Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Manuscript Category

OXFORD

Gene Expression

HyperGen: Compact and Efficient Genome Sketching using Hyperdimensional Vectors

Weihong Xu[1,*], Po-Kai Hsu[2], Niema Moshiri[1], Shimeng Yu[2], and Tajana Rosing[1]

[1] Department of Computer Science and Engineering, University of California San Diego, La Jolla, CA 92093, USA.
[2] School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA.

* To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: Genomic distance estimation is a critical workload since exact computation for whole-genome similarity metrics such as Average Nucleotide Identity (ANI) incurs exhibitive runtime overhead. Genome sketching is a fast and memory-efficient solution to estimate ANI similarity by distilling representative k-mers from the original sequences. In this work, we present HyperGen that improves accuracy, runtime performance, and memory efficiency for large-scale ANI estimation. Unlike existing genome sketching algorithms that convert large genome files into discrete k-mer hashes, HyperGen leverages the emerging hyperdimensional computing (HDC) to encode genomes into quasi-orthogonal vectors (Hypervector, HV) in high-dimensional space. HV is compact and can preserve more information, allowing for accurate ANI estimation while reducing required sketch sizes. In particular, the HV sketch representation in HyperGen allows efficient ANI estimation using vector multiplication, which naturally benefits from highly optimized general matrix multiply (GEMM) routines. As a result, HyperGen enables the efficient sketching and ANI estimation for massive genome collections.
Results: We evaluate HyperGen's sketching and database search performance using several genome datasets at various scales. HyperGen is able to achieve comparable or superior ANI estimation error and linearity compared to other sketch-based counterparts. Compared to other sketch-based baselines, HyperGen achieves comparable sketching speed and up to $4.3\times$ faster in database search. Meanwhile, HyperGen's sketch size is on average 1.8-$2.7\times$ smaller.
Availability: A Rust implementation of HyperGen is freely available under the MIT license as an open-source software project at https://github.com/wh-xu/Hyper-Gen.
Contact: wexu@ucsd.edu

1 Introduction

In recent years, the burgeoning field of genomics has been revolutionized by the advent of high-throughput sequencing technologies (Soon *et al.*, 2013), leading to exponential growth in genomic data (Stephens *et al.*, 2015). This deluge of data presents a significant challenge for traditional genomic analysis methods, particularly in terms of computational efficiency and storage requirements. Calculating the Average Nucleotide Identity (ANI) similarity of genome files is the key step for various downstream workloads in genome analysis, such as large-scale database search (Chaumeil *et al.*, 2022), clustering (Parks *et al.*, 2020), and taxonomy analysis (Hernández-Salmerón *et al.*, 2023). Traditional BLAST-based methods (Kurtz *et al.*, 2004; Lee *et al.*, 2016) rely on alignment to perform accurate ANI calculations. However, the alignment process is computationally expensive and requires hours or days to calculate ANIs. The slow speed of alignment-based approaches has become a major bottleneck for large-scale genome analysis.

Several state-of-the-art works have tried to speed up large-scale genome analysis by approximating the genome similarity using more efficient data structures. These works can be categorized into two types: mapping-based and sketch-based approaches as follows. FastANI (Jain *et al.*, 2018) and Skani (Shaw and Yu, 2023) are two representative mapping-based algorithms for ANI estimation. FastANI is built upon the Mashmap sequence mapping algorithm (Jain *et al.*, 2017) and achieves a significant speedup compared to the alignment-based baseline (Kurtz *et al.*, 2004). Skani uses the sparse chaining to increase mapping sensitivity, further improving accuracy and efficiency of ANI estimation. However, both FastANI and Skani suffer from high memory consumption. For example, Skani needs to store indexing files with a storage size comparable to the original dataset. FastANI encounters out-of-memory issues on large datasets as reported in (Shaw and Yu, 2023).

In this work, we focus on the "*genome sketching*," which is regarded as a promising solution to address the aforementioned challenges because it significantly reduces storage size while providing satisfactory accuracy of estimation (Hernández-Salmerón *et al.*, 2023). Unlike alignment-based or mapping-based tools (Kurtz *et al.*, 2004; Lee *et al.*, 2016; Jain *et al.*, 2018; Shaw and Yu, 2023) that require expensive computation or large memory space, sketch-based approaches (Ondov *et al.*, 2016; Brown and Irber, 2016; Baker and Langmead, 2019, 2023) only preserve the most essential features of the genome (called the "sketch"). The sketch's compact representation enables rapid and efficient ANI approximation for genome files. Mash (Ondov *et al.*, 2016) and Sourmash (Brown and Irber, 2016) represent groundbreaking efforts to use MinHash (Broder, 1997) and FracMinHash (Hera *et al.*, 2023) to estimate genomic similarity, respectively. Dashing 2 (Baker and Langmead, 2023) utilizes the SetSketch data structure (Ertl, 2021) and incorporates multiplicities to realize a memory-efficient genome sketch and accurate estimation of ANI.

1.1 Motivation

By transforming raw genome data into more compact data structures, genome sketching represents a paradigm shift in bioinformatics, paving the way for more scalable and rapid genomic analyses in the era of big data. Although previous sketch-based tools (Ondov *et al.*, 2016; Brown and Irber, 2016; Baker and Langmead, 2023) demonstrate a significant reduction in runtime and file sizes, our goal in this work is to create a more concise and informative sketching algorithm based on hyperdimensional computing (HDC) (Kanerva, 2009). Recent studies on HDC have demonstrated the effectiveness of using HDC to accelerate bioinformatics workloads, such as pattern matching (Zou *et al.*, 2022; Kang *et al.*, 2023) and spectral clustering (Xu *et al.*, 2023).

We take advantage of recent DotHash (Nunes *et al.*, 2023), which shows superior space and computation efficiency for the Jaccard similarity estimation. DotHash leverages the HDC-based random indexing (Sahlgren, 2005; Kanerva *et al.*, 2000) and is originally designed for fast set intersection estimation. The main difference between DotHash and MinHash lies in the format of generated sketch: MinHash represents a sketch as a hash set with discrete values, while DotHash represents a sketch with a nonbinary vector of high dimension. DotHash's vector representation of the sketch achieves faster processing speed since it can fully exploit the low-level hardware parallelism (such as CPU's Single Instruction Multiple Data (SIMD) and GPU) optimized for vector processing.

However, DotHash still suffers from two major limitations that hinder its application to genome sketching. First, DotHash is only applicable to non-genome data since it lacks an effective k-mer sampling strategy to generate genomic sketches. Second, DotHash uses high-precision floating point number to represent random vectors, exhibiting large runtime overhead and slow speed.

