Bioinformatics, 2026, **42**(3), btag087
https://doi.org/10.1093/bioinformatics/btag087
Published: 26 February 2026
Original Paper | Sequence Analysis

OXFORD

Rawsamble: overlapping raw nanopore signals using a hash-based seeding mechanism

Can Firtina[1,2,*], Maximilian Mordig[3,4], Harun Mustafa[3,5,6], Sayan Goswami[3], Nika Mansouri Ghiasi[1], Stefano Mercogliano[1], Furkan Eris[1], Joel Lindegger[1], André Kahles[3,5,6,*], Onur Mutlu[1,*]

[1]Department of Information Technology and Electrical Engineering, ETH Zurich, Zurich, 8092, Switzerland
[2]Department of Computer Science, University of Maryland, MD 20742, United States
[3]Department of Computer Science, ETH Zurich, Zurich, 8092, Switzerland
[4]Max Planck Institute for Intelligent Systems, Tubingen, 72076, Germany
[5]University Hospital Zurich, Zurich, 8091, Switzerland
[6]Swiss Institute of Bioinformatics, Zurich, 8057, Switzerland
*Corresponding authors. Can Firtina, Department of Computer Science, University of Maryland, MD 20742, United States. E-mail: firtina@umd.edu; André Kahles, Department of Computer Science, ETH Zurich, Zurich, 8092, Switzerland. Onur Mutlu, E-mail: andre.kahles@inf.ethz.ch; Onur Mutlu, Department of Information Technology and Electrical Engineering, ETH Zurich, Zurich, 8092, Switzerland. E-mail: omutlu@ethz.ch.
Associate Editor: Inanc Birol

Abstract

Motivation: **Raw nanopore signal analysis is a common approach in genomics to provide fast and resource-efficient analysis without translating the signals to bases (i.e. without basecalling). However, existing solutions cannot interpret raw signals directly if a reference genome is unknown due to a lack of accurate mechanisms to handle increased noise 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 identify regions of similarity between all raw signal pairs, known as *all-vs-all overlapping*, using a hash-based search mechanism.**

Results: **We use these overlaps to construct *de novo* assembly graphs with an existing assembler, miniasm, off-the-shelf. To our knowledge, these are the first *de novo* assemblies ever constructed directly from raw signals without basecalling. Our extensive evaluations across multiple genomes of varying sizes show that Rawsamble provides a significant speedup (on average by 5.01× and up to 23.10×) and reduces peak memory usage (on average by 5.74× and up to by 22.00×) compared to a conventional genome assembly pipeline using the state-of-the-art tools for basecalling (Dorado's fastest mode) and overlapping (minimap2) on a CPU. We find that around one-third of Rawsamble's overlapping pairs are also found by minimap2. We find that when we use overlapping reads from Rawsamble, we can construct unitigs that are (i) as accurate as those built from minimap2's overlaps and (ii) up to half a chromosome in length (e.g. 2.3 million bases for *E. coli*).**

Availability and implementation: **Rawsamble is available at https://github.com/CMU-SAFARI/RawHash. We also provide the scripts to fully reproduce our results on our GitHub page.**

1 Introduction

Nanopore sequencing technology can sequence long nucleic acid molecules, called *reads*, of up to a few million bases at high throughput (Jain *et al.* 2018, Senol Cali *et al.* 2019, Pugh 2023). As a molecule moves through a tiny *nanopore*, ionic current measurements, called *raw signals*, are generated (Deamer *et al.* 2016).

Nanopore sequencing provides two 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* (Loose *et al.* 2016), 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, compact nanopore sequencing devices enable on-site portable sequencing and analysis, which can be coupled with real-time analysis (Bloemen *et al.* 2023).

Existing works that analyze raw nanopore signals (Timp *et al.* 2012, Loman *et al.* 2015, Schreiber and Karplus 2015, Loose

Received: 22 July 2025. **Revised:** 9 January 2026. **Accepted:** 6 February 2026

et al. 2016, Boža et al. 2017, 2020, David et al. 2017, Teng et al. 2018, Miculinic et al. 2019, Zeng et al. 2019, Gamaarachchi et al. 2020, Lv et al. 2020, Zhang et al. 2020, Bao et al. 2021, Dunn et al. 2021, Konishi et al. 2021, Kovaka et al. 2021, Perešíni et al. 2021, Samarasinghe et al. 2021, Xu et al. 2021, Zhang et al. 2021, Huang et al. 2022, Shih et al. 2023, Yeh and Lu 2022, Noordijk et al. 2023, Sadasivan et al. 2023, Senanayake et al. 2023, Xu et al. 2023, Firtina et al. 2023a, 2024, Cavlak et al. 2024, Lindegger et al. 2024, Sadasivan et al. 2024, Shivakumar et al. 2024, Singh et al. 2024, Eris et al. 2025, Kovaka et al. 2025) mainly utilize complex deep learning mechanisms (Boža et al. 2017, 2020, Teng et al. 2018, Miculinic et al. 2019, Zeng et al. 2019, Lv et al. 2020, Zhang et al. 2020, Konishi et al. 2021, Perešíni et al. 2021, Xu et al. 2021, Huang et al. 2022, Yeh and Lu 2022, Noordijk et al. 2023, Xu et al. 2023, Cavlak et al. 2024, Singh et al. 2024) to translate these signals into nucleotides, a process called *basecalling*. Basecalling mechanisms usually (i) are designed to use large chunks of raw signal data for accurate analysis (Zhang et al. 2021), (ii) have high computational requirements (Senol Cali et al. 2019; Shih et al. 2023), and (iii) oblivious to the similarity information between raw signals that lead to redundant basecalling on these similar regions. Existing basecalling techniques and the lack of understanding of similarity between raw signals can impose limitations to enable (i) accurate real-time analysis (Zhang et al. 2021) and (ii) portable sequencing with constrained resources (Shih et al. 2023) and prevent future work on designing a new class of basecalling techniques that can utilize the similarity information between raw signals.

