

## Sequence analysis

# SKiM: accurately classifying metagenomic ONT reads in limited memory

Trevor Schneggenburger<sup>1</sup> and Jaroslaw Zola<sup>\*,1</sup>

<sup>1</sup>Department of Computer Science and Engineering, University at Buffalo, Buffalo, NY 14260, United States

\*Corresponding author. Department of Computer Science and Engineering, University at Buffalo, Davis Hall, Buffalo, NY 14260, United States.  
E-mail: jzola@buffalo.edu.

Associate Editor: Can Alkan

## Abstract

**Motivation:** Oxford Nanopore Technologies' devices, such as MinION, permit affordable, real-time DNA sequencing, and come with targeted sequencing capabilities. Such capabilities create new challenges for metagenomic classifiers that must be computationally efficient yet robust enough to handle potentially erroneous DNA reads, while ideally inspecting only a few hundred bases of a read. Currently available DNA classifiers leave room for improvement with respect to classification accuracy, memory usage, and the ability to operate in targeted sequencing scenarios.

**Results:** We present SKiM: Short K-mers in Metagenomics, a new lightweight metagenomic classifier designed for ONT reads. Compared to state-of-the-art classifiers, SKiM requires only a fraction of memory to run, and can classify DNA reads with higher accuracy after inspecting only their first few hundred bases. To achieve this, SKiM introduces new data compression techniques to maintain a reference database built from short *k*-mers, and treats classification as a statistical testing problem.

**Availability and implementation:** SKiM source code, documentation, and test data are available from: <https://gitlab.com/SCoRe-Group/skim>.

## 1 Introduction

Oxford Nanopore Technologies (ONTs) sequencing devices offer long-read DNA sequencing capabilities and can stream their sequencing data in real-time, thus allowing analysis to occur in step with sequencing. Real-time DNA analysis opens new possibilities, including mobile in-the-field deployments (Quick *et al.* 2016, Ko *et al.* 2018, Mikalsen and Zola 2023), as actionable information becomes available during sequencing. Such possibilities eliminate the usual delays of classic batch sequencing (Wang *et al.* 2021).

Real-time analysis becomes even more useful when combined with the ONT adaptive sampling feature (formerly "ReadUntil") (Loose *et al.* 2016). With adaptive sampling, a sequencer can eject a DNA molecule from a specific nanopore while it is being sequenced. After DNA ejection, the nanopore is freed up to sequence a different DNA molecule, allowing adaptive sampling to affect the content of a sequencing experiment *in silico* and improve the sequencing yield of targets of interest. So far, adaptive sampling has been used to enrich samples for low-abundance species (Martin *et al.* 2022), to enrich antimicrobial resistance (AMR) genes (Wrenn and Drown 2023, Ulrich *et al.* 2024), and to carry out metagenomic surveillance (Kipp *et al.* 2023), to name a few applications.

However, the effectiveness of adaptive sampling depends on the analysis that decides whether or not to eject a read. In the case where a read is not of interest, it is ideal to eject it as early as possible. In this way, both the hardware and software spend minimal time processing reads that are not of

interest. Therefore, this analysis must be efficient and accurate to handle the short starting fragments of a read.

Although the applications and benefits of real-time analysis are clear, it has some additional constraints. First, accuracy must be balanced with speed. In the worst case, the analysis stage must be able to process data at least as fast as the sequencing and basecalling, so as to not cause a bottleneck in the pipeline. Second, many of the applications of real-time analysis are done on-site. Although streaming data for off-site processing is sometimes possible, on-site applications are likely to have constrained computer hardware available, especially memory and storage. These two problems are further complicated by the rapidly increasing number of reference assemblies available (O'Leary *et al.* 2016).

Currently, real-time analysis and the read-ejection problem are solved in one of three ways. The first strategy is alignment of reads to a small number of reference sequences, for example, ONT's sequencing software (MinKNOW), which uses minimap2 (Li 2018) to perform alignment. While alignment is likely to produce correct matches, it is slow in practice and cannot be performed against a large number of reference sequences in real time. The second approach is to depend on a traditional classifier, for example, Readfish (Payne *et al.* 2021), which can use Centrifuge (Kim *et al.* 2016) to perform classification. However, traditional classifiers are designed to work with complete reads and are computationally expensive, requiring a significant amount of RAM to run with many reference sequences. The emerging third approach is to use a classifier designed for ONT real-time analysis and adaptive sampling, for example, SPUMONI (Ahmed *et al.* 2021),

preceded by SPUMONI 2 (Ahmed *et al.* 2023), or ReadBouncer (Ulrich *et al.* 2022), preceded by Taxor (Ulrich and Renard 2024). SPUMONI and SPUMONI 2 are both designed for a binary classification problem on pangenome references, i.e. they only indicate if the queried read is present in or absent from the pangenome reference database. While this information is enough for the read-ejection problem, it is insufficient for general real-time analyses, e.g. sample abundance estimation or pathogen identification without *a priori* knowledge of the pathogen. The ideas presented in Taxor directly build off those from ReadBouncer, a classifier designed for real-time classification. However, we believe there is still room for improvement over both these tools in terms of runtime, index size, and classification accuracy.

In this article, we introduce SKiM: Short K-mers in Metagenomics. SKiM is a  $k$ -mer-based metagenomic classifier designed from the ground up to address the challenges of classifying ONT reads during sequencing. Compared to popular traditional classifiers such as Kraken2 (Wood *et al.* 2019) or Centrifuge (Kim *et al.* 2016), SKiM requires only a fraction of memory to run, and can classify a DNA read with high accuracy after inspecting only its first few hundred bases. Moreover, it maintains a classification throughput comparable to that of other tools. To achieve this, SKiM uses short  $k$ -mers ( $k = 15$  or  $k = 16$ ), data compression techniques tailored for large reference databases, and statistical correction to test significance in  $k$ -mers matching.

## 2 Materials and methods

As a  $k$ -mer based classifier, SKiM works by decomposing both the input reads and the reference sequences into  $k$ -mers. What differentiates it from other  $k$ -mer based classifiers is how the database is stored and how  $k$ -mer queries are handled.

### 2.1 Database construction

To construct a SKiM database, we begin with an input collection,  $A$ , of reference genome assemblies that represent classification targets. Each assembly  $a_i \in A$  consists of one or more DNA strings, which may correspond to chromosomes, plasmids, etc. Typically, such input data will be represented by a set of FASTA files, where each file stores a single assembly  $a_i$ .

We convert each assembly  $a_i \in A$  into a set,  $K_i$ , of all  $k$ -mers that occur in  $a_i$ . Importantly, each  $K_i$  is strictly a set, not a multiset (i.e.  $K_i$  stores only one copy of every found  $k$ -mer). We want to represent each  $K_i$  as a bitmap. To do so, we map each  $k$ -mer onto an integer in the range  $[0, 4^k)$  using a bijective function  $f$ . While any function can be used here, in practice,  $f$  is such that each DNA base is assigned a 2-bit integer, and we concatenate all 2-bit integers for a  $k$ -mer into a  $2k$ -bit integer. With this function  $f$ ,  $K_i$  becomes a bitmap where  $K_i[f(j)] = 1$  iff the  $k$ -mer  $j$  is in the set  $K_i$  and 0 otherwise. Furthermore, to represent the entire collection of assemblies  $A$ , we combine all bitmaps into a  $4^k \times |A|$  matrix,  $M$ , where rows of  $M$  are indexed by  $f$  and columns are indexed by labels  $K_i$ .  $M$  is the complete, uncompressed input database for SKiM. An example matrix  $M$  is given in Fig. 1 (left). For simplicity, we use all  $k$ -mers in this example. In implementation, we only consider canonical  $k$ -mers (defined as the lexicographic minimum between a  $k$ -mer and its reverse complement)—a common way to account for DNA strandedness.

In its basic form, matrix  $M$  is memory inefficient. For example, all archaeal, bacterial, and viral genomes from NCBI RefSeq, which we will refer to as ABV, currently total  $|A| \approx 50,000$  entries. Even with  $k = 15$ , this gives us more than 536 million distinct canonical  $k$ -mers, and  $M$  would require more than 3TB of memory to store.