1.2 Contributions

In this work, we propose HyperGen, a novel tool for efficient genome sketching and ANI estimation. HyperGen exploits the emerging HDC (similar to DotHash (Nunes *et al.*, 2023)) to boost genomic ANI calculation. Specifically, we optimize DotHash's efficiency by converting the sketch generation process into a low bit-width integer domain. This allows us to represent the genome sketch using the high-dimensional vector (HV) at the cost of negligible runtime overhead. Based on the HV sketch, we propose an approach to estimate the Jaccard similarity using vector matrix multiplication. We also introduce a lossless compression scheme using bit-packing to further reduce the sketch size.

We benchmark HyperGen against several state-of-the-art tools (Jain *et al.*, 2018; Ondov *et al.*, 2016; Baker and Langmead, 2023; Kurtz *et al.*, 2004). For ANI estimation, HyperGen demonstrates comparable or lower ANI estimation errors compared to other baselines across different datasets. For generated sketch size, HyperGen achieves $1.8\times$ to $2.7\times$ sketch size reduction as compared to Mash (Ondov *et al.*, 2016) and Dashing 2 (Baker and Langmead, 2023), respectively. HyperGen also enjoys the benefits of the modern hardware architecture optimized for vector processing. HyperGen shows about $1.7\times$ sketch generation speedup over Mash and up to $4.3\times$ search speedup over Dashing 2. To the best of our knowledge, HyperGen offers the optimal trade-off between speed, accuracy, and memory efficiency for ANI estimation.

2 Methods

2.1 Preliminaries

Fast computation of Average Nucleotide Identity (ANI) is pivotal in genomic data analysis (microbial genomics to delineate species), as ANI serves as a standardized and genome-wide measure of similarity that helps facilitate genomic data analysis. Popular approaches to calculate ANI include: alignment (Kurtz *et al.*, 2004; Lee *et al.*, 2016), mapping (Jain *et al.*, 2018; Shaw and Yu, 2023), and sketch (Ondov *et al.*, 2016; Brown and Irber, 2016; Baker and Langmead, 2019, 2023). However, alignment-based and mapping-based methods involve either time-consuming pairwise alignments or memory-intensive mappings. In the following sections, we focus on the sketch-based ANI estimation with significantly better efficiency.

2.1.1 MinHash and Jaccard Similarity

Existing sketh-based approaches (Ondov *et al.*, 2016; Brown and Irber, 2016; Baker and Langmead, 2019, 2023) do not directly compute ANI. Instead, they compute the Jaccard similarity (Ondov *et al.*, 2016), which is used to measure the similarity of two given k-mer sets. Then the Jaccard similarity is converted to ANI as shown in Eq. (8). The conversion between Jaccard similarity and ANI is trivial, so most efforts in previous works (Ondov *et al.*, 2016; Brown and Irber, 2016; Baker and Langmead, 2019, 2023) are to find more efficient and accurate ways to estimate Jaccard similarity.

Without loss of generality, we denote k-mer as consecutive substrings with length k of the nucleotide alphabet, *e.g.* $\sum^k \in \{A, G, C, T\}^k$. $\mathcal{S}_k(X)$ denotes the set of k-mers sampled from genome sequence X based on a given condition. Therefore, the Jaccard similarity for two sequences, A and B, can be computed as follows:

$$J_k(A,B) = \frac{|\mathcal{S}_k(A) \cap \mathcal{S}_k(B)|}{|\mathcal{S}_k(A) \cup \mathcal{S}_k(B)|}, \tag{1}$$

where $J_k(A,B) \in [0,1]$ is the Jaccard similarity indicating the overlap between k-mer sets of two sequences. Note that HyperGen uses canonical k-mers by default.

A straightforward idea to sample k-mer sets in Eq. (1) is to keep all k-mers. However, this incurs prohibitive complexity since all unique k-mers need to be stored. The resulting complexity is $\mathcal{O}(L)$ for a sequence of length L. To alleviate the complexity issue, Mash (Ondov *et al.*, 2016) and its variants (Liu and Koslicki, 2022; Jain *et al.*, 2017) use MinHash (Broder, 1997) to approximate the Jaccard similarity by only preserving a tiny subset of k-mers. In particular, Mash keeps N k-mers that have the smallest hash values $h(\cdot)$. In this case, the Jaccard similarity is estimated as:

$$\hat{J}(A,B) = \mathcal{P}(\min_{a\in A} h(a) = \min_{b\in B} h(b)). \tag{2}$$

Here, using MinHash helps to reduce the sketch complexity from $\mathcal{O}(L)$ to a constant $\mathcal{O}(N)$. The sampled k-mer set $\mathcal{S}_k(X)$ that stores N smallest k-mer hash values is regarded as the genome file sketch required for ANI estimation.

2.1.2 Jaccard Similarity using DotHash

A recent work (Nunes *et al.*, 2023) demonstrates that the speed and memory efficiency of Jaccard similarity approximation can be improved by using the DotHash based on Random Indexing (Sahlgren, 2005). The key step to compute Jaccard similarity in Eq. (1) is computing the cardinality of set intersection $|A \cap B|$ while the cardinality of set union can be calculated through $|A \cup B| = |A| + |B| - |A \cap B|$.

In DotHash, each element of the set is mapped to a unique D-dimensional vector in real number using the mapping function $\phi(x)$. Each set is expressed as an aggregation vector $\mathbf{a} \in \mathbb{R}^D$ such that

$$\mathbf{a} = \sum_{a\in A} \phi(a), \tag{3}$$

where the aggregation vector sums all the elements' vectors generated by the mapping function $\phi(x)$. One necessary constraint for function $\phi(x)$ is: the generated vectors should satisfy the quasi-orthogonal properties:

$$\phi(a)\cdot\phi(b) = \begin{cases} 0, \text{if } a \neq b, \\ 1, \text{if } a == b. \end{cases} \tag{4}$$

DotHash (Nunes *et al.*, 2023) uses a pseudo random number generator (RNG) as the mapping function $\phi(x)$ because the RNG can generate uniform and quasi-orthogonal vectors in an efficient manner.

Using the quasi-orthogonal properties, the cardinality approximation for set intersection is transformed into the dot product of two aggregation vectors:

$$\begin{aligned} |A \cap B| &= \mathbb{E}[\mathbf{a}\cdot\mathbf{b}] \\ &= \mathbb{E}\left[\sum_{a\in A}\sum_{b\in B}\phi(a)\cdot\phi(b)\right] \\ &= \sum_{a\in A}\sum_{b\in B}\mathbb{1}(a == b) \\ &= \sum_{x\in A\cap B} 1, \end{aligned} \tag{5}$$

where those vectors not in the set intersection ($a \neq b$) have no contribution to the inner product due to their quasi-orthogonality as in Eq. (4). DotHash effectively aggregates all elements in a set to form an aggregation vector with D dimension. The computational and space complexity of cardinality estimation for set interaction is reduced from $\mathcal{O}(N)$ to $\mathcal{O}(D)$. Moreover, the computation process of DotHash is highly vectorized and can be easily boosted by existing hardware architecture optimized for general matrix multiply (GEMM).