To fully utilize the unique benefits of nanopore sequencing, it is necessary to analyze raw signals with (i) high accuracy and low latency for adaptive sampling and (ii) low resource usage for portability and efficiency. To achieve this, several mechanisms focus on analyzing raw nanopore signals *without* basecalling (Loose et al. 2016, Gamaarachchi et al. 2020, Bao et al. 2021, Dunn et al. 2021, Kovaka et al. 2021, Samarasinghe et al. 2021, Zhang et al. 2021, Shih et al. 2023, Sadasivan et al. 2023, Senanayake et al. 2023, Firtina et al. 2023a, 2024, Lindegger et al. 2024, Sadasivan et al. 2024, Shivakumar et al. 2024, Kovaka et al. 2025) by mapping these signals to a reference genome (We use the raw signal analysis term specifically for these mechanisms in the remainder of the paper). Although raw nanopore signal mapping is widely studied (Loose et al. 2016, Gamaarachchi et al. 2020, Dunn et al. 2021, Kovaka et al. 2021, Samarasinghe et al. 2021, Zhang et al. 2021, Shih et al. 2023, Firtina et al. 2023a, 2024, Lindegger et al. 2024, Sadasivan et al. 2024, Shivakumar et al. 2024), *none* of these works *without* a reference genome to identify similarities directly between reads, called *all-vs-all overlapping* (Li 2016, Senol Cali et al. 2019, Firtina et al. 2023b).

Although identifying all-vs-all overlapping between raw nanopore signals can be promising to enable *new* directions for *both* basecalling mechanisms (as we discuss in Section 4) and raw signal analysis (e.g. *de novo* assembly construction as we show in Section 3.4), it is challenging to do so for several reasons (Smeets et al. 2008, Kawano et al. 2009, Bhattacharya et al. 2012, Deamer et al. 2016). First, similarities between a pair of signals must be identified accurately when *both* raw signals are noisy as compared to identifying similarities between a noisy raw signal and an accurate signal generated from a reference genome (Zhang et al. 2021, Firtina et al. 2023a, 2024, Lindegger et al. 2024). 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 (Deamer et al. 2016)], variable speed of DNA molecules moving through nanopores (Kawano et al. 2009, Bhattacharya et al. 2012), and raw reads containing multiple split molecules), while raw signals are *not* free from such noise.

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 (Kovaka et al. 2021). 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 (Li 2016).

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 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 4).

Our goal is to enable (i) raw signal analysis *without* a reference genome and (ii) new use cases with raw nanopore signal analysis 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 (Firtina et al. 2023a, 2024) 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* (Zhang et al. 2021, Firtina et al. 2023a; Shivakumar et al. 2024). Although similar filtering strategies (Zhang et al. 2021, Firtina et al. 2023a; Shivakumar et al. 2024) 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 (Firtina et al. 2024). 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.

Rawsamble makes the following key contributions:

- We propose the *first* mechanism that can find all-vs-all overlapping of raw signals without basecalling to enable new use cases for raw signal analysis, such as *assembly from overlapping* raw signals and improving existing basecalling mechanisms.
- We show that identifying overlaps with Rawsamble (i) is faster (on average 5.01× and up to 23.10×) and (ii) reduces peak memory requirements (on average 5.74× and up to 22.00×) compared to the computational pipeline that includes the state-of-the-art basecalling running on a CPU with its fastest model followed by minimap2 to find overlaps. We evaluate the trade-offs between speed, accuracy, and hardware resources by running the basecaller on a CPU and GPU with fast and high-accuracy models.
- We show that 27.09% of overlapping pairs that Rawsamble generates are identical to the overlapping pairs generated by minimap2.
- We report the first *de novo* assembly graphs ever constructed directly from raw signal overlaps without basecalling. We show that we can construct long unitigs up to 2.3 million bases in length for *E. coli*, which constitutes half the length of the genome. We find that these assemblies constructed from raw signal overlaps are either as accurate or more accurate than those created by minimap2 overlaps.
- We identify new use cases that can be enabled by using overlapping raw signals and constructing assemblies from them and discuss the advantages of generating them, such as for improving the accuracy and performance of basecalling. We discuss current limitations and challenges to enable these new use cases as future work.

2 Methods

2.1 Overview

Rawsamble is a mechanism to find overlapping pairs between raw nanopore signals (i.e. *all-vs-all overlapping*), which can be used by existing assemblers to construct *de novo* assembly graphs without basecalling, as shown in Fig. 1. To achieve this, Rawsamble builds on the state-of-the-art raw signal mapper, RawHash2 (Firtina *et al.* 2023a, 2024). We provide the overview of RawHash2 in Supplementary Section A, where we explain the details related to processing raw signals such as generating *events* after the *segmentation* and generating hash values from these segmented raw signals (i.e. events). In the remainder of the paper, we use *raw signals* to refer to these events.

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 (i), Rawsamble aggressively filters the raw input after the signal-to-event conversion to avoid nanopore-related errors. Second, to improve the accuracy of overlapping (ii), 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

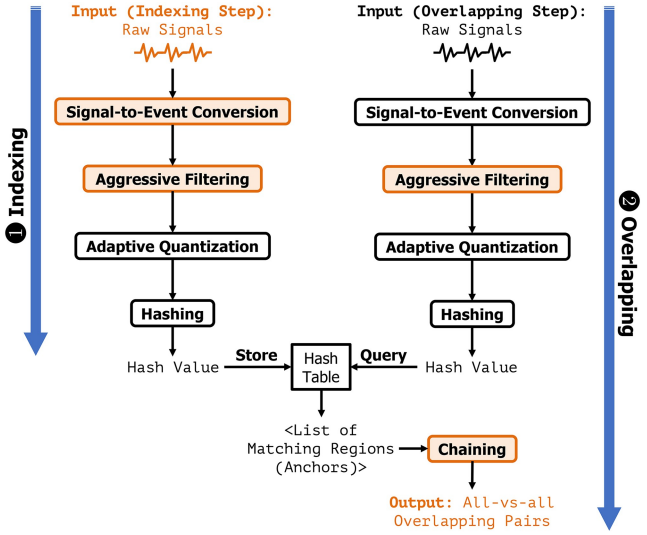

Figure 1 Overview of Rawsamble. We use colors for the inputs, steps, and outputs to highlight the parts that Rawsamble modifies over RawHash2.

adjusts the output strategy such that (i) *all* chains rather than only the best chain are reported and (ii) cyclic overlaps are avoided. Fourth, Rawsamble enables the use of existing *de novo* assemblers off-the-shelf, such as miniasm (Li, 2016), by providing the overlap information in a standardized format these assemblers use.

