PanSpace: Fast and Scalable Indexing for Massive Bacterial Databases

Jorge Avila Cartes[1,*], Simone Ciccolella[1], Luca Denti[2], Raghuram Dandinasivara[3], Gianluca Della Vedova[1], Paola Bonizzoni[1, †], Alexander Schönhuth[3, †]

[1]DISCo, Univ. Milano – Bicocca, Viale Sarca, Milano, 20126, Italy
[2]Department of Applied Informatics, Faculty of Mathematics, Physics and Informatics, Comenius University in Bratislava, Mlynská dolina F1, Bratislava, 84248, Slovakia
[3]Genome Data Science, Faculty of Technology, Bielefeld University, Bielefeld, Germany
[*]first author
[†]joint last authors

April 4, 2025

Abstract

Motivation: Species identification is a crucial task in fields such as agriculture, food processing, and healthcare. The rapid expansion of genomics databases, especially with the growing focus on investigating new bacterial genomes in clinical microbiology, has surpassed the capabilities of conventional tools like BLAST for basic search and query procedures. A major bottleneck in microbiome studies is building indexes that enable rapid identification and classification of species from assemblies while scaling efficiently to AllTheBacteria Database, the current larger massive bacterial databases, making large-scale analysis feasible on a common laptop.

Results: We introduce `PANSPACE`, the first convolutional neural network-based approach that leverages dense vector (embedding) indexing, proven to scale up to 1 billion embeddings, to index and query very large bacterial genome databases. `PANSPACE` is designed to classify (draft) assemblies of bacteria. Compared to the most recent and competitive tool for this task, our index requires only ∼2GB of disk space for the AllTheBacteria Database, more than 40× less. Additionally, `PANSPACE` is ultra-fast in genomic queries, processing over 1,000 queries in under two minutes and half while maintaining high accuracy compared to the current state-of-the-art tool for the same tasks.

Availability: `PANSPACE` is available at https://github.com/pg-space/panspace.

1 Introduction

Determining the originating species of a DNA fragment is one of the most fundamental tasks in biology. This problem is usually tackled by using BLAST [2], one of the most popular, if not the best known tool in bioinformatics. BLAST queries a database storing large collections of annotated genomes by aligning sequenced DNA against it. Several fine-tuned heuristics are necessary to guarantee that those queries are answered in a reasonable time. Afterwards, the result of the alignments can be analyzed to classify the fragment according to the originating species.

In bacteria, the first definition of *species* was based on DNA-DNA hybridization (DDH), where two strains with a DDH of 70% or higher are considered members of the same species [49]. This threshold has been used as a reference point for bacterial species delimitation. However, DDH is time-consuming and labor-intensive. Furthermore, with the ever-increasing number of sequenced genomes, DDH becomes practically infeasible. Therefore, computational alternatives are preferred. One of the most widely preferred computational alternatives is the Average Nucleotide Identity (ANI). This has proven useful for establishing boundaries to delineate between species using whole genome sequencing or genes. ANI has emerged as a computational alternative for establishing species boundary using whole genome sequences (WGS) or genes [13, 38, 18, 29], as studies indicate that a DDH threshold of 70% corresponds to an ANI threshold of approximately 95-96% [13].

Genomic databases are becoming increasingly larger, now storing millions of genomes — NCBI RefSeq has 2.3 million genomes [15, 40], GISAID 20 millions [1], and ENA 1.9 millions [17]. Querying DNA sequence databases with alignment-based search tools requires a lot of computational resources. In fact, it is impractical to run BLAST on any of those databases, outside of the very large NCBI cluster. When a research lab wants to complete the same task, it has to employ alignment-free approaches to classify query sequences, which is exactly the focus of this paper.

More precisely, we focus on the classification of prokaryotes at the taxonomic level of species. The field of bacterial taxonomy was significantly advanced by the seminal work [52, 51]. In that paper, they have constructed a phylogenetic tree using only the 16S rRNA gene, also known as the subunit rRNA gene, a highly conserved and ubiquitous gene in prokaryotes [45].