2.2 Proposed HyperGen Sketching

The aforementioned DotHash provides both good accuracy and runtime performance (Nunes *et al.*, 2023). However, we observe two major limitations of DotHash: 1. Although DotHash can be used to calculate the cardinality of set intersection, it cannot be applied to genomic sketching because DotHash lacks a k-mer sampling module that identifies the useful k-mers; 2. The computation and space efficiency can be further optimized

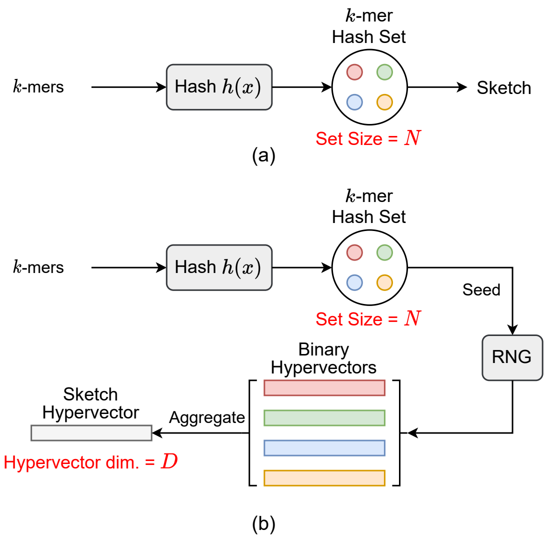

Fig. 1: Algorithm overview for (a) Mash-like sketching and (b) HyperGen sketching for genome sequences. Mash stores the genome sketch in a k-mer hash set with $\mathcal{O}(N)$ complexity while HyperGen aggregates N k-mer hashes into a D-dimensional sketch HV with $\mathcal{O}(D)$ complexity.

because the previous DotHash manages and processes all vectors in floating-point (FP) numbers. The mapping function $\phi(x)$ incurs significantly overhead.

We present HyperGen for genomic sketching applications that addresses the limitations of DotHash. Fig. 1 shows the algorithmic overview for (a) Mash-like sketching and (b) HyperGen sketching schemes. The first step of HyperGen is similar to Mash, where both Mash and HyperGen extract k-mers by sliding a window through given genome sequences. The extracted k-mers are uniformly hashed into the corresponding numerical values by a hash function $h(x)$. To ensure low memory complexity, most k-mer hashes are filtered and only a small portion of them are preserved in the k-mer hash set to work as the sketch (or signature) of the associative genome sequence. The key difference is that HyperGen adds a key step, called *Hyperdimensional Encoding for k-mer Hash*, to convert k-mer hash values into binary hypervectors (HVs) and aggregate to form the D-dimensional sketch HV. To distinguish itself from DotHash, the random vector in HyperGen is named HV. Algorithm 1 summarizes the flow of generating sketch hypervector in HyperGen. In the following sections, we explain the details of HyperGen.

2.2.1 Step 1: k-mer Hashing and Sampling

Mash uses MinHash that keeps the smallest N hash values as the genome sketch. In comparison, HyperGen adopts a different k-mer hashing and sampling scheme. Specifically, HyperGen performs a sparse k-mer sampling using FracMinHash (Hera *et al.*, 2023) (instead of MinHash in Mash). Given a hash function $h : \sum^k \mapsto [0, M]$ that maps k-mers into the corresponding nonnegative integer, the sampled k-mer hash set is expressed as Line 2-4 in Algorithm 1:

$$\mathcal{S}_k(A) = \{h(x) \mid \forall x \in A : h(x) \leq \frac{M}{S}\}, \tag{6}$$

where M is the maximum hash value while S denotes the scaled factor that determines the density of sampled k-mers in the set. FracMinHash has been widely adopted in other tools, such as Sourmash (Brown and Irber, 2016) and Skani (Shaw and Yu, 2023), due to its excellent performance. The advantage of using FracMinHash over MinHash (Broder, 1997) is that it ensures an unbiased estimation of the Jaccard similarity of k-mer sets with very dissimilar sizes (Hera *et al.*, 2023), providing better approximation

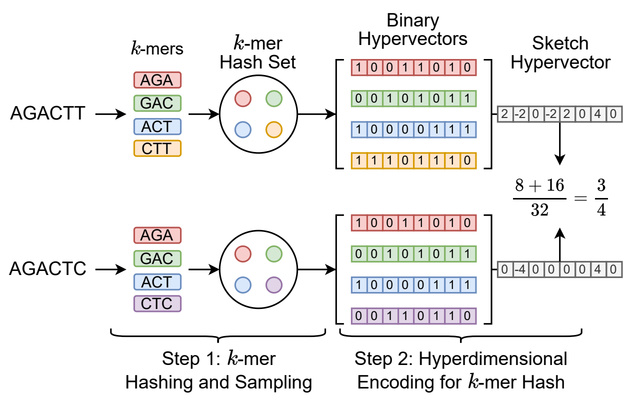

Fig. 2: Sketch hypervector generation and set intersection computation in HyperGen. Each k-mer with size $k = 3$ first passes through a hash function $h(x)$. The k-mers ($A = AGACTT$ and $B = AGACTC$) are hashed to hash set. Then each k-mer hash value is converted into the associated orthogonal binary HV. The set intersection between two k-mer hash sets is computed using the cosine similarity of aggregated sketch HVs.