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 is mainly free from certain types of noise that raw nanopore signals contain [e.g. stochastic signal fluctuations (Deamer *et al.* 2016) and variable speed of DNA molecules moving through nanopores (Kawano *et al.* 2009, Bhattacharya *et al.* 2012)], 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 Fig. 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 (Zhang *et al.* 2021, Firtina *et al.* 2023a, 2024) to reduce the stay errors [e.g. by aiming to perform homopolymer compression of raw signals (Shivakumar *et al.* 2024)], Rawsamble employs a substantially larger threshold (i.e. aggressive) for filtering to minimize noise

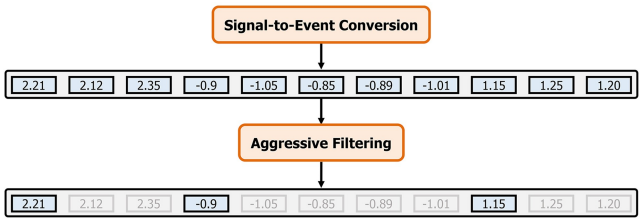

Figure 2 Filtering in Rawsamble. Values in gray boxes show the filtered signals as their values are close to the previous signal (in a blue box) that is not filtered out.

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.

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 (Li 2018, Firtina *et al.* 2024) 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.

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 the overlapping of 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 (Simpson *et al.* 2009, Sović *et al.* 2013, Li 2016). To avoid these trivial cyclic overlaps, Rawsamble implements 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.

Output Format. Rawsamble provides the overlapping information between raw signals using the Pairwise mApping Format (PAF) (Li 2016). To construct an assembly graph from the overlapping information that Rawsamble generates, any assembler that takes PAF files as input can be used, including miniasm (Li 2016).

3 Results

3.1 Evaluation methodology

We implement the improvements we propose in Rawsamble on the RawHash2 implementation (Firtina *et al.* 2024). Similar to RawHash2 and minimap2 (Li 2018), Rawsamble provides the mapping information in the standard Pairwise mApping Format (PAF) (Li 2016). We basecall the signals on two different hardware setups using Dorado's (i) high accuracy (HAC) model with an NVIDIA RTX A6000 GPU, and (ii) fast (Fast) and HAC models with an Intel Xeon Gold 6226R CPU. We include Dorado's results on a CPU to provide a fair comparison on the same hardware platform, as providing the GPU implementation for Rawsamble is a future work. We use POD5 files for all datasets, as suggested by Dorado 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 1. Although the R9.4.1 nanopore chemistry is deprecated, these raw datasets are commonly stored and basecalled for downstream analysis. We include datasets from both chemistries to show the feasibility and limitations of Rawsamble for each chemistry. For the R10.4.1 dataset, we use only the raw signals that contain a single molecule (i.e. non-chimeric multiple molecules can be contained in a single nanopore raw signal and later split by basecallers into multiple reads).

We evaluate computational requirements in terms of overall run time with 64 CPU threads, peak memory usage, and the estimated cloud computing costs to execute each tool. We use 64 CPU threads since the average thread utilization of CPU-based basecalling is around 64. We report the elapsed time, CPU time (when using only CPUs without GPUs), and peak memory usage of (i) Rawsamble and (ii) basecalling followed by minimap2 (`Dorado + Minimap2`). 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: (i) read overlapping and (ii) *de novo* assembly by constructing assembly graphs. We generate read overlaps from (i) raw signals using Rawsamble and (ii) basecalled sequences (with Dorado's HAC model) of corresponding strands of raw signals using minimap2. To evaluate the accuracy of all-vs-all overlapping, we calculate

Table 1 Details of datasets used in our evaluation.

	Organism	Chemistry Model	Reads (#)	Bases (#)	Avg. Read Length	Estimated Coverage (×)	SRA or DOI
D1	*E. coli*	R9.4	353,956	2,360M	6,668	451×	ERR9127551
D2	*Yeast*	R9.4	49,992	380M	7,609	31×	SRR8648503
D3	*Green Algae*	R9.4	30,012	622M	20,728	5.6×	ERR3237140
D4	*Human*	R9.4	270,012	1,776M	6,579	0.6×	FAB42260
D5	*E. coli*	R10.4.1	6,412,132	9,400M	1,466	1,797×	10.26188/25521892

Base counts in millions (M). Coverage is estimated using corresponding reference genomes.
All of the datasets are basecalled using Dorado (HAC).

the ratio of overlapping pairs (i) shared by both Rawsamble and minimap2, (ii) unique to Rawsamble, and (iii) unique to minimap2. For *de novo* assembly, we use off-the-shelf miniasm (Li 2016) to generate assembly graphs from the read overlaps that Rawsamble and minimap2 generate. We use miniasm because it enables us to evaluate the impact of overlaps that Rawsamble and minimap2 find 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 (Kolmogorov *et al.* 2019) to construct assemblies from these reads, as suggested in the guidelines by ONT. 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 contiguity of assemblies constructed from the overlaps that Rawsamble and minimap2 generate, we use several metrics. These metrics are (i) the total length of unitigs (Total Length), (ii) the number of nucleotides in the largest assembly graph component (Largest Comp.), (iii) unitig length at which 50% of the assembly is covered by unitigs of equal or greater length (N50), (iv) area under the Nx curve to generate a more robust evaluation of contiguity (auN), (v) longest unitig length, and (vi) the number of unitigs. We *estimate* the basecalled read length of raw signals based on the sample frequency (i.e. the number of signals produced every second) and the translocation speed (i.e. the number of bases estimated to pass through a nanopore in a single second). Estimating the read length based on these parameters is a common and relatively consistent approach widely adopted in previous related works (Kovaka *et al.* 2021, Zhang *et al.* 2021, Firtina *et al.* 2023a, 2024). We use Bandage (Wick *et al.* 2015) to visualize these assemblies. We note that our contiguity evaluations for Rawsamble and minimap2 are based on the assembly graphs constructed by miniasm.