One standard way to address this is to sub-sample the  $k$ -mers when forming the matrix (e.g. using minimizers (Roberts *et al.*

|    | $K_1$ | $K_2$ | $K_3$ | $K_4$ | $K_5$ | $K_6$ |
|----|-------|-------|-------|-------|-------|-------|
| AA | 1     | 0     | 0     | 1     | 0     | 0     |
| AC | 0     | 1     | 1     | 1     | 0     | 0     |
| AG | 0     | 1     | 0     | 0     | 1     | 0     |
| AT | 1     | 1     | 0     | 1     | 0     | 0     |
| CA | 0     | 1     | 0     | 1     | 0     | 0     |
| CC | 0     | 0     | 0     | 0     | 0     | 1     |
| CG | 0     | 0     | 1     | 0     | 0     | 1     |
| CT | 1     | 0     | 1     | 1     | 1     | 1     |
| GA | 0     | 1     | 1     | 0     | 1     | 0     |
| GC | 0     | 0     | 1     | 0     | 0     | 1     |
| GG | 0     | 0     | 1     | 0     | 0     | 1     |
| GT | 0     | 0     | 0     | 0     | 0     | 0     |
| TA | 1     | 1     | 0     | 1     | 1     | 0     |
| TC | 1     | 0     | 0     | 0     | 1     | 0     |
| TG | 0     | 0     | 0     | 0     | 0     | 1     |
| TT | 0     | 0     | 0     | 0     | 0     | 0     |

|    | $K_1$ | $K_4$ | $K_2$ | $K_5$ | $K_3$ | $K_6$ |
|----|-------|-------|-------|-------|-------|-------|
| AA | 1     | 1     | 0     | 0     | 0     | 0     |
| AC | 0     | 1     | 1     | 0     | 1     | 0     |
| AG | 0     | 0     | 1     | 1     | 0     | 0     |
| AT | 1     | 1     | 1     | 0     | 0     | 0     |
| CA | 0     | 1     | 1     | 0     | 0     | 0     |
| CC | 0     | 0     | 0     | 0     | 0     | 1     |
| CG | 0     | 0     | 0     | 0     | 1     | 1     |
| CT | 1     | 1     | 0     | 1     | 1     | 1     |
| GA | 0     | 0     | 1     | 1     | 1     | 0     |
| GC | 0     | 0     | 0     | 0     | 1     | 1     |
| GG | 0     | 0     | 0     | 0     | 1     | 1     |
| GT | 0     | 0     | 0     | 0     | 0     | 0     |
| TA | 1     | 1     | 1     | 1     | 0     | 0     |
| TC | 1     | 0     | 0     | 1     | 0     | 0     |
| TG | 0     | 0     | 0     | 0     | 0     | 1     |
| TT | 0     | 0     | 0     | 0     | 0     | 0     |

**Figure 1.** An example SKiM database with assemblies  $a_1 = \text{AATCTAA}$ ,  $a_2 = \text{TAGACAT}$ ,  $a_3 = \text{GGACGCT}$ ,  $a_4 = \text{CATAACT}$ ,  $a_5 = \text{CTCTAGA}$ ,  $a_6 = \text{CCGCTGG}$ , and  $k$ -mers of size  $k = 2$ . Left: initial matrix  $M$ . Right: matrix  $M'$  obtained by reordering columns of  $M$  using a greedy heuristic (see Section 2.2). Runs of three or more of the same bit are highlighted.

2004) or winnowing (Schleimer *et al.* 2003)). In our approach, we decided to use syncmers, which have been shown to better conserve biological sequences that have errors while having a lower density than minimizers (Edgar 2021). Specifically, a  $k$ -mer is a syncmer if its smallest  $s$ -mer (substring of length  $s < k$ ) starts exactly at position  $t$  within the  $k$ -mer, where  $t$  is an input parameter. In our case, we chose the default values of  $k = 15$ ,  $s = 9$ , and  $t = 2$  (zero indexed) based on the results from Edgar (2021). This enables us to reduce the number of stored  $k$ -mers by approximately 85%.

While sub-sampling provides significant memory reduction, it is not sufficient on its own. For example, the sub-sampled ABV matrix would still require around 400 GB to store. Hence, to further reduce the space requirements of  $M$ , we take advantage of two key observations. The first observation is that  $M$  is sparse. For instance, only about 15.5 billion bits are set in ABV, which is  $<0.5\%$  of the total bits in the matrix. The second observation is that  $M$  stores the same information regardless of the ordering of the columns, provided that the columns are properly labeled by each  $K_i$ . That is, each  $K_i$  can occupy an arbitrary column in  $M$  without affecting the way queries are processed during classification. As a consequence, we are free to decide which permutation of the columns to use. With these observations, we seek to design a compression scheme to further reduce the storage requirements of  $M$ .

## 2.2 Lossless database compression

Our goal now is to maximally compress the matrix  $M$  to reduce space requirements while still allowing the minimum possible theoretical runtime of queries. More specifically, the database will be queried with  $k$ -mers coming from DNA reads being classified (see Section 2.3). For each  $k$ -mer,  $j$ , that we query, the database should be able to retrieve all assemblies that  $j$  occurs in. As a lower runtime bound for such queries, consider that if  $k$ -mer  $j$  occurs in  $t_j$  assemblies where  $0 \leq t_j \leq |A|$ , any algorithm that delivers all  $t_j$  assemblies must take  $\Omega(t_j)$  runtime. With this bound in mind, we consider the fact that each  $k$ -mer row in  $M$  is also a bitmap, much like each column  $K_i$ , and has a set bit in each column (assembly) in which the  $k$ -mer occurs. Thus, we can compress  $M$  horizontally, i.e. per row (per  $k$ -mer), ensuring that the compression scheme allows rows to be iterable in  $O(t_j)$ . We note that compression in a similar context has been explored before in Taxor (Ulrich and Renard 2024). However, our approach differs in that we try to compress each row of  $M$  (that is, compression per  $k$ -mer), while Taxor tries to compress each column of the matrix (that is, compression per assembly). Hence, to perform the desired queries Taxor’s approach must take  $\Omega(|A|)$  in the worst case, or on the order of the number of indexed assemblies.

We decided to use a run-length encoding (RLE) for the row-wise compression of  $M$ , with two main considerations in mind. First, many compression schemes, e.g. succinct data structures (Navarro 2016) or roaring bitmaps (Chambi *et al.* 2016), are general purpose. This implies that auxiliary information may be stored to support lookup operations that we do not need (e.g. rank or select), which is contrary to our goal of efficiently compressing  $M$ . In contrast, a run-length encoding only stores the information necessary to iterate the bitmap, and it does so in the time constraints previously described. Second, there is an intuitive way to reorder columns in  $M$  to improve RLE compression effectiveness, which lends itself to elegant formalization. More specifically, we can try

to choose an ordering of columns that creates many long runs of the same bit within the rows of  $M$ . Such an ordering would allow an RLE representation to encode large portions of the matrix in a small amount of space. This intuition leads to clear and well-understood criteria (Olken and Rotem 1985, Johnson *et al.* 2004) to find a column ordering of  $M$  such that the compression effectiveness is improved.

To explain our compression scheme, we start with the idea of a “naive” RLE (NRLE). An NRLE represents a bitmap as a list of runs, where each run corresponds to a continuous sequence of the same bit, and it encodes information about the type of bit and the number of occurrences of that bit (length of the run). For example, row CC of the left matrix in Fig. 1 would consist of a run of zeros of length 5, followed by a run of ones of length 1. To achieve close to optimal compression with NRLE, we want  $M$  to have the consecutive ones property (Booth and Lueker 1976). This property is satisfied if there exists an ordering of the columns of  $M$  such that all ones (that is, set bits) in each row occur consecutively. In this case, each row of  $M$  would require at most three runs to store, and no more than  $\log(|A|)$  bits to encode a run. Interestingly, there exists a linear-time algorithm to test a binary matrix for the consecutive ones property using a PQ-tree data structure (Booth and Lueker 1976). Unfortunately, in practice, we found that  $M$  is too complex to have the property even for  $|A| \approx 10$ . Hence, we consider the following optimization problem instead: Let  $\text{Naive-Runs}(M)$  be the total number of runs required to represent all rows of the matrix  $M$  under NRLE. We want to find a new matrix  $M'$  by permuting the columns of  $M$  so that  $\text{Naive-Runs}(M')$  is minimized.