Algorithm 1: Generation of sketch hypervector in HyperGen

```
Input: Genome sequence $X$, Scaled factor $S$, Maximum hash
       value $M$, HV dimension $D$, Pseudo random number
       generator RNG
Output: Sketch HV $\mathbf{H}$ for sequence $X$
/* Sampling $k$-mers using FracMinHash */
1  $\mathcal{S}_k \leftarrow \{\}$
2  for $k$-mer $x \in X$ do
3      if $h(x) < \frac{M}{S}$ then
4          $\mathcal{S}_k \leftarrow h(x) \cup \mathcal{S}_k$
/* Hyperdimensional encoding for $k$-mer hash */
5  $\mathbf{H} \leftarrow \mathbf{0}$
6  for $seed \in \mathcal{S}_k$ do
       // Binary HV encoding for $k$-mer hash
7      $hv \leftarrow \mathbf{0}$
8      for $i \leftarrow 1$ to $D/64$ do
9          $rnd \leftarrow$ RNG($seed$)
10         $seed \leftarrow rnd$
11         $hv_{i*64\ldots(i+1)*64} \leftarrow rnd$
       // Binary HV aggregation
12     for $i \leftarrow 1$ to $D$ do
13         $\mathbf{H}_i \leftarrow \mathbf{H}_i + (hv_i \times 2 - 1)$
```

quality than MinHash and its variants (Ondov *et al.*, 2016; Jain *et al.*, 2017). However, FracMinHash usually produces a larger hash set compared to Mash (Hera *et al.*, 2023), requiring more memory space. Step 2 in HyperGen alleviates the increased memory issue.

2.2.2 Step 2: Hyperdimensional Encoding for *k*-mer Hash

In Fig. 1-(a), after the k-mer hashing and sampling process, Mash-like sketching algorithms (such as Mash (Ondov *et al.*, 2016), Sourmash (Brown and Irber, 2016), and Mash Screen (Ondov *et al.*, 2019)) directly use the sampled k-mer hash set as the sketch to compute the Jaccard similarity for given sequences.

In Fig. 1-(b), HyperGen adds an additional step, called *Hyperdimensional Encoding for k-mer Hash* (Line 5-13 in Algorithm 1), before the sketch is generated. This step essentially converts the discrete and numerical hashes in the k-mer hash set to a D-dimensional and nonbinary vector, called *sketch hypervector*. In particular, each hash value in the k-mer hash set is uniquely mapped to the associated binary HV hv as Line 6-11 of Algorithm 1. HyperGen relied on recursive random bit generation to produce binary HVs of arbitrary length: the k-mer hash value is set as the initial seed of the pseudo RNG$(seed) \mapsto rnd$ function. For each iterative step, a 64b random integer rnd is generated using $seed$. The generated integer rnd is not only assigned to the corresponding bits in hv, but is also set as the next $seed$.

The hash function RNG$(\cdot)$ that maps the k-mer hash value to the binary HV hv is the key component of HyperGen because it determines the speed and quality of genome sketch generation. The following factors should be considered when selecting a good RNG$(\cdot)$ function: 1. The function needs to be fast enough to reduce the additional overhead for sketch generation. 2. The generated random binary HVs need to be able to provide enough randomness (*i.e.*, the binary HVs are as orthogonal as possible). This is because binary HVs are essentially random binary bit streams that need to be nearly orthogonal to each other to satisfy the quasi-orthogonal requirements. 3. The sketches results should be reproducible (*i.e.*, the identical bit streams can be generated using the same seed). We adopt a fast and high-quality pseudo RNG[1] in Rust language (Matsakis and Klock, 2014), which passes two randomness tests: TestU01 and Practrand (Sleem and Couturier, 2020). In this case, we can use the pseudo RNG to stably generate high-quality and reproducible binary HVs.

Fig. 2 shows an example of generating the sketch HVs with dimension $D = 8$ for two genome sequences based on k-mer size $k = 3$ and k-mer hash set size $N = 4$. Each sampled k-mer hash value in the hash set is converted to the corresponding binary HV $hv \in \{0, 1\}^D$ using the function RNG(x). Then, all N binary HVs are aggregated into a single sketch HV $\mathbf{H} \in \mathbb{Z}^D$ based on the following point-wise vector addition:

$$\mathbf{H} = \sum_{i=1}^{N} 2 \times hv^i - 1, \tag{7}$$

where the binary HV $hv \in \{0, 1\}^D$ is first converted to $\{-1, +1\}^D$. hv^i denotes the i-th binary HV in the set. Then all binary HVs in the set are aggregated together to create the corresponding sketch HV. The aggregation step reduces the space complexity of the sketch from $\mathcal{O}(N)$ to $\mathcal{O}(D)$. Compared to Mash-liked sketching approaches (Ondov *et al.*, 2016; Hera *et al.*, 2023; Brown and Irber, 2016), HyperGen is more memory efficient because the sketch HV format is more compact with $\mathcal{O}(D)$ space complexity, which is independent of the k-mer hash set size N. Meanwhile, HyperGen's hyperdimensional encoding step helps to achieve better ANI similarity estimation quality (see Section 3).

2.2.3 Step 3: ANI Estimation using Sketch Hypervector

The generated sketch hypervector can be used to efficiently estimate the ANI similarity. HyperGen estimates ANI value using the same approach in (Ondov *et al.*, 2016). The ANI under the Poisson distribution is estimated as:

$$\text{ANI}(A, B) = 1 + \frac{1}{k} \cdot \log \frac{2 \cdot J_k(A, B)}{1 + J_k(A, B)}, \tag{8}$$

where $J_k(A, B)$ denotes the Jaccard similarity between genome sequence A and sequence B while k is the k-mer size.

Therefore, ANI estimation in HyperGen becomes calculating Jaccard similarity based on sketch HVs. Eq. (1) shows that the intersection size and the set size of two k-mer hash sets are the keys to calculating the Jaccard similarity. For $hv^i \in \{-1, +1\}^D$, the cardinality of a set $\mathcal{S}_k(A)$

[1] https://github.com/wangyi-fudan/wyhash

is computed as follows:

$$|\mathcal{S}_k(A)| = \frac{\|\mathbf{H}_A\|_2^2}{D} = \frac{\sum_{i=1}^{N} \|hv^i\|_2^2}{D} = \frac{N \cdot D}{D} = N, \qquad (9)$$

which shows the set cardinality can be computed based on the L^2 norm of sketch HV. The computation of set intersection in HyperGen is similar to DotHash (Nunes *et al.*, 2023)'s Eq. (5) because HVs in HyperGen share the same quasi-orthogonal properties as DotHash. Then, Eq. (5) becomes:

$$\begin{aligned} |\mathcal{S}_k(A) \cap \mathcal{S}_k(B)| &= \frac{\mathbf{H}_A \cdot \mathbf{H}_B^T}{D} \\ &= \frac{\sum_i hv^i \cdot \sum_j hv^j}{D} \\ &= \frac{\sum_i \sum_j D \cdot \mathbb{1}(hv^i == hv^j)}{D} \\ &= \sum_i \sum_j \mathbb{1}(hv^i == hv^j) \\ &= \sum_{x \in \mathcal{S}_k(A) \cap \mathcal{S}_k(B)} 1. \end{aligned} \qquad (10)$$

With Eq. (9) and Eq. (10), HyperGen first estimates the following Jaccard similarity using the derived sketch HVs:

$$\begin{aligned} J_k(A,B) &= \frac{|\mathcal{S}_k(A) \cap \mathcal{S}_k(B)|}{|\mathcal{S}_k(A)| + |\mathcal{S}_k(B)| - |\mathcal{S}_k(A) \cap \mathcal{S}_k(B)|} \\ &= \frac{\mathbf{H}_A \cdot \mathbf{H}_B^T}{\|\mathbf{H}_A\|_2^2 + \|\mathbf{H}_A\|_2^2 - \mathbf{H}_A \cdot \mathbf{H}_B^T}. \end{aligned} \qquad (11)$$

Then ANI in Eq. (8) can be easily calculated.

