast, V.-M. Sima, K. Bertels, and Z. Al-Ars, "Hardware acceleration of BWA-MEM genomic short read mapping for longer read lengths," *Computational Biology and Chemistry*, 2018.

[628] A. Zeni, G. Guidi, M. Ellis, N. Ding, M. D. Santambrogio, S. Hofmeyr, A. Buluç, L. Oliker, and K. Yelick, "Logan: High-performance GPU-based X-drop long-read alignment," in *IPDPS*, 2020.

[629] N. Ahmed, J. Lévy, S. Ren, H. Mushtaq, K. Bertels, and Z. Al-Ars, "GASAL2: A GPU accelerated sequence alignment library for high-throughput NGS data," *BMC Bioinformatics*, 2019.

[630] E. F. de Oliveira Sandes, G. Miranda, X. Martorell, E. Ayguade, G. Teodoro, and A. C. M. Melo, "CUDAlign 4.0: Incremental speculative traceback for exact chromosome-wide alignment in GPU clusters," *TPDS*, 2016.

[631] Y. Liu and B. Schmidt, "GSWABE: Faster GPU-accelerated sequence alignment with optimal alignment retrieval for short DNA sequences," *Concurrency and Computation: Practice and Experience*, 2015.

[632] Y. Liu, A. Wirawan, and B. Schmidt, "CUDASW++ 3.0: Accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions," *BMC Bioinformatics*, 2013.

[633] Y. Liu, D. L. Maskell, and B. Schmidt, "CUDASW++: Optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units," *BMC Research Notes*, 2009.

[634] Y. Liu, B. Schmidt, and D. L. Maskell, "CUDASW++ 2.0: Enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions," *BMC Research Notes*, 2010.

[635] R. Wilton, T. Budavari, B. Langmead, S. J. Wheelan, S. L. Salzberg, and A. S. Szalay, "Arioc: High-throughput read alignment with GPU-accelerated exploration of the seed-and-extend search space," *PeerJ*, 2015.

[636] S. D. Goenka, Y. Turakhia, B. Paten, and M. Horowitz, "SegAlign: A scalable GPU-based whole genome aligner," in *SC*, 2020.

[637] T. Nishimura, J. L. Bordim, Y. Ito, and K. Nakano, "Accelerating the Smith-waterman algorithm using bitwise parallel bulk computation technique on GPU," in *IPDPSW*, 2017.

[638] J. Lindegger, D. S. Cali, M. Alser, J. Gòmez-Luna, and O. Mutlu, "Algorithmic Improvement and GPU Acceleration of the GenASM Algorithm," in *IPDPSW*, 2022.

[639] S. Park, H. Kim, T. Ahmad, N. Ahmed, Z. Al-Ars, H. P. Hofstee, Y. Kim, and J. Lee, "SALoBa: Maximizing Data Locality and Workload Balance for Fast Sequence Alignment on GPUs," in *IPDPS*, 2022.

[640] S. Park, J. Hong, J. Song, H. Kim, Y. Kim, and J. Lee, "AGAThA: Fast and Efficient GPU Acceleration of Guided Sequence Alignment for Long Read Mapping," in *PPoPP*, 2024.

[641] X. Kong, C. Shen, and J. Tang, "CUK-Band: A CUDA-Based Multiple Genomic Sequence Alignment on GPU," in *ICIC*, D.-S. Huang, Y. Pan, and Q. Zhang, Eds., 2024.

[642] L. Guo, J. Lau, Z. Ruan, P. Wei, and J. Cong, "Hardware acceleration of long read pairwise overlapping in genome sequencing: a race between FPGA and GPU," in *FCCM*, 2019.

[643] H. Sadasivan, M. Maric, E. Dawson, V. Iyer, J. Israeli, and S. Narayanasamy, "Accelerating Minimap2 for accurate long read alignment on GPUs," *Journal of biotechnology and biomedicine*, 2023.

[644] Z. Bingöl, M. Alser, O. Mutlu, O. Ozturk, and C. Alkan, "GateKeeper-GPU: Fast and accurate pre-alignment filtering in short read mapping," in *IPDPSW*. IEEE, 2021.

[645] E. J. Houtgast, V. Sima, K. Bertels, and Z. AlArs, "An efficient GPU-accelerated implementation of genomic short read mapping with BWA-MEM," *CAN*, 2017.

[646] H. Cheng, Y. Zhang, and Y. Xu, "Bitmapper2: A GPU-accelerated all-mapper based on the sparse Q-gram index," *IEEE/ACM TCBB*, 2018.

[647] T. C. Pan, S. Misra, and S. Aluru, "Optimizing High Performance Distributed Memory Parallel Hash Tables for DNA k-mer Counting," in *SC*, 2018, pp. 135–147.

[648] I. Nisa, P. Pandey, M. Ellis, L. Oliker, A. Buluç, and K. Yelick, "Distributed-Memory k-mer Counting on GPUs," in *IPDPS*, 2021, pp. 527–536.

[649] T. Gao, Y. Guo, Y. Wei, B. Wang, Y. Lu, P. Cicotti, P. Balaji, and M. Taufer, "Bloomfish: A Highly Scalable Distributed K-mer Counting Framework," in *ICPADS*, 2017, pp. 170–179.

[650] L. Mocheng, C. Zhiguang, X. Nong, L. Yang, L. Xi, and C. Tao, "TopKmer: Parallel High Frequency K-mer Counting on Distributed Memory," in *Network and Parallel Computing*, S. Liu and X. Wei, Eds., 2022, pp. 96–107.

[651] Y. Cheng, X. Sun, and Q. Luo, "RapidGKC: GPU-Accelerated K-Mer Counting," in *ICDE*, 2024, pp. 3810–3822.

[652] E. Li, S. S. Banerjee, S. Huang, R. K. Iyer, and D. Chen, "Improved GPU Implementations of the Pair-HMM Forward Algorithm for DNA Sequence Alignment," in *ICCD*, 2021.

[653] M. J. Das, K. Shehzad, and P. Rao, "Efficient Variant Calling on Human Genome Sequences Using a GPU-Enabled Commodity Cluster," in *CIKM*, 2023.

[654] R. Kobus, A. Müller, D. Jünger, C. Hundt, and B. Schmidt, "MetaCache-GPU: Ultra-Fast Metagenomic Classification," in *ICPP*, New York, NY, USA, 2021.

[655] P. Jia, L. Xuan, L. Liu, and C. Wei, "MetaBinG: Using GPUs to accelerate metagenomic sequence classification," *PLOS ONE*, vol. 6, no. 11, p. e25353, 2011.

[656] X. Wang, T. Wang, Z. Xie, Y. Zhang, S. Xia, R. Sun, X. He, R. Xiang, Q. Zheng, Z. Liu, J. Wang, H. Wu, X. Jin, W. Chen, D. Li, and Z. He, "GPMeta: a GPU-accelerated method for ultrarapid pathogen identification from metagenomic sequences," *Briefings in Bioinformatics*, vol. 24, no. 2, p. bbad092, 2023.

[657] R. Kobus, C. Hundt, A. Müller, and B. Schmidt, "Accelerating Metagenomic Read Classification on CUDA-enabled GPUs," *BMC Bioinformatics*, 2017.

[658] X. Su, J. Xu, and K. Ning, "Parallel-META: efficient metagenomic data analysis based on high-performance computation," *BMC Systems Biology*, vol. 6, no. 1, p. S16, Jul. 2012.

[659] X. Su, X. Wang, G. Jing, and K. Ning, "GPU-Meta-Storms: computing the structure similarities among massive amount of microbial community samples using GPU," *Bioinformatics*, vol. 30, no. 7, pp. 1031–1033, Dec. 2013.

[660] M. Yano, H. Mori, Y. Akiyama, T. Yamada, and K. Kurokawa, "CLAST: CUDA implemented large-scale alignment search tool," *BMC Bioinformatics*, vol. 15, no. 1, p. 406, Dec. 2014.

[661] G. Singh, M. Alser, D. Senol Cali, D. Diamantopoulos, J. Gómez-Luna, H. Corporaal, and O. Mutlu, "FPGA-Based Near-Memory Acceleration of Modern Data-Intensive Applications," *IEEE Micro*, vol. 41, no. 4, pp. 39–48, Aug. 2021.

[662] Y.-L. Chen, B.-Y. Chang, C.-H. Yang, and T.-D. Chiueh, "A high-throughput FPGA accelerator for short-read mapping of the whole human genome," *TPDS*, 2021.

[663] Q. Lou and L. Jiang, "Brawl: A spintronics-based portable basecalling-in-memory architecture for nanopore genome sequencing," *CAL*, 2018.

[664] F. Zokaee, H. R. Zarandi, and L. Jiang, "AligneR: A process-in-Memory architecture for short read alignment in ReRAMs," *CAL*, 2018.

[665] R. Kaplan, L. Yavits, R. Ginosar, and U. Weiser, "A resistive CAM processing-in-storage architecture for DNA sequence alignment," *IEEE Micro*, 2017.

[666] R. Kaplan, L. Yavits, and R. Ginosar, "RASSA: Resistive pre-alignment accelerator for approximate DNA long read mapping," *IEEE Micro*, 2018.

[667] F. Zokaee, M. Zhang, and L. Jiang, "FindeR: Accelerating FM-index-based exact pattern matching in genomic sequences through ReRAM technology," in *PACT*, 2019.

[668] Z. I. Chowdhury, M. Zabihi, S. K. Khatamifard, Z. Zhao, S. Resch, M. Razaviyayn, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu, "A DNA read alignment accelerator based on computational RAM," *JXCDC*, 2020.

[669] K. Hammad, Z. Wu, E. Ghafar-Zadeh, and S. Magierowski, "A scalable hardware accelerator for mobile DNA sequencing," *TVLSI*, 2021.

[670] Z. Wu, K. Hammad, A. Beyene, Y. Dawji, E. Ghafar-Zadeh, and S. Magierowski, "An FPGA implementation of a portable DNA sequencing device based on RISC-V," in *Newcas*, 2022.

[671] Z. Wu, K. Hammad, E. Ghafar-Zadeh, and S. Magierowski, "FPGA-accelerated 3rd generation DNA sequencing," *TBCS*, 2020.

[672] Y. Chen, B. Schmidt, and D. L. Maskell, "A hybrid short read mapping accelerator," *BMC Bioinformatics*, 2013.

[673] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly," in *ASPLOS*, 2018.

[674] A. Goyal, H. J. Kwon, K. Lee, R. Garg, S. Y. Yun, Y. H. Kim, S. Lee, and M. S. Lee, "Ultra-fast next generation human genome sequencing data processing using DRAGEN Bio-IT processor for precision medicine," *OJGen*, 2017.

[675] S. S. Banerjee, M. El-Hadedy, J. B. Lim, Z. T. Kalbarczyk, D. Chen, S. S. Lumetta, and R. K. Iyer, "ASAP: Accelerated short-read alignment on programmable hardware," *TC*, 2019.

[676] X. Fei, Z. Dan, L. Lina, M. Xin, and Z. Chunlei, "FPGASW: Accelerating large-scale Smith–Waterman sequence alignment application with backtracking on FPGA linear systolic array," *Interdisciplinary Sciences: Computational Life Sciences*, 2018.

[677] H. M. Waidyasooriya and M. Hariyama, "Hardware-acceleration of short-read alignment based on the Burrows-wheeler transform," *TPDS*, 2015.

[678] E. Rucci, C. Garcia, G. Botella, A. De Giusti, M. Naiouf, and M. Prieto-Matias, "SWIFOLD: Smith-Waterman implementation on FPGA with OpenCL for long DNA sequences," *BMC Systems Biology*, 2018.

[679] L. Wu, D. Bruns-Smith, F. A. Nothaft, Q. Huang, S. Karandikar, J. Le, A. Lin, H. Mao, B. Sweeney, K. Asanović, and others, "FPGA accelerated indel realignment in the cloud," in *HPCA*, 2019.

[680] J.-U. Ulrich, A. Lutfi, K. Rutzen, and B. Y. Renard, "ReadBouncer: precise and scalable adaptive sampling for nanopore sequencing," *Bioinformatics*, vol. 38, no. Supplement_1, pp. i153–i160, Jul. 2022.

[681] Oxford Nanopore Technologies, "Bonito," 2021. [Online]. Available: https://github.com/nanoporetech/bonito

[682] Q. Lou, S. C. Janga, and L. Jiang, "Helix: Algorithm/Architecture Co-design for Accelerating Nanopore Genome Base-calling," in *PACT*, 2020.

[683] Z. Wu, K. Hammad, R. Mittmann, S. Magierowski, E. Ghafar-Zadeh, and X. Zhong, "FPGA-based DNA basecalling hardware acceleration," in *MWSCAS*. IEEE, 2018.

[684] H. Mao, M. Alser, M. Sadrosadati, C. Firtina, A. Baranwal, D. S. Cali, A. Manglik, N. A. Alserr, and O. Mutlu, "GenPIP: In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping," in *MICRO*, 2022.

[685] S. Gupta, M. Imani, B. Khaleghi, V. Kumar, and T. Rosing, "RAPID: A reRAM processing in-memory architecture for DNA sequence alignment," in *ISLPED*, 2019.

[686] F. Chen, L. Song, Y. Chen, and others, "PARC: A processing-in-CAM architecture for genomic long read pairwise alignment using ReRAM," in *ASP-DAC*, 2020.