The approach to this optimization problem has been first described by Johnson *et al.* (2004), and an early variant was considered by Olken and Rotem (1985). Briefly, we can imagine each column  $K_i$  of  $M$  as a point in a high-dimensional Hamming space. The Hamming distance  $d(K_i, K_j)$  between two bitmaps  $K_i$  and  $K_j$  in this space represents the number of bits that need to be substituted to edit one bitmap into the other. Imagine that we are constructing NRLEs for all rows in  $M$ , and we have considered bits up to column  $K_i$ . Then, the additional number of runs needed to add a column  $K_{i+1}$  to the compression is exactly  $d(K_i, K_{i+1})$ , i.e. one new run for each bit that differs from  $K_i$  to  $K_{i+1}$ . All other runs can simply be extended in length, and hence would not induce additional runs. Following this logic,  $\text{Naive-Runs}(M)$  is equivalent to the sum  $\sum d(K_i, K_{i+1})$ , or the sum of all distances between consecutive columns in  $M$ . This optimization problem can then be shown to be  $\mathcal{NP}$ -hard via a reduction from Hamiltonian Path (Johnson *et al.* 2004).

Due to its computational complexity, we solve NAIVE-RUNS in SKiM by treating it as an instance of the Traveling Salesperson Problem (TSP), and we apply a simple TSP greedy heuristic (the nearest-neighbor heuristic): Starting with a random point (column), at each iteration, we add the closest point to the last added one (ignoring points that have already been added). An example result of reordering columns according to this heuristic is shown in Fig. 1 (right). In this example, the reordered matrix exhibits more runs consisting of three or more bits. Moreover, we can verify that  $\text{Naive-Runs}(M) = 55$  while  $\text{Naive-Runs}(M') = 39$ .

Applying a TSP heuristic implies that we need to compute the pairwise distance between each pair of assembly bitmaps ( $K_i, K_j$ ). As stated previously, the matrix  $M$  is too large to fit into memory in its raw form. In practice, we load columns (assemblies) of the matrix as roaring bitmaps (Chambi *et al.* 2016), thus applying a generalized compression scheme to

reduce the peak memory usage of index construction. For an example about how this impacts index construction runtime and memory usage, see Section 4.

The number of runs required to represent  $M'$  can be provably reduced further. We achieve this by adjusting our compression scheme to include uncompressed “runs” of bits when it is inefficient to represent them as runs of zeros or ones, that is, encoding their length requires more bits than they take in their uncompressed form. More specifically, let  $w$  be the size of a word (e.g.  $w = 16$ ) to store a run in the RLE. A common weakness of RLEs is that very short runs from the original string, e.g. consisting of a single bit, must still be encoded using  $w$  bits. We bypass this limitation by allowing uncompressed “runs” within the RLE that simply store  $w$  bits directly as they appear in the original bitmap. We will refer to this new RLE scheme as an Adaptive RLE (ARLE). With the three types of runs (run of zeros, run of ones, uncompressed), we can use [Algorithm 1](#) to encode an arbitrary bit string,  $B$ , in the fewest possible number of runs.

**Algorithm 1** Encode-as-ARLE( $B, w$ )

```

1:  $i \leftarrow 0$ 
2: ARLE  $\leftarrow []$ 
3: while  $i < |B|$  do
4:    $j \leftarrow i + 1$ 
5:   while  $j < |B|$  and  $B[j] = B[i]$  and  $(j - i) < 2^{w-2} - 1$  do
6:      $j \leftarrow j + 1$ 
7:   if  $j < i + w$  then
8:      $k \leftarrow \min(i + w, |B|)$ 
9:     ARLE.push(Raw( $B[i..k]$ ))
10:     $i \leftarrow k$ 
11:   else
12:     ARLE.push(Encode( $B[i..j]$ ))
13:    $i \leftarrow j$ 
14: return ARLE

```

To compress the input bitmap  $B$ , the algorithm starts at the beginning of the bitmap ( $i = 0$ ), looks for the next bit that is opposite to the one at the current position  $i$ . If such a bit occurs at a position  $j < i + w$ , then it is more effective to store the run  $B[i..j]$  uncompressed (call to the function Raw in line 9). Otherwise, the run benefits from compression as a run of zeros or ones, and it is stored in encoded form (line 12). In SKiM, we use  $w = 16$  with one bit reserved to differentiate compressed and uncompressed runs, and then one more bit only for compressed runs to indicate whether it stores zeros or ones. With this, each uncompressed run represents up to  $w - 1$  bits from  $B$ , and each compressed run encodes up to  $2^{w-2} - 1$  bits (without loss of generality, runs with more than  $2^{w-2} - 1$  bits are treated as separate runs).

Our ARLE scheme provably improves over NRLE and leads to [Theorem 1](#) (we provide all proofs in the [Supplementary Information](#), available as [supplementary data](#) at [Bioinformatics](#) online):

**Theorem 1.** Let  $\text{Adaptive} - \text{Blocks}(M)$  be the total number of runs required to represent any bit matrix  $M$ , obtained by running [Algorithm 1](#) on each of the rows

of  $M$ . Then  $\text{Adaptive} - \text{Blocks}(M) \leq \text{Naive} - \text{Runs}(M)$  for any bit matrix  $M$ .

We apply [Theorem 1](#) to matrix  $M'$ , i.e. we run [Algorithm 1](#) on each row of  $M'$ , to arrive at our final encoding. A summary of the complete SKiM database construction is as follows: First, each input assembly  $a_i \in A$  is converted to a bitmap  $K_i$  by enumerating all sub-sampled canonical  $k$ -mers present in  $a_i$ . Next, we compute the pairwise edit distances between all pairs of bitmaps  $(K_i, K_j)$ . These distances become an input to the greedy nearest-neighbor heuristic to find an ordering of all columns to generate matrix  $M'$ . With this ordering, we compute an ARLE for each row in the matrix  $M'$  according to [Algorithm 1](#). The resulting compressed matrix is the SKiM reference database.

### 2.3 Classification

The use of short  $k$ -mers allows us to drastically reduce the size of the SKiM reference database. Moreover, in a sequence containing errors, a short  $k$ -mer is more likely to be error-free than a longer  $k$ -mer. Thus, shorter  $k$ -mers have an advantage during adaptive sampling, where we want to classify a sequence after inspecting only a limited number of its nucleotides. However, it also creates challenges. First, short  $k$ -mers are more likely to match with any reference sequence just by chance. For example, assuming our reference assembly set is  $A$ , if we try to classify a read that is derived from a genome not in  $A$ , there is a high chance that the read will contain a  $k$ -mer in  $A$  and the read will be incorrectly classified. Second, false positive  $k$ -mer matching may introduce a significant bias toward larger reference sequences. For example, a bacterial sequence may contain  $100 \times$  as many  $k$ -mers as a viral sequence. In such a scenario, intuitively, 10 matches to the bacterial sequence are less significant than 9 matches to the viral sequence.

To address these challenges, we model the read classification as a binomial experiment. We assume that the incoming DNA read is a random sequence of nucleotides, and hence the  $k$ -mers are random as well. We also assume that all  $k$ -mers from the read are independent, despite the fact that some will contain overlapping base pairs. Lastly, for both reads and references, we consider only canonical syncmers, which form a subset,  $U$ , of all possible  $4^k$   $k$ -mers. Given these assumptions, the probability that a random  $k$ -mer from the read is in the  $k$ -mer set  $K_i$  of assembly  $a_i$ , is simply  $p_i = \frac{|K_i|}{|U|}$ . For the remainder of this section, we will refer to the canonical syncmers extracted from a read simply as  $k$ -mers.