2.3 Software Implementation and Optimization

HyperGen is developed using the Rust language, and the code is available at https://github.com/wh-xu/Hyper-Gen. We present the following optimizations to improve the computing speed and efficiency of HyperGen:

2.3.1 Sketch Quantization and Compression

Although the sketch HV has a compact data format with high memory efficiency, there still exists data redundancy in sketch HVs that can be utilized for further sketch compression. Our experimental observation is that the value range of sketch HVs is distributed within a bell curve (see Supplementary Fig. 1). Rather than store the full-precision sketch hypervector (*e.g.*, INT32), we perform lossless compression by quantizing the HV to a lower bit width. The quantized bits are concatenated together using bit-packing.

2.3.2 Fast HV Aggregation using SIMD

The inner loop of binary HV aggregation step in Algorithm 1 incurs significant runtime overhead when a large HV dimension D is applied. We develop a parallelized HV aggregation using single instruction, multiple data (SIMD) instruction to reduce the impact of increased HV aggregation time. As shown in Supplementary Fig. 2, the HV aggregation optimized by SIMD only takes negligible portion of the total sketching time.

2.3.3 Pre-computation for HV Sketch Norm

The L^2 norm of each sketch hypervector, $\|\mathbf{H}\|_2$, is precomputed during sketch generation phase. The L^2 norm value is stored along with the sketch hypervector to reduce redundant computations for the ANI calculation phase.

3 Evaluation and Results

3.1 Evaluation Methodology

3.1.1 Genome Dataset and Hardware Setting

The evaluation is conducted on a machine with a 16-core Intel i7-11700K CPU with up to 5.0GHz frequency, 2TB NVMe PCIe 4.0 storage, and 64GB of DDR4 memory. Unless otherwise specified, all programs are allowed to use 16 threads with their default parameters. Five genome datasets in Supplementary Table 2 are adopted for benchmarking. The datasets include: *Bacillus cereus*, *Escherichia coli*, NCBI RefSeq (Jain *et al.*, 2018), Parks MAGs (Parks *et al.*, 2017), and GTDB MAGs (Parks *et al.*, 2018). These datasets vary in terms of # of genomes, lengths, and sizes.

3.1.2 Benchmarking Tools

We compare HyperGen with five state-of-the-art tools, including Mash (Ondov *et al.*, 2016), Dashing 2 (Baker and Langmead, 2023), FastANI (Jain *et al.*, 2018), and ANIm (Kurtz *et al.*, 2004). Mash, and Dashing 2 are sketch-based tools for ANI estimation. In comparison, FastANI uses a mapping-based method while ANIm adopts the most accurate alignment-based method to calculate the ANI values. ANIm results are regarded as the ground truth. Specifically, we use NUCleotide MUMmer (Kurtz *et al.*, 2004) to generate the alignment results and then convert the alignment data into the corresponding ground-truth ANIs. HyperGen (similar to Mash and Dashing 2) is an ANI approximation tool for the high ANI regime. We follow previous work (Ondov *et al.*, 2016) and only preserve ANI values > 85. The versions and commands used are summarized in Supplementary Table 1. HyperGen uses k-mer size $k = 21$, scaled factor $S = 1500$ as suggested in previous works (Shaw and Yu, 2023; Hera *et al.*, 2023; Brown and Irber, 2016). Our analysis in Section 3.2.1 shows that the HV dimension $D = 4096$ achieves a good balance between ANI estimation error and sketching complexity. So we set it as the default parameter.

3.1.3 Evaluation Metrics

ANI Precision. One of the critical metrics for evaluating the effectiveness of a genome sketching tool is the precision of ANI estimation. We use three metrics to evaluate the ANI approximation errors: 1. mean absolute error (MAE), 2. root mean squared error (RMSE), and 3. mean percentage absolute error (MPAE). We also adopt the Pearson correlation coefficient to assess the linearity of the ANI estimate with respect to ground truth.
Computation and Memory Efficiency. An ideal genome sketching scheme should be able to generate compact sketch files at the cost of short runtime, especially for large-scale genomic analysis. To compare the computation and memory efficiency of evaluated tools, we measure and report the wall-clock runtime and sketch sizes during database search.

3.2 ANI Estimation Quality

In this section, we study the quality of ANI estimation by performing the following pairwise ANI experiment. First, 100 genome files are collected from each dataset. Then, each batch of 100 genome files is used to calculate the pairwise 100×100 ANI matrix.

3.2.1 HyperGen ANI Quality using Different Parameters

We first evaluate the impact of HyperGen's two algorithmic parameters: scaled factor S and HV dimension D on the final ANI estimation errors and linearity. The experimental results are depicted in Fig. 3, where the scaled factor S and the HV dimension D vary from 800 to 2000 and from 256 to 16384, respectively. It shows that: for all scaled factors, the ANI approximation errors decrease significantly as D increases from 256 to 4096. This is because a larger HV dimension can produce better orthogonality, which is helpful to reduce the approximation error of the set

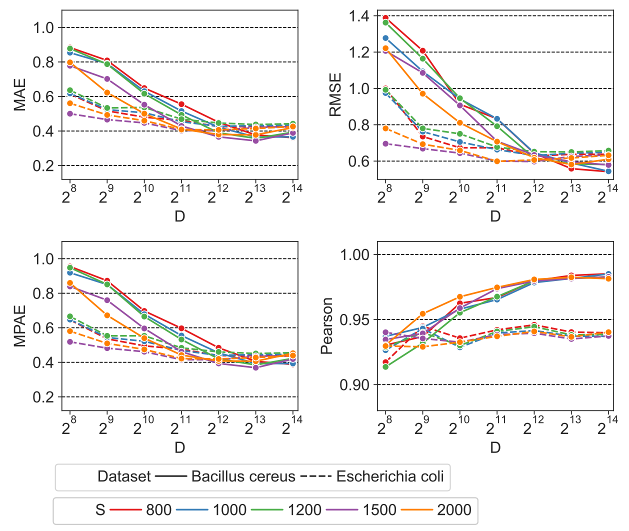

Fig. 3: Error metrics (MAE, RMSE, MPAE) and ANI linearity (Pearson coefficient) as a function of scaled factor S and HV dimension D.

