Published online 20 January 2023

NAR Genomics and Bioinformatics, 2023, Vol. 5, No. 1 **1**
https://doi.org/10.1093/nargab/lqad004

BLEND: a fast, memory-efficient and accurate mechanism to find fuzzy seed matches in genome analysis

Can Firtina [1,*], Jisung Park [1,2], Mohammed Alser [1], Jeremie S. Kim [1], Damla Senol Cali [3], Taha Shahroodi [4], Nika Mansouri Ghiasi [1], Gagandeep Singh [1], Konstantinos Kanellopoulos [1], Can Alkan [5] and Onur Mutlu [1,*]

[1]ETH Zurich, Zurich 8092, Switzerland, [2]POSTECH, Pohang 37673, Republic of Korea, [3]Carnegie Mellon University, Pittsburgh, PA 15213, USA, [4]TU Delft, 2600 AA Delft, Netherlands and [5]Bilkent University, Ankara 06800, Turkey

Received July 29, 2022; Revised December 16, 2022; Editorial Decision December 24, 2022; Accepted January 10, 2023

ABSTRACT

Generating the hash values of short subsequences, called 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 exact-matching seeds as the conventional hashing methods assign distinct hash values for different seeds, including highly similar seeds. Finding only exact-matching seeds causes either (i) increasing the use of the costly sequence alignment or (ii) limited sensitivity. We introduce *BLEND*, the first efficient and accurate mechanism that can identify *both* exact-matching and highly similar seeds with a single lookup of their hash values, called fuzzy seed matches. BLEND (i) utilizes a technique called SimHash, that can generate the same hash value for similar sets, and (ii) provides the proper mechanisms for using seeds as sets with the SimHash technique to find fuzzy seed matches efficiently. We show the benefits of BLEND when 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. Source code is available at https://github.com/CMU-SAFARI/BLEND.

INTRODUCTION

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 (1). Although HTS technologies are key enablers of almost *all* genomics studies (2–7), HTS technology-provided data comes with two key shortcomings. First, HTS technologies sequence short fragments of genome sequences. These short fragments are called *reads*, which cover only a smaller region of a genome and contain from about one hundred up to a million bases depending on the technology (1). Second, HTS technologies can misinterpret signals during sequencing and thus provide reads that contain *sequencing errors* (8). The average frequency of sequencing errors in a read highly varies from 0.1% up to 15% depending on the HTS technology (9–13). 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 (i) read mapping (14–18), (ii) *de novo* assembly (19–21), (iii) read classification in metagenomic studies (22–24), (iv) correcting sequencing errors (25–27).

At the core of these computational approaches, similarities between sequences must be identified to overcome the fundamental 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 (28,29). To practically identify similarities, it is essential to avoid calculating the distance between dissimilar sequence pairs. A common heuristic is to find matching *short* subsequences, called *seeds*, between sequence pairs by using a hash table (14,15,30–54). Sequences that have no or few seed matches are quickly filtered out from performing costly sequence alignment. There are several techniques that

*To whom correspondence should be addressed. Tel: +41 44 632 64 29; Email: firtinac@ethz.ch
Correspondence may also be addressed to Onur Mutlu. Tel: +41 44 632 64 29; Email: omutlu@ethz.ch

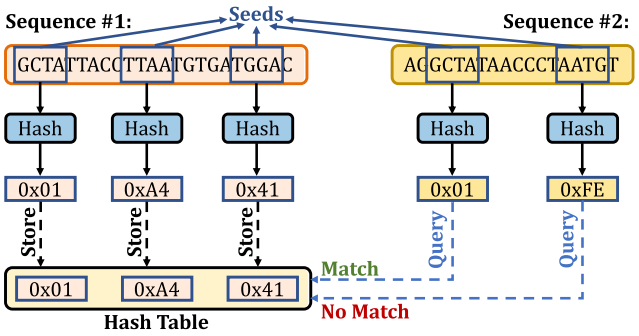

Figure 1. Finding seed matches with a single lookup of hash values.

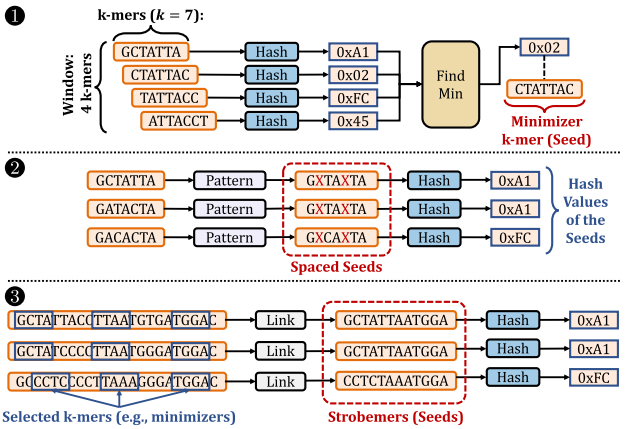

Figure 2. Examples of common seeding techniques. 1. Finding the minimizer *k*-mers. 2. A spaced seeding technique. Masked characters are highlighted by X in red. 3. A simple example of the strobemers technique.

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. Figure 1 shows an overview of how hash tables are used to find seed matches between two sequences. Seeds in Figure 1 are extracted from sequences based on a seeding technique. These seeds are used to find matches between sequences. To find seed matches, the hash values of seeds are used for filling and querying the hash table, as shown in Figure 1. Querying the hash table with hash values enables finding the positions where a seed from the second sequence appears in the first sequence with a single lookup. The use of seeds drastically reduces the search space from all possible sequence pairs to the similar sequence pairs to facilitate efficient distance calculations over many sequence pairs (55–57).

Figure 2 shows the three main directions that existing seeding techniques take. The first direction aims to minimize the computational overhead of using and storing seeds by selectively choosing fewer seeds from all fixed-length subsequences of reads, called *k-mers*, where the fixed length is *k*. The existing works such as minimap2 (15), MHAP (58), Winnowmap2 (59,60), re$M_u$val (61) and CAS (62) 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 only the *k*-mers with the *minimum* hash value in a window of *w* consecutive *k*-mers, known as the *minimizerk*-mers (56) (1 in Figure 2). Such a sampling approach guarantees that one *k*-mer is sampled in each window to provide a fixed sampling ratio that can be tuned to increase the probability of matching *k*-mers between reads. Alternatively, MHAP uses the MinHash technique (63) to generate many hash values from each *k*-mer of a read using many hash functions. For each hash function, only the *k*-mer with the minimum hash value is used as a seed with no windowing guarantees. MHAP is mainly effective for matching sequences with similar lengths since the number of hash functions is fixed for all sequences, whereas it can generate too many seeds for shorter sequences when the sequence lengths vary greatly (14). While these *k*-mer selection approaches reduce the number of seeds to use, all of these existing works find *only* exact-matching k-mers with a single lookup, as they use hash functions with *low-collision* rates to generate the hash values of these *k*-mers. The exact-matching requirement imposes challenges when determining the *k*-mer length. Longer *k*-mer lengths significantly decrease the probability of finding exact-matching *k*-mers between sequences due to genetic variations and sequencing errors. Short *k*-mer lengths (e.g., 8–21 bp) result in matching a large number of k-mers due to both the repetitive nature of most genomes and the high probability of finding the same short k-mer frequently in a long sequence of DNA letters (64). Although *k*-mers are commonly used as seeds, a seed is a more general concept that can allow substitutions, insertions and deletions (indels) when matching short subsequences between sequence pairs.

The second direction is to allow substitutions when matching *k*-mers by *masking* (i.e., ignoring) certain characters of *k*-mers and using the masked *k*-mers as seeds2. Predefined *patterns* determine the fixed masking positions for all *k*-mers. Seeds generated from masked *k*-mers are known as *spaced seeds* (34). The tools such as ZOOM! (41) and SHRiMP2 (52) use spaced seeds to improve the sensitivity when mapping short reads (i.e., Illumina paired-end reads). S-conLSH (65,66) 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 (67,68). Unfortunately, spaced seeds cannot find *any arbitrary* fuzzy matches of *k*-mers with a single lookup due to (i) fixed patterns that allow mismatches only at certain positions of *k*-mers and (ii) *low-collision hashing* techniques that can be used for finding *only* exact-matching spaced seeds, which are key limitations in improving the sensitivity of spaced seeds.

The third direction aims to allow both substitutions and indels when matching *k*-mers. A common approach is to link a few selected *k*-mers of a sequence to use these linked *k*-mers as seeds, such as paired-minimizers (69) and strobemers (70,71). These approaches can ignore large gaps between the linked *k*-mers. For example, the strobemer technique concatenates a subset of selected *k*-mers of a sequence to generate a strobemer sequence, which is used as a seed. Strobealign (71) uses these strobemer seeds for mapping short reads with high accuracy and performance. Strobe-

mers 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.

To our knowledge, there is no work that can *efficiently* find fuzzy matches of seeds *without* requiring (i) *exact matches* of all k-mers (i.e., any *k*-mer can mismatch) and (ii) 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 such a requirement so that similar seeds can have the same hash value has the potential to improve further the performance and sensitivity of the applications that use seeds with their ability to allow substitutions and indels at any arbitrary position when matching seeds.