[687] S. K. Khatamifard, Z. Chowdhury, N. Pande, M. Razaviyayn, C. H. Kim, and U. R. Karpuzcu, "GeNVoM: Read mapping near non-volatile memory," *IEEE/ACM TCBB*, 2021.

[688] S. Angizi, W. Zhang, and D. Fan, "Exploring DNA alignment-in-memory leveraging emerging SOT-MRAM," in *GLSVLSI*, 2020.

[689] N. Mansouri Ghiasi, J. Park, H. Mustafa, J. Kim, A. Olgun, A. Gollwitzer, D. Senol Cali, C. Firtina, H. Mao, N. Almadhoun Alserr, R. Ausavarungnirun, N. Vijaykumar, M. Alser, and O. Mutlu, "GenStore: A high-performance in-storage processing system for genome sequence analysis," in *ASPLOS*, 2022.

[690] M. Khalifa, R. Ben-Hur, R. Ronen, O. Leitersdorf, L. Yavits, and S. Kvatinsky, "FiltPIM: In-memory filter for DNA sequencing," in *ICECS*, 2021.

[691] W. Huangfu, S. Li, X. Hu, and Y. Xie, "RADAR: A 3D-ReRAM based DNA alignment accelerator architecture," in *DAC*, 2018.

[692] C. N. Ramachandra, A. Nag, R. Balasubramonion, G. Kalsi, K. Pillai, and S. Subramoney, "ONT-X: An FPGA approach to real-time portable genomic analysis," in *FCCM*, 2021.

[693] A. Nag, C. N. Ramachandra, R. Balasubramonian, R. Stutsman, E. Giacomin, H. Kambalasubramanyam, and P.-E. Gaillardon, "GenCache: Leveraging in-Cache operators for efficient sequence alignment," in *MICRO*, 2019.

[694] A. Haghi, S. Marco-Sola, L. Alvarez, D. Diamantopoulos, C. Hagleitner, and M. Moreto, "An FPGA accelerator of the wavefront algorithm for genomics pairwise alignment," in *FPL*, 2021.

[695] D. Fujiki, A. Subramaniyan, T. Zhang, Y. Zeng, R. Das, D. Blaauw, and S. Narayanasamy, "GenAx: A genome sequencing accelerator," in *ISCA*, 2018.

[696] Y.-T. Chen, J. Cong, Z. Fang, J. Lei, and P. Wei, "When Spark Meets FPGAs: A case study for next-generation DNA sequencing acceleration," in *HotCloud*, 2016.

[697] P. Chen, C. Wang, X. Li, and X. Zhou, "Accelerating the next generation long read mapping with the FPGA-based system," *IEEE/ACM TCBB*, 2014.

[698] D. Fujiki, S. Wu, N. Ozog, K. Goliya, D. Blaauw, S. Narayanasamy, and R. Das, "SeedEx: A genome sequencing accelerator for optimal alignments in subminimal space," in *MICRO*, 2020.

[699] Y.-T. Chen, J. Cong, J. Lei, and P. Wei, "A novel high-throughput acceleration engine for read alignment," in *FCCM*, 2015.

[700] L. Li, J. Lin, and Z. Wang, "PipeBSW: A two-stage pipeline structure for banded Smith-Waterman algorithm on FPGA," in *ISVLSI*, 2021.

[701] S. Diab, A. Nassereldine, M. Alser, J. Gómez Luna, O. Mutlu, and I. El Hajj, "A framework for high-throughput sequence alignment using real processing-in-memory systems," *Bioinform.*, 2024.

[702] S. Diab, A. Nassereldine, M. Alser, J. G. Luna, O. Mutlu, and I. E. Hajj, "High-throughput Pairwise Alignment with the Wavefront Algorithm using Processing-in-Memory," in *IPDPSW*, 2022.

[703] X. Wu, A. Subramaniyan, Z. Wang, S. Narayanasamy, R. Das, and D. Blaauw, "A High-Throughput Pruning-Based Pair-Hidden-Markov-Model Hardware Accelerator for Next-Generation DNA Sequencing," *IEEE Solid-State Circuits Letters*, 2021.

[704] Y.-C. Wu, Y.-L. Chen, C.-H. Yang, C.-H. Lee, C.-Y. Yu, N.-S. Chang, L.-C. Chen, J.-R. Chang, C.-P. Lin, H.-L. Chen, C.-S. Chen, J.-H. Hung, and C.-H. Yang, "A 975-mW Fully Integrated Genetic Variant Discovery System-on-Chip in 28 nm for Next-Generation Sequencing," *IEEE J. Solid-State Circuits*, 2021.

[705] R. Wertenbroek and Y. Thoma, "Acceleration of the Pair-HMM forward algorithm on FPGA with cloud integration for GATK," in *BIBM*, 2019.

[706] S. Huang, G. J. Manikandan, A. Ramachandran, K. Rupnow, W.-m. W. Hwu, and D. Chen, "Hardware Acceleration of the Pair-HMM Algorithm for DNA Variant Calling," in *FPGA*, 2017.

[707] S. S. Banerjee, M. el Hadedy, C. Y. Tan, Z. T. Kalbarczyk, S. Lumetta, and R. K. Iyer, "On accelerating pair-HMM computations in programmable hardware," in *FPL*, 2017.

[708] S. Angizi, J. Sun, W. Zhang, and D. Fan, "PIM-Aligner: A processing-in-MRAM platform for biological sequence alignment," in *DATE*, 2020.

[709] A. Madhavan, T. Sherwood, and D. Strukov, "Race Logic: A Hardware Acceleration for Dynamic Programming Algorithms," in *ISCA*, 2014.

[710] N. M. Ghiasi, M. Sadrosadati, H. Mustafa, A. Gollwitzer, C. Firtina, J. Eudine, H. Mao, J. Lindegger, M. B. Cavlak, M. Alser, J. Park, and O. Mutlu, "Megis: High-performance, energy-efficient, and low-cost metagenomic analysis with in-storage processing," in *ISCA*, 2024.

[711] T. Shahroodi, G. Singh, M. Zahedi, H. Mao, J. Lindegger, C. Firtina, S. Wong, O. Mutlu, and S. Hamdioui, "Swordfish: A framework for evaluating deep neural network-based basecalling using computation-in-memory with non-ideal memristors," in *MICRO*, 2023.

[712] W. Huangfu, K. T. Malladi, A. Chang, and Y. Xie, "BEACON: Scalable Near-Data-Processing Accelerators for Genome Analysis near Memory Pool with the CXL Support," in *MICRO*, 2022, pp. 727–743.

[713] W. Huangfu, K. T. Malladi, S. Li, P. Gu, and Y. Xie, "NEST: DIMM based near-data-processing accelerator for K-mer counting," in *ICCAD*, New York, NY, USA, 2020.

[714] M. Khalifa, B. Hoffer, O. Leitersdorf, R. Hanhan, B. Perach, L. Yavits, and S. Kvatinsky, "ClaPIM: Scalable Sequence Classification Using Processing-in-Memory," *VLSI*, vol. 31, no. 9, pp. 1347–1357, 2023.

[715] T. Shahroodi, M. Miao, M. Zahedi, S. Wong, and S. Hamdioui, "RattlesnakeJake: A Fast and Accurate Pre-alignment Filter Suitable for Computation-in-Memory," in *SAMOS*, C. Silvano, C. Pilato, and M. Reichenbach, Eds., 2023, pp. 209–221.

[716] T. Shahroodi, M. Miao, M. Zahedi, S. Wong, and S. Hamdioui, "SieveMem: A Computation-in-Memory Architecture for Fast and Accurate Pre-Alignment," in *ASAP*, 2023, pp. 156–164.

[717] N. Akbari, M. Modarressi, M. Daneshtalab, and M. Loni, "A Customized Processing-in-Memory Architecture for Biological Sequence Alignment," in *ASAP*, 2018, pp. 1–8.

[718] V. Elisseev, L.-J. Gardiner, and R. Krishna, "Scalable in-memory processing of omics workflows," *CSBJ*, vol. 20, pp. 1914–1924, Jan. 2022.

[719] R. Kaplan, L. Yavits, and R. Ginosasr, "BioSEAL: In-memory biological sequence alignment accelerator for large-scale genomic data," in *SYSTOR*, 2020.

[720] T. Shahroodi, M. Zahedi, A. Singh, S. Wong, and S. Hamdioui, "KrakenOnMem: a memristor-augmented HW/SW framework for taxonomic profiling," in *ICS*, New York, NY, USA, 2022.

[721] A. Sneddon, P. Acera Mateos, N. Shirokikh, and E. Eyras, "Language-Informed Basecalling Architecture for Nanopore Direct RNA Sequencing," in *MLCB*, D. A. Knowles, S. Mostafavi, and S.-I. Lee, Eds., vol. 200, Nov. 2022, pp. 150–165.

[722] P. Grzesik and D. Mrozek, "Serverless Nanopore Basecalling with AWS Lambda," in *ICCS*, M. Paszynski, D. Kranzlmüller, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. A. Sloot, Eds., 2021, pp. 578–586.

[723] A. Saavedra, H. Lehnert, C. Hernández, G. Carvajal, and M. Figueroa, "Mining discriminative k-mers in DNA sequences using sketches and hardware acceleration," *IEEE Access*, vol. 8, pp. 114 715–114 732, 2020.

[724] T. Zhang, A. González, N. Moshiri, R. Knight, and T. Rosing, "GenoMiX: Accelerated Simultaneous Analysis of Human Genomics, Microbiome Metagenomics, and Viral Sequences," in *BioCAS*, 2023.

[725] G. H. Cervi, C. D. Flores, and C. E. Thompson, "Metagenomic Analysis: A Pathway Toward Efficiency Using High-Performance Computing," in *ICICT*, 2022.

[726] L. Wu, R. Sharifi, M. Lenjani, K. Skadron, and A. Venkat, "Sieve: Scalable in-situ DRAM-based accelerator designs for massively parallel k-mer matching," in *ISCA*, 2021, pp. 251–264.

[727] Z. Jahshan, I. Merlin, E. Garzón, and L. Yavits, "DASH-CAM: Dynamic Approximate SearcH Content Addressable Memory for genome classification," in *MICRO*, 2023.

[728] R. Hanhan, E. Garzón, Z. Jahshan, A. Teman, M. Lanuzza, and L. Yavits, "EDAM: edit distance tolerant approximate matching content addressable memory," in *ISCA*, 2022, pp. 495–507.

[729] Z. Zou, H. Chen, P. Poduval, Y. Kim, M. Imani, E. Sadredini, R. Cammarota, and M. Imani, "BioHD: an efficient genome sequence search platform using hyperdimensional memorization," in *ISCA*, 2022, pp. 656–669.

[730] J. E. Soto, C. Hernández, and M. Figueroa, "JACC-FPGA: A hardware accelerator for Jaccard similarity estimation using FPGAs in the cloud," *Future Generation Computer Systems*, vol. 138, pp. 26–42, Jan. 2023.

[731] Y. Harary, P. Snapir, S. S. Tov, C. Kruphman, E. Rechef, Z. Jahshan, E. Garzón, and L. Yavits, "GCOC: A Genome Classifier-On-Chip based on Similarity Search Content Addressable Memory," *TBCAS*, pp. 1–12, 2024.

[732] T. B. Preuber, O. Knodel, and R. G. Spallek, "Short-Read Mapping by a Systolic Custom FPGA Computation," in *FCCM*, 2012, pp. 169–176.

[733] M. Doblas, O. Lostes-Cazorla, Q. Aguado-Puig, N. Cebry, P. Fontova-Musté, C. F. Batten, S. Marco-Sola, and M. Moretó, "GMX: Instruction Set Extensions for Fast, Scalable, and Efficient Genome Sequence Alignment," in *MICRO*, 2023.

[734] K. Liyanage, H. Samarakoon, S. Parameswaran, and H. Gamaarachchi, "Efficient end-to-end long-read sequence mapping using minimap2-fpga integrated with hardware accelerated chaining," *Scientific Reports*, vol. 13, no. 1, p. 20174, Nov. 2023.

[735] C. Lanius and T. Gemmeke, "Fully Digital, Standard-Cell-Based Multifunction Compute-in-Memory Arrays for Genome Sequencing," *TVLSI*, vol. 32, no. 1, pp. 30–41, 2024.

[736] Y.-L. Liao, Y.-C. Li, N.-C. Chen, and Y.-C. Lu, "Adaptively Banded Smith-Waterman Algorithm for Long Reads and Its Hardware Accelerator," in *ASAP*, 2018, pp. 1–9.

[737] W. Xu, S. Gupta, N. Moshiri, and T. S. Rosing, "RAPIDx: High-Performance ReRAM Processing In-Memory Accelerator for Sequence Alignment," *TCAD*, vol. 42, no. 10, pp. 3275–3288, 2023.

[738] Y. Zhang, D. Chen, G. Zeng, J. Zhu, Z. Li, L. Chen, S. Wei, and L. Liu, "Harp: Leveraging Quasi-Sequential Characteristics to Accelerate Sequence-to-Graph Mapping of Long Reads," in *ASPLOS*, New York, NY, USA, 2024.

[739] G. Zeng, J. Zhu, Y. Zhang, G. Chen, Z. Yuan, S. Wei, and L. Liu, "A High-Performance Genomic Accelerator for Accurate Sequence-to-Graph Alignment Using Dynamic Programming Algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 2, pp. 237–249, 2024.