Table 1. Error and linearity metrics for pairwise ANI estimation. (Underline: the best among sketch-based algorithms. Bold: the best among all algorithms.)

Dataset: *Bacillus cereus*					
Algorithm	k	MAE ↓	RMSE ↓	MPAE ↓	Pearson ↑
FastANI	16	**0.312**	**0.368**	**0.334**	**0.999**
Mash	21	0.399	0.591	0.430	0.981
Dashing 2	21	0.500	0.650	0.537	0.981
HyperGen	21	0.380	0.619	0.408	0.980

Dataset: *Escherichia coli*					
Algorithm	k	MAE ↓	RMSE ↓	MPAE ↓	Pearson ↑
FastANI	16	0.680	1.152	0.705	0.899
Mash	21	0.456	0.686	0.470	0.930
Dashing 2	21	0.464	0.704	0.479	0.930
HyperGen	21	**0.407**	**0.606**	**0.419**	**0.941**

Table 2. Sketch size, error, and linearity metrics for database search. (Underline: the best among sketch-based algorithms. bold: the best among all algorithms.)

Dataset: *Bacillus cereus*					
Algorithm	Sketch Size	MAE ↓	RMSE ↓	MPAE ↓	Pearson ↑
FastANI	-	**0.218**	**0.296**	**0.235**	**0.999**
Mash	4.7MB (1.8×)	0.542	0.678	0.586	0.996
Dashing 2	6.7MB (2.6×)	0.576	0.715	0.622	0.993
HyperGen	**2.6MB (1.0×)**	0.318	0.424	0.342	0.996

Dataset: *Escherichia coli*					
Algorithm	Sketch Size	MAE ↓	RMSE ↓	MPAE ↓	Pearson ↑
FastANI	-	0.215	**0.391**	0.221	**0.950**
Mash	36MB (1.8×)	0.226	0.529	0.231	0.877
Dashing 2	51MB (2.6×)	0.234	0.536	0.239	0.873
HyperGen	**20MB (1.0×)**	**0.153**	0.491	**0.156**	0.851

Dataset: NCBI RefSeq					
Algorithm	Sketch Size	MAE ↓	RMSE ↓	MPAE ↓	Pearson ↑
FastANI	-	0.443	0.522	0.452	0.968
Mash	14MB (1.9×)	0.204	0.251	0.208	0.983
Dashing 2	20MB (2.7×)	0.167	0.189	0.171	0.972
HyperGen	**7.4MB (1.0×)**	**0.135**	**0.164**	**0.138**	**0.991**

Dataset: Parks MAGs					
Algorithm	Sketch Size	MAE ↓	RMSE ↓	MPAE ↓	Pearson ↑
FastANI	-	**0.457**	**0.551**	**0.490**	**0.998**
Mash	65MB (1.9×)	1.090	1.298	1.137	0.990
Dashing 2	93MB (2.7×)	2.163	2.466	2.251	0.921
HyperGen	**34MB (1.0×)**	1.199	1.340	1.255	0.985

Dataset: GTDB MAGs					
Algorithm	Sketch Size	MAE ↓	RMSE ↓	MPAE ↓	Pearson ↑
FastANI	-	**0.436**	**0.592**	**0.469**	0.976
Mash	533MB (1.9×)	0.584	0.668	0.632	**0.980**
Dashing 2	770MB (2.7×)	0.994	1.283	1.078	0.892
HyperGen	**287MB (1.0×)**	1.001	1.124	1.085	0.963

intersection according to the theory in (Nunes *et al.*, 2023). But increasing the HV dimension larger than $D = 4096$ does not yield a significant error reduction or linearity improvement.

It is also observed that a smaller scaled factor S generally leads to a worse ANI approximation error when using the same HV dimension D. The reason behind this is: a smaller S that produces a larger hash threshold value as in Eq. (2), will generate a denser sampling of k-mers. This increases the size of sampled k-mer hash set. As a result, more binary HVs need to be aggregated to the sketch HV. The excessive number of binary HVs degrades the orthogonality between binary HVs, reducing the approximation accuracy for set cardinality. To balance between the quality and complexity of the ANI approximation, we choose $S = 1500$ and $D = 4096$ as the default scaled factor and HV dimension, respectively.

3.2.2 Comparison with Other Sketching Tools

We also compare the quality of the ANI estimation for various tools, including HyperGen, Mash, Dashing 2, and FastANI. For fair comparison, the three sketch-based tools (HyperGen, Mash, and Dashing 2) use the same sketch size. Other parameters are the same as their default parameters. Specifically, HyperGen uses $D = 4096$, while Mash and Dashing 2 use a sketch size of 1024. The k-mer size is $k = 21$ for HyperGen, Mash, and Dashing 2.

Table 1 summarizes the ANI error and linearity metrics with respect to the ground truth values on *Bacillus cereus* and *Escherichia coli* datasets. For the *Bacillus cereus* dataset, HyperGen is slightly inferior to the mapping-based FastANI and yields the lowest or second lowest ANI estimation errors with a comparable Pearson correlation coefficient compared to the other sketch-based tools (Mash and Dashing 2). In the *Escherichia coli* dataset, HyperGen consistently surpasses all other tools, providing both lower ANI approximation errors and better linearity. These experiments demonstrate that HyperGen is able to deliver high ANI estimation quality.

3.3 Genome Database Search

One critical workload that genome sketching tools can accelerate is the genome database search. Meanwhile, the genome database search can be extended to multiple downstream applications.

3.3.1 ANI Linearity and Quality

We extensively consider the five evaluated datasets as reference databases. We run FastANI, Mash, Dashing 2, and proposed HyperGen using the commands and queries listed in Supplementary Table 2. The execution consists of two steps: 1. All tools first generate reference sketches for the target database, 2. The second step is to search for the query genomes against the built reference sketches. Note that FastANI was unable to complete the database search on the Parks MAGs and GTDB datasets in one shot because it requires more memory than the available 64GB and experienced out of memory issues. We divided FastANI execution into smaller batches and measured the accumulative runtime.