A hashing technique, SimHash (72,73), provides useful properties for efficiently detecting highly similar seeds from their hash values. The SimHash technique can generate similar hash values for similar real-valued vectors or sets (72). Such a property enables estimating the cosine similarity between a pair of vectors (74) based on the Hamming distance of their hash values that SimHash generates (i.e., *SimHash values*) (72,75). Although MinHash can provide better cosine similarity estimations than SimHash (76), SimHash enables generating compact hash values that are practically useful for similarity estimations based on the Hamming distance. 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 (73). This *permutation-based* approach enables exploiting the Hamming distance similarity properties of the SimHash technique for various applications that find near-duplicate items (73,77–80).

In genomics, the properties of the SimHash and the permutation-based techniques are used for cell type classification (81) and short sequence alignment (82). In read alignment, the permutation-based approach (73) is applied for detecting mismatches by permuting the sequences *without* generating the hash values using the SimHash technique. This approach can find the longest prefix matches between a reference genome and a read since the mismatches between a pair of sequences *may* move to the last positions of these sequences after applying different permutations while keeping the Hamming distance between sequences the same. This approach uses various versions of permutations to find the prefix matches. Apart from the permutation-based technique, a pigeonhole principle is also used for tolerating mismatches in read alignment (39,40,42,62,83). Unfortunately,

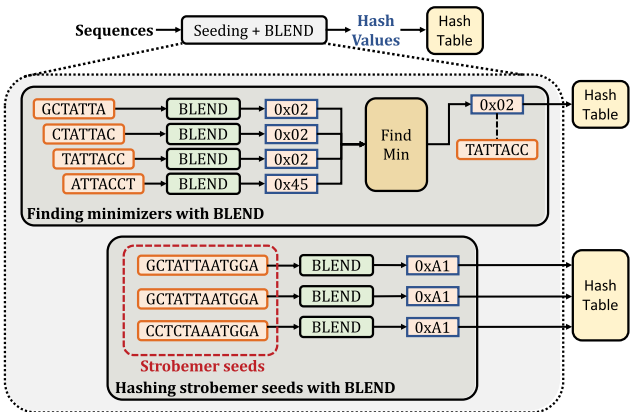

Figure 3. Replacing the hash functions in seeding techniques with BLEND.

none of these works can find highly similar seed matches that have the same hash value with a single lookup, which we call *fuzzy seed matches*.

Our goal in this work is to enable finding *fuzzy* matches of seeds as well as exact-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 exact-matching and highly similar seeds with a single lookup of their hash values. The key idea in BLEND is to enable assigning the same hash value for highly similar seeds. To this end, BLEND (i) exploits the SimHash technique (72,73) and (ii) provides proper mechanisms for using any seeding technique with SimHash to find fuzzy seed matches with a single lookup of their hash values. 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 (i) using various permutations to find the longest prefix matches (82) or (ii) matching many hash values for calculating costly similarity scores (e.g., Jaccard similarity (84)) that the conventional locality-sensitive hashing-based methods use, such as MHAP (58) or S-conLSH (65,66). These two ideas ensure that BLEND can efficiently find both (i) all exact-matching seeds that a seeding technique finds using a conventional hash function with a low-collision rate and (ii) approximate seed matches that these conventional hashing mechanisms cannot find with a single lookup of a hash value.

Figure 3 shows two examples of how BLEND can replace the conventional hash functions that the seeding techniques use in Figure 2. The key challenge is to accurately and efficiently define the items of sets from seeds that the SimHash technique requires. To achieve this, BLEND provides two mechanisms for converting seeds into sets of items: (i) `BLEND-I` and (ii) `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 set 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 set 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 set of items.

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: (i) read overlapping and (ii) read mapping. First, read overlapping aims to find overlaps between all pairs of reads based on seed matches. These overlapping reads are mainly useful for generating an assembly of the sequenced genome (14,85). 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 (86)). We compare BLEND with minimap2, LRA (87), Winnowmap2, S-conLSH and Strobealign by mapping long and paired-end short reads to their reference genomes. We evaluate the effect of the long read mapping results on downstream analysis by calling structural variants (SVs) and calculating the accuracy of SVs. This paper provides the following key contributions and major results:

- We introduce BLEND, the *first* mechanism that can quickly and efficiently find *fuzzy* seed matches between sequences with a single lookup.
- We propose two mechanisms for converting seeds into a set of items that the SimHash technique requires: (i) BLEND-I and (ii) BLEND-S. We show that BLEND-S provides better speedup and accuracy than BLEND-I when using PacBio HiFi reads for read overlapping and read mapping. When using ONT, PacBio CLR and short reads, BLEND-I provides significantly better accuracy than BLEND-S with similar performance.
- For read overlapping, we show that BLEND provides speedup compared to minimap2 and MHAP by 2.4×–83.9 × (on average 19.3×), 28.4×–4367.8× (on average 808.2×) while reducing the memory overhead by 0.9×–14.1 × (on average 3.8×), 36.0×–234.7× (on average 127.8×), respectively.
- We show that BLEND usually finds *longer* overlaps between reads while using *fewer* seed matches than other tools, which improves the performance and memory space efficiency for read overlapping.
- We find that we can construct more accurate assemblies with similar contiguity by using the overlapping reads that BLEND finds compared to those that minimap2 finds.
- For read mapping, we show that BLEND provides speedup compared to minimap2, LRA, Winnowmap2 and S-conLSH by 0.8×–4.1× (on average 1.7×), 1.2×–18.6× (on average 6.8×), 1.1×–9.9× (on average 4.3×), 1.4×–29.8× (on average 13.3×) while maintaining a similar memory overhead by 0.5×–1.1× (on average 1.0×), 0.3×–1.0× (on average 0.6×), 0.9×–4.1× (on average 1.5×), 0.2×–4.2× (on average 1.6×), respectively.
- We show that BLEND provides a read mapping accuracy similar to minimap2, and Winnowmap2 usually provides the best read mapping accuracy.
- We show that BLEND enables calling structural variants with the highest F1 score compared to minimap2, LRA and Winnowmap2.
- We open source our BLEND implementation as integrated into minimap2.
- We provide the open-source SIMD implementation of the SimHash technique that BLEND employs.

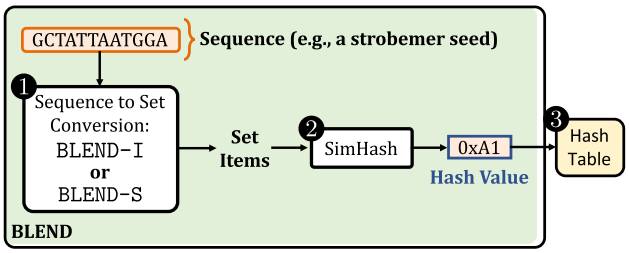

Figure 4. Overview of BLEND. 1. BLEND uses BLEND-I or BLEND-S for converting a sequence into its set of items. 2. BLEND generates the hash value of the input sequence using its set of items with the SimHash technique. 3. BLEND uses hash tables for finding fuzzy seed matches with a single lookup of the hash values that BLEND generates.

MATERIALS AND METHODS

We propose *BLEND*, a mechanism that can efficiently find fuzzy (i.e., approximate) seed matches with a single lookup of their hash values. To find fuzzy seed matches, BLEND introduces a new mechanism that enables generating the same hash values for highly similar seeds. By combining this mechanism with any seeding approach (e.g., minimizer *k*-mers or strobemers), BLEND can find fuzzy seed matches between sequences with a single lookup of hash values.

Figure 4 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 3) to its set representation as the SimHash technique generates the hash value of the set using its items1. To enable effective and efficient integration of seeds with the SimHash technique, BLEND proposes two mechanisms for identifying the items of the set of the input sequence: (i) BLEND-I and (ii) BLEND-S. Second, after identifying the items of the set, BLEND uses this set with the SimHash technique to generate the hash value for the input sequence2. BLEND uses the SimHash technique as it allows for generating the same hash value for highly similar sets. 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.

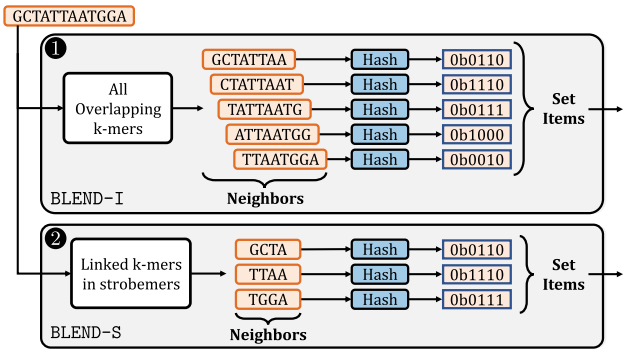

Figure 5. Overview of two mechanisms used for determining the set items of input sequences. 1. `BLEND-I` uses the hash values of all the overlapping *k*-mers of an input sequence as the set items. 2. `BLEND-S` uses the hash values of only the *k*-mers selected by the strobemer seeding mechanism.

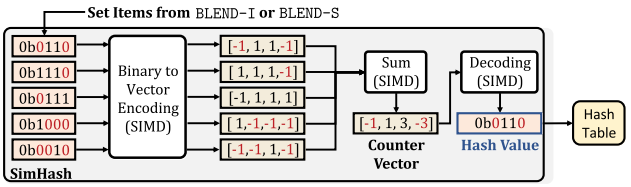

Figure 6. The overview of the steps in the SimHash technique for calculating the hash value of a given set of items. The set items are the hash values represented in their binary form. Binary to Vector Encoding converts these set items to their corresponding vector representations. Sum performs the vector additions and stores the result in a separate vector that we call the *counter vector*. Decoding generates the hash value of the set 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.