The 16S rRNA gene has been used for the identification of bacterial species [36, `TYGS`] not only because it is present in all prokaryotes, but due to its efficiency in conducting pairwise comparisons since it is short (around 1485 bp long). However, in real settings, especially when working with draft assemblies, the 16S rRNA gene may be missing, making the classification task unfeasible. Moreover, the short length of the 16S rRNA gene also means that its ability to discriminate between similar species is limited. For this reason, we have recently witnessed a move towards methods that use longer regions, and whole genomes if possible. In this case, the length of the genome sequences stored in the database results in computational issues, since pairwise alignments are usually infeasible to compute, leading to alignment-free methods. Alignment-free methods, such as [27, `fIDBAC`] and [32, `GAMBIT`], are based on the analysis of k-mers, that is the short portions of the sequences involved. All of these approaches need a curated database storing thousands of genomes. The increasing number of available data has opened the door to exploring the use of a larger set of sequences, a pangenome; a collection of genomic sequences to be analyzed jointly or to be used as a reference [1].

The Type Genome Server database (`TYGS`)[2] [36] contains more than 20 000 microbial type-strain genomes and is updated weekly [3, 8]. `TYGS` incorporates quality checks to reject poor quality assemblies, and incorrectly identified species.

`fIDBAC` [27] is another curated database storing bacterial genome assemblies and metadata including the strain. `fIDBAC` uses CheckM [37] to evaluate the completeness and contamination of each genome.

Some additional curation steps, such as using the information of the LTP [55] database and strain information from LPSN [55], allow to discard wrongly labeled genomes. Overall, the `fIDBAC` database contains more than 12 000 bacterial genome assemblies, covering 9 827 species and 2 448 genera. In this case the classification is performed by aliging the 16S sequences against the LTP database, and by matching k-mers from the query sequence against the k-mers in the curated database using KmerFinder[7].

`GAMBIT` [32] is a curated database of more than 48 000 bacterial genomes from the NCBI RefSeq database spanning 1 414 species and 454 unique genera. To perform species identification, an approximate Jaccard similarity between the query and the k-mers of the genomes in the database is computed.

Most recently, the largest available bacterial dataset, AllTheBacteria [17], was released, and it comprises all known bacterial genomes (more than 1.9 million draft assemblies) — it is available in the European Nucleotide Archive (ENA).

AllTheBacteria has been used by two alignment tools — `Phylign` [5] and `LexicMap` [43] — that first query the database from a set of reads or contigs, then compute a posterior alignment against queried assemblies. Both construct k-mer-based indexes, `Phylign` leveraging a 248GB COBS index [4], while `LexicMap` extracts some k-mers for each genome that guarantee matches in short windows, which needs a 2.46TB index. The alignment step is done by minimap2 [26] in the case of `Phylign`, and with the wavefront alignment algorithm [34] in the case of `LexicMap`. Even though classification is not an explicit part of those tools, it can be easily added by aggregating the query results of each contig from a draft assembly.

Several read classification tools are in the realm of metagenomics, where the goal is to label each read in a sample at the species level. Examples of these tools are `Kraken` [54], `Kraken2` [53], `Centrifuge` [22], and `Centrifuger` [44], all of them work with short reads, while `Centrifuger` can work with long reads as well. `Taxor` [46] is another tool that only handles long reads.

The current state of the art for (bacterial) genome classification is `GSearch` [57], an alignment-free

[1] gisaid.org ACCESSION DATE 03-03-2025
[2] https://tygs.dsmz.de/

genome search tool that can work at pangenome scale, by combining some memory-efficient sketching algorithms such as MinHash [6] or HyperLogLog [11] for genomic distance estimation, and Hierarchical Navigable Small World graphs (HNSW) [33] for finding nearest neighbors and identifying the originating species. A subset of the k-mers of the query genome and its occurrences is selected as a feature representation of each assembly, and an index is created based on HNSW ($\sim$ 80GB of disk space for the AllTheBacteria dataset). The index is then queried using a weighted Jaccard-like similarity of their feature representation.

Machine learning tools have been proposed to speed up classification and avoid dependency on databases, thus decreasing the amount of computational resources needed for classification. Most notably, `HiTaxon` [47] and `Taxometer` [24] are designed to work on bacterial genomes, while `DeepMicrobes` [28] is restricted to gut microbiota. Some other tools with a more limited scope are `EukRep` [50] and `Tiara` [21], which classify at the domain level, and `MycoAI` [39] which targets fungal sequences. These tools have as main purpose the work on eukaryotic sequences, however, they have also been shown to work on bacterial data.