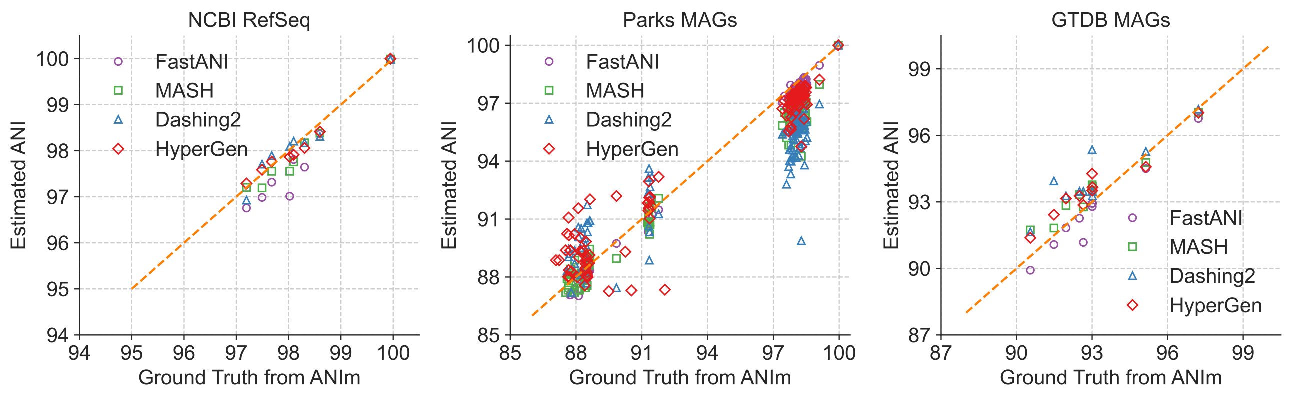

Fig. 4: Database search ANI comparison for FastANI, Mash, Dashing 2, HyperGen, and ground-truth ANIm on NCBI RefSeq, Parks MAGs, and GTDB MAGs datasets.

The estimated ANI values generated from each tool in NCBI RefSeq, Parks MAGs, and GTDB MAGs datasets are visualized in Fig. 4 by comparing with corresponding ground truth values from ANIm. It shows that HyperGen produces good ANI linearity compared to the ground truth results. Quantitative results in terms of numerical error and linearity metrics are summarized in Table 2.

In datasets *Bacillus cereus*, *Escherichia coli*, and NCBI RefSeq, HyperGen achieves the lowest ANI errors among the three sketching tools. This suggests that HyperGen delivers more accurate ANI estimations on most of the evaluated datasets compared to Mash and Dashing 2. Compared to mapping-based FastANI, HyperGen still shows competitive accuracy. In *Escherichia coli* and NCBI RefSeq, HyperGen outperforms FastANI in terms of most error metrics and produces comparable Pearson coefficients. This indicates that HyperGen is capable of achieving state-of-the-art error and linearity for large-scale genome search.

3.3.2 Memory Efficiency

The file sizes of the reference sketches generated by Mash, Dashing 2, and HyperGen, are listed in Table 2. We apply the *Sketch Quantization and Compression* technique to HyperGen. As a result, HyperGen consumes the smallest memory space among the three sketch-based tools. The sketch sizes produced by Mash and Dashing 2 are $1.8\times$ to $2.6\times$ of HyperGen's sketch sizes. This suggests that HyperGen is the most space-efficient sketching algorithm. Compared to original datasets with GB sizes, a compression ratio of $600 - 1200\times$ can be achieved by only processing the sketch files. This enables the large-scale genome search on portable devices with memory constraints. HyperGen's memory efficiency comes from two factors. First, the *Hyperdimensional Encoding for k-mer Hash* step converts discrete hash values into continuous high-dimensional sketch HVs, which are more compact than hash values. Second, HyperGen's *Sketch Quantization and Compression* provides additional $1.3\times$ compression through further removing redundant information in sketch HVs.

3.3.3 Runtime Performance

The wall-clock time spent on two major steps during database search: reference sketch generation and query search, is illustrated in Fig. 5. The reference sketching step is mainly bounded by the sketch generation process, while the search step is bounded by the sketch file loading and ANI calculation. HyperGen's reference sketching speed is the 2nd fastest and slightly (2% to 5%) slower than Dashing 2. The sketching speed of Mash is $1.4\times$ to $1.5\times$ slower than HyperGen. FastANI is a mapping-based tool that requires expensive sequence mappings. FastANI's sketching speed $3.4\times$ to $5.5\times$ slower than HyperGen. For query search speed, HyperGen has is the fastest one among all benchmarked tools. The search speedup of HyperGen over FastANI is $100\times$ to $> 3000\times$ because HyperGen does not need the expensive mapping process. Dashing 2's sketch search is less efficient than HyperGen. HyperGen's search speed is $1.3\times$ to $4.3\times$ of Dashing 2. Moreover, the speedup of HyperGen is more significant for larger datasets. Dashing 2 sketch size is about $2.6\times$ of HyperGen's so it takes more time to load sketch files. The reduced sketch size helps to save sketch loading time. Meanwhile, the HV sketch format of HyperGen allows us to adopt highly vectorized programs to compute ANI with a short processing latency.

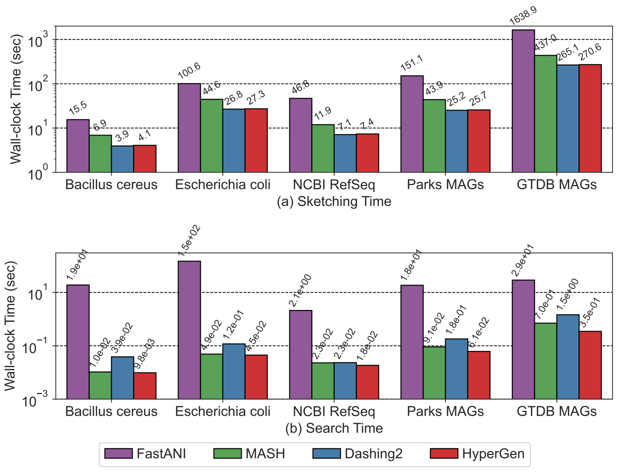

Fig. 5: Runtime performance comparison for FastANI (Jain *et al.*, 2018), Mash (Ondov *et al.*, 2016), Dashing 2 (Baker and Langmead, 2023), and HyperGen: (a) Reference sketching time, (b) Query search time.