Sequence to set conversion

Our goal is to convert the input sequences that BLEND receives from any seeding technique (Figure 3) to their proper set representations so that BLEND can use the items of sets for generating the hash values of input sequences with the SimHash technique. To achieve effective and efficient conversion of sequences into their set representations in different scenarios, BLEND provides two mechanisms: (1) `BLEND-I` and (2) `BLEND-S`, as we show in Figure 5.

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 1 of Figure 5. For simplicity, we use the *neighbors* term to refer to all the k-mers that `BLEND-I` extracts from an input sequence (Figure 5). 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 set 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 3 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 (70) as neighbors, as shown in 2 of Figure 5. `BLEND-S` can enable a few of these linked *k*-mers to mismatch between strobemer sequences because a single character difference does not propagate to the other linked *k*-mers as opposed to the effect of a single character difference propagating to several overlapping *k*-mers in `BLEND-I`. 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 set items of the input sequence for SimHash.

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 (72). The SimHash technique takes a set of items and generates a hash value for the set using its items. The key benefit of the SimHash technique is that it allows generating the same hash value for highly similar sets while enabling any *arbitrary* items to mismatch between sets. To exploit the key benefit of the SimHash technique, BLEND efficiently and effectively integrates the SimHash technique with the set items that `BLEND-I` or `BLEND-S` determine. BLEND uses these set 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: (i) encoding the set items as vectors, (ii) performing vector additions, and (iii) decoding the vector to generate the hash value for the set that `BLEND-I` or `BLEND-S` determine, as we show in Figure 6. 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 Supplementary Section S3 and Supplementary Figures S1 and S2.

First, the goal of the *binary to vector encoding* step is to transform all the hash values of set items from the binary form into their corresponding 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 set item, the encoding can be done in two steps. The first step creates a vector of n elements for an n-bit hash value. We assume that all the elements in the vector are initially set to 1. For each bit position t of the hash value, the second step assigns -1 to the tth element in the vector if the bit at position t is 0, as we highlight in Figure 6 with red colors of 0 bits and their corresponding -1 values in the vector space. For each hash value

in set items, the resulting 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 set items, which we call determining the *majority* bits. The key insight in determining these majority bits is that highly similar sets are likely to result in *similar* majority results because a few differences between two similar sets 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 vectors it generates in the vector encoding step, as shown with the addition step (Sum) in Figure 6. The vector addition performs simple additions of +1 or −1 values between the vector elements and stores the result in a separate *counter* vector. The values in this counter vector show the majority of bits at each position of the set items. Since BLEND assigns −1 for 0 bits and 1 for 1 bits, the majority of bits at a position is either (i) 1 if the corresponding value in the counter vector is greater than 0 or (ii) 0 if the values are less than or equal to 0.

Third, to generate the hash value of a set, 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 6 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 (i) 1 if the value at the corresponding position of the counter vector is greater than 0 or (ii) 0 if otherwise. Thus, each bit of the final hash value of the set shows the majority voting result of set 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* set items by using `BLEND-I` or `BLEND-S`. Properly identifying the set 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 S2 and Supplementary Tables S3–S10.

Using the hash tables

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 (15).

RESULTS

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 S16 and the parameter configurations we use for each tool and dataset in Supplementary Tables S17 and S18. 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 S14. We show the trade-offs between the seeding mechanisms `BLEND-I` and `BLEND-S` in Supplementary Figures S3 and S4 and Supplementary Tables S11–S13 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 1. To evaluate BLEND in several common scenarios in read overlapping and read mapping, we classify our datasets into three categories: (i) highly accurate long reads (i.e., PacBio HiFi), (ii) erroneous long reads (i.e., PacBio CLR and Oxford Nanopore Technologies) and (iii) short reads (i.e., Illumina). We use PBSIM2 (88) to simulate the erroneous PacBio and Oxford Nanopore Technologies (ONT) reads from the Yeast genome. To use realistic depth of coverage, we use SeqKit (89) to down-sample the original *E. coli*, and

Table 1. Details of datasets used in our 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	(90)
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	(90)
	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.

D. ananassae reads to 100× and 50× sequencing depth of coverage, respectively.

We evaluate BLEND based on two use cases: (i) read overlapping and (ii) 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 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 (14). 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 (91) 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 100 kb, and GC content (i.e., the ratio of G and C bases in an assembly). We use dnadiff (92) to measure the accuracy of *de novo* assemblies based on (i) the average identity of an assembly when compared to its reference genome and (ii) the fraction of overall bases in a reference genome that align to a given assembly (i.e., genome fraction). We compare BLEND with minimap2 (15) and MHAP (58) for read overlapping. For the human genomes, MHAP either (i) requires a memory space larger than what we have in our system (i.e., 1TB) or (ii) generates a large output such that we cannot generate the assembly as miniasm exceeds the memory space we have.

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 (87), Winnowmap2 (59,60), S-conLSH (65,66) and Strobealign (71). We do not evaluate (i) LRA, Winnowmap2 and S-conLSH for short reads as these tools do not support mapping paired-end short reads, (ii) Strobealign for long reads as it is a short read aligner, (iii) 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 (iv) 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 (i) LRA generates a CIGAR string with characters that the `paftools mapeval` tool cannot parse to calculate alignment positions, and (ii) S-conLSH due to its poor accuracy results we observe in our preliminary analysis.

Read mapping accuracy. We measure (i) the overall read mapping error rate and (ii) 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 (i) the breadth of coverage (i.e., percentage of bases in a reference genome covered by at least one read), (ii) the average depth of coverage (i.e., the average number of read alignments per base in a reference genome), (iii) mapping rate (i.e., number of aligned reads) and (iv) rate of properly paired reads for paired-end mapping. To measure the breadth and depth of coverage of read mapping, we use BEDTools (93) and Mosdepth (94), respectively. To measure the mapping rate and properly paired reads, we use BAMUtil (95).

Downstream analysis. We use sniffles2 (96,97) to call structural variants (SVs) from the HG002 long read mappings. We use Truvari (98) 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 (99) 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

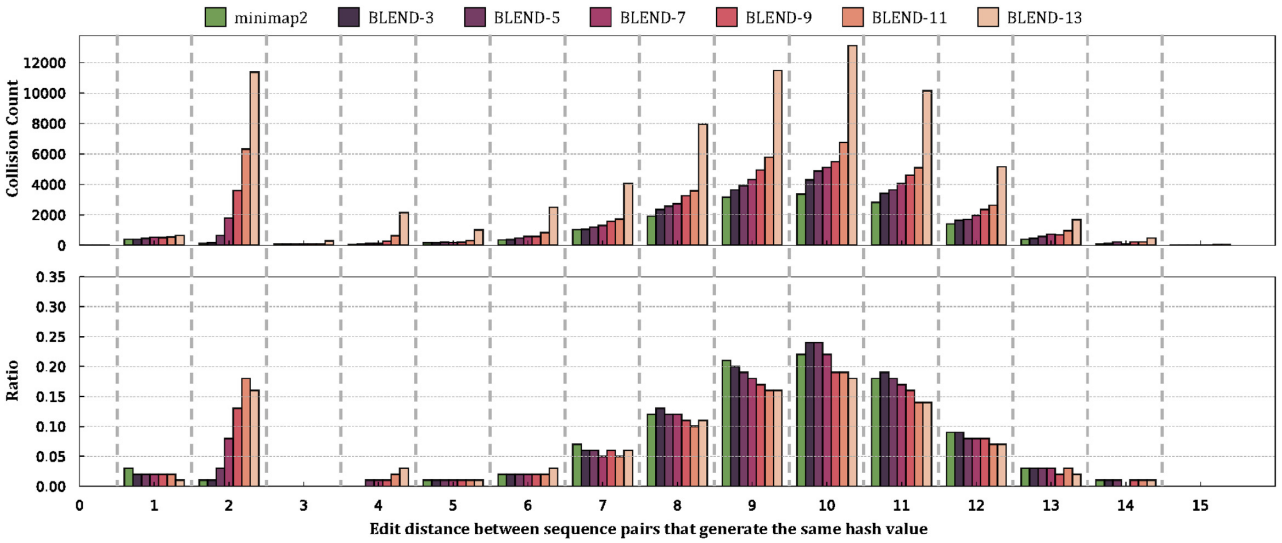

Figure 7. Fuzzy seed matching statistics. Collision count shows the number of non-identical seeds that generate the same hash value and the edit distance between these sequences. Ratio is the proportion of collisions between non-identical sequences at a certain edit distance over all collisions. BLEND-*n* shows the number of neighbors (*n*) that BLEND uses.

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 (99), 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 (100) 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 S17–S19. 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 S5 and Supplementary Table S15. For read mapping, the comparable default parameters in BLEND are already the same as in minimap2.

Empirical analysis of fuzzy seed matching

We evaluate the effectiveness of fuzzy seed matching by finding non-identical seeds with the same hash value (i.e., collisions) when using a low-collision hash function that minimap2 uses (`hash64`) and BLEND in two ways.