[740] C. Teng, R. W. Achjian, J. C. Wang, and F. J. Fonseca, "Adapting the GACT-X Aligner to Accelerate Minimap2 in an FPGA Cloud Instance," *Applied Sciences*, vol. 13, no. 7, 2023.

[741] Y. Gu, A. Subramaniyan, T. Dunn, A. Khadem, K.-Y. Chen, S. Paul, M. Vasimuddin, S. Misra, D. Blaauw, S. Narayanasamy, and R. Das, "GenDP: A Framework of Dynamic Programming Acceleration for Genome Sequencing Analysis," in *ISCA*, 2023.

[742] A. Haghi, S. Marco-Sola, L. Alvarez, D. Diamantopoulos, C. Hagleitner, and M. Moreto, "WFA-FPGA: An efficient accelerator of the wavefront algorithm for short and long read genomics alignment," *Future Generation Computer Systems*, vol. 149, pp. 39–58, Dec. 2023.

[743] A. Haghi, L. Alvarez, J. Front, J. M. De Haro Ruiz, R. Figueras, M. Doblas, S. Marco-Sola, and M. Moreto, "WFAsic: A High-Performance ASIC Accelerator for DNA Sequence Alignment on a RISC-V SoC," in *ICPP*, 2023.

[744] S. Walia, C. Ye, A. Bera, D. Lodhavia, and Y. Turakhia, "TALCO: Tiling Genome Sequence Alignment Using Convergence of Traceback Pointers," in *HPCA*, 2024, pp. 91–107.

[745] C. Lanius and T. Gemmeke, "Multi-Function CIM Array for Genome Alignment Applications built with Fully Digital Flow," in *NorCAS*, 2022.

[746] S. Han, S. Moon, T. Suh, J. Heo, and J.-Y. Kim, "BLESS: Bandwidth and Locality Enhanced SMEM Seeding Acceleration for DNA Sequencing," in *ISCA*, 2024.

[747] L. Wu, M. Zhou, W. Xu, A. Venkat, T. Rosing, and K. Skadron, "Abakus: Accelerating k-mer Counting with Storage Technology," *ACM Trans. Archit. Code Optim.*, vol. 21, no. 1, Jan. 2024.

[748] J. Pavon, I. V. Valdivieso, C. Rojas, C. Hernandez, M. Aslan, R. Figueras, Y. Yuan, J. Lindegger, M. Alser, F. Moll, S. Marco-Sola, O. Ergin, N. Talati, O. Mutlu, O. Unsal, M. Valero, and A. Cristal, "QUETZAL: Vector Acceleration Framework for Modern Genome Sequence Analysis Algorithms," in *ISCA*, 2024.

[749] T. Wu, C.-F. Nien, K.-C. Chou, and H.-Y. Cheng, "RePAIR: A ReRAM-based Processing-in-Memory Accelerator for Indel Realignment," in *DATE*, 2022.

[750] T. J. Ham, D. Bruns-Smith, B. Sweeney, Y. Lee, S. H. Seo, U. G. Song, Y. H. Oh, K. Asanovic, J. W. Lee, and L. W. Wills, "Genesis: A Hardware Acceleration Framework for Genomic Data Analysis," in *ISCA*, 2020.

[751] M. Lo, Z. Fang, J. Wang, P. Zhou, M.-C. F. Chang, and J. Cong, "Algorithm-Hardware Co-design for BQSR Acceleration in Genome Analysis ToolKit," in *FCCM*, 2020.

[752] T. Xu, S. Rixner, and A. L. Cox, "An FPGA Accelerator for Genome Variant Calling," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 4, Sep. 2023.

[753] M.-H. Chen, M.-J. Lin, Y.-C. Li, and Y.-C. Lu, "Banded Pair-HMM Algorithm for DNA Variant Calling and Its Hardware Accelerator Design," in *BIBE*, 2019.

[754] M. Roodi and A. Moshovos, "MemAlign: A Memory Structure to Accelerate Gene Sequencing," in *BIBE*, 2019.

[755] K. Liyanage, H. Gamaarachchi, H. Saadat, T. Li, H. Samarakoon, and S. Parameswaran, "Accelerating Chaining in Genomic Analysis Using RISC- V Custom Instructions," in *DATE*, 2024.

[756] M. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, vol. 54, no. 12, pp. 1901–1909, 1966.

[757] Z. Jahshan and L. Yavits, "MajorK: Majority Based kmer Matching in Commodity DRAM," *IEEE Computer Architecture Letters*, vol. 23, no. 1, pp. 83–86, 2024.

[758] N. Abecassis, J. Gómez-Luna, O. Mutlu, R. Ginosar, A. Moisson-Franckhauser, and L. Yavits, "GAPiM: Discovering Genetic Variations on a Real Processing-in-Memory System," *bioRxiv*, p. 2023.07.26.550623, 2023.

[759] I. Merlin, E. Garzón, A. Fish, and L. Yavits, "DIPER: Detection and Identification of Pathogens Using Edit Distance-Tolerant Resistive CAM," *IEEE Transactions on Computers*, vol. 73, no. 10, pp. 2463–2473, 2024.

[760] E. Garzón, R. Golman, Z. Jahshan, R. Hanhan, N. Vinshtok-Melnik, M. Lanuzza, A. Teman, and L. Yavits, "Hamming Distance Tolerant Content-Addressable Memory (HD-CAM) for DNA Classification," *IEEE Access*, vol. 10, pp. 28 080–28 093, 2022.

[761] M. Besta, R. Kanakagiri, H. Mustafa, M. Karasikov, G. Rätsch, T. Hoefler, and E. Solomonik, "Communication-Efficient Jaccard similarity for High-Performance Distributed Genome Comparisons," in *IPDPS*, 2020, pp. 1122–1132.

[762] S. Angizi, N. A. Fahmi, D. Najafi, W. Zhang, and D. Fan, "PANDA: Processing in Magnetic Random-Access Memory-Accelerated de Bruijn Graph-Based DNA Assembly," *Journal of Low Power Electronics and Applications*, vol. 14, no. 1, 2024.

[763] M. Zhou, L. Wu, M. Li, N. Moshiri, K. Skadron, and T. Rosing, "Ultra Efficient Acceleration for De Novo Genome Assembly via Near-Memory Computing," in *PACT*, 2021, pp. 199–212.

[764] B. S. C. Varma, K. Paul, and M. Balakrishnan, "Accelerating Genome Assembly Using Hard Embedded Blocks in FPGAs," in *VLSID*, 2014, pp. 306–311.

[765] B. S. C. Varma, K. Paul, and M. Balakrishnan, "High Level Design Approach to Accelerate De Novo Genome Assembly Using FPGAs," in *DSD*, 2014, pp. 66–73.

[766] B. S. C. Varma, K. Paul, M. Balakrishnan, and D. Lavenier, "Hardware acceleration of de novo genome assembly," *International Journal of Embedded Systems*, vol. 9, no. 1, pp. 74–89, Jan. 2017. [Online]. Available: https://www.inderscienceonline.com/doi/abs/10.1504/IJES.2017.081729

[767] B. S. C. Varma, K. Paul, and M. Balakrishnan, "FPGA-Based Acceleration of De Novo Genome Assembly," in *Architecture Exploration of FPGA Based Accelerators for BioInformatics Applications*, B. S. C. Varma, K. Paul, and M. Balakrishnan, Eds. Singapore: Springer Singapore, 2016, pp. 55–79.

[768] C. Poirier, B. Gosselin, and P. Fortier, "DNA assembly with de bruijn graphs on FPGA," in *EMBC*, 2015, pp. 6489–6492.

[769] C. Poirier, B. Gosselin, and P. Fortier, "DNA Assembly with De Bruijn Graphs Using an FPGA Platform," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 15, no. 3, pp. 1003–1009, 2018.

[770] Y.-H. Kuo, C.-S. Liu, Y.-C. Li, and Y.-C. Lu, "Parallel architecture and hardware implementation of pre-processor and post-processor for sequence assembly," in *ICASSP*, 2013, pp. 1158–1161.

[771] S. Angizi, N. A. Fahmi, W. Zhang, and D. Fan, "PIM-Assembler: A Processing-in-Memory Platform for Genome Assembly," in *DAC*, 2020, pp. 1–6.

[772] Y. Huang, L. Kong, D. Chen, Z. Chen, X. Kong, J. Zhu, K. Mamouras, S. Wei, K. Yang, and L. Liu, "CASA: An Energy-Efficient and High-Speed CAM-based SMEM Seeding Accelerator for Genome Alignment," in *MICRO*, 2023, pp. 1423–1436.

[773] F. Zhang, S. Angizi, N. A. Fahmi, W. Zhang, and D. Fan, "PIM-Quantifier: A Processing-in-Memory Platform for mRNA Quantification," in *DAC*, 2021, pp. 43–48.

[774] W. Huangfu, X. Li, S. Li, X. Hu, P. Gu, and Y. Xie, "MEDAL: Scalable DIMM based Near Data Processing Accelerator for DNA Seeding Algorithm," in *MICRO*, 2019.

[775] F. Zhang, S. Angizi, J. Sun, W. Zhang, and D. Fan, "Aligner-D: Leveraging In-DRAM Computing to Accelerate DNA Short Read Alignment," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 13, no. 1, pp. 332–343, 2023.

[776] F. Hameed, A. A. Khan, and J. Castrillon, "ALPHA: A Novel Algorithm-Hardware Co-Design for Accelerating DNA Seed Location Filtering," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 3, pp. 1464–1475, 2022.

[777] B. S. C. Varma, K. Paul, M. Balakrishnan, and D. Lavenier, "FAssem: FPGA Based Acceleration of De Novo Genome Assembly," in *FCCM*, 2013, pp. 173–176.

[778] Y. Hu and P. Georgiou, "A Real-Time de novo DNA Sequencing Assembly Platform Based on an FPGA Implementation," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 13, no. 2, pp. 291–300, 2016.

[779] G. Galanos, P. Malakonakis, and A. Dollas, "An FPGA-Based Data Pre-Processing Architecture to Accelerate De-Novo Genome Assembly," in *BIBE*, 2021, pp. 1–6.

[780] K. Sahlin, T. Baudeau, B. Cazaux, and C. Marchet, "A survey of mapping algorithms in the long-reads era," *Genome Biology*, vol. 24, no. 1, p. 133, Jun. 2023.

[781] H. Gamaarachchi, J. M. Ferguson, H. Samarakoon, K. Liyanage, and I. W. Deveson, "Simulation of nanopore sequencing signal data with tunable parameters," *Genome Research*, 2024.

[782] C. Wen, D. Dematties, and S.-L. Zhang, "A Guide to Signal Processing Algorithms for Nanopore Sensors," *ACS Sensors*, vol. 6, no. 10, pp. 3536–3555, Oct. 2021.

[783] R. Munro, S. Wibowo, A. Payne, and M. Loose, "Icarust, a real-time simulator for Oxford Nanopore adaptive sampling," *Bioinformatics*, vol. 40, no. 4, p. btae141, Apr. 2024.

[784] M. Mordig, G. Rätsch, and A. Kahles, "SimReadUntil for benchmarking selective sequencing algorithms on ONT devices," *Bioinformatics*, vol. 40, no. 5, p. btae199, May 2024.

[785] F. J. Rang, W. P. Kloosterman, and J. de Ridder, "From Squiggle to Basepair: Computational Approaches for Improving Nanopore Sequencing Read Accuracy," *Genome Biology*, vol. 19, no. 1, p. 90, Jul. 2018.

[786] A. Payne, N. Holmes, T. Clarke, R. Munro, B. J. Debebe, and M. Loose, "Readfish enables targeted nanopore sequencing of gigabase-sized genomes," *Nature Biotechnology*, vol. 39, no. 4, pp. 442–450, Apr. 2021.

[787] H. S. Edwards, R. Krishnakumar, A. Sinha, S. W. Bird, K. D. Patel, and M. S. Bartsch, "Real-Time Selective Sequencing with RUBRIC: Read Until with Basecall and Reference-Informed Criteria," *Scientific Reports*, vol. 9, no. 1, p. 11475, Aug. 2019.

[788] A. J. Mikalsen and J. Zola, "Coriolis: enabling metagenomic classification on lightweight mobile devices," *Bioinform.*, 2023.

[789] H. Samarakoon, S. Punchihewa, A. Senanayake, J. M. Hammond, I. Stevanovski, J. M. Ferguson, R. Ragel, H. Gamaarachchi, and I. W. Deveson, "Genopo: a nanopore sequencing analysis toolkit for portable Android devices," *Communications Biology*, vol. 3, no. 1, p. 538, Sep. 2020.

[790] L. Weilguny, N. De Maio, R. Munro, C. Manser, E. Birney, M. Loose, and N. Goldman, "Dynamic, adaptive sampling during nanopore sequencing using Bayesian experimental design," *Nature Biotechnology*, Jan. 2023.

[791] "BLEND," 2023. [Online]. Available: https://github.com/CMU-SAFARI/BLEND

[792] C. Firtina, J. Park, M. Alser, J. S. Kim, D. S. Cali, T. Shahroodi, N. M. Ghiasi, G. Singh, K. Kanellopoulos, C. Alkan, and O. Mutlu, "BLEND: a fast, memory-efficient and accurate mechanism to find fuzzy seed matches in genome analysis," in *RECOMB*, Istanbul, Turkey, Apr. 2023, Conference Program: http://recomb2023.bilkent.edu.tr/program.html. Slides (PPT): https://people.ee.ethz.ch/~firtinac/pub/blend-firtina-2023_04_19-recomb.pptx. Slides (PDF): https://people.ee.ethz.ch/~firtinac/pub/blend-firtina-2023_04_19-recomb.pdf. Talk Video: https://youtu.be/k9NzGwaF_mE.