Some considerations on database size are in order. Small and manually curated databases are easier to manage and require smaller computational resources, but they are subject to selection bias. This is especially troubling in bacteria, due to their high mutation rate. In fact, two strains from the same species can differ up to 45% [9]. Comprehensive databases inherently avoids this bias, but going to pangenome scale poses other challenges. Most notably, they require huge computational resources, which limits their accessibility and practical use to the broader research community. In contrast, machine learning classification methods have the advantage of being faster and generalizable to unseen data (if well trained), but are limited to the task of classification, *i.e.* the assignment of a taxonomy level given a query sequence. In bioinformatics pipelines it is desirable to identify not only a taxonomy level but also sequences in the database that better align with the query, in order to perform downstream analyses, a feature that index-based methods preserve.

In this work we propose `PANSPACE`, a tool designed to classify bacterial genomic sequences by providing new efficient approaches for both indexing and querying bacterial pangenomes, obtaining accurate classification at the taxonomy level of species. We show that `PANSPACE` can index the AllTheBacteria dataset using less than 2GB of disk space. Moreover, can answer a thousand queries in less than 3 minutes, `PANSPACE` queries require less than 6GB of RAM for indexing and querying, making it possible to run `PANSPACE` on a personal computer, without uploading data to a cloud server. We compare our results with `GSearch`, showing that the `PANSPACE` index is 45$\times$ smaller and requires 22$\times$ less RAM for its construction, with comparable classification accuracy at the species level, and better accuracy at the genus level.

1.1 Overview of our approach

We propose `PANSPACE`, a framework to index genomic sequences in a continuous n-dimensional space, referred to as *Embedding space*. Genomic sequences are represented by the Frequency matrix of the Chaos Game Representation of DNA (FCGR) [10], where the frequency of k-mers (4^k in total) are placed in a $2^k \times 2^k$ matrix, where each cell correspond to a k-mer scaled frequency. In this representation, k-mers sharing an l-long suffix (with $l < k$) are placed contiguously in a $2^{k-l} \times 2^{k-l}$ square matrix (see Figure 2). The FCGRs are used as feature representations of the sequences and then mapped to the Embedding space ($n \ll 4^k$) by leveraging a Convolutional Neural Network designed to exploit the k-mer ordering in the FCGR with Metric Learning for clustering based on species labels, and optionally the use of Autoencoders when no labels are available. The embedding representations are indexed with FAISS [20], an index for dense vectors, proven to scale up to 1 billion embeddings. The species identification step is done by querying the index and assigning the most common species among its closest neighbors w.r.t. the vector representation of the assemblies.

1.2 Our contribution

- We propose the first embedding-based index for draft assemblies using the FAISS [20] index, with a Metric learning-based method that exploits the FCGR representation of assemblies for the construction of these embeddings.

- We index the largest publicly available database of bacterial draft assemblies (the AllTheBacteria

Tool	Taxonomy Classification	Type Query	Machine Learning	Index-based method	Minimum Taxonomy Level
DeepMicrobes [28]	Yes	short read	Yes	No	species
Kraken [54]	Yes	short read	No	Yes	species
Kraken2 [53]	Yes	short read	No	Yes	species
Centrifuge [22]	Yes	short read	No	Yes	species
Centrifuger [44]	Yes	short and long reads	No	Yes	species
Taxor [46]	Yes	long read	No	Yes	species
HiTaxon [47]	Yes	short read	Hybrid	Hybrid	species
Taxometer [24]	Yes	contig	Yes	No	species
EukRep [50]	Yes	5000 bp	Yes	No	domain
Tiara [21]	Yes	contigs	Yes	No	domain
MycoAI [39]	Yes	type query	Yes	No	species
LexicMap [43]	No	short and long reads (>500bp)	No	Yes	species
Phylign [5]	No	reads/contigs	No	Yes	species
TYGS [36]	Yes	assembly	Partial	Yes	species or subspecies
fIDBAC [27]	Yes	assembly	No	Yes	species
GAMBIT [32]	Yes	assembly	No	Yes	genus or species
GSearch [57]	Yes	assembly	No	Yes	species
PANSPACE	Yes	assembly	Yes	Yes	species

Table 1: List of tools tailored and/or adaptable for taxonomy classification in bacteria. For each tool, we point out if: it can be used for taxonomy classification, the type of query sequence, if it is a machine learning-based method (Hybrid means that part of the approach is machine learning-based), if it rely on an index (Hybrid means that part of the approach is index-based), and the minimum taxonomy level of the query results (when more than one taxonomy level is defined, it means that the tool can output either of the two depending on its confidence). Tiara and EukRep have as their main purpose the classification of eukaryotic sequences, while MycoAI fungal sequences, but have included or can be extended for prokaryotic taxonomy classification. LexicMap and Phylign are alignment-based tools where query results can be used for taxonomy classification. For Taxometer, the minimum taxonomy level depends on the dataset used to train the model. TYGS is provided as a web service. HiTaxon is an ensemble method that combines machine learning and index-based methods.

data set [17]), requiring approximately 2 GB of disk space, a 40× improvement compared to the state-of-the-art `GSearch`.