4 Discussion and Conclusion

Fast and accurate estimation of Average Nucleotide Identity (ANI) is considered crucial in genomic analysis because ANI is widely adopted as a standardized measure of genome file similarity. In this work, we present HyperGen: a genome sketching tool based on hyperdimensional computing (HDC) (Nunes *et al.*, 2023; Kanerva, 2009) that improves accuracy, runtime performance, and memory efficiency for large-scale genomic analysis. HyperGen inherits the advantages of both FracMinHash-based sketching (Hera *et al.*, 2023) and DotHash (Nunes *et al.*, 2023). HyperGen first samples the k-mer set using FracMinHash. Then, the discrete k-mer hash set is encoded into the corresponding sketch HV in hyperdimensional space. This allows the genome sketch to be presented in compact vectors without sacrificing accuracy. HyperGen software implemented in Rust language deploys vectorized routines for both sketch and search steps. The evaluation results show that HyperGen offers superior ANI estimation quality over state-of-the-art sketch-based tools (Ondov *et al.*, 2016; Baker and Langmead, 2023). Meanwhile, HyperGen delivers not only the fastest sketch and search speed, but also the highest memory efficiency in terms of sketch file size.

Future directions of HyperGen include the following aspects. HyperGen can be extended to support a wider range of genomic applications. For example, in metagenome analysis, we can utilize HyperGen to perform the containment analysis for genome files as (Ondov *et al.*, 2019). To realize this, sketch HVs generated by HyperGen can be used to calculate max-containment index instead of ANI. The ANI estimation error and memory requirements of HyperGen can be reduced by considering the more accurate ANI estimation based on multi-resolution k-mers (Liu and Koslicki, 2022). On the other hand, the HV representation of sketch allows us to further accelerate HyperGen in advanced hardware architectures with high data parallelism. Previous work (Xu *et al.*, 2023) demonstrates that deploying HDC-based bioinformatics analysis on GPU exhibits at least one order of magnitude speedup over CPU.

Supplementary data

Supplementary data are available at Bioinformatics online.

Data availability

The source code of HyperGen used in this work is freely available at https://github.com/wh-xu/Hyper-Gen. All used datasets can be downloaded from https://gtdb.ecogenomic.org and http://enve-omics.ce.gatech.edu/data/fastani.

Conflict of interest

None declared.

Funding

This work was supported in part by the Center for Processing with Intelligent Storage and Memory (PRISM) SRC grant number 2023-JU-3135, CoCoSys, centers in JUMP 2.0, an SRC program sponsored by DARPA, and TILOS AI Research Institute (NSF CCF-2112665).

References

Baker, D. N. and Langmead, B. (2019). Dashing: fast and accurate genomic distances with hyperloglog. *Genome biology*, **20**, 1–12.

Baker, D. N. and Langmead, B. (2023). Genomic sketching with multiplicities and locality-sensitive hashing using dashing 2. *Genome Research*, **33**(7), 1218–1227.

Broder, A. Z. (1997). On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE.

Brown, C. T. and Irber, L. (2016). sourmash: a library for minhash sketching of dna. *Journal of open source software*, **1**(5), 27.

Chaumeil, P.-A. *et al.* (2022). Gtdb-tk v2: memory friendly classification with the genome taxonomy database. *Bioinformatics*, **38**(23), 5315–5316.

Ertl, O. (2021). Setsketch: filling the gap between minhash and hyperloglog. *Proceedings of the VLDB Endowment*, **14**(11), 2244–2257.

Hera, M. R. *et al.* (2023). Deriving confidence intervals for mutation rates across a wide range of evolutionary distances using fracminhash. *Genome Research*, pages gr–277651.

Hernández-Salmerón, J. E. *et al.* (2023). Fast genome-based delimitation of enterobacterales species. *Plos one*, **18**(9), e0291492.

Jain, C. *et al.* (2017). A fast approximate algorithm for mapping long reads to large reference databases. In *International Conference on Research in Computational Molecular Biology*, pages 66–81. Springer.

Jain, C. *et al.* (2018). High throughput ani analysis of 90k prokaryotic genomes reveals clear species boundaries. *Nature communications*, **9**(1), 5114.

Kanerva, P. (2009). Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation*, **1**, 139–159.

Kanerva, P. *et al.* (2000). Random indexing of text samples for latent semantic analysis. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 22.

Kang, J. *et al.* (2023). Accelerating open modification spectral library searching on tensor core in high-dimensional space. *Bioinformatics*, **39**(7), btad404.

Kurtz, S. *et al.* (2004). Versatile and open software for comparing large genomes. *Genome biology*, **5**, 1–9.

Lee, I. *et al.* (2016). Orthoani: an improved algorithm and software for calculating average nucleotide identity. *International journal of systematic and evolutionary microbiology*, **66**(2), 1100–1103.

Liu, S. and Koslicki, D. (2022). Cmash: fast, multi-resolution estimation of k-mer-based jaccard and containment indices. *Bioinformatics*, **38**(Supplement_1), i28–i35.

Matsakis, N. D. and Klock, F. S. (2014). The rust language. *ACM SIGAda Ada Letters*, **34**(3), 103–104.

Nunes, I. *et al.* (2023). Dothash: Estimating set similarity metrics for link prediction and document deduplication. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1758–1769.

Ondov, B. D. *et al.* (2016). Mash: fast genome and metagenome distance estimation using minhash. *Genome biology*, **17**(1), 1–14.

Ondov, B. D. *et al.* (2019). Mash screen: high-throughput sequence containment estimation for genome discovery. *Genome biology*, **20**, 1–13.

Parks, D. H. *et al.* (2017). Recovery of nearly 8,000 metagenome-assembled genomes substantially expands the tree of life. *Nature microbiology*, **2**(11), 1533–1542.

Parks, D. H. *et al.* (2018). A standardized bacterial taxonomy based on genome phylogeny substantially revises the tree of life. *Nature biotechnology*, **36**(10), 996–1004.

Parks, D. H. *et al.* (2020). A complete domain-to-species taxonomy for bacteria and archaea. *Nature biotechnology*, **38**(9), 1079–1086.

Sahlgren, M. (2005). An introduction to random indexing. In *Methods and applications of semantic indexing workshop at the 7th international conference on terminology and knowledge engineering*.

Shaw, J. and Yu, Y. W. (2023). Fast and robust metagenomic sequence comparison through sparse chaining with skani. *Nature Methods*, **20**, 1661–1665.

Sleem, L. and Couturier, R. (2020). Testu01 and practrand: Tools for a randomness evaluation for famous multimedia ciphers. *Multimedia Tools and Applications*, **79**, 24075–24088.

Soon, W. W. *et al.* (2013). High-throughput sequencing for biology and medicine. *Molecular systems biology*, **9**(1), 640.

Stephens, Z. D. *et al.* (2015). Big data: astronomical or genomical? *PLoS biology*, **13**(7), e1002195.

Xu, W. *et al.* (2023). Hyperspec: Ultrafast mass spectra clustering in hyperdimensional space. *Journal of Proteome Research*.

Zou, Z. *et al.* (2022). Biohd: an efficient genome sequence search platform using hyperdimensional memorization. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 656–669.