Finding minimizer collisions. Our goal is to evaluate the effects of using a low-collision hash function and the BLEND mechanism on the hash value collisions between non-identical minimizers. We use minimap2 and BLEND to find all the minimizer seeds in the *E. coli* reference genome (90), as explained in Supplementary Section S1.1. Figure 7 shows the edit distance between non-identical seeds with hash collision when using minimap2 and BLEND. We evaluate BLEND for various numbers of *neighbors* (*n*) as explained in the *Sequence to set conversion* section, which we show as BLEND-*n* in Figure 7, Supplementary Tables S1 and S2. We make three key observations. First, BLEND significantly increases the ratio of hash 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 Supplementary Table S1. 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 provid-

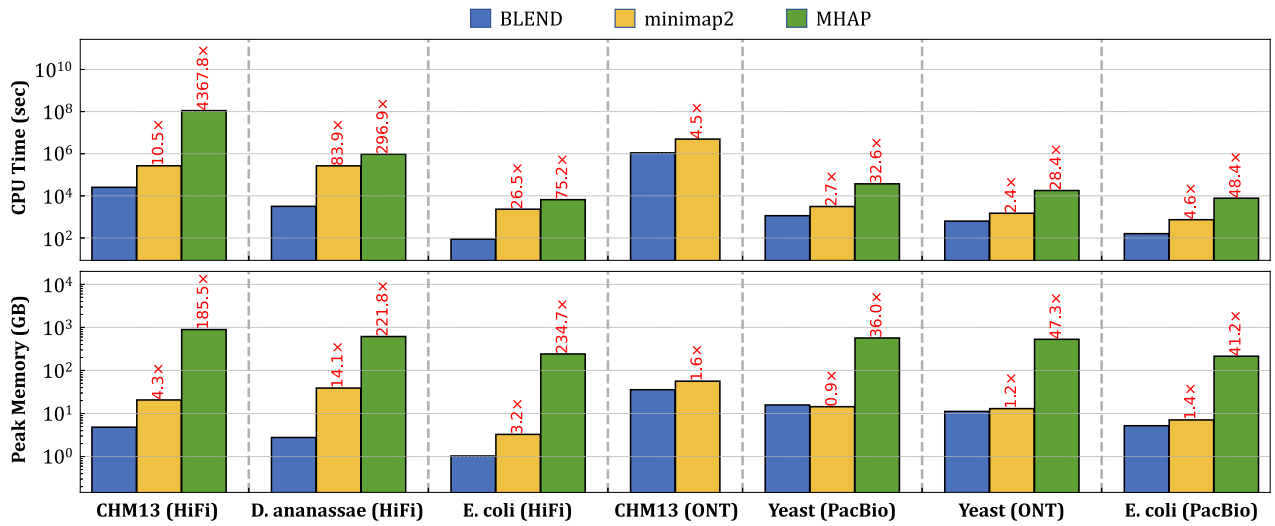

Figure 8. CPU time and peak memory footprint comparisons of read overlapping.

ing a collision ratio similar to minimap2 for dissimilar seeds.

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, as explained in Supplementary Section S1.2. Supplementary Table S2 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.

Use Case 1: read overlapping

Performance. Figure 8 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 (i) 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 (ii) 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 S5 that minimap2-Eq performs, on average, ∼ 5% better than BLEND with similar memory space requirements when using the same set

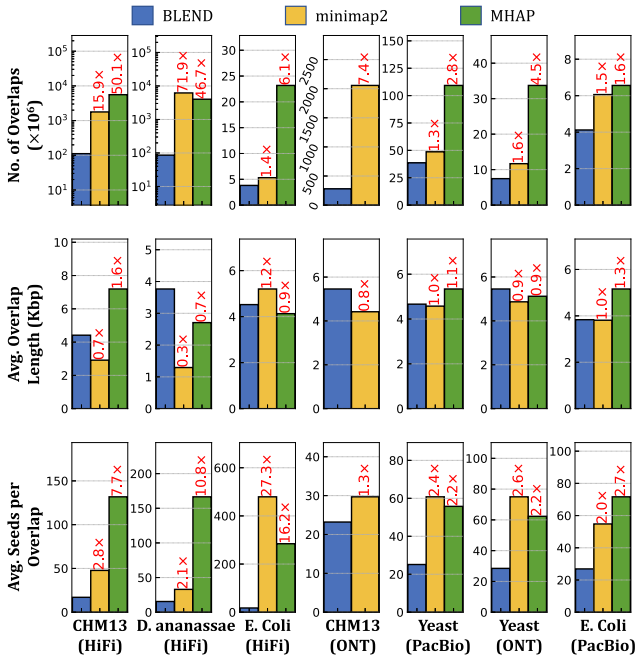

Figure 9. Average number and length of overlaps, and average number of seeds used to find a single overlap.

of parameters with the `BLEND-I` technique. Minimap2-Eq provides worse accuracy than BLEND when generating the ONT assemblies, as shown in Supplementary Table S15, 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.

Overlap statistics. Figure 9 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. The combination of the overall number of overlaps and the average number of seed matches provides the overall 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 (i) uniquely find the fuzzy seed matches that the exact-matching-based tools cannot find and (ii) 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 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 (i) larger window lengths than minimap2 and (ii) provides windowing guarantees, unlike MHAP. Third, finding fewer seed matches per overlap leads to (i) finding fewer overlaps than minimap2 and MHAP find and (ii) 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 2). 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 2. 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.

Assembly quality assessment. Our goal is to assess the quality of assemblies generated using the overlapping reads found by BLEND, minimap2, and MHAP. Table 2 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 we can construct more accurate assemblies in terms of average identity and k-mer completeness when we use the overlapping reads that BLEND finds than those minimap2 and MHAP find. 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 be-

Table 2. Assembly quality comparisons

Dataset	Tool	Average Identity (%)	Genome Fraction (%)	*k*-mer Compl. (%)	Aligned Length (Mb)	Mismatch per 100 kb (#)	Average GC (%)	Assembly Length (Mb)	Largest Contig (Mb)	NGA50 (kb)	NG50 (kb)
CHM13 (HiFi)	BLEND	**99.8526**	**98.4847**	**90.15**	3092.54	**22.02**	**40.78**	**3095.21**	22.8397	5442.25	5442.31
	minimap2	99.7421	97.1493	83.05	**3094.79**	55.96	40.71	3100.97	**47.1387**	**7133.43**	**7134.31**
	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	3054.83	0.00	40.85	3054.83	248.387	154 260	154 260
D. ananassae (HiFi)	BLEND	**99.7856**	**97.2308**	**86.43**	240.391	**143.13**	**41.75**	**247.153**	**6.23256**	**792.407**	**798.913**
	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 (HiFi)	BLEND	**99.8320**	**99.8801**	**87.91**	**5.12155**	**3.77**	**50.53**	5.12155	**3.41699**	**3416.99**	**3416.99**
	minimap2	99.7064	99.8748	79.27	5.09249	19.71	50.47	**5.09436**	3.08849	3087.05	3087.05
	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	4944.46	4944.46
CHM13 (ONT)	BLEND	N/A	N/A	**29.26**	**2891.28**	**4077.53**	**41.32**	2897.87	25.2071	5061.52	5178.59
	minimap2	N/A	N/A	28.32	2860.26	4660.73	41.36	**2908.55**	**66.7564**	**13 189.2**	**13 820.3**
	Reference	100	100	100	3117.29	0.00	40.75	3117.29	248.387	150 617	150 617
Yeast (PacBio)	BLEND	89.1677	**97.0854**	**33.81**	**12.3938**	2672.37	38.84	12.4176	1.54807	635.966	636.669
	minimap2	88.9002	96.9709	33.38	12.0128	2684.38	38.85	**12.3325**	**1.56078**	**810.046**	**828.212**
	MHAP	**89.2182**	88.5928	32.39	10.9039	**2552.05**	**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 (ONT)	BLEND	**89.6889**	99.2974	**35.95**	**12.3222**	**2529.47**	**38.64**	**12.3225**	1.10582	793.046	793.046
	minimap2	88.9393	**99.6878**	34.84	12.304	2782.59	38.74	12.3725	**1.56005**	**796.718**	**941.588**
	MHAP	89.1970	89.2785	33.58	10.8302	2647.19	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 (PacBio)	BLEND	**88.5806**	**96.5238**	**32.32**	**5.90024**	**1857.56**	**49.81**	6.21598	2.40671	**769.981**	2060.4
	minimap2	88.1365	92.7603	30.74	5.37728	2005.72	49.66	**6.02707**	**3.77098**	367.442	**3770.98**
	MHAP	88.4883	90.5533	31.32	5.75159	1999.48	49.69	6.26216	1.04286	110.535	456.01
	Reference	100	100	100	5.6394	0.00	50.43	5.6394	5.54732	5547.32	5547.32

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.

lieve 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 2 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.