- We describe a method for the rapid identification of the species associated with an isolated (draft) assembly; the idenfication of species is based on a database of labeled assemblies.
- We experimentally show that the ANI values between a query assembly and the assemblies resulting from querying our index are in correspondence with the 95 ANI value threshold defined for ascertaining that two assemblies belong to the same species.

2 Material and Methods

Let us define by $\Sigma = \{A, C, G, T\}$ the DNA alphabet, and the extended DNA alphabet by $\Sigma_N = \{A, C, G, T, N\}$, where N indicates a null call. Consider a collection $\mathcal{S}$ of (possibly draft) assemblies, where each assembly $s \in \mathcal{S}$ could be a set of more than one (contig) sequence over Σ_N, and a set $\mathcal{L}$ of species (labels). We assume there exists a labeling function $\lambda : \mathcal{S} \mapsto \mathcal{L}$ that assigns a species in $\mathcal{L}$ to each assembly $s \in \mathcal{S}$ — learning such a function is the main goal of our method. The mapping $\mathcal{F}_k : \mathcal{S} \mapsto (\mathbb{N} \cup \{0\})^{4^k}$ computes the ocurrences of all k-mers for an assembly.

2.1 Chaos Game Representation of DNA

Let us now introduce the key ingredients related to the FCGR representation of an assembly, which is the matrix representation of its k-mer occurrences mentioned in the previous section. We first define the Chaos Game Representation (CGR) of a DNA sequence s over the alphabet Σ. Observe that, as will be explained later, the FCGR k-mer representation is essentially a 2D-image (see Figure 1).

The CGR encoding of a DNA (and RNA) sequence is a coordinate in $\mathbb{R}^2$, and is formally defined [19] as follows.

Definition 1 (Chaos Game Representation of DNA (CGR))**.** *Given a sequence $s = s_1 \cdots s_n \in \Sigma^+$, the CGR encoding of s is the ordered pair (x_n, y_n) which is defined iteratively as*

$$(x_i, y_i) = \frac{1}{2}\Big((x_{i-1}, y_{i-1}) + g(s_i)\Big), \text{ if } i \geq 1 \tag{1}$$

where $(x_0, y_0) = (0, 0)$ and,

$$g(s_i) = \begin{cases} (1,1) & s_i = A \\ (-1,1) & s_i = C \\ (-1,-1) & s_i = G \\ (1,-1) & s_i = T \end{cases} \tag{2}$$

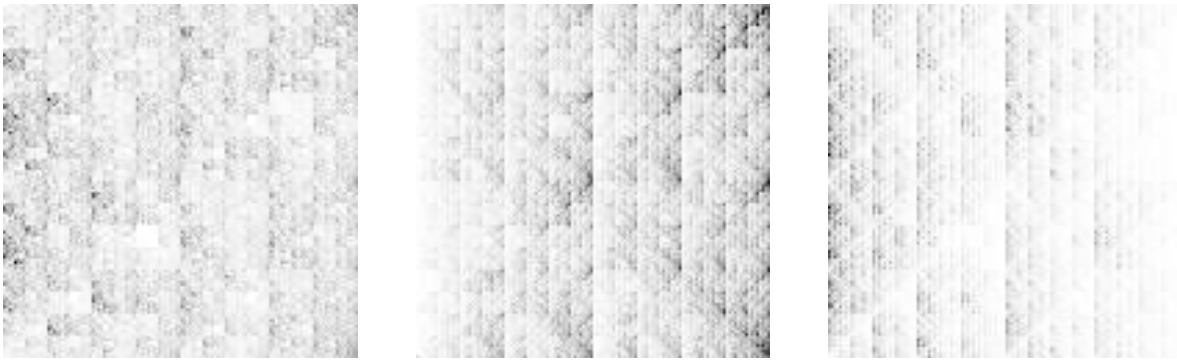
When plotting all the points resulting from iteratively encoding all characters in a DNA sequence, we obtain images like the ones shown in Figure 1. This requires an entire sequence over the alphabet Σ, and poses two challenges in real settings: sequences usually have missing data, *i.e.* they are defined over the alphabet Σ_N, and they are often present as draft assemblies; as collections of contigs. The CGR encoding of a sequence is reversible, which means that each sequence has a unique 2-dimensional position in the square $[-1, 1]$, in particular, we can use it to find the positions of k-mers in the square (see Figure 2). As shown in [10, 48] the CGR visual representation is well approximated by k-mer occurrences, where k-mers containing an N are excluded.