To evaluate the accuracy of the constructed graphs from Rawsamble and minimap2 overlaps, we perform read mapping from their corresponding basecalled reads to their corresponding reference genomes. For each unitig generated in a graph, we measure how well its reads line up on the reference genome in four steps. First, we list the reads of the unitig in the exact order given by the assembly graph. Second, for each read, we collect every alignment of that read to the reference genome and create an anchor $\langle i, s \rangle$, where i is the ordinal position of the read in the unitig and s is the reference start coordinate. Third, we find the longest strictly increasing sequence of anchors that satisfies two constraints: (i) all anchors in the chain must map to the same reference chromosome and in the same left-to-right order, and (ii) the start position of anchor $j+1$ must lie in the interval $[s_j, s_j + \text{readlen}_j + \delta)$, where δ bp is a tolerance threshold for the range. Our dynamic programming implementation on this chaining algorithm provides the number of reads in the best chain, k, where the goal of the best chain is to maximize k. Fourth, we record the ratio $R = k/n$ where n is the total number of reads in the unitig (unitigs containing no more than two reads are skipped). Summing k and n over all multi-read unitigs gives overall totals all_k and all_n. We estimate the accuracy of the assembly graph as $100 \times \text{all\_k}/\text{all\_n}$, i.e. the percentage of all reads within the same unitig that can be placed in a consistent, chromosome-specific, nearly contiguous order with respect to the reference genome, which we report as `Chained Read Percentage`. The goal of generating these percentages is to provide our best attempt for a *meaningful comparison* between assembly graphs in terms of their *estimated* accuracy. We cannot align the assembled raw signals in an assembly graph to a reference genome to measure their actual identity, since assembled raw signals cannot be reported. We leave (i) constructing the assembled signal sequence (i.e. spelling the assembly from the graph), (ii) designing a consensus mechanism for error correction, and (iii) evaluating the accuracy of these assembled signals as future work, which we discuss in more detail in Section 4.

We provide the parameter settings and versions for each tool as well as the details of the preset parameters in Rawsamble in Supplementary Tables S3 (parameters), S4 (details of presets), and S5 (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 S5. We provide the detailed reasoning behind the choice of Flye in order to generate the gold standard in Supplementary Material C.

3.2 Performance and memory

3.2.1 Throughput

Table 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 (i) provide better scalability (i.e. analyzing a larger amount of data with the same computation capabilities) and (ii) reduce the overall computational requirements and corresponding energy consumption (i.e.

Table 2 Rawsamble throughput (signals processed per second per CPU thread).

	D1 *E. coli*	D2 *Yeast*	D3 *Green Algae*	D4 *Human*	D5 *E. coli* (R10.4.1)
Throughput	2,171,615	1,304,910	1,117,183	1,295,240	1,776,225

Table 3 Comparison of various tools across different organisms in terms of elapsed time, CPU time, and peak memory usage.

Organism	Tool	Elapsed time (hh: mm: ss)	CPU time (sec)	Peak Mem. (GB)
D1	Rawsamble	0:52:07	184,938	7.41
E. coli	Minimap2 + Dorado CPU (Fast)	2:56:45 (3.39×)	593,764 (3.21×)	36.73 (4.96×)
	Minimap2 + Dorado CPU (HAC)	13:31:19 (15.57×)	1,408,712 (7.62×)	62.34 (8.41×)
	Minimap2 + Dorado GPU (HAC)	0:26:48 (0.51×)	NA	26.73 (3.61×)
D2	Rawsamble	0:01:21	3,753	6.68
Yeast	Minimap2 + Dorado CPU (Fast)	0:31:11 (23.10×)	98,558 (26.26×)	146.95 (22.00×)
	Minimap2 + Dorado CPU (HAC)	2:30:28 (111.46×)	548,495 (146.15×)	290.85 (43.54×)
	Minimap2 + Dorado GPU (HAC)	0:02:00 (1.48×)	NA	5.98 (0.90×)
D3	Rawsamble	0:06:20	21,769	10.87
Green Algae	Minimap2 + Dorado CPU (Fast)	0:30:17 (4.78×)	103,102 (4.74×)	38.18 (3.51×)
	Minimap2 + Dorado CPU (HAC)	7:29:20 (70.95×)	178,106 (8.18×)	20.76 (1.91×)
	Minimap2 + Dorado GPU (HAC)	0:03:04 (0.48×)	NA	10.08 (0.93×)
D4	Rawsamble	0:45:08	136,966	6.39
Human	Minimap2 + Dorado CPU (Fast)	5:54:27 (7.85×)	1,284,069 (9.38×)	113.14 (17.71×)
	Minimap2 + Dorado CPU (HAC)	18:45:18 (24.93×)	4,226,869 (30.86×)	130.87 (20.48×)
	Minimap2 + Dorado GPU (HAC)	0:13:48 (0.31×)	NA	17.16 (2.69×)
D5	Rawsamble	16:04:39	2,672,608	112.31
E. coli	Minimap2 + Dorado CPU (Fast)	17:16:52 (1.07×)	3,478,136 (1.30×)	103.1 (0.92×)
(R10.4.1)	Minimap2 + Dorado CPU (HAC)	135:05:47 (8.40×)	19,051,635 (7.13×)	103.1 (0.92×)
	Minimap2 + Dorado GPU (HAC)	4:59:49 (0.31×)	NA	103.1 (0.92×)

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 tools provide a better and worse results with colors.

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 5000 signals per second, 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) (Magi *et al.* 2018). We find that Rawsamble provides an average throughput of 1,533,035 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.

3.2.2 Computational resources

Table 3 shows the computational resources, and Supplementary Table S1 shows the estimated cloud computing costs to execute each tool for various datasets. Inside the parentheses provided with `Dorado + Minimap2` results, we provide the ratio between the reported result and the corresponding Rawsamble result (if higher than 1×, Rawsamble is better).