Use Case 2: read mapping

Performance. Figure 10 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. Although BLEND performs better than most of these tools, the speedups we see are usually lower than those we observe 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

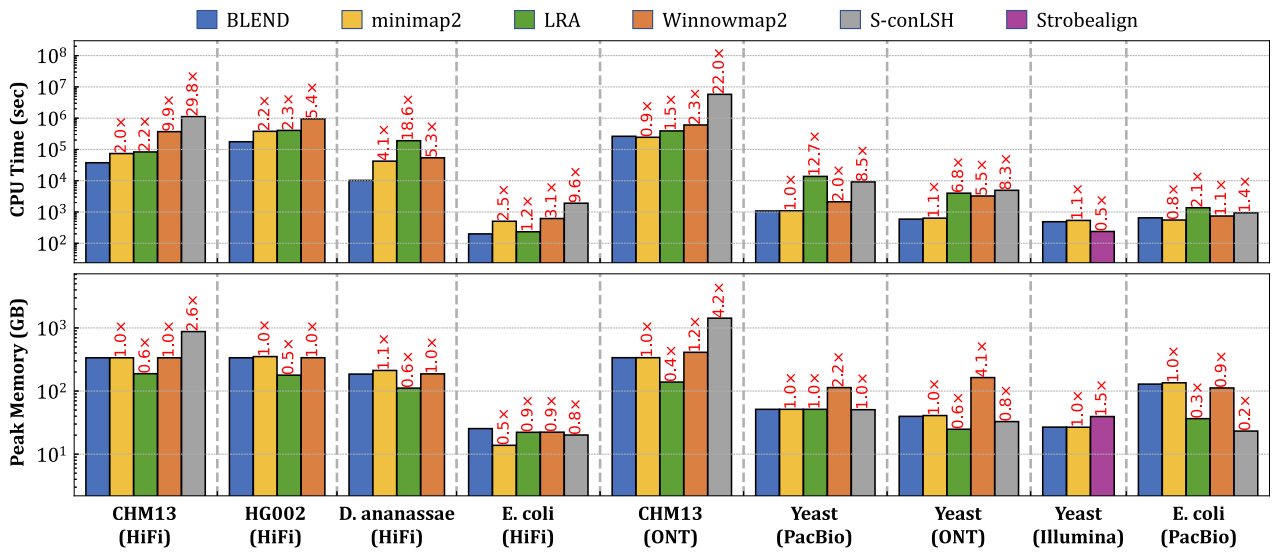

Figure 10. CPU time and peak memory footprint comparisons of read mapping.

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, (i) performs better than all tools for mapping long reads and (ii) provides a memory footprint similar to or better than minimap2, Winnowmap2, S-conLSH, and Strobealign, while LRA is the most memory-efficient tool.

Read mapping accuracy. Table 3 and Figure 11 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 that 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.

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

Table 3. Read mapping accuracy comparisons

	Overall error rate (%)		
Dataset	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.

paired in mapping. Table 4 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 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. We investigate if such a smaller number of aligned reads leads to a coverage bias genome-wide in Supplementary Figures S6–S8. We find that the distribution of the depth of coverage of BLEND is mostly simi-

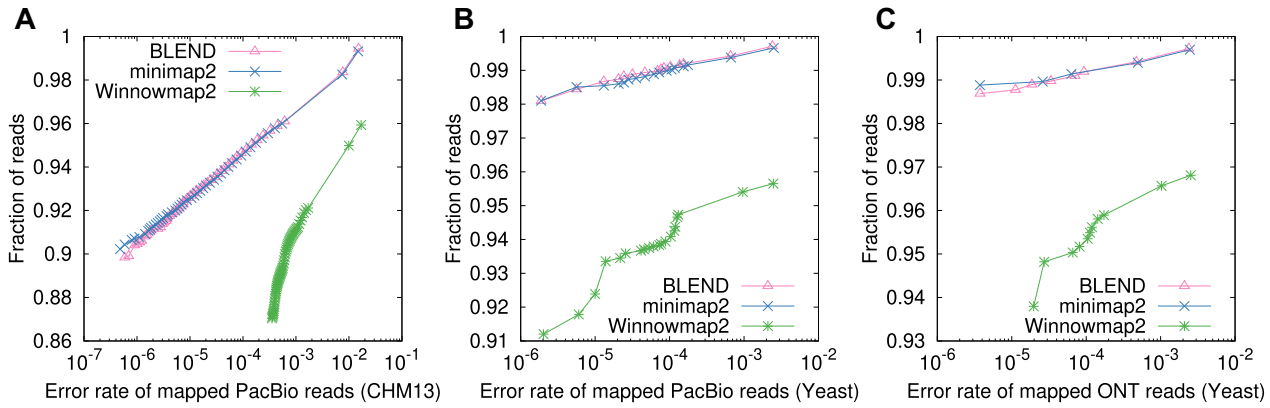

Figure 11. Fraction of simulated reads with an average mapping error rate. Reads are binned by their mapping quality scores. There is a bin for each mapping quality score as reported by the read mapper, and bins are sorted based on their mapping quality scores in descending order. For each tool, the *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.

lar 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 S8, 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 (i) short reads mappings to the reference genome and (ii) properly paired reads compared to BLEND and minimap2. Strobealign can map more reads using less time (Figure 10, 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 exact-matching seed requirements.

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 5 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 5, which leads to providing the best overall F_1 accuracy in structural variant calling.

DISCUSSION

We demonstrate that there are usually too many redundant *short* and *exact-matching* seeds used to find overlaps between sequences, as shown in Figure 9. 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 (15). 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 (28). To achieve this, earlier works mainly focus on either (i) chaining the exact *k*-mer matches by tolerating the gaps between them to increase the seed region or (ii) 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 (101). Linking multiple minimizer k-mers enables tolerating indels when finding the matches of short subsequences 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 seed-

Table 4. 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.

Table 5. Benchmarking the structural variant (SV) calling results

	HG002 SVs (high-confidence tier 1 SV set)					
Tool	TP (#)	FP (#)	FN (#)	Precision	Recall	F_1
BLEND	**9229**	855	**412**	0.9152	**0.9573**	**0.9358**
minimap2	9222	915	419	0.9097	0.9565	0.9326
LRA	9155	**830**	486	**0.9169**	0.9496	0.9329
Winnowmap2	9170	1029	471	0.8991	0.9511	0.9244

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

ing techniques to generate exactly the same seed to find either exact-matching or approximate matches of short subsequences. 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 (102–114), GPUs (115–117), and FPGAs and ASICs (118–123) 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) (71). It is worth exploring and rethinking the hash functions used in these seeding techniques. Fourth, potential machine learning applications can be used to gener-

ate more sensitive hash values for fuzzy seed matching based on learning-to-hash approaches (124) and recent improvements on SimHash for identifying nearest neighbors in machine learning and bioinformatics (125–127).

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 Supplementary Table S2. Such 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 (125) or FlyHash (128) 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.

Conclusion. 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: (i) read overlapping and (ii) 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 (i) 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.

DATA AVAILABILITY

We provide the accession numbers of all the public datasets we use in Table 1. We make the simulated datasets we use available on the Zenodo website. The human CHM13 (simulated ONT) dataset is available at https://doi.org/10.5281/zenodo.7261610. The Yeast (simulated PacBio CLR) dataset is available at https://doi.org/10.5281/zenodo.7261660. The Yeast (simulated ONT) dataset is available at https://doi.org/10.5281/zenodo.7261655. We also provide all the scripts (i) with the Zenodo links to download real and simulated datasets and (ii) to fully reproduce our results and figures at https://github.com/CMUSAFARI/BLEND/tree/master/test. The source code of BLEND is available at https://github.com/CMU-SAFARI/BLEND and at https://doi.org/10.5281/zenodo.7502134. For easy installation, we also make BLEND available in Docker (firtinac/blend) and bioconda (`blend-bio`).

SUPPLEMENTARY DATA

Supplementary Data are available at NARGAB Online.

ACKNOWLEDGEMENTS

We thank the SAFARI Research Group members for their valuable feedback and the stimulating intellectual and scholarly environment they provide. SAFARI Research Group acknowledges the generous gifts of our industrial partners, including Intel and VMware. We are grateful for the detailed comments that Kristoffer Sahlin provided, which improved our mechanism and the manuscript greatly.

FUNDING

Intel [to O.M.]; VMware [to O.M.].
Conflict of interest statement. None declared.

REFERENCES

1. Shendure,J., Balasubramanian,S., Church,G.M., Gilbert,W., Rogers,J., Schloss,J.A. and Waterston,R.H. (2017) DNA sequencing at 40: past, present and future. *Nature*, **550**, 345–353.
2. Aynaud,M.-M., Hernandez,J.J., Barutcu,S., Braunschweig,U., Chan,K., Pearson,J.D., Trcka,D., Prosser,S.L., Kim,J., Barrios-Rodiles,M. *et al.* , (2021) A multiplexed, next generation sequencing platform for high-throughput detection of SARS-CoV-2. *Nat. Commun.*, **12**, 1405.
3. Logsdon,G.A., Vollger,M.R. and Eichler,E.E. (2020) Long-read human genome sequencing and its applications. *Nat. Rev. Genet.*, **21**, 597–614.
4. Mantere,T., Kersten,S. and Hoischen,A. (2019) Long-read sequencing emerging in medical genetics. *Front. Genet.*, **10**, 426.
5. B.M. Knoppers for the Paediatric Task Team of the Global Alliance for Genomics and Health Regulatory and Ethics Work Stream, Friedman,J.M., Bombard,Y., Cornel,M.C., Fernandez,C.V., Junker,A.K., Plon,S.E. and Stark,Z. (2019) Genome-wide sequencing in acutely ill infants: genomic medicine's critical application? *Genet. Med.*, **21**, 498–504.