[793] "RawHash," 2024. [Online]. Available: https://github.com/CMU-SAFARI/RawHash

[794] C. Firtina, N. Mansouri Ghiasi, J. Lindegger, G. Singh, M. B. Cavlak, H. Mao, and O. Mutlu, "RawHash: enabling fast and accurate real-time analysis of raw nanopore signals for large genomes," in *ISMB/ECCB*, Lyon, France, Jul. 2023, Conference Program: https://www.iscb.org/ismbeccb2023-programme/tracks/hitseq. Slides (PPT): https://people.ee.ethz.ch/%7Efirtinac/pub/rawhash-firtina-2023-ismbeccb.pptx. Slides (PDF): https://people.ee.ethz.ch/%7Efirtinac/pub/rawhash-firtina-2023-ismbeccb.pdf. Talk Video: https://youtu.be/ti0M6TvRkTs.

[795] C. Firtina, M. Mordig, H. Mustafa, J. Lindegger, S. Goswami, S. Mercogliano, Y. Zhu, A. Kahles, and O. Mutlu, "Rawsamble: Overlapping and Assembling Raw Nanopore Signals using a Hash-based Seeding Mechanism," in *ISMB (HiTSeq)*, Montreal, QC, Canada, Jul. 2024, Conference Program: https://www.iscb.org/ismb2024/programme-schedule/scientific-programme/hitseq. Slides (PPT): https://people.ee.ethz.ch/%7Efirtinac/pub/rawsamble-firtina-2024-ismb.pptx. Slides (PDF): https://people.ee.ethz.ch/%7Efirtinac/pub/rawsamble-firtina-2024-ismb.pdf.

[796] W.-C. Kao and Y. S. Song, "naiveBayesCall: An Efficient Model-Based Base-Calling Algorithm for High-Throughput Sequencing," in *Research in Computational Molecular Biology*, B. Berger, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 233–247.

[797] A. Cacho, W. Yao, and X. Cui, "Base-Calling Using a Random Effects Mixture Model on Next-Generation Sequencing Data," *Statistics in Biosciences*, vol. 10, no. 1, pp. 3–19, Apr. 2018.

[798] Y. Erlich, P. P. Mitra, M. delaBastide, W. R. McCombie, and G. J. Hannon, "Alta-Cyclic: a self-optimizing base caller for next-generation sequencing," *Nature Methods*, vol. 5, no. 8, pp. 679–682, Aug. 2008.

[799] B. Wang, L. Wan, A. Wang, and L. M. Li, "An adaptive decorrelation method removes Illumina DNA base-calling errors caused by crosstalk between adjacent clusters," *Scientific Reports*, vol. 7, no. 1, p. 41348, Feb. 2017.

[800] J. Rougemont, A. Amzallag, C. Iseli, L. Farinelli, I. Xenarios, and F. Naef, "Probabilistic base calling of Solexa sequencing data," *BMC Bioinformatics*, vol. 9, no. 1, p. 431, Oct. 2008.

[801] X. Shen and H. Vikalo, "ParticleCall: A particle filter for base calling in next-generation sequencing systems," *BMC Bioinformatics*, vol. 13, no. 1, p. 160, Jul. 2012.

[802] Y. Ji, R. Mitra, F. Quintana, A. Jara, P. Mueller, P. Liu, Y. Lu, and S. Liang, "BM-BC: a Bayesian method of base calling for Solexa sequence data," *BMC Bioinformatics*, vol. 13, no. 13, p. S6, Aug. 2012.

[803] S. Das and H. Vikalo, "Base calling for high-throughput short-read sequencing: dynamic programming solutions," *BMC Bioinformatics*, vol. 14, no. 1, p. 129, Apr. 2013.

[804] M. Kircher, U. Stenzel, and J. Kelso, "Improved base calling for the Illumina Genome Analyzer using machine learning strategies," *Genome Biology*, vol. 10, no. 8, p. R83, Aug. 2009.

[805] T. Massingham and N. Goldman, "All Your Base: a fast and accurate probabilistic approach to base calling," *Genome Biology*, vol. 13, no. 2, p. R13, Feb. 2012.

[806] C. Ye, C. Hsiao, and H. Corrada Bravo, "BlindCall: ultra-fast base-calling of high-throughput sequencing data by blind deconvolution," *Bioinformatics*, vol. 30, no. 9, pp. 1214–1219, May 2014.

[807] G. Renaud, M. Kircher, U. Stenzel, and J. Kelso, "freeIbis: an efficient basecaller with calibrated quality scores for Illumina sequencers," *Bioinformatics*, vol. 29, no. 9, pp. 1208–1209, May 2013.

[808] S. Das and H. Vikalo, "OnlineCall: fast online parameter estimation and base calling for illumina's next-generation sequencing," *Bioinformatics*, vol. 28, no. 13, pp. 1677–1683, Jul. 2012.

[809] F. Menges, G. Narzisi, and B. Mishra, "TotalReCaller: improved accuracy and performance via integrated alignment and base-calling," *Bioinformatics*, vol. 27, no. 17, pp. 2330–2337, Sep. 2011.

[810] H. C. Bravo and R. A. Irizarry, "Model-Based Quality Assessment and Base-Calling for Second-Generation Sequencing Data," *Biometrics*, vol. 66, no. 3, pp. 665–674, Sep. 2010.

[811] W.-C. Kao, K. Stevens, and Y. S. Song, "BayesCall: A model-based base-calling algorithm for high-throughput short-read sequencing," *Genome Research*, vol. 19, no. 10, pp. 1884–1895, Oct. 2009.

[812] A. Cacho, E. Smirnova, S. Huzurbazar, and X. Cui, "A Comparison of Base-calling Algorithms for Illumina Sequencing Technology," *Briefings Bioinform.*, 2016.

[813] C. Firtina and C. Alkan, "On genomic repeats and reproducibility," *Bioinformatics*, vol. 32, no. 15, pp. 2243–2247, Aug. 2016.

[814] G. A. Logsdon, M. R. Vollger, and E. E. Eichler, "Long-read human genome sequencing and its applications," *Nature Reviews Genetics*, vol. 21, no. 10, pp. 597–614, Oct. 2020.

[815] M. J. Levene, J. Korlach, S. W. Turner, M. Foquet, H. G. Craighead, and W. W. Webb, "Zero-Mode Waveguides for Single-Molecule Analysis at High Concentrations," *Science*, vol. 299, no. 5607, pp. 682–686, Jan. 2003.

[816] K. J. Travers, C.-S. Chin, D. R. Rank, J. S. Eid, and S. W. Turner, "A flexible and efficient template format for circular consensus sequencing and SNP detection," *Nucleic Acids Research*, vol. 38, no. 15, pp. e159–e159, Aug. 2010.

[817] D. Sharon, H. Tilgner, F. Grubert, and M. Snyder, "A single-molecule long-read survey of the human transcriptome," *Nature Biotechnology*, vol. 31, no. 11, pp. 1009–1014, Nov. 2013.

[818] A. M. Wenger, P. Peluso, W. J. Rowell, P.-C. Chang, R. J. Hall, G. T. Concepcion, J. Ebler, A. Fungtammasan, A. Kolesnikov, N. D. Olson, A. Töpfer, M. Alonge, M. Mahmoud, Y. Qian, C.-S. Chin, A. M. Phillippy, M. C. Schatz, G. Myers, M. A. DePristo, J. Ruan, T. Marschall, F. J. Sedlazeck, J. M. Zook, H. Li, S. Koren, A. Carroll, D. R. Rank, and M. W. Hunkapiller, "Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome," *Nat. Biotechnol.*, 2019.

[819] C.-S. Chin, D. H. Alexander, P. Marks, A. A. Klammer, J. Drake, C. Heiner, A. Clum, A. Copeland, J. Huddleston, E. E. Eichler, S. W. Turner, and J. Korlach, "Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data," *Nat. Methods*, 2013.

[820] J. Pugh, "The Current State of Nanopore Sequencing," in *Nanopore Sequencing: Methods and Protocols*, K. Arakawa, Ed. New York, NY: Springer US, 2023, pp. 3–14.

[821] H. Samarakoon, Y. K. Wan, S. Parameswaran, J. Göke, H. Gamaarachchi, and I. W. Deveson, "Leveraging Basecaller's Move Table to Generate a Lightweight k-mer Model," *bioRxiv*, p. 2024.06.30.601452, 2024.

[822] R. M. M. Smeets, U. F. Keyser, N. H. Dekker, and C. Dekker, "Noise in solid-state nanopores," *Proceedings of the National Academy of Sciences*, vol. 105, no. 2, pp. 417–421, Jan. 2008.

[823] S. Bhattacharya, I. M. Derrington, M. Pavlenok, M. Niederweis, J. H. Gundlach, and A. Aksimentiev, "Molecular Dynamics Study of MspA Arginine Mutants Predicts Slow DNA Translocations and Ion Current Blockades Indicative of DNA Sequence," *ACS Nano*, vol. 6, no. 8, pp. 6960–6968, Aug. 2012.

[824] R. Kawano, A. E. P. Schibel, C. Cauley, and H. S. White, "Controlling the Translocation of Single-Stranded DNA through a-Hemolysin Ion Channels Using Viscosity," *Langmuir*, vol. 25, no. 2, pp. 1233–1237, Jan. 2009.

[825] M. Pagès-Gallego and J. de Ridder, "Comprehensive benchmark and architectural analysis of deep learning models for nanopore sequencing basecalling," *Genome Biol.*, 2023.

[826] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *ICML*, 2006.

[827] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data," in *ICML*, 2001.

[828] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," in *ASPLOS*, 2018.

[829] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, "Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks," in *PACT*, Sep. 2021.

[830] Oxford Nanopore Technologies, "Scrappie," 2019. [Online]. Available: https://github.com/nanoporetech/scrappie

[831] M. Stoiber, J. Quick, R. Egan, J. Eun Lee, S. Celniker, R. K. Neely, N. Loman, L. A. Pennacchio, and J. Brown, "De novo Identification of DNA Modifications Enabled by Genome-Guided Nanopore Signal Processing," *bioRxiv*, p. 094672, 2017.

[832] Q. Liu, D. C. Georgieva, D. Egli, and K. Wang, "NanoMod: a computational tool to detect DNA modifications using Nanopore long-read sequencing data," *BMC Genomics*, vol. 20, no. 1, p. 78, Feb. 2019.

[833] Y. K. Wan, C. Hendra, P. N. Pratanwanich, and J. Göke, "Beyond sequencing: machine learning algorithms extract biology hidden in Nanopore signal data," *Trends in Genetics*, vol. 38, no. 3, pp. 246–257, Mar. 2022.

[834] B. D. Sigurpalsdottir, O. A. Stefansson, G. Holley, D. Beyter, F. Zink, M. Hardarson, S. Sverrisson, N. Kristinsdottir, D. N. Magnusdottir, O. Magnusson, D. F. Gudbjartsson, B. V. Halldorsson, and K. Stefansson, "A comparison of methods for detecting DNA methylation from long-read sequencing of human genomes," *Genome Biology*, vol. 25, no. 1, p. 69, Mar. 2024.

[835] J. T. Simpson, R. E. Workman, P. C. Zuzarte, M. David, L. J. Dursi, and W. Timp, "Detecting DNA cytosine methylation using nanopore sequencing," *Nature Methods*, vol. 14, no. 4, pp. 407–410, Apr. 2017.

[836] B. Yao, C. Hsu, G. Goldner, Y. Michaeli, Y. Ebenstein, and J. Listgarten, "Effective training of nanopore callers for epigenetic marks with limited labelled data," *Open Biology*, vol. 14, no. 6, p. 230449, Jun. 2024.

[837] X. Bai, H.-C. Yao, B. Wu, L.-R. Liu, Y.-Y. Ding, and C.-L. Xiao, "DeepBAM: a high-accuracy single-molecule CpG methylation detection tool for Oxford nanopore sequencing," *Briefings in Bioinformatics*, vol. 25, no. 5, p. bbae413, Sep. 2024.

[838] D. Stanojević, Z. Li, S. Bakić, R. Foo, and M. Šikić, "Rockfish: A transformer-based model for accurate 5-methylcytosine prediction from nanopore sequencing," *Nature Communications*, vol. 15, no. 1, p. 5580, Jul. 2024.

[839] Y.-Z. Zhang, K. Yamaguchi, S. Hatakeyama, Y. Furukawa, S. Miyano, R. Yamaguchi, and S. Imoto, "On the application of BERT models for nanopore methylation detection," in *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2021, pp. 320–327.

[840] J. Bonet, M. Chen, M. Dabad, S. Heath, A. Gonzalez-Perez, N. Lopez-Bigas, and J. Lagergren, "DeepMP: a deep learning tool to detect DNA base modifications on Nanopore sequencing data," *Bioinformatics*, vol. 38, no. 5, pp. 1235–1243, Mar. 2022.

[841] Z. W.-S. Yuen, A. Srivastava, R. Daniel, D. McNevin, C. Jack, and E. Eyras, "Systematic benchmarking of tools for CpG methylation detection from nanopore sequencing," *Nature Communications*, vol. 12, no. 1, p. 3438, Jun. 2021.