When classifying a read, for each assembly  $a_i$  we assume a null hypothesis,  $H_{i0}$ , that the matches between the read's  $k$ -mers and the assembly  $a_i$  are not statistically significant. Then, for a given read, we enumerate all its  $k$ -mers, and use each enumerated  $k$ -mer to query our reference database. For each assembly  $a_i$  we maintain a counter,  $x_i$ , that indicates the number of times a  $k$ -mer from the read matched to  $a_i$ . When there are no more  $k$ -mers in the read, we calculate the  $P$ -value of observing  $x_i$  or more  $k$ -mer matches for each  $a_i$  under the assumption of a binomial experiment with statistic  $P(\text{Bin}(n, p_i) \geq x_i)$ , where  $n$  is the total number of  $k$ -mers extracted from the read being classified. Put differently, for a read with  $n$   $k$ -mers,  $P(\text{Bin}(n, p_i) \geq x_i)$  gives us the probability that we observed  $x_i$  or more  $k$ -mer matches with  $a_i$ , given that the probability of match success for  $a_i$  is  $p_i$ .

If the observed  $P$ -value for any  $a_i$  is lower than a predefined cutoff threshold, we reject the null hypothesis. We conclude that this is good evidence that the matching of  $k$ -mers between the read and the assembly  $a_i$  is significant, and we classify the read as coming from  $a_i$ . If more than one null hypothesis is rejected, we choose the assembly with the lowest  $P$ -value.

We further modify this procedure to account for the fact that ONT reads usually have drastically different lengths, resulting in significant variability in the number of  $k$ -mers,  $n$ , extracted from each read. Since  $n$  is the number of binomial trials in our tests, it can affect the comparability of the corresponding  $P$ -values. To address this, instead of considering  $n$  directly, we calculate our classification probability  $P$  based on a maximum likelihood estimate for the number of matching  $k$ -mers expected if exactly  $n_{\text{fixed}}$   $k$ -mers were considered.

More precisely, we use  $P(\text{Bin}(n_{\text{fixed}}, p_i) \geq \bar{x}_i)$  to obtain the  $P$ -value for  $a_i$ , where  $\bar{x}_i = \lfloor \frac{x_i}{n} \cdot n_{\text{fixed}} \rfloor$ . If less than  $n_{\text{fixed}}$   $k$ -mers were queried, we do not calculate  $\bar{x}_i$  (we simply use  $\bar{x}_i = x_i$ ), but we still use  $n_{\text{fixed}}$  to calculate the  $P$ -value (i.e.  $P(\text{Bin}(n_{\text{fixed}}, p_i) \geq x_i)$ ). The reason for this is explained in Section 4. Overall, this makes a single cutoff threshold more comparable between reads of differing lengths. Moreover, because  $n_{\text{fixed}}$  can be relatively small, for instance we use  $n_{\text{fixed}} = 100$  by default, it allows us to reduce runtime by pre-computing a  $P$ -value lookup table for all possible number of matches in the range  $[0, n_{\text{fixed}}]$  for each assembly  $a_i$ .

Putting this all together, the complete classification process for a single read works as follows: First, for each assembly  $a_i$  we pre-compute both  $p_i$  and the  $P$ -value lookup table. Each  $p_i$  is stored in the SKiM database, and the lookup table is quickly calculated at runtime using the  $n_{\text{fixed}}$  parameter. For each  $k$ -mer in the read, we use the bijection  $f$  to find its corresponding ARLE-compressed row in our reference database. For each  $a_i$  present in the row, we increment counter  $x_i$ , and once all  $n$   $k$ -mers from the read have been searched, for each  $x_i$  we calculate  $\bar{x}_i = \lfloor \frac{x_i}{n} \cdot n_{\text{fixed}} \rfloor$  (if needed) and use  $\bar{x}_i$  to lookup the pre-computed  $P$ -value. If the smallest  $P$ -value found is lower than the predefined cutoff threshold, we classify the read as coming from the assembly  $a_i$  for which we obtained that  $P$ -value. Otherwise, we report the read as unclassified. Here, the cutoff threshold is defined as  $10^{-e}$ , where  $e$  is user-defined at runtime ( $e = 12$  by default).

### 3 Results

To assess the effectiveness and performance of our proposed solutions, we compared SKiM with other metagenomic classifiers under conditions that mimic adaptive sampling and real-time classification applications. In the tests, we used Centrifuge 1.0.4 (Kim *et al.* 2016), Kraken2 2.1.3 (Wood *et al.* 2019), KrakenUniq 1.0.4 (Breitwieser *et al.* 2018), and CLARK 1.2.6 (Ounit *et al.* 2015). We tried to compare to the more recent Taxor 0.2.1 (Ulrich and Renard 2024), however, we found it unable to perform classification in any tests we conducted.

We created four SKiM databases with different sub-sampling parameters to observe the impacts of these parameters on both classification and performance. When reporting results, we use SKiM to denote a reference database constructed with the default parameters ( $k = 15$ ,  $s = 9$  and  $t = 2$ ), and SKiM- $k$ - $s$ - $t$  when using other configurations. For example, SKiM- $k_{15}$ - $s_{13}$ - $t_1$  refers to the configuration with

$k = 15$ ,  $s = 13$ , and  $t = 1$ . In all SKiM classification experiments, we use the cutoff value of  $10^{-12}$ .

For all other  $k$ -mer based tools, we tested multiple sets of parameters as well. Kraken2- $k$ - $l$ - $g$  denotes a Kraken2 reference database with  $k$ -mer length  $k$ , minimizer length  $l$ , and minimizer spaces  $g$  ( $g = 0$  if absent), and we use Kraken2 to denote the default parameters ( $k = 35$ ,  $l = 31$ ,  $g = 7$ ). Because Kraken2 uses minimizers for indexing, not  $k$ -mers, in our tests we matched the minimizer length to  $k$ -mer length in other tools. KrakenUniq- $k$ - $l$  denotes a KrakenUniq reference database with  $k$ -mer length  $k$  and minimizer length  $l$ , and we use KrakenUniq to denote the default parameters ( $k = 31$ ,  $l = 15$ ). CLARK- $k$  denotes a CLARK reference database with  $k$ -mer length  $k$ , and we use CLARK to denote the default ( $k = 31$ ). Where possible, we selected alternate parameters that sub-sample to 12% of  $k$ -mers to match SKiM's default parameters. For Centrifuge, we limited the number of classification results reported per read to one (using the `-k 1` switch).

#### 3.1 Test data

In all tests, we ran all the tools with the same reference database. This database consisted of all archaeal, bacterial, and viral sequences from NCBI RefSeq with the annotation "Complete Genome," downloaded together with the NCBI taxonomy on 12 June 2023. We further extended the database with 12 reference sequences for the synthetic mock microbial community from Sevim *et al.* (2019) that were not covered by the NCBI sequences. We refer to the resulting database as ABV.

This database represents a large, quality reference suitable for a comprehensive metagenomic classification without prior knowledge of the composition of the sequenced samples. Collectively, it spans 48 214 assemblies that take 145 GB when stored in FASTA files.

As input data, we used reads from three actual ONT metagenomic sequencing experiments. The first dataset was from the GridION sequencing of a ZymoBIOMICS Microbial Community Standard with known composition, publicly available from Nicholls *et al.* (2019) (European Read Archive accession ERR2887847). This standard contains eight species of bacteria and two species of yeast, where all bacteria have even abundance. The dataset has 3 491 390 reads and contains raw ONT signals. This enabled us to re-basecall the raw signals with Dorado using model `dna_r9.4.1_e8_fast@v3.4` (originally, the reads were basecalled with Guppy, an earlier generation of ONT basecallers). We used the fast model as it is the most likely choice when performing basecalling on the computationally lightweight hardware we target. We will refer to this dataset as Even.

For the second dataset, we used MinION sequencing from Leidenfrost *et al.* (2020) where the sequencing sample was a synthetic mock community of 12 bacteria with even abundance. The reads were originally basecalled with Albacore, another early generation ONT basecaller with a higher error rate than Dorado. Since the raw ONT signals were provided, we re-basecalled and demultiplexed them with Dorado (model `dna_r9.4.1_e8_fast@v3.4`) as well. The resulting dataset, which we will refer to as Bench, contains 431 287 reads.