We make three key observations. First, Rawsamble provides a substantial speedup and lower peak memory usage compared to Dorado's fast model on a CPU followed by minimap2 [`Dorado CPU (Fast) + Minimap2`] by, on average, 5.01× (elapsed time), 5.46× (CPU time), and 5.74× (peak memory usage). When using Dorado's HAC model on a CPU, Rawsamble provides even better results on average by 30.36× (elapsed time), 18.21× (CPU time), and 6.67× (peak memory usage), since running the HAC model is computationally more costly than running the fast model.

Second, GPU-based basecalling followed by minimap2 takes 0.51× of the time that Rawsamble takes while requiring higher peak memory usage by 1.49× compared to Rawsamble. 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 performance given the limited parallelism it uses on CPUs (i.e. 64 threads) compared to the massive 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 based on its comparison when basecalling is done using CPUs and GPUs, which we leave as future work.

Third, in most cases where Rawsamble is slower than GPU-based configurations, Rawsamble still provides a cheaper solution when using cloud computing services. This is mainly because GPU usage in the cloud is associated with substantially

higher monetary costs than using a CPU-only solution under today's pricing scheme. We find that Rawsamble provides the cheapest solution for identifying overlaps in most cases, even when it is slower than GPU-based configurations.

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 4.

3.3 All-vs-all overlapping statistics

Table 4 shows the all-vs-all overlapping statistics between Rawsamble and minimap2. We make two key observations. First, on average, 27.09% of the 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, there are still overlapping pairs unique to Rawsamble (12.87% on average) and minimap2 (60.04% on average). These differences are likely due to differences in (i) certain parameters (e.g. chaining scores) and (ii) increased noise inherent in raw signals compared to basecalled sequences, which can become more pronounced in genomes with greater size, complexity, or repetitiveness. These factors overall result in fewer overlaps found by Rawsamble. Although certain parameters can be configured (e.g. minimum chaining scores) to find more overlaps, this strategy usually results in finding a substantially larger number of erroneous overlaps in Rawsamble. Designing robust mechanisms that can handle noise better and more accurate hashing mechanisms with fewer collisions can provide better trade-offs. Although maximizing the shared overlaps can provide insights regarding the accuracy of overlapping information that Rawsamble finds, it is not necessary to provide near-identical shared overlap statistics for certain use cases such as constructing *de novo* assemblies (Vaser and Šikić 2021). Instead, contiguous and more accurate assemblies can still be constructed using a smaller but useful portion of shared overlapping pairs (Vaser and Šikić 2021). We conclude that Rawsamble provides a mechanism that shares a large portion of overlaps with minimap2, while the shared portions decrease due to the differences in parameters and the noise in raw signal datasets.

3.4 Contiguity and estimated accuracy of the assembly graphs

To evaluate the impact of overlaps that Rawsamble and minimap2 find, we construct *de novo* assembly graphs from these overlaps by using miniasm off-the-shelf. Table 5 shows the contiguity statistics and estimated accuracy of these *de novo* assembly graphs (please see Section 3.1 for our explanation and discussion related to using these accuracy ratios *only* for comparison purposes, not for calculating the actual assembly identities). We note that the assembly graphs constructed using miniasm provide the connection information used to assemble unitigs, as well as the connections between unitigs that make up subgraph components in the assembly graph. Since we use the miniasm assembler without any modifications, we cannot generate the assembled sequences from raw signals as miniasm supports generating assembled sequences from basecalled sequencing reads. This mainly prevents us from evaluating the actual identity of these assemblies from raw signals, which we further discuss in Section 4.

First, we find that we can construct long unitigs from the raw signal overlaps that Rawsamble finds. The unitigs we can construct from these raw signal overlaps are substantially longer than the average read length of their corresponding datasets (e.g. the longest unitig length in D1 is 413.25× longer than the average read length of the D1 dataset). 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 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 5. We believe this is mainly in line with the all-vs-all overlapping statistics where minimap2 finds more overlaps than Rawsamble, leading to longer assemblies. 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 D1 dataset (*E.*

Table 4 All-vs-all overlapping statistics.

	Organism	Unique to Rawsamble (%)	Unique to Minimap2 (%)	Shared Overlaps (%)
D1	*E. coli*	8.33	50.62	41.05
D2	*Yeast*	26.54	28.62	44.84
D3	*Green Algae*	3.76	78.64	17.61
D4	*Human*	25.65	62.62	11.73
D5	*E. coli* (R10.4.1)	0.06	79.69	20.25

Percentages show the overlapping pairs that are (i) unique to Rawsamble, (ii) unique to minimap2, and (iii) reported in both tools (shared overlaps).

Table 5 Assembly statistics.

Dataset	Tool	Chained read Percentage (%)	Largest Comp. (bp)	N50 (bp)	auN (bp)	Longest Unitig (bp)	Unitig Count
D1	Rawsamble	34.0	4,841,669	1,535,079	1,187,229	2,347,310	32
E. coli	minimap2	24.3	5,207,206	5,204,754	5,194,495	5,207,206	4
	Gold standard	NA	5,235,343	5,235,343	5,235,343	5,235,343	1
D2	Rawsamble	43.5	362,050	41,118	48,106	161,883	396
Yeast	minimap2	43.2	1,611,876	134,050	137,172	64,054	282
	Gold standard	NA	11,835,059	640,934	623,210	1,073,346	68
D3	Rawsamble	41.3	448,422	93,111	107,914	252,038	50
Green Algae	minimap2	53.2	198,709	63,310	91,360	198,709	55
	Gold standard	NA	2,255,807	452,774	538,136	1,667,975	420
D4	Rawsamble	61.2	183,402	47,397	66,574	183,402	48
Human	minimap2	72.7	81,017	19,459	26,925	81,017	64
	Gold standard	NA	367,305	19,329	29,697	150,470	592
D5	Rawsamble	34.6	22,921,373	27,460	43,505	410,905	2,194
E. coli	minimap2	16.3	27,250,402	24,111	814,807	5,117,413	2,888
(R10.4.1)	Gold standard	NA	5,230,610	5,230,610	5,230,610	5,230,610	1

Chained read percentage provides our estimate in terms of assembly graph accuracy. Other columns provide our evaluations in terms of assembly contiguity. For the accuracy, we highlight the cells that tools provide a better and worse results with colors.

coli). For example, Rawsamble constructs unitigs with the longest unitig length of 2.3 million bases, which is almost 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, in terms of the accuracy estimate of the assembly graphs (i.e. *Chained Read Percentage*), we find that a larger percentage of *unitig reads* generated by the Rawsamble overlaps appear in close proximity when these reads are mapped to their corresponding reference genome compared to the unitig reads generated by the minimap2 overlaps, except for two cases (i.e. D3 and D4). Although this observation by itself does not show that the Rawsamble overlaps can lead to assemblies with higher identity, it provides the accuracy estimate that these assemblies share similar accuracy to those generated by the minimap2 overlaps. For the cases where Rawsamble provides a lower chained read percentage, Rawsamble leads to better contiguity. These datasets are from larger genomes with lower sequencing depth. We believe that spurious overlaps are more common at lower coverage when using raw signals, leading to contiguous but erroneous assemblies.