6. Merker,J.D., Wenger,A.M., Sneddon,T., Grove,M., Zappala,Z., Fresard,L., Waggott,D., Utiramerur,S., Hou,Y., Smith,K.S. *et al.* (2018) Long-read genome sequencing identifies causal structural variation in a Mendelian disease. *Genet. Med.*, **20**, 159–163.
7. Alkan,C., Coe,B.P. and Eichler,E.E. (2011) Genome structural variation discovery and genotyping. *Nat. Rev. Genet.*, **12**, 363–376.
8. Goodwin,S., McPherson,J.D. and McCombie,W.R. (2016) Coming of age: ten years of next-generation sequencing technologies. *Nat. Rev. Genet.*, **17**, 333–351.
9. Stoler,N. and Nekrutenko,A. (2021) Sequencing error profiles of Illumina sequencing instruments. *NAR Genom. Bioinform.*, **3**, lqab019.
10. Zhang,H., Jain,C. and Aluru,S. (2020) A comprehensive evaluation of long read error correction methods. *BMC Genom.*, **21**, 889.
11. Hon,T., Mars,K., Young,G., Tsai,Y.-C., Karalius,J.W., Landolin,J.M., Maurer,N., Kudrna,D., Hardigan,M.A., Steiner,C.C. *et al.* (2020) Highly accurate long-read HiFi sequencing data for five complex genomes. *Sci. Data*, **7**, 399.
12. Ma,X., Shao,Y., Tian,L., Flasch,D.A., Mulder,H.L., Edmonson,M.N., Liu,Y., Chen,X., Newman,S., Nakitandwe,J. *et al.* (2019) Analysis of error profiles in deep next-generation sequencing data. *Genome Biol.*, **20**, 50.
13. Senol Cali,D., Kim,J.S., Ghose,S., Alkan,C. and Mutlu,O. (2019) Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions. *Brief. Bioinform.*, **20**, 1542–1559.
14. Li,H. (2016) Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, **32**, 2103–2110.
15. Li,H. (2018) Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, **34**, 3094–3100.
16. Canzar,S. and Salzberg,S.L. (2017) Short read mapping: an algorithmic tour. *Proc. IEEE*, **105**, 436–458.
17. Kim,J.S., Firtina,C., Cavlak,M.B., Senol Cali,D., Hajinazar,N., Alser,M., Alkan,C. and Mutlu,O. (2021) AirLift: a fast and comprehensive technique for remapping alignments between reference genomes. bioRxiv doi: https://doi.org/10.1101/2021.02.16.431517, 17 February 2021, preprint: not peer reviewed.
18. Kim,J.S., Firtina,C., Cavlak,M.B., Senol Cali,D., Alkan,C. and Mutlu,O. (2022) FastRemap: a tool for quickly remapping reads between genome assemblies. *Bioinformatics*, **38**, 4633–4635.
19. Ekim,B., Berger,B. and Chikhi,R. (2021) Minimizer-space de Bruijn graphs: whole-genome assembly of long reads in minutes on a personal computer. *Cell Syst.*, **12**, 958–968.
20. Cheng,H., Concepcion,G.T., Feng,X., Zhang,H. and Li,H. (2021) Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. *Nat. Methods*, **18**, 170–175.
21. Robertson,G., Schein,J., Chiu,R., Corbett,R., Field,M., Jackman,S.D., Mungall,K., Lee,S., Okada,H.M., Qian,J.Q. *et al.* (2010) De novo assembly and analysis of RNA-seq data. *Nat. Methods*, **7**, 909–912.
22. Meyer,F., Fritz,A., Deng,Z.-L., Koslicki,D., Lesker,T.R., Gurevich,A., Robertson,G., Alser,M., Antipov,D., Beghini,F. *et al.* (2022) Critical assessment of metagenome interpretation: the second round of challenges. *Nat. Methods*, **19**, 429–440.
23. LaPierre,N., Alser,M., Eskin,E., Koslicki,D. and Mangul,S. (2020) Metalign: efficient alignment-based metagenomic profiling via containment min hash. *Genome Biol.*, **21**, 242.
24. Wood,D.E., Lu,J. and Langmead,B. (2019) Improved metagenomic analysis with Kraken 2. *Genome Biol.*, **20**, 257.
25. Firtina,C., Kim,J.S., Alser,M., Senol Cali,D., Cicek,A.E., Alkan,C. and Mutlu,O. (2020) Apollo: a sequencing-technology-independent, scalable and accurate assembly polishing algorithm. *Bioinformatics*, **36**, 3669–3679.
26. Vaser,R., Sović,I., Nagarajan,N. and Šikić,M. (2017) Fast and accurate de novo genome assembly from long uncorrected reads. *Genome Res.*, **27**, 737–746.
27. Loman,N.J., Quick,J. and Simpson,J.T. (2015) A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nat. Methods*, **12**, 733–735.
28. Alser,M., Rotman,J., Deshpande,D., Taraszka,K., Shi,H., Baykal,P.I., Yang,H.T., Xue,V., Knyazev,S., Singer,B.D. *et al.* (2021) Technology dictates algorithms: recent developments in read alignment. *Genome Biol.*, **22**, 249.
29. Alser,M., Lindegger,J., Firtina,C., Almadhoun,N., Mao,H., Singh,G., Gomez-Luna,J. and Mutlu,O. (2022) From Molecules to Genomic Variations: Accelerating Genome Analysis via Intelligent Algorithms and Architectures. *Comput. Struct. Biotechnol. J.*, **20**, 4579–4599.
30. Altschul,S.F., Gish,W., Miller,W., Myers,E.W. and Lipman,D.J. (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.
31. Altschul,S.F., Madden,T.L., Schäffer,A.A., Zhang,J., Zhang,Z., Miller,W. and Lipman,D.J. (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.*, **25**, 3389–3402.
32. Ning,Z., Cox,A.J. and Mullikin,J.C. (2001) SSAHA: a fast search method for large DNA databases. *Genome Res.*, **11**, 1725–1729.
33. Kent,W.J. (2002) BLAT—the BLAST-Like alignment tool. *Genome Res.*, **12**, 656–664.
34. Ma,B., Tromp,J. and Li,M. (2002) PatternHunter: faster and more sensitive homology search. *Bioinformatics*, **18**, 440–445.
35. Schwartz,S., Kent,W.J., Smit,A., Zhang,Z., Baertsch,R., Hardison,R.C., Haussler,D. and Miller,W. (2003) Human—mouse alignments with BLASTZ. *Genome Res.*, **13**, 103–107.
36. Slater,G. S.C. and Birney,E. (2005) Automated generation of heuristics for biological sequence comparison. *BMC Bioinformatics*, **6**, 31.
37. Wu,T.D. and Watanabe,C.K. (2005) GMAP: a genomic mapping and alignment program for mRNA and EST sequences. *Bioinformatics*, **21**, 1859–1875.
38. Ondov,B.D., Varadarajan,A., Passalacqua,K.D. and Bergman,N.H. (2008) Efficient mapping of applied biosystems SOLiD sequence data to a reference genome for functional genomic applications. *Bioinformatics*, **24**, 2776–2777.
39. Li,R., Li,Y., Kristiansen,K. and Wang,J. (2008) SOAP: short oligonucleotide alignment program. *Bioinformatics*, **24**, 713–714.
40. Jiang,H. and Wong,W.H. (2008) SeqMap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics*, **24**, 2395–2396.
41. Lin,H., Zhang,Z., Zhang,M.Q., Ma,B. and Li,M. (2008) ZOOM! Zillions of oligos mapped. *Bioinformatics*, **24**, 2431–2437.
42. Smith,A.D., Xuan,Z. and Zhang,M.Q. (2008) Using quality scores and longer reads improves accuracy of Solexa read mapping. *BMC Bioinformatics*, **9**, 128.
43. Alkan,C., Kidd,J.M., Marques-Bonet,T., Aksay,G., Antonacci,F., Hormozdiari,F., Kitzman,J.O., Baker,C., Malig,M., Mutlu,O. and et,al. (2009) Personalized copy number and segmental duplication maps using next-generation sequencing. *Nat. Genet.*, **41**, 1061–1067.
44. Homer,N., Merriman,B. and Nelson,S.F. (2009) BFAST: an alignment tool for large scale genome resequencing. *PLOS One*, **4**, e7767.
45. Schneeberger,K., Hagmann,J., Ossowski,S., Warthmann,N., Gesing,S., Kohlbacher,O. and Weigel,D. (2009) Simultaneous alignment of short reads against multiple genomes. *Genome Biol.*, **10**, R98.
46. Weese,D., Emde,A.-K., Rausch,T., Döring,A. and Reinert,K. (2009) RazerS—fast read mapping with sensitivity control. *Genome Res.*, **19**, 1646–1654.
47. Rumble,S.M., Lacroute,P., Dalca,A.V., Fiume,M., Sidow,A. and Brudno,M. (2009) SHRiMP: accurate mapping of short color-space reads. *PLoS Comput. Biol.*, **5**, e1000386.
48. Li,R., Yu,C., Li,Y., Lam,T.-W., Yiu,S.-M., Kristiansen,K. and Wang,J. (2009) SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, **25**, 1966–1967.
49. Hach,F., Hormozdiari,F., Alkan,C., Hormozdiari,F., Birol,I., Eichler,E.E. and Sahinalp,S.C. (2010) mrsFAST: a cache-oblivious algorithm for short-read mapping. *Nat. Methods*, **7**, 576–577.
50. Wu,T.D. and Nacu,S. (2010) Fast and SNP-tolerant detection of complex variants and splicing in short reads. *Bioinformatics*, **26**, 873–881.
51. Rizk,G. and Lavenier,D. (2010) GASSST: global alignment short sequence search tool. *Bioinformatics*, **26**, 2534–2540.
52. David,M., Dzamba,M., Lister,D., Ilie,L. and Brudno,M. (2011) SHRiMP2: Sensitive yet Practical Short Read Mapping. *Bioinformatics*, **27**, 1011–1012.
53. Egidi,L. and Manzini,G. (2013) Better spaced seeds using quadratic residues. *J. Comp. Syst. Sci.*, **79**, 1144–1155.