For the third dataset, we used the MinION sequencing from Sevim *et al.* (2019) (NCBI Sequence Read Archive accession SRX4901586), where the sequencing sample was another synthetic mock community of 12 bacteria. More specifically, the dataset contains 10 kb size-selected reads (i.e.

only reads longer than 10 kb), and the abundance of each bacteria is uneven, although no single species accounts for the majority of reads. The dataset has 187 507 reads base-called with Albacore, but did not include raw ONT signals that we could re-basecall. We will refer to this dataset as BMock12-10kb.

To mimic the classification conditions of an adaptive sampling experiment, we took each of the three datasets and generated new datasets that consisted of all the same reads, but with a constrained maximum read length  $\ell$ . During classification, for every read longer than the maximum read length  $\ell$ , we considered only the first  $\ell$  base pairs, and for reads shorter than  $\ell$ , we considered the entire read. ONT's current protocol for adaptive sampling only provides a single chunk at the start of the read to the base-caller and classifier. By default, this chunk is about 400 bp, or  $\sim 1$  second of sequencing (<https://nanoporetech.com/document/adaptive-sampling#targeting-and-buffering-advanced>). If the decision to eject the read cannot be made based on this first chunk, the read will continue sequencing as normal. It has also been noted that the reported optimal fragment size when working with the ONT interface for adaptive sampling is around 180 bp (see, e.g. [Payne et al. 2021](#)). Therefore, to test what chunk size is required for an accurate classification, we consider  $\ell$  of 180, 360, 720, and 1440 bp, along with the unconstrained full read length.

### 3.2 Classification effectiveness

To assess the effectiveness of each classifier, we used commonly accepted statistics, used by, for example, CAMI ([Li 2018](#)). Specifically, to establish a reference point (ground truth) for the tests, we used minimap2 ([Li 2018](#)), with ONT-specific parameters, and aligned the full input reads against only those reference sequences that should be present in a sample based on its known composition. If a given read did not have a significant alignment or if there were alignments of the same quality to multiple references, we discarded it from the subsequent analysis. Otherwise, we assigned its ground truth classification to be the taxonomic identifier of the reference sequence with the most significant alignment.

We counted a classification as a true positive (TP) when its classifier-assigned identifier matched the ground truth identifier, a false positive (FP) when the classifier-assigned identifier was different from the ground truth, a false negative (FN) when the classifier did not assign any identifier, but the ground truth identifier was known, and finally, a true

negative (TN) when the classifier did not assign an identifier and the ground truth identifier was not in the classifier's reference database. We note that TNs occur only in the case of the Even dataset, since this dataset contains yeast sequences that we did not include in the ABV reference. Hence, any read classified as yeast by minimap2 would ideally remain unclassified by any classifier being tested.

To assess classification effectiveness at different taxonomic levels (i.e. species, genus), we relied on the NCBI taxonomy. This required additional care to account for sporadic situations where some species-level taxonomic identifiers have no genus-level assignment (i.e. their parent node in the taxonomic tree is labeled as "no rank"). For all such cases, we assumed that the genus-level identifier is the parent identifier of the selected species-level identifier.

The results of our experiments are summarized in [Table 1](#) and [Figs 2–4](#), as well as [Figs 1–3](#), available as [supplementary data](#) at *Bioinformatics* online. The [Supplementary Excel](#), available as [supplementary data](#) at *Bioinformatics* online file contains a complete set of raw statistics for all experiments as well. We define Recall as  $\frac{TP}{TP+FN}$ , Precision as  $\frac{TP}{TP+FP}$ , and Accuracy as  $\frac{TP+TN}{TP+FP+FN+TN}$ .

### 3.3 Performance statistics

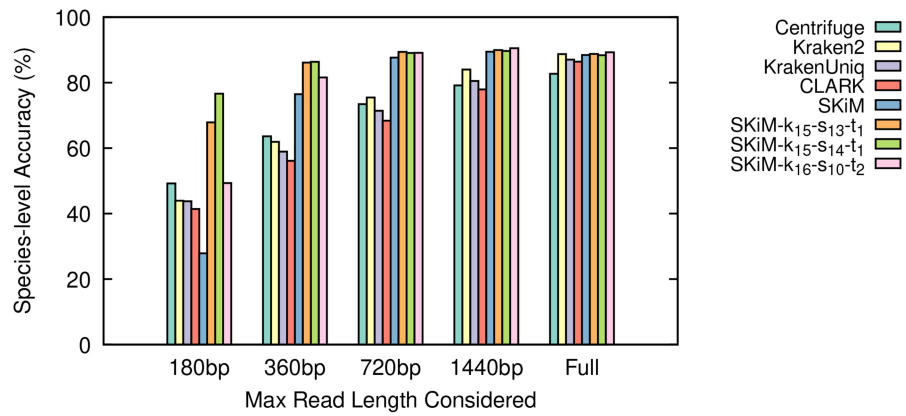
To assess the computational performance of the tested classifiers, we ran all tests on a server with 512 GB of RAM and an AMD EPYC 7203P 8-core processor that supports 8 threads. To report the peak memory usage of each classifier when the classifier is running, we extracted the maximum resident set size (RSS) provided by the `time -v` UNIX tool. To obtain the classification throughput, we used the self-reported throughput for the classifiers that provided such information, or we calculated it based on the walltime used (excluding the time to load the reference database). We also included the performance of minimap2 using the same reference database as a comparison. These results are summarized in [Tables 2](#) and [3](#). The throughput tables for Bench and BMock12-10kb, which are almost identical to [Table 2](#), are available in [Tables 1](#) and [2](#), available as [supplementary data](#) at *Bioinformatics* online.

To understand the performance of our proposed compression method, we created variants of the default SKiM database using: a random column ordering of input assemblies and the NRLE compression scheme (denoted as Random NRLE), a random column ordering and our ARLE

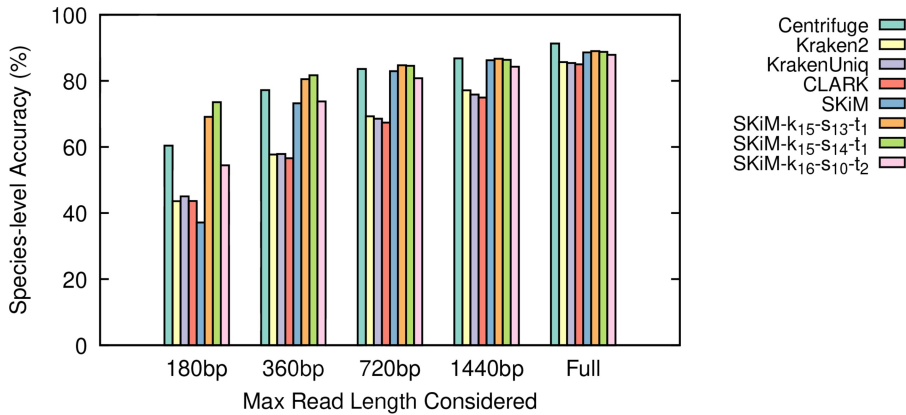
**Table 1.** Classifier statistics for the Even dataset when  $\ell = 360$ bp.<sup>a</sup>