We conclude that Rawsamble generates useful overlapping information, enabling the construction of assemblies directly from raw signals. The assembly graphs generated by the Rawsamble overlaps mainly provide comparable contiguity and accuracy compared to those generated by minimap2. We discuss the potential next steps to enable evaluating the accuracy of these assemblies in Section 4.

4 Discussion and future work

4.1 Limitations

Our evaluation demonstrates that Rawsamble achieves high throughput when finding overlaps between raw signal pairs, making it a viable candidate for real-time analysis. 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 increase 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 (Myers 2005)] 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 (El-Metwally *et al.* 2016, Rozov *et al.* 2018, Scott 2022).

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 matches on the other strand. Rawsamble cannot detect reverse-complemented strands, since this requires modifying the original raw signal to generate its reverse-complemented version. Our preliminary results show that simple one-to-one mapping strategies are not practical, potentially due to noise. Lacking reverse complemented signals 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.

4.2. Challenges for constructing and evaluating *de novo* assemblies

Although the main focus of our work is enabling all-vs-all overlapping between raw signals, we identify *de novo* assembly construction as a natural use case that can utilize the all-vs-all overlap information from raw signals that Rawsamble finds. To show that the overlaps from Rawsamble can lead to unitigs that are much longer than average read lengths, we construct and evaluate the *de novo* assembly graphs from the Rawsamble overlaps, as discussed in Section 3.4. However, we identify two key challenges that limit the scope of our evaluation of these *de novo* assemblies, which we leave as future work.

First, we cannot generate the sequence of assembled signals from the assembly graphs (i.e. spelling the sequence from the assembly graph). This is because we use the miniasm assembler to construct these assembly graphs from the raw signal overlaps. Although miniasm can use the overlap position information to construct the assembly graph, it cannot output the sequence of assemblies, as it is mainly designed for basecalled sequences. This is challenging because each raw signal contains metadata used in basecalling mechanisms (e.g. nanopore-specific information such as the baseline current levels). Carrying the metadata information effectively from many overlapping signals is essential to enable further analysis of these assembled signals.

Second, to evaluate the identity of the constructed assemblies, these assemblies need to be aligned to a ground truth assembly. This can be done by either (i) basecalling the assembled signal or (ii) aligning the assembled signal using dynamic time warping (DTW) (Lindegger *et al.* 2024) to the ground truth assembly. These approaches require constructing the assembled signal (i.e. the first challenge mentioned above) and designing a new class of basecallers to directly basecall assembled signals, which we discuss as future work below. Although the missing identity evaluations of these assemblies can be a major limitation of our work, we believe the assemblies that are constructed from the Rawsamble overlaps are likely to be close to the assembly accuracy that can be achieved by using the minimap2 overlaps based on our estimated accuracy evaluation.

4.3 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 (e.g. basecalling the overlapped region only once or designing basecallers such that they collectively take overlapping region for more accurate basecalling). 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 (Li 2016). 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. Fourth, existing assemblies generated from basecalled sequences can leverage the raw signal information to resolve complex regions and perform phasing by utilizing the rich information in signals that may resolve certain repeats. Fifth, more accurate and robust segmentation approaches for signal processing (Bakić *et al.* 2026, Spangenberg *et al.* 2026) can lead to better overlapping and assembly construction, which requires further investigation for this particular use case.

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.

5 Conclusion

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 (i) find overlaps while meeting the real-time requirements with an average throughput of $1,533,035$ signals/sec per CPU thread, (ii) reduce the overall time needed for finding overlaps (on average by $5.01\times$ and up to by $23.10\times$) and peak memory usage (on average by $5.74\times$ and up to by $22.00\times$) 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, (iii) share a large portion of overlapping pairs with minimap2 (27.09% on average), and (iv) 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 2.3 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.

Funding

C.F. acknowledges gift funding provided by AMD. SAFARI Research Group acknowledges the generous gift funding provided by industrial partners (especially by Google, Intel, Microsoft, VMware), which has been instrumental in enabling the 20+ year-long research SAFARI Research Group has been

conducting on accelerating genome analysis. SAFARI Research Group is also partially supported by the Semiconductor Research Corporation (SRC), the European Union's Horizon programme for research and innovation [101047160 – BioPIM] and the Swiss National Science Foundation (SNSF) [200021_213084]. M.M. has been supported by the Max Planck ETH Center for Learning Systems and by SNSF Project Grant #200550 to A.K. S. G. was funded through SNSF Project Grant #200550 to A.K.

Supplementary material

Supplementary material is available at *Bioinformatics* online.