54. Liu,B., Guan,D., Teng,M. and Wang,Y. (2016) rHAT: fast alignment of noisy long reads with regional hashing. *Bioinformatics*, **32**, 1625–1631.
55. Baichoo,S. and Ouzounis,C.A. (2017) Computational complexity of algorithms for sequence comparison, short-read assembly and genome alignment. *Biosystems*, **156-157**, 72–85.
56. Roberts,M., Hayes,W., Hunt,B.R., Mount,S.M. and Yorke,J.A. (2004) Reducing storage requirements for biological sequence comparison. *Bioinformatics*, **20**, 3363–3369.
57. Schleimer,S., Wilkerson,D.S. and Aiken,A. (2003) Winnowing: local algorithms for document fingerprinting. In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. pp. 76–85.
58. Berlin,K., Koren,S., Chin,C.-S., Drake,J.P., Landolin,J.M. and Phillippy,A.M. (2015) Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nat. Biotechnol.*, **33**, 623–630.
59. Jain,C., Rhie,A., Hansen,N.F., Koren,S. and Phillippy,A.M. (2022) Long-read mapping to repetitive reference sequences using Winnowmap2. *Nat. Methods*, **19**, 705–710.
60. Jain,C., Rhie,A., Zhang,H., Chu,C., Walenz,B.P., Koren,S. and Phillippy,A.M. (2020) Weighted minimizer sampling improves long read mapping. *Bioinformatics*, **36**, i111–i118.
61. DeBlasio,D., Gbosibo,F., Kingsford,C. and Marçais,G. (2019) Practical universal K-Mer sets for minimizer schemes. In: *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics, BCB '19*. Association for Computing Machinery, NY, pp. 167–176.
62. Xin,H., Shao,M. and Kingsford,C. (2020) Context-aware seeds for read mapping. *Algorithm. Mol. Biol.*, **15**, 10.
63. Broder,A. (1997) On the resemblance and containment of documents. In: *Proceedings. Compression and Complexity of Sequences 1997 (Cat. No. 97TB100171)*. pp. 21–29.
64. Xin,H., Lee,D., Hormozdiari,F., Yedkar,S., Mutlu,O. and Alkan,C. (2013) Accelerating read mapping with FastHASH. *BMC Genom.*, **14**, S13.
65. Chakraborty,A. and Bandyopadhyay,S. (2020) conLSH: context based locality sensitive hashing for mapping of noisy SMRT reads. *Comput. Biol. Chem.*, **85**, 107206.
66. Chakraborty,A., Morgenstern,B. and Bandyopadhyay,S. (2021) S-conLSH: alignment-free gapped mapping of noisy long reads. *BMC Bioinformatics*, **22**, 64.
67. Petrucci,E., Noé,L., Pizzi,C. and Comin,M. (2020) Iterative spaced seed hashing: closing the gap between spaced seed hashing and K-mer hashing. *J. Comput. Biol.*, **27**, 223–233.
68. Mallik,A. and Ilie,L. (2021) ALeS: adaptive-length spaced-seed design. *Bioinformatics*, **37**, 1206–1210.
69. Chin,C.-S. and Khalak,A. (2019) Human genome assembly in 100 minutes. bioRxiv doi: https://doi.org/10.1101/705616, 17 July 2019, preprint: not peer reviewed.
70. Sahlin,K. (2021) Effective sequence similarity detection with strobemers. *Genome Res.*, **31**, 2080–2094.
71. Sahlin,K. (2022) Strobealign: flexible seed size enables ultra-fast and accurate read alignment. *Genome Biol.*, **23**, 260.
72. Charikar,M.S. (2002) Similarity estimation techniques from rounding algorithms. In: *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing, STOC '02*. Association for Computing Machinery, NY , pp. 380–388.
73. Manku,G.S., Jain,A. and Das Sarma,A. (2007) Detecting near-duplicates for web crawling. In: *Proceedings of the 16th International Conference on World Wide Web, WWW '07*. Association for Computing Machinery, NY, pp. 141–150.
74. Goemans,M.X. and Williamson,D.P. (1995) Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, **42**, 1115–1145.
75. Pratap,R., Deshmukh,A., Nair,P. and Ravi,A. (2020) Scaling up simhash. In: *Proceedings of the 12th Asian Conference on Machine Learning. PMLR Vol. 129 of Proceedings of Machine Learning Research*, pp. 705–720.
76. Shrivastava,A. and Li,P. (2014) In defense of minhash over simhash. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics. PMLR Vol.33 of Proceedings of Machine Learning Research*, Reykjavik, Iceland, pp. 886–894.
77. Uddin,M.S., Roy,C.K., Schneider,K.A. and Hindle,A. (2011) On the effectiveness of simhash for detecting near-miss clones in large scale software systems. In: *2011 18th Working Conference on Reverse Engineering*. pp. 13–22.
78. Sood,S. and Loguinov,D. (2011) Probabilistic near-duplicate detection using simhash. In: *Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM '11*. Association for Computing Machinery, NY, pp. 1117–1126.
79. Feng,X., Jin,H., Zheng,R. and Zhu,L. (2014) Near-duplicate detection using GPU-based simhash scheme. In: *2014 International Conference on Smart Computing*. pp. 223–228.
80. Fröbe,M., Bevendorff,J., Gienapp,L., Völske,M., tein,B., Potthast,M. and Hagen,M. (2021) 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, SIGIR '21*. Association for Computing Machinery, NY, pp. 2398–2404.
81. Sun,Q., Peng,Y. and Liu,J. (2021) A reference-free approach for cell type classification with scRNA-seq. *iScience*, **24**, 102855.
82. Lederman,R. (2013) A random-permutations-based approach to fast read alignment. *BMC Bioinformatics*, **14**, S8.
83. Xin,H., Greth,J., Emmons,J., Pekhimenko,G., Kingsford,C., Alkan,C. and Mutlu,O. (2015) Shifted Hamming distance: a fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping. *Bioinformatics*, **31**, 1553–1560.
84. Jaccard,P. (1908) Nouvelles recherches sur la distribution florale. *Bull. Soc. Vaud. Sci. Nat.*, **44**, 223–270.
85. Pop,M., Phillippy,A., Delcher,A.L. and Salzberg,S.L. (2004) Comparative genome assembly. *Brief. Bioinform.*, **5**, 237–248.
86. McKenna,A., Hanna,M., Banks,E., Sivachenko,A., Cibulskis,K., Kernytsky,A., Garimella,K., Altshuler,D., Gabriel,S., Daly,M. *et al.* (2010) The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res.*, **20**, 1297–1303.
87. Ren,J. and Chaisson,M. J.P. (2021) lra: A long read aligner for sequences and contigs. *PLOS Comput. Biol.*, **17**, e1009078.
88. Ono,Y., Asai,K. and Hamada,M. (2021) PBSIM2: a simulator for long-read sequencers with a novel generative model of quality scores. *Bioinformatics*, **37**, 589–595.
89. Shen,W., Le,S., Li,Y. and Hu,F. (2016) SeqKit: A Cross-Platform and Ultrafast Toolkit for FASTA/Q File Manipulation. *PLOS One*, **11**, e0163962.
90. Tvedte,E.S., Gasser,M., Sparklin,B.C., Michalski,J., Hjelmen,C.E., Johnston,J.S., Zhao,X., Bromley,R., Tallon,L.J., Sadzewicz,L. *et al.* (2021) Comparison of long-read sequencing technologies in interrogating bacteria and fly genomes. *G3 Genes|Genomes|Genetics*, **11**, jkab083.
91. Gurevich,A., Saveliev,V., Vyahhi,N. and Tesler,G. (2013) QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, **29**, 1072–1075.
92. Marçais,G., Delcher,A.L., Phillippy,A.M., Coston,R., Salzberg,S.L. and Zimin,A. (2018) MUMmer4: A fast and versatile genome alignment system. *PLoS Comput. Biol.*, **14**, e1005944.
93. Quinlan,A.R. and Hall,I.M. (2010) BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, **26**, 841–842.
94. Pedersen,B.S. and Quinlan,A.R. (2018) Mosdepth: quick coverage calculation for genomes and exomes. *Bioinformatics*, **34**, 867–868.
95. Jun,G., Wing,M.K., Abecasis,G.R. and Kang,H.M. (2015) An efficient and scalable analysis framework for variant extraction and refinement from population scale DNA sequence data. *Genome Res.*, **25**, 918–925.
96. Sedlazeck,F.J., Rescheneder,P., Smolka,M., Fang,H., Nattestad,M., von Haeseler,A. and Schatz,M.C. (2018) Accurate detection of complex structural variations using single-molecule sequencing. *Nat. Methods*, **15**, 461–468.
97. Smolka,M., Paulin,L.F., Grochowski,C.M., Mahmoud,M., Behera,S., Gandhi,M., Hong,K., Pehlivan,D., Scholz,S.W., Carvalho,C.M. *et al.* (2022) Comprehensive structural variant detection: from mosaic to population-level. bioRxiv doi: https://doi.org/10.1101/2022.04.04.487055, 05 April 2022, preprint: not peer reviewed.