[842] A. B. R. McIntyre, N. Alexander, K. Grigorev, D. Bezdan, H. Sichtig, C. Y. Chiu, and C. E. Mason, "Single-molecule sequencing detection of N6-methyladenine in microbial reference materials," *Nature Communications*, vol. 10, no. 1, p. 579, Feb. 2019.

[843] A. C. Rand, M. Jain, J. M. Eizenga, A. Musselman-Brown, H. E. Olsen, M. Akeson, and B. Paten, "Mapping DNA methylation with high-throughput nanopore sequencing," *Nature Methods*, vol. 14, no. 4, pp. 411–413, Apr. 2017.

[844] P. Ni, N. Huang, Z. Zhang, D.-P. Wang, F. Liang, Y. Miao, C.-L. Xiao, F. Luo, and J. Wang, "DeepSignal: detecting DNA methylation state from Nanopore sequencing reads using deep-learning," *Bioinformatics*, vol. 35, no. 22, pp. 4586–4595, Nov. 2019.

[845] Y. Liu, W. Rosikiewicz, Z. Pan, N. Jillette, P. Wang, A. Taghbalout, J. Foox, C. Mason, M. Carroll, A. Cheng, and S. Li, "DNA methylation-calling tools for Oxford Nanopore sequencing: a survey and human epigenome-wide evaluation," *Genome Biology*, vol. 22, no. 1, p. 295, Oct. 2021.

[846] Q. Liu, L. Fang, G. Yu, D. Wang, C.-L. Xiao, and K. Wang, "Detection of DNA base modifications by deep recurrent neural network on Oxford Nanopore sequencing data," *Nature Communications*, vol. 10, no. 1, p. 2449, Jun. 2019.

[847] M. U. Ahsan, A. Gouru, J. Chan, W. Zhou, and K. Wang, "A signal processing and deep learning framework for methylation detection using Oxford Nanopore sequencing," *Nature Communications*, vol. 15, no. 1, p. 1448, Feb. 2024.

[848] P. Ni, J. Xu, Z. Zhong, F. Luo, and J. Wang, "RNA m6A detection using raw current signals and basecalling errors from Nanopore direct RNA sequencing reads," *Bioinformatics*, vol. 40, no. 6, p. btae375, Jun. 2024.

[849] M. Krause, A. M. Niazi, K. Labun, Y. N. Torres Cleuren, F. S. Müller, and E. Valen, "tailfindr: alignment-free poly(A) length measurement for Oxford Nanopore RNA and DNA sequencing," *RNA*, vol. 25, no. 10, pp. 1229–1241, Oct. 2019.

[850] R. E. Workman, A. D. Tang, P. S. Tang, M. Jain, J. R. Tyson, R. Razaghi, P. C. Zuzarte, T. Gilpatrick, A. Payne, J. Quick, N. Sadowski, N. Holmes, J. G. de Jesus, K. L. Jones, C. M. Soulette, T. P. Snutch, N. Loman, B. Paten, M. Loose, J. T. Simpson, H. E. Olsen, A. N. Brooks, M. Akeson, and W. Timp, "Nanopore native RNA sequencing of a human poly(A) transcriptome," *Nature Methods*, vol. 16, no. 12, pp. 1297–1305, Dec. 2019.

[851] P. Giesselmann, B. Brändl, E. Raimondeau, R. Bowen, C. Rohrandt, R. Tandon, H. Kretzmer, G. Assum, C. Galonska, R. Siebert, O. Ammerpohl, A. Heron, S. A. Schneider, J. Ladewig, P. Koch, B. M. Schuldt, J. E. Graham, A. Meissner, and F.-J. Müller, "Analysis of short tandem repeat expansions and their methylation state with nanopore sequencing," *Nature Biotechnology*, vol. 37, no. 12, pp. 1478–1481, Dec. 2019.

[852] A. De Roeck, W. De Coster, L. Bossaerts, R. Cacace, T. De Pooter, J. Van Dongen, S. D'Hert, P. De Rijk, M. Strazisar, C. Van Broeckhoven, and K. Sleegers, "NanoSatellite: accurate characterization of expanded tandem repeat length and sequence through whole genome long-read sequencing on PromethION," *Genome Biology*, vol. 20, no. 1, p. 239, Nov. 2019.

[853] J. Sitarčík, T. Vinař, B. Brejová, W. Krampl, J. Budiš, J. Radvánszky, and M. Lucká, "WarpSTR: determining tandem repeat lengths using raw nanopore signals," *Bioinformatics*, vol. 39, no. 6, p. btad388, Jun. 2023.

[854] S. Baichoo and C. A. Ouzounis, "Computational complexity of algorithms for sequence comparison, short-read assembly and genome alignment," *Biosystems*, vol. 156-157, pp. 72–85, Jun. 2017.

[855] D. Lee, F. Hormozdiari, H. Xin, F. Hach, O. Mutlu, and C. Alkan, "Fast and accurate mapping of Complete Genomics reads," *Methods*, 2015.

[856] K. Sahlin, "Flexible seed size enables ultra-fast and accurate read alignment," *bioRxiv*, p. 2021.06.18.449070, 2022.

[857] M. X. Goemans and D. P. Williamson, "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming," *J. ACM*, vol. 42, no. 6, pp. 1115–1145, Nov. 1995.

[858] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[859] R. Pratap, A. Deshmukh, P. Nair, and A. Ravi, "Scaling up Simhash," in *Proceedings of The 12th Asian Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 129. PMLR, Nov. 2020, pp. 705–720.

[860] M. S. Uddin, C. K. Roy, K. A. Schneider, and A. Hindle, "On the Effectiveness of Simhash for Detecting Near-Miss Clones in Large Scale Software Systems," in *2011 18th Working Conference on Reverse Engineering*, 2011, pp. 13–22.

[861] S. Sood and D. Loguinov, "Probabilistic Near-Duplicate Detection Using Simhash," in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, ser. CIKM '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 1117–1126.

[862] X. Feng, H. Jin, R. Zheng, and L. Zhu, "Near-duplicate detection using GPU-based simhash scheme," in *2014 International Conference on Smart Computing*, 2014, pp. 223–228.

[863] M. Fröbe, J. Bevendorff, L. Gienapp, M. Völske, B. Stein, M. Potthast, and M. Hagen, "CopyCat: Near-Duplicates Within and Between the ClueWeb and the Common Crawl," in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 2398–2404.

[864] Q. A. Song, N. S. Catlin, W. Brad Barbazuk, and S. Li, "Computational analysis of alternative splicing in plant genomes," *Gene*, vol. 685, pp. 186–195, Feb. 2019.

[865] L. Denti, R. Rizzi, S. Beretta, G. D. Vedova, M. Previtali, and P. Bonizzoni, "ASGAL: aligning RNA-Seq data to a splicing graph to detect novel alternative splicing events," *BMC Bioinformatics*, vol. 19, no. 1, p. 444, Nov. 2018.

[866] D. Stanojević, D. Lin, P. Florez de Sessions, and M. Šikić, "Telomere-to-telomere phased genome assembly using error-corrected Simplex nanopore reads," *bioRxiv*, p. 2024.05.18.594796, 2024.

[867] S. Nabavi and F. Zare, "Identification of Copy Number Alterations from Next-Generation Sequencing Data," in *Computational Methods for Precision Oncology*, A. Laganà, Ed. Cham: Springer International Publishing, 2022, pp. 55–74.

[868] T. Kanagawa, "Bias and artifacts in multitemplate polymerase chain reactions (PCR)," *Journal of Bioscience and Bioengineering*, vol. 96, no. 4, pp. 317–323, Jan. 2003.

[869] M. J. P. Chaisson, R. K. Wilson, and E. E. Eichler, "Genetic variation and the de novo assembly of human genomes," *Nature Reviews Genetics*, vol. 16, no. 11, pp. 627–640, Nov. 2015.

[870] K. H. Y. Wong, W. Ma, C.-Y. Wei, E.-C. Yeh, W.-J. Lin, E. H. F. Wang, J.-P. Su, F.-J. Hsieh, H.-J. Kao, H.-H. Chen, S. K. Chow, E. Young, C. Chu, A. Poon, C.-F. Yang, D.-S. Lin, Y.-F. Hu, J.-Y. Wu, N.-C. Lee, W.-L. Hwu, D. Boffelli, D. Martin, M. Xiao, and P.-Y. Kwok, "Towards a reference genome that captures global genetic diversity," *Nature Communications*, vol. 11, no. 1, p. 5482, Oct. 2020.

[871] N. Bexfield and P. Kellam, "Metagenomics and the molecular identification of novel viruses," *The Veterinary Journal*, vol. 190, no. 2, pp. 191–198, Nov. 2011.

[872] Cangelosi Gerard A. and Meschke John S., "Dead or Alive: Molecular Assessment of Microbial Viability," *Applied and Environmental Microbiology*, vol. 80, no. 19, pp. 5884–5891, Oct. 2014.

[873] Y. Dong, H. Li, L. Zhao, P. Koopman, F. Zhang, and J. X. Huang, "Genome-Wide Off-Target Analysis in CRISPR-Cas9 Modified Mice and Their Offspring," *G3*, 2019.

[874] K. Sangkuhl, M. Whirl-Carrillo, R. M. Whaley, M. Woon, A. Lavertu, R. B. Altman, L. Carter, A. Verma, M. D. Ritchie, and T. E. Klein, "Pharmacogenomics Clinical Annotation Tool (PharmCAT)," *Clin. Pharm. Therap.*, 2020.

[875] S. Zverinova and V. Guryev, "Variant calling: Considerations, practices, and developments," *Human Mutation*, 2022.

[876] N.-C. Chen, L. F. Paulin, F. J. Sedlazeck, S. Koren, A. M. Phillippy, and B. Langmead, "Improved sequence mapping using a complete reference genome and lift-over," *Nature Methods*, vol. 21, no. 1, pp. 41–49, Jan. 2024.

[877] S. P. Sadedin and A. Oshlack, "Bazam: a rapid method for read extraction and realignment of high-throughput sequencing data," *Genome Biology*, vol. 20, no. 1, p. 78, Apr. 2019.

[878] A. Shumate and S. L. Salzberg, "Liftoff: accurate mapping of gene annotations," *Bioinformatics*, vol. 37, no. 12, pp. 1639–1643, Jul. 2021.

[879] H. Zhao, Z. Sun, J. Wang, H. Huang, J.-P. Kocher, and L. Wang, "CrossMap: a versatile tool for coordinate conversion between genome assemblies," *Bioinformatics*, vol. 30, no. 7, pp. 1006–1007, Apr. 2014.

[880] A. Talenti and J. Prendergast, "nf-LO: A Scalable, Containerized Workflow for Genome-to-Genome Lift Over," *Genome Biology and Evolution*, vol. 13, no. 9, p. evab183, Sep. 2021.

[881] T. Mun, N.-C. Chen, and B. Langmead, "LevioSAM: fast lift-over of variant-aware reference alignments," *Bioinformatics*, vol. 37, no. 22, pp. 4243–4245, Nov. 2021.

[882] B. Gao, Q. Huang, and M. Baudis, "segment_liftover : a Python tool to convert segments between genome assemblies [version 2; peer review: 2 approved]," *F1000Research*, vol. 7, no. 319, 2018.

[883] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "A Modern Primer on Processing in Memory," in *Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann*, 2023.

[884] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu, "Processing-in-Memory: A Workload-Driven Perspective," *IBM JRD*, 2019.

[885] O. Mutlu and others, "Processing Data Where It Makes Sense: Enabling In-Memory Computation," *MicPro*, 2019.

[886] V. Seshadri and O. Mutlu, "Simple Operations in Memory to Reduce Data Movement," in *Advances in Computers, Volume 106*, 2017.

[887] S. Ghose, K. Hsieh, A. Boroumand, R. Ausavarungnirun, and O. Mutlu, "The Processing-in-Memory Paradigm: Mechanisms to Enable Adoption," in *Beyond-CMOS Technologies for Next Generation Computer Design*, 2019.

[888] V. Seshadri and O. Mutlu, "In-DRAM Bulk Bitwise Execution Engine," *arXiv*, 2020.

[889] G. F. Oliveira, J. Gómez-Luna, S. Ghose, A. Boroumand, and O. Mutlu, "Accelerating Neural Network Inference With Processing-in-DRAM: From the Edge to the Cloud," *IEEE Micro*, vol. 42, no. 6, pp. 25–38, 2022.

[890] G. F. Oliveira, J. Gómez-Luna, L. Orosa, S. Ghose, N. Vijaykumar, I. Fernandez, M. Sadrosadati, and O. Mutlu, "DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks," *IEEE Access*, 2021.

[891] R. Bera, A. Ranganathan, J. Rakshit, S. Mahto, A. V. Nori, J. Gaur, A. Olgun, K. Kanellopoulos, M. Sadrosadati, S. Subramoney, and O. Mutlu, "Constable: Improving Performance and Power Efficiency by Safely Eliminating Load Instruction Execution," in *ISCA*, 2024, pp. 88–102.

[892] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *ISCA*, 2010, pp. 37–47.

[893] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 37–47, Jun. 2010.

[894] H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI)," vol. 1. Society for industrial and applied mathematics Philadelphia, PA, USA, 1979, pp. 256–282.

[895] Kung, "Why systolic architectures?" *Computer*, vol. 15, no. 1, pp. 37–46, 1982.

[896] J. Gómez-Luna, Y. Guo, S. Brocard, J. Legriel, R. Cimadomo, G. F. Oliveira, G. Singh, and O. Mutlu, "An Experimental Evaluation of Machine Learning Training on a Real Processing-in-Memory System," *arXiv*, 2022.