Author contributions

Can Firtina (Conceptualization [Lead], Data curation [Lead], Formal analysis [Lead], Funding acquisition [Equal], Investigation [Equal], Methodology [Lead], Project administration [Lead], Software [Lead], Validation [Lead], Visualization [Lead], Writing—original draft [Lead], Writing—review & editing [Lead]), Maximilian Mordig (Conceptualization [Supporting], Methodology [Supporting], Software [Supporting], Writing—original draft [Supporting], Writing—review & editing [Supporting]), Harun Mustafa (Data curation [Supporting], Methodology [Supporting], Writing—original draft [Supporting], Writing—review & editing [Supporting]), Sayan Goswami (Methodology [Supporting], Writing—original draft [Supporting], Writing—review & editing [Supporting]), Nika Mansouri Ghiasi (Methodology [Supporting], Writing—original draft [Supporting], Writing—review & editing [Supporting]), Stefano Mercogliano (Software [Supporting], Writing—original draft [Supporting]), Furkan Eris (Formal analysis [Supporting], Software [Supporting], Visualization [Supporting], Writing—review & editing [Supporting]), Joël Lindegger (Methodology [Supporting], Writing—original draft [Supporting]), Andre Kahles (Funding acquisition [Equal], Investigation [Equal], Project administration [Equal], Resources [Equal], Supervision [Equal], Writing—original draft [Supporting], Writing—review & editing [Supporting]), and Onur Mutlu (Conceptualization [Equal], Funding acquisition [Lead], Investigation [Lead], Project administration [Equal], Resources [Equal], Supervision [Lead], Writing—original draft [Equal], Writing—review & editing [Equal])

Conflict of interests

None declared.

Data availability

Source code, scripts to reproduce the results, and scripts to download the datasets are available at https://github.com/CMU-SAFARI/RawHash. Scripts for additional datasets are available at https://github.com/STORMgroup/RawHash2.

References

Bakić S, Friganović K, Hooi B *et al.* Campolina: a deep neural framework for accurate segmentation of nanopore signals. *Genome Biol* 2026;**27**:46. https://doi.org/10.1186/s13059-026-03950-1.

Bao Y, Wadden J, Erb-Downward JR *et al.* SquiggleNet: real-time, direct classification of nanopore signals. *Genome Biol* 2021;**22**:298.

Bhattacharya S, Derrington IM, Pavlenok M *et al.* Molecular dynamics study of MspA arginine mutants predicts slow DNA translocations and ion current blockades indicative of DNA sequence. *ACS Nano* 2012;**6**:6960–8. https://doi.org/10.1021/nn3019943

Bloemen B, Gand M, Vanneste K *et al.* Development of a portable on-site applicable metagenomic data generation workflow for enhanced pathogen and antimicrobial resistance surveillance. *Sci Rep* 2023;**13**:19656.

Boža V, Brejová B, Vinař T. DeepNano: deep recurrent neural networks for base calling in MinION nanopore reads. *PLoS One* 2017;**12**:e0178751.

Boža V, Perešíni P, Brejová B *et al.* DeepNano-blitz: a fast base caller for MinION nanopore sequencers. *Bioinformatics* 2020;**36**:4191–2.

Cavlak MB, Singh G, Alser M *et al.* TargetCall: eliminating the wasted computation in basecalling via pre-basecalling filtering. *Front Genet* 2024;**15**:1429306.

David M, Dursi LJ, Yao D *et al.* Nanocall: an open source basecaller for oxford nanopore sequencing data. *Bioinformatics* 2017;**33**:49–55.

Deamer D, Akeson M, Branton D. Three decades of nanopore sequencing. *Nat Biotechnol* 2016;**34**:518–24. https://doi.org/10.1038/nbt.3423

Dunn T, Sadasivan H, Wadden J *et al.* SquiggleFilter: an accelerator for portable virus detection. In: *MICRO*, 2021.

El-Metwally S, Zakaria M, Hamza T. LightAssembler: fast and memory-efficient assembly algorithm for high-throughput sequencing reads. *Bioinformatics* 2016;**32**:3215–3223.

Eris F, McConnell U, Firtina C *et al.* RawBench: a comprehensive benchmarking framework for raw nanopore signal analysis techniques. In: *ACM-BCB*, New York, NY, 2025. https://doi.org/10.1145/3765612.3767302

Firtina C, Mansouri Ghiasi N, Lindegger J *et al.* RawHash: enabling fast and accurate real-time analysis of raw nanopore signals for large genomes. *Bioinformatics* 2023a;**39**:i297–307.

Firtina C, Park J, Alser M *et al.* BLEND: a fast, memory-efficient and accurate mechanism to find fuzzy seed matches in genome analysis. *NAR Genom Bioinform* 2023b;**5**:lqad004.

Firtina C, Soysal M, Lindegger J *et al.* RawHash2: mapping raw nanopore signals using hash-based seeding and adaptive quantization. *Bioinformatics* 2024;**40**:btae478.

Gamaarachchi H, Lam CW, Jayatilaka G *et al.* GPU accelerated adaptive banded event alignment for rapid comparative nanopore signal analysis. *BMC Bioinform* 2020;**21**:343. https://doi.org/10.1186/s12859-020-03697-x

Huang N, Nie F, Ni P *et al.* SACall: a neural network basecaller for oxford nanopore sequencing data based on self-attention mechanism. *IEEE/ACM Trans Comput Biol Bioinform* 2022;**19**:614–23. https://doi.org/10.1109/TCBB.2020.3039244

Sović I, Skala K, Šikić M. Approaches to DNA de novo assembly. In: *MIPRO*, 2013.

Jain M, Koren S, Miga KH *et al.* Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nat Biotechnol* 2018;**36**:338–45.

Kawano R, Schibel AEP, Cauley C *et al.* Controlling the translocation of single-stranded DNA through a-hemolysin ion channels

using viscosity. *Langmuir* 2009;**25**:1233–7. https://doi.org/10.1021/la803556p

Kolmogorov M, Yuan J, Lin Y *et al.* Assembly of long, error-prone reads using repeat graphs. *Nat Biotechnol* 2019;**37**:540–6. https://doi.org/10.1038/s41587-019-0072-8

Konishi H, Yamaguchi R, Yamaguchi K *et al.* Halcyon: an accurate basecaller exploiting an encoder–decoder model with monotonic attention. *Bioinformatics* 2021;**37**:1211–7. https://doi.org/10.1093/bioinformatics/btaa953

Kovaka S, Fan Y, Ni B *et al.* Targeted nanopore sequencing by real-time mapping of raw electrical signal with UNCALLED. *Nat Biotechnol* 2021;**39**:431–41.

Kovaka S, Hook PW, Jenike KM *et al.* Uncalled4 improves nanopore DNA and RNA modification detection via fast and accurate signal alignment. *Nat Methods* 2025;**22**:681–91. https://doi.org/10.1038/s41592-025-02631-4.

Li H. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics* 2016;**32**:2103–10.