| Classifier                                               | Genus        |              |              |                      | Species      |              |              |                      |
|----------------------------------------------------------|--------------|--------------|--------------|----------------------|--------------|--------------|--------------|----------------------|
|                                                          | Recall       | Precision    | Accuracy     | F <sub>1</sub> score | Recall       | Precision    | Accuracy     | F <sub>1</sub> score |
| Centrifuge                                               | 81.36        | 90.82        | 76.06        | 85.83                | 67.67        | 88.54        | 63.58        | 76.72                |
| Kraken2                                                  | 86.71        | 96.63        | 84.88        | 91.40                | 63.18        | 92.49        | 61.91        | 75.08                |
| Kraken2-k <sub>37</sub> -l <sub>27</sub> -g <sub>3</sub> | 85.54        | 96.79        | 83.94        | 90.81                | 56.77        | <b>93.79</b> | 56.46        | 70.44                |
| Kraken2-k <sub>32</sub> -l <sub>22</sub>                 | 89.35        | 95.42        | 86.27        | 92.28                | 59.71        | 90.92        | 59.19        | 72.08                |
| KrakenUniq                                               | 87.41        | <b>97.02</b> | 85.81        | 91.96                | 59.22        | 93.73        | 58.95        | 72.59                |
| KrakenUniq-k <sub>27</sub> -l <sub>13</sub>              | 90.41        | 95.43        | 87.15        | 92.85                | 63.11        | 90.91        | 60.88        | 74.50                |
| KrakenUniq-k <sub>22</sub> -l <sub>11</sub>              | <b>95.01</b> | 91.12        | 87.21        | 93.02                | 69.41        | 84.19        | 62.27        | 76.09                |
| CLARK                                                    | 56.87        | 95.82        | 57.56        | 71.38                | 56.20        | 93.22        | 56.12        | 70.12                |
| CLARK-k <sub>27</sub>                                    | 62.57        | 89.90        | 60.18        | 73.78                | 61.75        | 86.81        | 58.25        | 72.17                |
| CLARK-k <sub>22</sub>                                    | 86.05        | 63.35        | 58.00        | 72.98                | 85.53        | 60.71        | 55.63        | 71.02                |
| SKiM                                                     | 80.45        | 94.65        | 77.98        | 86.98                | 80.12        | 92.66        | 76.43        | 85.93                |
| SKiM-k <sub>15</sub> -s <sub>13</sub> -t <sub>1</sub>    | 92.37        | 94.18        | 87.90        | 93.27                | 92.22        | 92.18        | 86.12        | 92.20                |
| SKiM-k <sub>15</sub> -s <sub>14</sub> -t <sub>1</sub>    | 94.95        | 92.13        | <b>88.23</b> | <b>93.52</b>         | <b>94.84</b> | 90.10        | <b>86.35</b> | <b>92.40</b>         |
| SKiM-k <sub>16</sub> -s <sub>10</sub> -t <sub>2</sub>    | 85.82        | 95.22        | 83.06        | 90.28                | 85.59        | 93.42        | 81.57        | 89.33                |

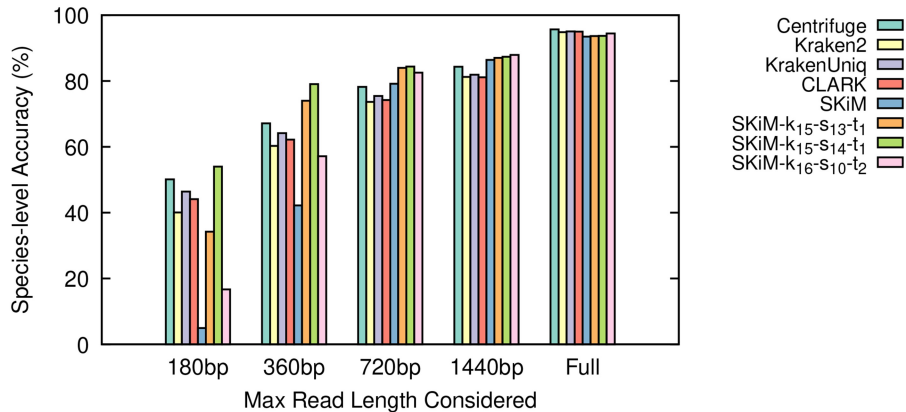
<sup>a</sup> The best value for each column is in bold.



**Figure 2.** Species-level classifier accuracy on the Even reads.



**Figure 3.** Species-level classifier accuracy on the Bench reads.



**Figure 4.** Species-level classifier accuracy on the BMock12-10kb reads.

compression scheme (denoted as Random ARLE), the TSP-heuristic induced column ordering and NRLE scheme (denoted as SKiM NRLE), and the TSP-heuristic ordering with ARLE scheme (which is simply referred to as SKiM). The properties of each resulting database are summarized in Table 4. For the Random NRLE and Random ARLE tests, we report the average over five different random orderings.

## 4 Discussion

SKiM outperforms the tested classifiers in terms of recall and accuracy when limited to the first 720 bp of a read while

maintaining comparable precision. Moreover, its classification performance when considering at least 720 bp is comparable to that of the entire read, which is not the case for other tools. Furthermore, for Even and Bench datasets, which were base-called using Dorado, SKiM achieves similar performance when considering only 360 bp. These findings remain true irrespective of how other tools are configured, as shown in Figs 1–3, available as [supplementary data](#) at *Bioinformatics* online.

Table 1 shows that when limited to 360 bp on the Even dataset, SKiM has the highest species-level recall, accuracy, and  $F_1$ -score of all classifiers, while the precision is comparable to that of other tools. As stated in Section Test Data,

**Table 2.** Classifier throughput on the Even reads.<sup>a</sup>

| Classifier                                               | Max Read Length Considered |                |                |                 |              |
|----------------------------------------------------------|----------------------------|----------------|----------------|-----------------|--------------|
|                                                          | 180 bp (Mbp/s)             | 360 bp (Mbp/s) | 720 bp (Mbp/s) | 1440 bp (Mbp/s) | Full (Mbp/s) |
| Centrifuge                                               | 12.1                       | 13.9           | 15.0           | 11.3            | 15.0         |
| Kraken2                                                  | 107.1                      | 101.8          | 99.5           | 98.6            | 97.6         |
| Kraken2-k <sub>37</sub> -l <sub>27</sub> -g <sub>3</sub> | <b>146.6</b>               | <b>143.4</b>   | <b>141.6</b>   | <b>139.9</b>    | <b>140.9</b> |
| Kraken2-k <sub>32</sub> -l <sub>22</sub>                 | 131.6                      | 132.2          | 130.1          | 122.7           | 128.2        |
| KrakenUniq                                               | 21.5                       | 20.9           | 20.6           | 21.2            | 20.9         |
| KrakenUniq-k <sub>27</sub> -l <sub>13</sub>              | 11.3                       | 10.8           | 10.5           | 7.8             | 9.3          |
| KrakenUniq-k <sub>22</sub> -l <sub>11</sub>              | 7.3                        | 6.9            | 6.8            | 5.6             | 6.2          |
| CLARK                                                    | 26.5                       | 24.9           | 24.3           | 24.1            | 24.2         |
| CLARK-k <sub>27</sub>                                    | 26.3                       | 24.6           | 24.3           | 24.0            | 18.6         |
| CLARK-k <sub>22</sub>                                    | 25.3                       | 24.0           | 23.8           | 23.8            | 23.5         |
| SKiM                                                     | 9.8                        | 13.0           | 17.5           | 19.9            | 26.0         |
| SKiM-k <sub>15</sub> -s <sub>13</sub> -t <sub>1</sub>    | 7.0                        | 9.3            | 10.5           | 12.7            | 15.0         |
| SKiM-k <sub>15</sub> -s <sub>14</sub> -t <sub>1</sub>    | 5.7                        | 6.5            | 8.2            | 9.4             | 10.6         |
| SKiM-k <sub>16</sub> -s <sub>10</sub> -t <sub>2</sub>    | 14.4                       | 19.5           | 24.7           | 27.9            | 38.2         |
| Minimap2                                                 | 0.3                        | 0.3            | 0.2            | 0.2             | 0.1          |

<sup>a</sup> The best value for each column is in bold.**Table 3.** Database size and memory use for each classifier on ABV.