[897] J. Gómez-Luna, Y. Guo, S. Brocard, J. Legriel, R. Cimadomo, G. F. Oliveira, G. Singh, and O. Mutlu, "Machine Learning Training on a Real Processing-in-Memory System," in *ISVLSI*, 2022.

[898] G. Singh, D. Diamantopoulos, J. Gómez-Luna, C. Hagleitner, S. Stuijk, H. Corporaal, and O. Mutlu, "Accelerating Weather Prediction using Near-Memory Reconfigurable Fabric," *ACM TRETS*, 2021.

[899] A. Boroumand, S. Ghose, G. F. Oliveira, and O. Mutlu, "Polynesia: Enabling Effective Hybrid Transactional Analytical Databases with Specialized Hardware Software Co-Design," in *ICDE*, 2022.

[900] C. Giannoula, I. Fernandez, J. Gómez-Luna, N. Koziris, G. Goumas, and O. Mutlu, "SparseP: Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-In-Memory Systems," *arXiv*, 2022.

[901] C. Giannoula, I. Fernandez, J. Gómez-Luna, N. Koziris, G. Goumas, and O. Mutlu, "Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-in-Memory Architectures," in *SIGMETRICS*, 2022.

[902] A. Denzler, R. Bera, N. Hajinazar, G. Singh, G. F. Oliveira, J. Gómez-Luna, and O. Mutlu, "Casper: Accelerating Stencil Computation using Near-cache Processing," *arXiv*, 2021.

[903] L. Orosa, Y. Wang, M. Sadrosadati, J. S. Kim, M. Patel, I. Puddu, H. Luo, K. Razavi, J. Gomez-Luna, H. Hassan, N. Mansouri-Ghiasi, S. Ghose, and O. Mutlu, "CODIC: A Low-Cost Substrate for Enabling Custom In-DRAM Functionalities and Optimizations," in *ISCA*, 2021.

[904] O. Mutlu, "Intelligent Architectures for Intelligent Computing Systems," in *DATE*, 2021.

[905] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System," *IEEE Access*, 2022.

[906] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking Memory-Centric Computing Systems: Analysis of Real Processing-In-Memory Hardware," in *IGSC*, 2021.

[907] O. Mutlu, "Main Memory Scaling: Challenges and Solution Directions," in *More than Moore Technologies for Next Generation Computer Design*. Springer, 2015.

[908] J. D. Ferreira, G. Falcao, J. Gómez-Luna, M. Alser, L. Orosa, M. Sadrosadati, J. S. Kim, G. F. Oliveira, T. Shahroodi, A. Nori, and O. Mutlu, "pLUTo: In-DRAM Lookup Tables to Enable Massively Parallel General-Purpose Computation," *arXiv*, 2021.

[909] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, "Mitigating Edge Machine Learning Inference Bottlenecks: An Empirical Study on Accelerating Google Edge Models," *arXiv*, 2021.

[910] N. Hajinazar, G. F. Oliveira, S. Gregorio, J. D. Ferreira, N. M. Ghiasi, M. Patel, M. Alser, S. Ghose, J. Gómez-Luna, and O. Mutlu, "SIMDRAM: A Framework for Bit-Serial SIMD Processing Using DRAM," in *ASPLOS*, 2021.

[911] C. Giannoula, N. Vijaykumar, N. Papadopoulou, V. Karakostas, I. Fernandez, J. Gómez-Luna, L. Orosa, N. Koziris, G. Goumas, and O. Mutlu, "SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures," in *HPCA*, 2021.

[912] N. Vijaykumar, E. Ebrahimi, K. Hsieh, P. B. Gibbons, and O. Mutlu, "The Locality Descriptor: A Holistic Cross-layer Abstraction to Express Data Locality in GPUs," in *ISCA*, 2018.

[913] M. Kim, J. Park, G. Cho, Y. Kim, L. Orosa, O. Mutlu, and J. Kim, "Evanesco: Architectural Support for Efficient Data Sanitization in Modern Flash-Based Storage Systems," in *ASPLOS*, 2020.

[914] S. Song, A. Das, O. Mutlu, and N. Kandasamy, "Improving Phase Change Memory Performance with Data Content Aware Access," in *ISMM*, 2020.

[915] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda, "Reducing Memory Interference in Multicore Systems via Application-aware Memory Channel Partitioning," in *MICRO*, 2011.

[916] N. Hajinazar, P. Patel, M. Patel, K. Kanellopoulos, S. Ghose, R. Ausavarungnirun, G. F. d. Oliveira Jr, J. Appavoo, V. Seshadri, and O. Mutlu, "The Virtual Block Interface: A Flexible Alternative to the Conventional Virtual Memory Framework," in *ISCA*, 2020.

[917] V. Seshadri, O. Mutlu, M. A. Kozuch, and T. C. Mowry, "The Evicted-address Filter: A Unified Mechanism to Address both Cache Pollution and Thrashing," in *PACT*, 2012.

[918] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu, "WARM: Improving NAND Flash Memory Lifetime with Write-hotness Aware Retention Management," in *MSST*, 2015.

[919] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Crista, O. S. Unsal, and K. Mai, "Error Analysis and Retention-Aware Error Management for NAND Flash Memory," *Intel Technology Journal*, 2013.

[920] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis," in *DATE*, 2012.

[921] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. S. Unsal, and K. Mai, "Flash Correct-and-Refresh: Retention-aware Error Management for Increased Flash Memory Lifetime," in *ICCD*, 2012.

[922] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, "Program Interference in MLC NAND Flash Memory: Characterization, Modeling, and Mitigation," in *ICCD*, 2013.

[923] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Threshold Voltage Distribution in MLC NAND Flash Memory: Characterization, Analysis, and Modeling," in *DATE*, 2013.

[924] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu, "Data Retention in MLC NAND Flash Memory: Characterization, Optimization, and Recovery," in *HPCA*, 2015.

[925] Y. Cai, S. Ghose, Y. Luo, K. Mai, O. Mutlu, and E. F. Haratsch, "Vulnerabilities in MLC NAND Flash Memory Programming: Experimental Analysis, Exploits, and Mitigation Techniques," in *HPCA*, 2017.

[926] Y. Li, S. Ghose, J. Choi, J. Sun, H. Wang, and O. Mutlu, "Utility-based Hybrid Memory Management," in *CLUSTER*, 2017.

[927] H. Yoon, J. Meza, R. Ausavarungnirun, R. A. Harding, and O. Mutlu, "Row Buffer Locality Aware Caching Policies for Hybrid Memories," in *ICCD*, 2012.

[928] J. Haj-Yahya, Y. Sazeides, M. Alser, E. Rotem, and O. Mutlu, "Techniques for Reducing the Connected-Standby Energy Consumption of Mobile Devices," in *HPCA*, 2020.

[929] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," in *MICRO*, 2010.

[930] A. Boroumand, S. Ghose, M. Patel, H. Hassan, B. Lucia, R. Ausavarungnirun, K. Hsieh, N. Hajinazar, K. T. Malladi, H. Zheng, and O. Mutlu, "CoNDA: Efficient Cache Coherence Support for near-Data Accelerators," in *ISCA*, 2019.

[931] H. Luo, T. Shahroodi, H. Hassan, M. Patel, A. G. Yaglikci, L. Orosa, J. Park, and O. Mutlu, "CLR-DRAM: A Low-Cost DRAM Architecture Enabling Dynamic Capacity-Latency Trade-Off," in *ISCA*, 2020.

[932] G. Pekhimenko, T. C. Mowry, and O. Mutlu, "Linearly Compressed Pages: A Main Memory Compression Framework with Low Complexity and Low Latency," in *PACT*, 2012.

[933] G. Pekhimenko, V. Seshadri, Y. Kim, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Linearly Compressed Pages: A Low-Complexity, Low-Latency Main Memory Compression Framework," in *MICRO*, 2013.

[934] G. Singh, D. Diamantopoulos, C. Hagleitner, J. Gomez-Luna, S. Stuijk, O. Mutlu, and H. Corporaal, "NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling," in *FPL*, 2020.

[935] G. Singh, J. Gomez-Luna, G. Mariani, G. F. Oliveira, S. Corda, S. Stujik, O. Mutlu, and H. Corporaal, "NAPEL: Near-memory Computing Application Performance Prediction via Ensemble Learning," in *DAC*, 2019.

[936] I. Fernandez, R. Quislant, C. Giannoula, M. Alser, J. Gomez-Luna, E. Gutierrez, O. Plata, and O. Mutlu, "NATSA: A Near-Data Processing Accelerator for Time Series Analysis," in *ICCD*, 2020.

[937] S. H. S. Rezaei, M. Modarressi, R. Ausavarungnirun, M. Sadrosadati, O. Mutlu, and M. Daneshtalab, "NoM: Network-on-Memory for Inter-Bank Data Transfer in Highly-Banked Memories," *CAL*, 2020.

[938] J. Kim, M. Patel, H. Hassan, and O. Mutlu, "Solar-DRAM: Reducing DRAM Access Latency by Exploiting the Variation in Local Bitlines," in *ICCD*, 2018.

[939] O. Mutlu and J. S. Kim, "RowHammer: A Retrospective," *IEEE TCAD*, 2019.

[940] Y. Wang, A. Tavakkol, L. Orosa, S. Ghose, N. M. Ghiasi, M. Patel, J. S. Kim, H. Hassan, M. Sadrosadati, and O. Mutlu, "Reducing DRAM Latency via Charge-Level-Aware Look-Ahead Partial Restoration," in *MICRO*, 2018.

[941] O. Mutlu, S. Ghose, and R. Ausavarungnirun, "Recent Advances in DRAM and Flash Memory Architectures," *Invited Journal Issue IPSI Transactions on Internet Research*, 2018.

[942] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Enabling Practical Processing in and near Memory for Data-Intensive Computing," in *DAC*, 2019.

[943] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu, "A Workload and Programming Ease Driven Perspective of Processing-in-Memory," *arXiv*, 2019.

[944] R. Ausavarungnirun, V. Miller, J. Landgraf, S. Ghose, J. Gandhi, A. Jog, C. Rossbach, and O. Mutlu, "MASK: Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency," in *ASPLOS*, 2018.

[945] R. Ausavarungnirun, J. Landgraf, V. Miller, S. Ghose, J. Gandhi, C. J. Rossbach, and O. Mutlu, "Mosaic: Enabling Application-Transparent Support for Multiple Page Sizes in Throughput Processors," *SIGOPS Oper. Syst. Rev.*, 2018.

[946] A. Jog, O. Kayiran, N. C. Nachiappan, A. K. Mishra, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance," in *ASPLOS*, 2013.

[947] A. Jog, O. Kayiran, A. Pattnaik, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "Exploiting Core Criticality for Enhanced GPU Performance," in *SIGMETRICS*, 2016.

[948] S. Ghose, K. Hsieh, A. Boroumand, R. Ausavarungnirun, and O. Mutlu, "Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions," *arXiv*, 2018.

[949] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Errors in Flash-Memory-Based Solid-State Drives: Analysis, Mitigation, and Recovery," *arXiv*, 2018.

[950] K. Hsieh, S. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation," in *ICCD*, 2016.

[951] H. Hassan, N. Vijaykumar, S. Khan, S. Ghose, K. Chang, G. Pekhimenko, D. Lee, O. Ergin, and O. Mutlu, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," in *HPCA*, 2017.

[952] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," *CAL*, 2015.

[953] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Prefetch-Aware Shared Resource Management for Multi-core Systems," in *ISCA*, 2011.

[954] Z. Liu, I. Calciu, M. Herlihy, and O. Mutlu, "Concurrent Data Structures for Near-Memory Computing," in *SPAA*, 2017.

[955] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," in *ISCA*, 2012.

[956] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," in *ISCA*, 2009.

[957] H. Yoon, J. Meza, N. Muralimanohar, N. P. Jouppi, and O. Mutlu, "Efficient Data Mapping and Buffering Techniques for Multilevel Cell Phase-Change Memories," *ACM TACO*, 2014.

[958] M. Hashemi, O. Mutlu, and Y. N. Patt, "Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads," in *MICRO*, 2016.

[959] D. Lee, Y. Kim, V. Seshadri, J. Liu, L. Subramanian, and O. Mutlu, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," in *HPCA*, 2013.

[960] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, 2012.

[961] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," in *ISCA*, 2014.

[962] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field," in *DSN*, 2015.

[963] O. Mutlu and L. Subramanian, "Research Problems and Opportunities in Memory Systems," *SUPERFRI*, 2014.

[964] O. Mutlu, "Memory Scaling: A Systems Architecture Perspective," *IMW*, 2013.

[965] K. K. Chang, A. Kashyap, H. Hassan, S. Ghose, K. Hsieh, D. Lee, T. Li, G. Pekhimenko, S. Khan, and O. Mutlu, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," in *SIGMETRICS*, 2016.

[966] A. Boroumand, S. Ghose, M. Patel, H. Hassan, B. Lucia, K. Hsieh, K. T. Malladi, H. Zheng, and O. Mutlu, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," *CAL*, 2016.

[967] K. Hsieh and others, "A Pointer Chasing Accelerator for 3D-Stacked Memory," *CMU SAFARI Technical Report No. 2016-007*, 2016.

[968] A. Pattnaik, X. Tang, A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, and C. R. Das, "Scheduling Techniques for GPU Architectures with Processing-in-Memory Capabilities," in *PACT*, 2016.