Li H. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* 2018;**34**:3094–100.

Lindegger J, Firtina C, Ghiasi NM *et al.* RawAlign: accurate, fast, and scalable raw nanopore signal mapping via combining seeding and alignment. *IEEE Access* 2024;**12**:196855–65.

Loman NJ, Quick J, Simpson JT. A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nat Methods* 2015;**12**:733–5.

Loose M, Malla S, Stout M. Real-time selective sequencing using nanopore technology. *Nat Methods* 2016;**13**:751–4.

Lv X, Chen Z, Lu Y *et al.* An end-to-end Oxford nanopore basecaller using convolution-augmented transformer. In: *BIBM*, 2020.

Magi A, Semeraro R, Mingrino A *et al.* Nanopore sequencing data analysis: state of the art, applications and challenges. *Brief Bioinform* 2018;**19**:1256–72.

Miculinic N, Ratkovic M, Sikic M. MinCall—MinION end2end convolutional deep learning basecaller. arXiv, 2019, preprint: not peer reviewed.

Myers EW. The fragment assembly string graph. *Bioinformatics* 2005;**21** Suppl 2:ii79–85.

Noordijk B, Nijland R, Carrion VJ *et al.* baseLess: lightweight detection of sequences in raw MinION data. *Bioinform Adv* 2023;**3**:vbad017.

Perešíni P, Boža V, Brejová B *et al.* Nanopore base calling on the edge. *Bioinformatics* 2021;**37**:4661–7.

Pugh J. The current state of nanopore sequencing. In: Arakawa K (ed.), *Nanopore Sequencing: Methods and Protocols*. New York, NY: Springer US, 2023, 3–14.

Rozov R, Goldshlager G, Halperin E *et al.* Faucet: streaming de novo assembly graph construction. *Bioinformatics* 2018;**34**:147–154.

Sadasivan H, Wadden J, Goliya K *et al.* Rapid real-time squiggle classification for read until using RawMap. *Arch Clin Biomed Res* 2023;**7**:45–57.

Sadasivan H, Stiffler D, Tirumala A *et al.* Accelerated dynamic time warping on GPU for selective nanopore sequencing. *J Biotechnol Biomed* 2024;**07**:137–48.

Samarasinghe S, Premathilaka P, Herath W *et al.* Energy efficient adaptive banded event alignment using OpenCL on FPGAs. In: *ICIAfS*, pp.369–374, 2021. https://doi.org/10.1109/ICIAfS52090.2021.9606056

Schreiber J, Karplus K. Analysis of nanopore data using hidden markov models. *Bioinformatics* 2015;**31**:1897–903. https://doi.org/10.1093/bioinformatics/btv046

Scott C. Streaming methods for assembly graph analysis. PhD, University of California, Davis, 2022.

Senanayake A, Gamaarachchi H, Herath D *et al.* DeepSelectNet: deep neural network based selective sequencing for oxford nanopore sequencing. *BMC Bioinformatics* 2023;**24**:31.

Senol Cali D, Kim JS, Ghose S *et al.* Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions. *Brief Bioinform* 2019;**20**:1542–59.

Shih PJ, Saadat H, Parameswaran S *et al.* Efficient real-time selective genome sequencing on resource-constrained devices. *GigaScience* 2023;**12**:giad046.

Shivakumar VS, Ahmed OY, Kovaka S *et al.* Sigmoni: classification of nanopore signal with a compressed pangenome index. *Bioinformatics* 2024;**40**:i287–96.

Simpson JT, Wong K, Jackman SD *et al.* ABySS: a parallel assembler for short read sequence data. *Genome Res* 2009;**19**:1117–23.

Singh G, Alser M, Denolf K *et al.* RUBICON: a framework for designing efficient deep learning-based genomic basecallers. *Genome Biol* 2024;**25**:49.

Smeets RMM, Keyser UF, Dekker NH *et al.* Noise in solid-state nanopores. *Proc Natl Acad Sci USA* 2008;**105**:417–21. https://doi.org/10.1073/pnas.0705349105

Spangenberg J, Siederdissen CHz, Goettsch W *et al.* Dynamont: a comprehensive cross-species comparison of ONT segmentation tools. *GigaScience* 2026;giag005. https://doi.org/10.1093/gigascience/giag005

Teng H, Cao MD, Hall MB *et al.* Chiron: translating nanopore raw signal directly into nucleotide sequence using deep learning. *GigaScience* 2018;**7**:giy037. https://doi.org/10.1093/gigascience/giy037

Timp W, Comer J, Aksimentiev A. DNA base-calling from a nanopore using a Viterbi algorithm. *Biophys J* 2012;**102**:L37–9.

Vaser R, Šikić M. Time- and memory-efficient genome assembly with Raven. *Nat Comput Sci* 2021;**1**:332–6. https://doi.org/10.1038/s43588-021-00073-4

Wick RR, Schultz MB, Zobel J *et al.* Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics* 2015;**31**:3350–2.

Xu X, Bhalla N, Ståhl P *et al.* Lokatt: a hybrid DNA nanopore basecaller with an explicit duration hidden markov model and a residual LSTM network. *BMC Bioinform* 2023;**24**:461. https://doi.org/10.1186/s12859-023-05580-x

Xu Z, Mai Y, Liu D *et al.* Fast-bonito: a faster deep learning based basecaller for nanopore sequencing. *Artif Intell Life Sci* 2021;**1**:100011.

Yeh Y-M, Lu Y-C. MSRCall: a multi-scale deep neural network to basecall oxford nanopore sequences. *Bioinformatics* 2022;**38**:3877–84. https://doi.org/10.1093/bioinformatics/btac435

Zeng J, Cai H, Peng H *et al.* Causalcall: nanopore basecalling using a temporal convolutional network. *Front Genet* 2019;**10**:1332–8021.

Zhang H, Li H, Jain C *et al.* Real-time mapping of nanopore raw signals. *Bioinformatics* 2021;**37**:i477–83.

Zhang Y-Z, Akdemir A, Tremmel G *et al.* Nanopore basecalling from a perspective of instance segmentation. *BMC Bioinform* 2020;**21**:136. https://doi.org/10.1186/s12859-020-3459-0