| Classifier                                               | Size on disk (GB) | Peak RSS (GB) |
|----------------------------------------------------------|-------------------|---------------|
| Centrifuge                                               | 81.5              | 87.9          |
| Kraken2                                                  | 62.0              | 62.9          |
| Kraken2-k <sub>37</sub> -l <sub>27</sub> -g <sub>3</sub> | 32.0              | 33.7          |
| Kraken2-k <sub>32</sub> -l <sub>22</sub>                 | 29.4              | 31.3          |
| KrakenUniq                                               | 432.7             | 430.6         |
| KrakenUniq-k <sub>27</sub> -l <sub>13</sub>              | 374.4             | 375.2         |
| KrakenUniq-k <sub>22</sub> -l <sub>11</sub>              | 307.1             | 312.5         |
| CLARK                                                    | 184.8             | 184.8         |
| CLARK-k <sub>27</sub>                                    | 175.8             | 179.1         |
| CLARK-k <sub>22</sub>                                    | 102.2             | 114.6         |
| SKiM                                                     | 12.4              | 14.6          |
| SKiM-k <sub>15</sub> -s <sub>13</sub> -t <sub>1</sub>    | 24.1              | 28.3          |
| SKiM-k <sub>15</sub> -s <sub>14</sub> -t <sub>1</sub>    | 35.2              | 40.2          |
| SKiM-k <sub>16</sub> -s <sub>10</sub> -t <sub>2</sub>    | 17.9              | 26.5          |
| Minimap2                                                 | 320.0             | 28.9          |

**Table 4.** Effects of compression on the size of the SKiM ABV database.<sup>a</sup>

| Database                                              | Size on disk (GB) | # RLE runs     |
|-------------------------------------------------------|-------------------|----------------|
| Random NRLE                                           | —                 | 30 102 012 410 |
| Random ARLE                                           | 44.1              | 22 068 369 680 |
| SKiM NRLE                                             | —                 | 8 309 409 513  |
| SKiM                                                  | 12.4              | 5 846 074 003  |
| SKiM-k <sub>15</sub> -s <sub>13</sub> -t <sub>1</sub> | 24.1              | 11 390 492 474 |
| SKiM-k <sub>15</sub> -s <sub>14</sub> -t <sub>1</sub> | 35.2              | 16 674 228 653 |
| SKiM-k <sub>16</sub> -s <sub>10</sub> -t <sub>2</sub> | 17.9              | 7 206 746 522  |

<sup>a</sup> NRLE databases are never saved to disk. They are created in memory as an intermediate during database construction.

ONT's current adaptive sampling protocol uses the first 400 bp to decide if a read should be ejected. Therefore, SKiM's superior classification accuracy at 360 bp, without sacrificing precision, means it is better suited to handle adaptive sampling than the other tested classifiers.

However, SKiM is not without limitations. From Figs 2–4 we can see that SKiM with default parameters has significantly lower accuracy when reads are restricted to the first 180 bp. This is caused by a larger number of false negative assignments (see raw data in [Supplementary Information](#), available as [supplementary data](#) at *Bioinformatics* online), suggesting that when the read length is too short we have

insufficient information to accurately estimate  $\bar{x}_i$ . More specifically, with the default sub-sampling parameters, only 12% of all possible  $k$ -mers are canonical syncmers. This means that a 180 bp read would only provide approximately 21  $k$ -mer queries to calculate  $\bar{x}_i$ . Because this is below the default  $n_{\text{fixed}} = 100$ , we do not perform any scaling on the number of matches  $x_i$ , but we still calculate  $P$ -values based on  $n_{\text{fixed}}$  (recall Section 2.3). We could instead calculate the  $P$ -value based on the true number of queries  $n$ , use a larger cut-off value, or always calculate  $\bar{x}_i$ . Indeed, we observed that each possible solution decreased the number of false negatives, increased the number of true positives, and increased the overall accuracy to outperform all other classifiers. But, it also introduced a significant number of false positives. Within the context of adaptive sampling, false positive decisions are more problematic than false negatives. If a read is falsely classified, it may be incorrectly ejected, discontinuing the sequencing of potentially important DNA.

SKiM's performance on the BMock12-10kb dataset (Fig. 4) is slightly worse compared to the other two test datasets. This dataset is the only dataset basecalled with Albacore, a very early ONT basecaller that has a high error rate compared to the newest basecaller Dorado. More specifically, the fastest and least accurate model for Dorado (used to basecall Even and Bench) has an error rate under 10%, while Albacore has an error rate around 15% (Kuřmirek 2023). This difference in the error rate can explain SKiM's poorer performance on the BMock12-10kb dataset, and hence SKiM's performance should not be affected in the up-to-date basecalling workflows. To confirm this, we simulated three datasets using Badread (Wick 2019) at average error rates of 5%, 10%, and 15%. As expected, SKiM's classification performance relative to other classifiers is only significantly worse on the 15% error rate reads, while SKiM's relative performance on the 5% and 10% error rate reads resemble the Even and Bench datasets. We provide additional details on how the simulated datasets were generated, along with species-level accuracy graphs (Figs 4–9, available as [supplementary data](#) at *Bioinformatics* online), in the [Supplementary Information](#), available as [supplementary data](#) at *Bioinformatics* online.

Table 3 shows that SKiM runs with a fraction of the memory, as low as 14.6 GB for all of ABV, while the next best classifier, Kraken2-k<sub>32</sub>-l<sub>22</sub>, requires 31.3 GB. As Table 4

shows, the proposed lossy compression methods play a crucial role in SKiM's database sizes. More specifically, it shows that a random column ordering of ABV requires around four times as many runs as the nearest neighbor heuristic ordering (for both NRLE and ARLE). It also shows that the number of runs required by the ARLE representations is around 70% of the corresponding NRLE representation (for both the random and TSP-heuristic ordering). Without these optimizations, SKiM's default parameter database would be comparable in size to Kraken2. We provide further discussion on how  $k$ -mer size affects the size of reference database in the [Supplementary Information](#), available as [supplementary data at Bioinformatics online](#).

While the resulting database is small, peak memory usage during SKiM database construction is higher than the database size. For example, when building ABV with default parameters, the max RSS was 71.9 GB. This is not too problematic, because databases can be constructed on high-memory hardware and run in lower memory environments. However, the pairwise distance computation required to use the TSP-heuristic can take over a day for large assembly sets, even on high performance hardware, and is the dominating step in database construction runtime. However, this step is embarrassingly parallel and the distance matrix construction can be delegated to dedicated tools like, e.g. Dashing ([Baker and Langmead 2019](#)) (which SKiM supports out of the box). Dashing uses the HyperLogLog ([Flajolet et al. 2007](#)) algorithm to estimate the pairwise Jaccard indices, which in turn can be used instead of distances for the TSP-heuristic. Dashing is both space-efficient and fast, reducing the runtime of computing the pairwise distances (Jaccard indices) to less than an hour. Moreover, it produces databases that are only 5% larger than the true pairwise distances, a percentage that we believe would be negligible if Dashing was tuned to use syncmers instead of  $k$ -mers.

[Table 2](#) shows that SKiM has a classification throughput of 10–25 Mbp/s, which is comparable to other tools, and is significantly outperformed only by Kraken2. For comparison, a MinION flow cell can concurrently sequence 512 molecules at 450 bp/s each ([Wang et al. 2021](#)) at maximum theoretical throughput, for a total of 230 kb/s. While hardware at the point of sequencing may not be as fast or have as many available cores as the processor used in our experiments, SKiM's throughput is still comfortably faster than the maximum theoretical sequencing throughput. This is true even for GridION devices, capable of sequencing on five MinION flow cells concurrently. Consequently, we would not expect SKiM to cause a bottleneck during adaptive sampling or real-time classification.

Lastly, [Table 2](#) also shows that SKiM is the only classifier whose throughput changes significantly as read length changes. Specifically, throughput increases as read length increases. This is due to finding the lowest probability value after all  $k$ -mer queries are complete (see Section 2.3), which is a non-negligible overhead of the classification algorithm, and it occurs per read. This overhead takes up a larger portion of the runtime on short reads than on longer ones.

#### 4.1 Implications for pangenomes and strain-level classification

Reference databases used in metagenomics frequently involve highly similar sequences, for example, coming from multiple strains of the same bacterial species. When these sequences

are treated as the same classification target, as in the case for tools like CLARK, Kraken2, Centrifuge, etc., significant index reduction can be achieved. However, this design does not allow for classification at the level of reference assembly or sequence. Because SKiM indexes each reference assembly individually, it may be better equipped for pangenome and strain-level classification experiments. Furthermore, highly similar assemblies are likely to be significantly compressed in a SKiM database but without information loss.