[969] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback Directed Prefetching: Improving the Performance and Bandwidth-Efficiency of Hardware Prefetchers," in *HPCA*, 2007.

[970] O. Mutlu, H. Kim, and Y. N. Patt, "Techniques for Efficient Processing in Runahead Execution Engines," in *ISCA*, 2005.

[971] D. Lee, S. Ghose, G. Pekhimenko, S. Khan, and O. Mutlu, "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," *TACO*, 2016.

[972] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," in *ISCA*, 2015.

[973] K. Hsieh, E. Ebrahimi, G. Kim, N. Chatterjee, M. O'Conner, N. Vijaykumar, O. Mutlu, and S. Keckler, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," in *ISCA*, 2016.

[974] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," in *ISCA*, 2015.

[975] V. Seshadri, A. Bhowmick, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "The Dirty-Block Index," in *ISCA*, 2014.

[976] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, M. A. Kozuch, P. B. Gibbons, and T. C. Mowry, "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization," in *MICRO*, 2013.

[977] V. Seshadri, K. Hsieh, A. Boroumand, D. Lee, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Fast Bulk Bitwise AND and OR in DRAM," *CAL*, 2015.

[978] J. Joao, O. Mutlu, and Y. N. Patt, "Flexible Reference-Counting-Based Hardware Acceleration for Garbage Collection," in *ISCA*, 2009.

[979] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Buddy-RAM: Improving the Performance and Efficiency of Bulk Bitwise Operations Using DRAM," *arXiv*, 2016.

[980] V. Seshadri and O. Mutlu, "The Processing Using Memory Paradigm: In-DRAM Bulk Copy, Initialization, Bitwise AND and OR," *arXiv*, 2016.

[981] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM," in *HPCA*, 2016.

[982] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *MICRO*, 2017.

[983] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," in *ISPASS*, 2013.

[984] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-Change Technology and the Future of Main Memory," *IEEE Micro*, 2010.

[985] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Phase Change Memory Architecture and the Quest for Scalability," *CACM*, 2010.

[986] J. Meza, Y. Luo, S. Khan, J. Zhao, Y. Xie, and O. Mutlu, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," in *WEED*, 2013.

[987] J. Zhao, O. Mutlu, and Y. Xie, "FIRM: Fair and High-Performance Memory Control for Persistent Memory Systems," in *MICRO*, 2014.

[988] J. Ren, J. Zhao, S. Khan, J. Choi, Y. Wu, and O. Mutlu, "ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems," in *MICRO*, 2015.

[989] J. Kim, M. Patel, H. Hassan, and O. Mutlu, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency–Reliability Tradeoff in Modern DRAM Devices," in *HPCA*, 2018.

[990] L. Subramanian, V. Seshadri, Y. Kim, B. Jaiyen, and O. Mutlu, "MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems," in *HPCA*, 2013.

[991] L. Subramanian, V. Seshadri, A. Ghosh, Samira Khan, and O. Mutlu, "The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory," in *MICRO*, 2015.

[992] O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," in *MICRO*, 2007.

[993] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems," in *ISCA*, 2008.

[994] R. Ausavarungnirun, S. Ghose, O. Kayıran, G. H. Loh, C. R. Das, M. T. Kandemir, and O. Mutlu, "Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance," in *PACT*, 2015.

[995] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives," *Proc. IEEE*, 2017.

[996] A. Tavakkol, M. Sadrosadati, S. Ghose, J. Kim, Y. Luo, Y. Wang, N. M. Ghiasi, L. Orosa, J. Gómez-Luna, and O. Mutlu, "FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives," in *ISCA*, 2018.

[997] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-Optimizing Memory Controllers: A Reinforcement Learning Approach," in *ISCA*, 2008.

[998] G. Singh, R. Nadig, J. Park, R. Bera, N. Hajinazar, D. Novo, J. Gómez-Luna, S. Stuijk, H. Corporaal, and O. Mutlu, "Sibyl: Adaptive and Extensible Data Placement in Hybrid Storage Systems Using Online Reinforcement Learning," in *ISCA*, 2022.

[999] O. Mutlu, "Lightning Talk: Memory-Centric Computing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–2.

[1000] C. Zou and A. A. Chien, "ASSASIN: Architecture Support for Stream Computing to Accelerate Computational Storage," in *MICRO*, 2022, pp. 354–368.

[1001] J. Park, R. Azizi, G. F. Oliveira, M. Sadrosadati, R. Nadig, D. Novo, J. Gómez-Luna, M. Kim, and O. Mutlu, "Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory," in *MICRO*, 2022, pp. 937–955.

[1002] A. C. Elster and T. A. Haugdahl, "Nvidia Hopper GPU and Grace CPU Highlights," *Computing in Science & Engineering*, vol. 24, no. 2, pp. 95–100, 2022.

[1003] D. R. Solli and B. Jalali, "Analog optical computing," *Nature Photonics*, vol. 9, no. 11, pp. 704–706, Nov. 2015.

[1004] C. Herzeel, P. Costanza, D. Decap, J. Fostier, R. Wuyts, and W. Verachtert, "Multithreaded variant calling in elPrep 5," *PLOS ONE*, 2021.

[1005] C. Jain, A. Rhie, N. F. Hansen, S. Koren, and A. M. Phillippy, "Long-read mapping to repetitive reference sequences using Winnowmap2," *Nature Methods*, Apr. 2022.

[1006] D. DeBlasio, F. Gbosibo, C. Kingsford, and G. Marçais, "Practical Universal K-Mer Sets for Minimizer Schemes," in *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, ser. BCB '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 167–176.

[1007] M. S. Lindner, B. Strauch, J. M. Schulze, S. H. Tausch, P. W. Dabrowski, A. Nitsche, and B. Y. Renard, "HiLive: real-time mapping of illumina reads while sequencing," *Bioinformatics*, vol. 33, no. 6, pp. 917–319, Mar. 2017.

[1008] T. P. Loka, S. H. Tausch, and B. Y. Renard, "Reliable variant calling during runtime of Illumina sequencing," *Scientific Reports*, vol. 9, no. 1, p. 16502, Nov. 2019.

[1009] S. H. Tausch, B. Strauch, A. Andrusch, T. P. Loka, M. S. Lindner, A. Nitsche, and B. Y. Renard, "LiveKraken−−real-time metagenomic classification of illumina data," *Bioinformatics*, vol. 34, no. 21, pp. 3750–3752, Nov. 2018.

[1010] H. Stranneheim, M. Engvall, K. Naess, N. Lesko, P. Larsson, M. Dahlberg, R. Andeer, A. Wredenberg, C. Freyer, M. Barbaro, H. Bruhn, T. Emahazion, M. Magnusson, R. Wibom, R. H. Zetterström, V. Wirta, U. von Döbeln, and A. Wedell, "Rapid pulsed whole genome sequencing for comprehensive acute diagnostics of inborn errors of metabolism," *BMC Genomics*, vol. 15, no. 1, p. 1090, Dec. 2014.

[1011] D. Zhang, J. Zhang, J. Du, Y. Zhou, P. Wu, Z. Liu, Z. Sun, J. Wang, W. Ding, J. Chen, J. Wang, Y. Xu, C. Ouyang, and Q. Yang, "Optimized Sequencing Adaptors Enable Rapid and Real-Time Metagenomic Identification of Pathogens during Runtime of Sequencing," *Clinical Chemistry*, vol. 68, no. 6, pp. 826–836, Jun. 2022.

[1012] S. H. Tausch, T. P. Loka, J. M. Schulze, A. Andrusch, J. Klenner, P. W. Dabrowski, M. S. Lindner, A. Nitsche, and B. Y. Renard, "PathoLive—Real-Time Pathogen Identification from Metagenomic Illumina Datasets," *Life*, vol. 12, no. 9, 2022.

[1013] T. P. Loka, S. H. Tausch, P. W. Dabrowski, A. Radonić, A. Nitsche, and B. Y. Renard, "PriLive: privacy-preserving real-time filtering for next-generation sequencing," *Bioinformatics*, vol. 34, no. 14, pp. 2376–2383, Jul. 2018.

[1014] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[1015] R. Bellman, "Dynamic Programming," *Science*, vol. 153, no. 3731, pp. 34–37, Jul. 1966.

[1016] R. Bellman and R. Kalaba, "On adaptive control processes," *IRE Transactions on Automatic Control*, vol. 4, no. 2, pp. 1–9, 1959.

[1017] M.-M. Aynaud, J. J. Hernandez, S. Barutcu, U. Braunschweig, K. Chan, J. D. Pearson, D. Trcka, S. L. Prosser, J. Kim, M. Barrios-Rodiles, M. Jen, S. Song, J. Shen, C. Bruce, B. Hazlett, S. Poutanen, L. Attisano, R. Bremner, B. J. Blencowe, T. Mazzulli, H. Han, L. Pelletier, and J. L. Wrana, "A multiplexed, next generation sequencing platform for high-throughput detection of SARS-CoV-2," *Nature Communications*, vol. 12, no. 1, p. 1405, Mar. 2021.

[1018] T. Mantere, S. Kersten, and A. Hoischen, "Long-Read Sequencing Emerging in Medical Genetics," *Frontiers in Genetics*, vol. 10, p. 426, 2019.

[1019] J. M. Friedman, Y. Bombard, M. C. Cornel, C. V. Fernandez, A. K. Junker, S. E. Plon, Z. Stark, B. M. Knoppers, and for the Paediatric Task Team of the Global Alliance for Genomics and Health Regulatory and Ethics Work Stream, "Genome-wide sequencing in acutely ill infants: genomic medicine's critical application?" *Genetics in Medicine*, vol. 21, no. 2, pp. 498–504, Feb. 2019.

[1020] J. D. Merker, A. M. Wenger, T. Sneddon, M. Grove, Z. Zappala, L. Fresard, D. Waggott, S. Utiramerur, Y. Hou, K. S. Smith, S. B. Montgomery, M. Wheeler, J. G. Buchan, C. C. Lambert, K. S. Eng, L. Hickey, J. Korlach, J. Ford, and E. A. Ashley, "Long-read genome sequencing identifies causal structural variation in a Mendelian disease," *Genetics in Medicine*, vol. 20, no. 1, pp. 159–163, Jan. 2018.

[1021] S. Goodwin, J. D. McPherson, and W. R. McCombie, "Coming of age: ten years of next-generation sequencing technologies," *Nature Reviews Genetics*, vol. 17, no. 6, pp. 333–351, May 2016.

[1022] N. Stoler and A. Nekrutenko, "Sequencing error profiles of Illumina sequencing instruments," *NAR Genomics and Bioinformatics*, vol. 3, no. 1, Mar. 2021.

[1023] H. Zhang, C. Jain, and S. Aluru, "A comprehensive evaluation of long read error correction methods," *BMC Genomics*, vol. 21, no. 6, p. 889, Dec. 2020.

[1024] T. Hon, K. Mars, G. Young, Y.-C. Tsai, J. W. Karalius, J. M. Landolin, N. Maurer, D. Kudrna, M. A. Hardigan, C. C. Steiner, S. J. Knapp, D. Ware, B. Shapiro, P. Peluso, and D. R. Rank, "Highly accurate long-read HiFi sequencing data for five complex genomes," *Scientific Data*, vol. 7, no. 1, p. 399, Nov. 2020.

[1025] X. Ma, Y. Shao, L. Tian, D. A. Flasch, H. L. Mulder, M. N. Edmonson, Y. Liu, X. Chen, S. Newman, J. Nakitandwe, Y. Li, B. Li, S. Shen, Z. Wang, S. Shurtleff, L. L. Robison, S. Levy, J. Easton, and J. Zhang, "Analysis of error profiles in deep next-generation sequencing data," *Genome Biology*, vol. 20, no. 1, p. 50, Mar. 2019.

[1026] S. Canzar and S. L. Salzberg, "Short Read Mapping: An Algorithmic Tour," *Proceedings of the IEEE*, vol. 105, no. 3, pp. 436–458, Mar. 2017.

[1027] G. Robertson, J. Schein, R. Chiu, R. Corbett, M. Field, S. D. Jackman, K. Mungall, S. Lee, H. M. Okada, J. Q. Qian, M. Griffith, A. Raymond, N. Thiessen, T. Cezard, Y. S. Butterfield, R. Newsome, S. K. Chan, R. She, R. Varhol, B. Kamoh, A.-L. Prabhu, A. Tam, Y. Zhao, R. A. Moore, M. Hirst, M. A. Marra, S. J. M. Jones, P. A. Hoodless, and I. Birol, "De novo assembly and analysis of RNA-seq data," *Nature Methods*, vol. 7, no. 11, pp. 909–912, Nov. 2010.