98. English,A.C., Menon,V.K., Gibbs,R., Metcalf,G.A. and Sedlazeck,F.J. (2022) Truvari: refined structural variant comparison preserves allelic diversity. *Genome Biol.*, **23**, 271.
99. Zook,J.M., Hansen,N.F., Olson,N.D., Chapman,L., Mullikin,J.C., Xiao,C., Sherry,S., Koren,S., Phillippy,A.M., Boutros,P.C. *et al.* (2020) A robust benchmark for detection of germline large deletions and insertions. *Nat. Biotechnol.*, **38**, 1347–1355.
100. Nurk,S., Walenz,B.P., Rhie,A., Vollger,M.R., Logsdon,G.A., Grothe,R., Miga,K.H., Eichler,E.E., Phillippy,A.M. and Koren,S. (2020) HiCanu: accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads. *Genome Res.*, **30**, 1291–1305.
101. Guo,L., Lau,J., Ruan,Z., Wei,P. and Cong,J. (2019) Hardware acceleration of long read pairwise overlapping in genome sequencing: a race between FPGA and GPU. In: *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. pp. 127–135.
102. Senol Cali,D., Kanellopoulos,K., Lindegger,J., Bingöl,Z., Kalsi,G.S., Zuo,Z., Firtina,C., Cavlak,M.B., Kim,J., Ghiasi,N.M. *et al.* (2022) SeGraM: A universal hardware accelerator for genomic sequence-to-graph and sequence-to-sequence mapping. In: *Proceedings of the 49th Annual International Symposium on Computer Architecture, ISCA '22*. Association for Computing Machinery, NY, pp. 638–655.
103. Mansouri Ghiasi,N., Park,J., Mustafa,H., Kim,J., Olgun,A., Gollwitzer,A., Senol Cali,D., Firtina,C., Mao,H., Almadhoun Alserr,N. *et al.* , (2022) GenStore: A high-performance in-storage processing system for genome sequence analysis. In: *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. Association for Computing Machinery ASPLOS, NY, pp. 635–654.
104. Shahroodi,T., Zahedi,M., Firtina,C., Alser,M., Wong,S., Mutlu,O. and Hamdioui,S. (2022) Demeter: a fast and energy-efficient food profiler using hyperdimensional computing in memory. *IEEE Access*, **10**, 82493–82510.
105. Diab,S., Nassereldine,A., Alser,M., Luna,J.G., Mutlu,O. and Hajj,I.E. (2022) High-throughput pairwise alignment with the wavefront algorithm using processing-in-memory. arXiv doi: https://arxiv.org/abs/2204.02085, 05 April 2022, preprint: not peer reviewed.
106. Khalifa,M., Ben-Hur,R., Ronen,R., Leitersdorf,O., Yavits,L. and Kvatinsky,S. (2022) FiltPIM: In-memory filter for DNA sequencing. arXiv doi: https://arxiv.org/abs/2205.15140, 30 May 2022, preprint: not peer reviewed.
107. Khatamifard,S.K., Chowdhury,Z., Pande,N., Razaviyayn,M., Kim,C.H. and Karpuzcu,U.R. (2021) GeNVoM: Read mapping near non-volatile memory. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, **19**, 3482–3496.
108. Senol Cali,D., Kalsi,G.S., Bingöl,Z., Firtina,C., Subramanian,L., Kim,J.S., Ausavarungnirun,R., Alser,M., Gomez-Luna,J., Boroumand,A. *et al.* (2020) 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)*. pp. 951–966.
109. Chen,F., Song,L., Li,H.H. and Chen,Y. (2020) PARC: A Processing-in-CAM architecture for genomic long read pairwise alignment using ReRAM. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. pp. 175–180.
110. Kaplan,R., Yavits,L. and Ginosasr,R. (2020) BioSEAL: In-memory biological sequence alignment accelerator for large-scale genomic data. In: *Proceedings of the 13th ACM International Systems and Storage Conference*. Association for Computing Machinery, Haifa, Israel, pp. 36–48.
111. Laguna,A.F., Gamaarachchi,H., Yin,X., Niemier,M., Parameswaran,S. and Hu,X.S. (2020) Seed-and-Vote based in-memory accelerator for DNA read mapping. In: *IEEE/ACM International Conference On Computer Aided Design*. IEEE, San Diego, CA, USA, pp. 1–9.
112. Angizi,S., Sun,J., Zhang,W. and Fan,D. (2020) PIM-Aligner: A processing-in-MRAM platform for biological sequence alignment. In: *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*. pp. 1265–1270.
113. Nag,A., Ramachandra,C.N., Balasubramonian,R., Stutsman,R., Giacomin,E., Kambalasubramanyam,H. and Gaillardon,P.-E. (2019) GenCache: Leveraging in-cache operators for efficient sequence alignment. In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '52*. Association for Computing Machinery, NY, pp. 334–346.
114. Kim,J.S., Senol Cali,D., Xin,H., Lee,D., Ghose,S., Alser,M., Hassan,H., Ergin,O., Alkan,C. and Mutlu,O. (2018) GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies. *BMC Genom.*, **19**, 89.
115. Sadasivan,H., Maric,M., Dawson,E., Iyer,V., Israeli,J. and Narayanasamy,S. (2022) Accelerating Minimap2 for accurate long read alignment on GPUs. bioRxiv doi: https://doi.org/10.1101/2022.03.09.483575, 10 March 2022, preprint: not peer reviewed.
116. Zeni,A., Guidi,G., Ellis,M., Ding,N., Santambrogio,M.D., Hofmeyr,S., Buluç,A., Oliker,L. and Yelick,K. (2020) LOGAN: High-performance GPU-based X-Drop long-read alignment. In: *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. pp. 462–471.
117. Goenka,S.D., Turakhia,Y., Paten,B. and Horowitz,M. (2020) SegAlign: A scalable gpu-based whole genome aligner. In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. pp. 1–13.
118. Singh,G., Alser,M., Senol Cali,D., Diamantopoulos,D., Gómez-Luna,J., Corporaal,H. and Mutlu,O. (2021) FPGA-based near-memory acceleration of modern data-intensive applications. *IEEE Micro.*, **41**, 39–48.
119. Chen,Y.-L., Chang,B.-Y., Yang,C.-H. and Chiueh,T.-D. (2021) A high-throughput FPGA accelerator for short-read mapping of the whole human genome. *IEEE Transactions on Parallel and Distributed Systems*, **32**, 1465–1478.
120. Yan,Y., Chaturvedi,N. and Appuswamy,R. (2021) Accel-Align: a fast sequence mapper and aligner based on the seed–embed–extend method. *BMC Bioinformatics*, **22**, 257.
121. Fujiki,D., Wu,S., Ozog,N., Goliya,K., Blaauw,D., Narayanasamy,S. and Das,R. (2020) SeedEx: A genome sequencing accelerator for optimal alignments in subminimal space. In: *53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, Athens, Greece, pp. 937–950.
122. Alser,M., Shahroodi,T., Gómez-Luna,J., Alkan,C. and Mutlu,O. (2020) SneakySnake: a fast and accurate universal genome pre-alignment filter for CPUs, GPUs and FPGAs. *Bioinformatics*, **36**, 5282–5290.
123. Turakhia,Y., Bejerano,G. and Dally,W.J. (2018) Darwin: A genomics Co-processor provides up to 15,000X acceleration on long read assembly. *SIGPLAN Not.*, **53**, 199–213.
124. Wang,J., Zhang,T., Song,J., Sebe,N. and Shen,H.T. (2018) A survey on learning to hash. *IEEE Trans. Pattern Anal. Mach. Intell.*, **40**, 769–790.
125. Sharma,J. and Navlakha,S. (2018) Improving similarity search with high-dimensional locality-sensitive hashing. arXiv doi: https://arxiv.org/abs/1812.01844, 05 December 2018, preprint: not peer reviewed.
126. Chen,Y., Chen,S. and Zhang,X. (2020) Using DenseFly algorithm for cell searching on massive scRNA-seq datasets. *BMC Genom.*, **21**, 222.
127. Sinha,K. and Ram,P. (2021) Fruit-Fly inspired neighborhood encoding for classification. In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, KDD '21*. Association for Computing Machinery, NY, pp. 1470–1480.
128. Dasgupta,S., Stevens,C.F. and Navlakha,S. (2017) A neural algorithm for a fundamental computing problem. *Science*, **358**, 793–796.