As a proof of concept, we used SKiM with default parameters to index all DNA sequences for SARS-CoV-2 deposited in the NCBI SARS-CoV-2 Data Hub (<https://www.ncbi.nlm.nih.gov/labs/virus/vssi>) throughout 2020. With a total of 47 160 sequences (about 1.4 GB when stored as FASTA files), SKiM was able to create an index of just 5MB. This is a promising result. For example, such a 5MB index could simply be added to the existing ABV database with minimal additional overhead, allowing for more robust classification of many SARS-CoV-2 variants.

To further explore the impact of pangenome collections on database size, we took the two largest groups of organisms in ABV by the NCBI species taxonomy id, 562 (*Escherichia coli*) and 573 (*Klebsiella pneumoniae*), and created SKiM references from each. There were 2289 files with the taxonomy id 562 in the ABV reference, totaling 11.9 GB of raw FASTA files. SKiM's database, with default parameters, was 311MB on disk. Likewise, there were 1400 files with the taxonomy id 573 in the ABV reference, totaling 8.0 GB of raw FASTA files. SKiM's database, with default parameters, was 186MB on disk. Again, these files were a subset of the ABV reference, meaning that these files are likely compressed to the same degree within the full ABV SKiM reference database. This suggests that adding many strains of the same species compresses well within our database matrix.

#### Author contributions

Trevor Schneggenburger (Conceptualization [equal], Data curation [lead], Formal analysis [lead], Methodology [equal], Software [lead], Visualization [equal], Writing—original draft [equal], Writing—review & editing [equal]) and Jaroslaw Zola (Conceptualization [equal], Data curation [supporting], Formal analysis [supporting], Funding acquisition [lead], Methodology [equal], Resources [lead], Software [supporting], Supervision [lead], Visualization [equal], Writing—original draft [equal], Writing—review & editing [equal])

#### Supplementary data

[Supplementary data](#) are available at [Bioinformatics online](#).

Conflict of interest: None declared.

#### Funding

This work is part of the SMARTEn Project (<http://score-group.org/?id=smarten>), supported in part by funds from the National Science Foundation (CNS-1910193).

#### Data availability

SKiM is available at <https://gitlab.com/SCoRe-Group/SKiM>. All datasets were derived from sources in the public domain: Even ([Nicholls et al. 2019](#)) (<https://www.ebi.ac.uk/ena/browser/view/>)

ERR2887847), Bench (Leidenfrost *et al.* 2020) (<https://zenodo.org/records/3600229>), and BMock12-10kb (Sevim *et al.* 2019) (<https://www.ncbi.nlm.nih.gov/sra/SRX4901586>).

## References

- Ahmed O, Rossi M, Gagne T *et al.* SPUMONI 2: improved classification using a pangenome index of minimizer digests. *Genome Biol* 2023; 24:122.
- Ahmed O, Rossi M, Kovaka S *et al.* Pan-genomic matching statistics for targeted nanopore sequencing. *iScience* 2021;24:102696.
- Baker D, Langmead B. Dashing: fast and accurate genomic distances with HyperLogLog. *Genome Biol* 2019;20:265.
- Booth K, Lueker G. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J Comput Syst Sci* 1976;13:335–79.
- Breitwieser F, Baker D, Salzberg S. KrakenUniq: confident and fast metagenomics classification using unique k-mer counts. *Genome Biol* 2018;19:198.
- Chambi S, Lemire D, Kaser O *et al.* Better bitmap performance with roaring bitmaps. *Softw Pract Exp* 2016;46:709–19.
- Edgar R. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ* 2021;9:e10805.
- Flajolet P, Fusy E, Gandouet O, Meunier F. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In: Jacquet P (ed.), *Discrete Mathematics & Theoretical Computer Science*. Juan les Pins, FR: Episciences, 2007.
- Johnson D, Krishnan S, Chhugani J *et al.* Compressing large Boolean matrices using reordering techniques. In: *International Conference on Very Large Data Bases*. Toronto, CA: VLDB Endowment, 2004, 13–23.
- Kim D, Song L, Breitwieser F *et al.* Centrifuge: rapid and sensitive classification of metagenomic sequences. *Genome Res* 2016;26:1721–9.
- Kipp E, Lindsey L, Khoo B *et al.* Metagenomic surveillance for bacterial tick-borne pathogens using nanopore adaptive sampling. *Sci Rep* 2023;13:10991.
- Ko S, Sassoubre L, Zola J. Applications and challenges of real-time mobile DNA analysis. In: *International Workshop on Mobile Computing Systems & Applications*. Tempe, AZ, USA: Association for Computing Machinery, 2018, 1–6.
- Kuśmirek W. Estimated nucleotide reconstruction quality symbols of basecalling tools for oxford nanopore sequencing. *Sensors* 2023; 23:6787.
- Leidenfrost R, Pöther D, Jäckel U *et al.* Benchmarking the MinION: evaluating long reads for microbial profiling. *Sci Rep* 2020; 10:5125.
- Li H. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* 2018;34:3094–100.
- Loose M, Malla S, Stout M. Real-time selective sequencing using nanopore technology. *Nat Methods* 2016;13:751–4.
- Martin S, Heavens D, Lan Y *et al.* Nanopore adaptive sampling: a tool for enrichment of low abundance species in metagenomic samples. *Genome Biol* 2022;23:11.
- Mikalsen A, Zola J. Coriolis: enabling metagenomic classification on lightweight mobile devices. *Bioinformatics* 2023;39:i66–75.
- Navarro G. *Compact Data Structures: A Practical Approach*. New York, NY, USA: Cambridge University Press, 2016.
- Nicholls S, Quick J, Tang S *et al.* Ultra-deep, long-read nanopore sequencing of mock microbial community standards. *Gigascience* 2019;8:giz043.
- O’Leary NA, Wright MW, Brister JR *et al.* Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation. *Nucleic Acids Res* 2016;44:D733–45.
- Olken F, Rotem D. Rearranging Data to Maximize the Efficiency of Compression. In: *ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*. Cambridge, MA, USA: Association for Computing Machinery, 1985.
- Ounit R, Wanamaker S, Close T *et al.* CLARK: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers. *BMC Genomics* 2015;16:236.
- Payne A, Holmes N, Clarke T *et al.* Readfish enables targeted nanopore sequencing of gigabase-sized genomes. *Nat Biotechnol* 2021; 39:442–50.
- Quick J, Loman N, Duraffour S *et al.* Real-time, portable genome sequencing for ebola surveillance. *Nature* 2016;530:228–32.
- Roberts M, Hayes W, Hunt B *et al.* Reducing storage requirements for biological sequence comparison. *Bioinformatics* 2004;20:3363–9.
- Schleimer A, Wilkerson D, Aiken A. Winnowing: local algorithms for document fingerprinting. In: *ACM SIGMOD International Conference on Management of Data*. San Diego, CA, USA: Association for Computing Machinery, 2003, 76–85.
- Sevim V, Lee J, Egan R *et al.* Shotgun metagenome data of a defined mock community using oxford nanopore, PacBio and Illumina technologies. *Sci Data* 2019;6:285.
- Ulrich J, Renard B. Fast and space-efficient taxonomic classification of long reads with hierarchical interleaved XOR filters. *Genome Res* 2024;34:914–24.
- Ulrich J, Epping L, Pilz T *et al.* Nanopore adaptive sampling effectively enriches bacterial plasmids. *mSystems* 2024;9:e00945-23.
- Ulrich J, Lutfi A, Rutzen K *et al.* ReadBouncer: precise and scalable adaptive sampling for nanopore sequencing. *Bioinformatics* 2022; 38:i153–60.
- Wang Y, Zhao Y, Bollas A *et al.* Nanopore sequencing technology, bioinformatics and applications. *Nat Biotechnol* 2021;39:1348–65.
- Wick R. Badread: simulation of error-prone long reads. *J Open Source Softw* 2019;4:1316.
- Wood D, Lu J, Langmead B. Improved metagenomic analysis with Kraken 2. *Genome Biol* 2019;20:257.
- Wrenn D, Drown D. Nanopore adaptive sampling enriches for antimicrobial resistance genes in microbial communities. *GigaByte* 2023; 2023:gigabyte103.