[1028] F. Meyer, A. Fritz, Z.-L. Deng, D. Koslicki, T. R. Lesker, A. Gurevich, G. Robertson, M. Alser, D. Antipov, F. Beghini, D. Bertrand, J. J. Brito, C. T. Brown, J. Buchmann, A. Buluç, B. Chen, R. Chikhi, P. T. L. C. Clausen, A. Cristian, P. W. Dabrowski, A. E. Darling, R. Egan, E. Eskin, E. Georganas, E. Goltsman, M. A. Gray, L. H. Hansen, S. Hofmeyr, P. Huang, L. Irber, H. Jia, T. S. Jørgensen, S. D. Kieser, T. Klemetsen, A. Kola, M. Kolmogorov, A. Korobeynikov, J. Kwan, N. LaPierre, C. Lemaitre, C. Li, A. Limasset, F. Malcher-Miranda, S. Mangul, V. R. Marcelino, C. Marchet, P. Marijon, D. Meleshko, D. R. Mende, A. Milanese, N. Nagarajan, J. Nissen, S. Nurk, L. Oliker, L. Paoli, P. Peterlongo, V. C. Piro, J. S. Porter, S. Rasmussen, E. R. Rees, K. Reinert, B. Renard, E. M. Robertsen, G. L. Rosen, H.-J. Ruscheweyh, V. Sarwal, N. Segata, E. Seiler, L. Shi, F. Sun, S. Sunagawa, S. J. Sørensen, A. Thomas, C. Tong, M. Trajkovski, J. Tremblay, G. Uritskiy, R. Vicedomini, Z. Wang, Z. Wang, Z. Wang, A. Warren, N. P. Willassen, K. Yelick, R. You, G. Zeller, Z. Zhao, S. Zhu, J. Zhu, R. Garrido-Oter, P. Gastmeier, S. Hacquard, S. Häußler, A. Khaledi, F. Maechler, F. Mesny, S. Radutoiu, P. Schulze-Lefert, N. Smit, T. Strowig, A. Bremges, A. Sczyrba, and A. C. McHardy, "Critical Assessment of Metagenome Interpretation: the second round of challenges," *Nature Methods*, vol. 19, no. 4, pp. 429–440, Apr. 2022.

[1029] R. Vaser, I. Sović, N. Nagarajan, and M. Šikić, "Fast and accurate de novo genome assembly from long uncorrected reads," *Genome Research*, vol. 27, no. 5, pp. 737–746, May 2017.

[1030] R. Lederman, "A random-permutations-based approach to fast read alignment," *BMC Bioinformatics*, vol. 14, no. 5, p. S8, Apr. 2013.

[1031] P. Jaccard, "Nouvelles recherches sur la distribution florale," *Bull. Soc. Vaud. Sci. Nat.*, vol. 44, pp. 223–270, 1908.

[1032] M. Pop, A. Phillippy, A. L. Delcher, and S. L. Salzberg, "Comparative genome assembly," *Briefings in Bioinformatics*, vol. 5, no. 3, pp. 237–248, Sep. 2004.

[1033] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, and M. A. DePristo, "The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data," *Genome Research*, vol. 20, no. 9, pp. 1297–1303, Sep. 2010.

[1034] Y. Ono, K. Asai, and M. Hamada, "PBSIM2: a simulator for long-read sequencers with a novel generative model of quality scores," *Bioinformatics*, vol. 37, no. 5, pp. 589–595, Mar. 2021.

[1035] W. Shen, S. Le, Y. Li, and F. Hu, "SeqKit: A Cross-Platform and Ultrafast Toolkit for FASTA/Q File Manipulation," *PLOS ONE*, vol. 11, no. 10, p. e0163962, Oct. 2016.

[1036] E. S. Tvedte, M. Gasser, B. C. Sparklin, J. Michalski, C. E. Hjelmen, J. S. Johnston, X. Zhao, R. Bromley, L. J. Tallon, L. Sadzewicz, D. A. Rasko, and J. C. Dunning Hotopp, "Comparison of long-read sequencing technologies in interrogating bacteria and fly genomes," *G3 Genes\textbarGenomes\textbarGenetics*, vol. 11, no. 6, Jun. 2021.

[1037] A. Gurevich, V. Saveliev, N. Vyahhi, and G. Tesler, "QUAST: quality assessment tool for genome assemblies," *Bioinformatics*, vol. 29, no. 8, pp. 1072–1075, Apr. 2013.

[1038] A. R. Quinlan and I. M. Hall, "BEDTools: a flexible suite of utilities for comparing genomic features," *Bioinformatics*, vol. 26, no. 6, pp. 841–842, Mar. 2010.

[1039] B. S. Pedersen and A. R. Quinlan, "Mosdepth: quick coverage calculation for genomes and exomes," *Bioinformatics*, vol. 34, no. 5, pp. 867–868, Mar. 2018.

[1040] G. Jun, M. K. Wing, G. R. Abecasis, and H. M. Kang, "An efficient and scalable analysis framework for variant extraction and refinement from population scale DNA sequence data," *Genome Research*, Apr. 2015.

[1041] M. Smolka, L. F. Paulin, C. M. Grochowski, M. Mahmoud, S. Behera, M. Gandhi, K. Hong, D. Pehlivan, S. W. Scholz, C. M. Carvalho, C. Proukakis, and F. J. Sedlazeck, "Comprehensive Structural Variant Detection: From Mosaic to Population-Level," *bioRxiv*, p. 2022.04.04.487055, 2022.

[1042] A. C. English, V. K. Menon, R. Gibbs, G. A. Metcalf, and F. J. Sedlazeck, "Truvari: Refined Structural Variant Comparison Preserves Allelic Diversity," *bioRxiv*, p. 2022.02.21.481353, 2022.

[1043] J. M. Zook, N. F. Hansen, N. D. Olson, L. Chapman, J. C. Mullikin, C. Xiao, S. Sherry, S. Koren, A. M. Phillippy, P. C. Boutros, S. M. E. Sahraeian, V. Huang, A. Rouette, N. Alexander, C. E. Mason, I. Hajirasouliha, C. Ricketts, J. Lee, R. Tearle, I. T. Fiddes, A. M. Barrio, J. Wala, A. Carroll, N. Ghaffari, O. L. Rodriguez, A. Bashir, S. Jackman, J. J. Farrell, A. M. Wenger, C. Alkan, A. Soylev, M. C. Schatz, S. Garg, G. Church, T. Marschall, K. Chen, X. Fan, A. C. English, J. A. Rosenfeld, W. Zhou, R. E. Mills, J. M. Sage, J. R. Davis, M. D. Kaiser, J. S. Oliver, A. P. Catalano, M. J. P. Chaisson, N. Spies, F. J. Sedlazeck, and M. Salit, "A robust benchmark for detection of germline large deletions and insertions," *Nature Biotechnology*, vol. 38, no. 11, pp. 1347–1355, 2020.

[1044] A. Zeni, G. Guidi, M. Ellis, N. Ding, M. D. Santambrogio, S. Hofmeyr, A. Buluç, L. Oliker, and K. Yelick, "LOGAN: High-Performance GPU-Based X-Drop Long-Read Alignment," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2020, pp. 462–471.

[1045] Y. Yan, N. Chaturvedi, and R. Appuswamy, "Accel-Align: a fast sequence mapper and aligner based on the seed–embed–extend method," *BMC Bioinformatics*, 2021.

[1046] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen, "A Survey on Learning to Hash," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 769–790, 2018.

[1047] S. Dasgupta, C. F. Stevens, and S. Navlakha, "A neural algorithm for a fundamental computing problem," *Science*, vol. 358, no. 6364, pp. 793–796, Nov. 2017.

[1048] F. Ramírez, D. P. Ryan, B. Grüning, V. Bhardwaj, F. Kilpert, A. S. Richter, S. Heyne, F. Dündar, and T. Manke, "deepTools2: a next generation web server for deep-sequencing data analysis," *Nucleic Acids Research*, vol. 44, no. W1, pp. W160–W165, Jul. 2016.

[1049] Oxford Nanopore Technologies, "K-mer models," 2023. [Online]. Available: https://github.com/nanoporetech/kmer_models/

[1050] M. Jain, S. Koren, K. H. Miga, J. Quick, A. C. Rand, T. A. Sasani, J. R. Tyson, A. D. Beggs, A. T. Dilthey, I. T. Fiddes, S. Malla, H. Marriott, T. Nieto, J. O'Grady, H. E. Olsen, B. S. Pedersen, A. Rhie, H. Richardson, A. R. Quinlan, T. P. Snutch, L. Tee, B. Paten, A. M. Phillippy, J. T. Simpson, N. J. Loman, and M. Loose, "Nanopore sequencing and assembly of a human genome with ultra-long reads," *Nature Biotechnology*, vol. 36, no. 4, pp. 338–345, Apr. 2018.

[1051] H. Gamaarachchi, H. Samarakoon, S. P. Jenner, J. M. Ferguson, T. G. Amos, J. M. Hammond, H. Saadat, M. A. Smith, S. Parameswaran, and I. W. Deveson, "Fast nanopore sequencing data analysis with SLOW5," *Nat. Biotechnol.*, 2022.

[1052] R. Flynn, S. Washer, A. R. Jeffries, A. Andrayas, G. Shireby, M. Kumari, L. C. Schalkwyk, J. Mill, and E. Hannon, "Evaluation of nanopore sequencing for epigenetic epidemiology: a comparison with DNA methylation microarrays," *Hum. Mol. Genet.*, 2022.

[1053] B. Bloemen, M. Gand, K. Vanneste, K. Marchal, N. H. Roosens, and S. C. De Keersmaecker, "Development of a portable on-site applicable metagenomic data generation workflow for enhanced pathogen and antimicrobial resistance surveillance," *Sci. Rep.*, 2023.

[1054] I. Sović, K. Skala, and M. Šikić, "Approaches to DNA de novo assembly," in *MIPRO*, 2013.

[1055] H. Li, "GFA: Graphical Fragment Assembly (GFA) Format Specification," 2023. [Online]. Available: https://github.com/GFA-spec/GFA-spec

[1056] S. Mercogliano, "Rawasm," 2024. [Online]. Available: https://github.com/CMU-SAFARI/rawasm

[1057] "NVIDIA RTX A6000," 2024. [Online]. Available: https://www.nvidia.com/en-us/design-visualization/rtx-a6000

[1058] "Intel® Xeon® Gold 6226R Processor," 2024. [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/199347/intel-xeon-gold-6226r-processor-22m-cache-2-90-ghz/specifications.html

[1059] Oxford Nanopore Technologies, "Human assembly workflow," 2022. [Online]. Available: https://a.storyblok.com/f/196663/cd1c1c07ec/human-assembly-workflow.pdf

[1060] H. Li, "aun: a new metric to measure assembly contiguity," 2020. [Online]. Available: https://lh3.github.io/2020/04/08/a-new-metric-on-assembly-contiguity

[1061] R. R. Wick, M. B. Schultz, J. Zobel, and K. E. Holt, "Bandage: interactive visualization of de novo genome assemblies," *Bioinform.*, 2015.

[1062] A. Magi, R. Semeraro, A. Mingrino, B. Giusti, and R. D'Aurizio, "Nanopore sequencing data analysis: state of the art, applications and challenges," *Briefings in Bioinformatics*, 2018.

[1063] S. El-Metwally, M. Zakaria, and T. Hamza, "LightAssembler: fast and memory-efficient assembly algorithm for high-throughput sequencing reads," *Bioinform.*, 2016.

[1064] R. Rozov, G. Goldshlager, E. Halperin, and R. Shamir, "Faucet: streaming de novo assembly graph construction," *Bioinform.*, 2018.

[1065] C. Scott, "Streaming Methods for Assembly Graph Analysis," Ph.D., University of California, Davis, 2022.

[1066] J. Sun, R. Li, C. Chen, J. D. Sigwart, and K. M. Kocot, "Benchmarking oxford nanopore read assemblers for high-quality molluscan genomes," *Philosophical Transactions of the Royal Society*, Apr. 2021.

[1067] R. R. Wick and K. E. Holt, "Benchmarking of long-read assemblers for prokaryote whole genome sequencing," *F1000Research*, 2021.

[1068] B.-M. Cosma, R. Shirali HosseinZade, E. N. Jordan, P. vanLent, C. Peng, S. Pillay, and T. Abeel, "Evaluating long-read de novo assembly tools for eukaryotic genomes: insights and considerations," *GigaScience*, 2022.

[1069] W. Yu, H. Luo, J. Yang, S. Zhang, H. Jiang, X. Zhao, X. Hui, D. Sun, L. Li, X.-q. Wei, S. Lonardi, and W. Pan, "Comprehensive assessment of 11 de novo hifi assemblers on complex eukaryotic genomes and metagenomes," *Genome Research*, 2024.

[1070] M. Alser, B. Lawlor, R. J. Abdill, S. Waymost, R. Ayyala, N. Rajkumar, N. LaPierre, J. Brito, A. M. Ribeiro-dos Santos, N. Almadhoun, V. Sarwal, C. Firtina, T. Osinski, E. Eskin, Q. Hu, D. Strong, B.-D. B. Kim, M. S. Abedalthagafi, O. Mutlu, and S. Mangul, "Packaging and containerization of computational methods," *Nature Protocols*, Apr. 2024.

[1071] K. Kanellopoulos, R. Bera, K. Stojiljkovic, F. N. Bostanci, C. Firtina, R. Ausavarungnirun, R. Kumar, N. Hajinazar, M. Sadrosadati, N. Vijaykumar, and O. Mutlu, "Utopia: Fast and efficient address translation via hybrid restrictive & flexible virtual-to-physical address mappings," in *MICRO*, 2023.

[1072] M.-D. Rumpf, M. Alser, A. E. Gollwitzer, J. Lindegger, N. Almadhoun, C. Firtina, S. Mangul, and O. Mutlu, "Sequencelab: A comprehensive benchmark of computational methods for comparing genomic sequences," *arXiv*, 2023.

[1073] A. E. Gollwitzer, M. Alser, J. Bergtholdt, J. Lindegger, M.-D. Rumpf, C. Firtina, S. Mangul, and O. Mutlu, "Metafast: Enabling fast metagenomic classification via seed counting and edit distance approximation," *arXiv*, 2023.