Definition 2 (Frequency matrix of CGR (FCGR))**.** *Let $s = s_1 \cdots s_n \in \Sigma_N^+$ be a sequence, and let k be an integer. Then the Frequency matrix of CGR, in short FCGR, of the sequence s is a $2^k \times 2^k$ 2-D matrix $F = (a_{i,j}), 1 \leq i, j \leq 2^k, i, j \in \mathbb{N}$. For each k-mer $b \in \{A, C, G, T\}^k$, we have an element $a_{i,j}$ in the matrix F, that is equal to the number of occurrences of b as a substring of s. Moreover, the position (i, j) of such element is computed as follows:*

$$i = \quad 2^k - \lceil 2^{k-1}(y+1) \rceil + 1 \tag{3}$$

$$j = \quad \lceil 2^{k-1}(x+1) \rceil \tag{4}$$

where (x, y) is the CGR encoding for the k-mer b.



(a) Salmonella enterica (b) Streptococcus pneumoniae (c) Mycobacterium tuberculosis

Figure 1: **Examples of FCGR** of assemblies from 3 species from the AllTheBacteria dataset. From left to right, Salmonella enterica (SAMD00003784), Streptococcus pneumoniae (SAMD00032020), and Mycobacterium tuberculosis (SAMD00111306). Each image corresponds to the visualization of the FCGR in 8-bits (k-mer frequencies rescaled to the interval $[0, 255]$) for $k = 7$. White color in a pixel means that the k-mer encoded has the minimum value of frequency found in the assembly. Black color represents k-mers with the maximum frequency in the assembly. Gray colors correspond to k-mer frequencies lying between the minimum and maximum values.

It is immediate to obtain a 2-D image from an FCGR matrix; it suffices to rescale all its values to the interval $[0, 255]$ (8 bits). Examples of these images for three different species are shown in Figure 1, and the ordering that k-mers adopt in the representation is shown in Figure 2, for $k \in \{1, 2, 3\}$, we can see that k-mers sharing their suffix are close in the matrix.

2.2 Convolutional Neural network for the FCGR

Convolutional layers [25], the building block of Convolutional Neural Networks (CNN), are commonly used to process spatially structured data, such as images. Formally, given an input tensor $X \in \mathbb{R}^{H \times W \times C_{in}}$, where H and W denote the height and width of the input, respectively, and C_{in} the number of input channels, a convolutional layer applies a set of K learnable filters $W_j \in \mathbb{R}^{F \times F \times C_{in}}$, with $j \in [1, K]$ and F known as the kernel size, to generate an output feature map $Y \in \mathbb{R}^{H_{out} \times W_{out} \times K}$. The dimensions H_{out} and W_{out} depend on the kernel size F, and another parameter called stride, which controls the step size of the filter as it moves across the input, in addition, can be affected if padding of the output is used across the height or width dimensions.

When the FCGR is the input tensor X, the filters W_k correspond to a 2-dimensional matrix, and we have a single channel. As we can see in Figure 1, different species show different patterns when seen through the lens of the FCGR, these patterns are the result of how k-mer occurrences are placed in the 2-dimensional matrix. In Figure 2 we can illustrate this ordering for $k = 3$, notice for example that in the top-left, the sub-square with center CC contains the 3-mers ACC, CCC, GCC, and TCC, all sharing the same 2-long suffix CC. More generally, given k, in the FCGR representation, k-mers sharing the l-long suffix ($l < k$) are contained in a sub-square of dimension $2^{k-l} \times 2^{k-l}$ in the FCGR. This observation is useful for using the k-mer ordering in the FCGR when defining a convolutional layer to exploit this ordering. We define a convolutional layer, named $ConvFCGR$, with kernel size 2^{k-l} and stride 2^{k-l} to be applied over the FCGR matrix, in this case, each convolutional filter processes only the number of occurrences of the k-mers sharing a given l-long suffix, for $0 < l < k$.

Under the umbrella of supervised metric learning, we propose the `CNNFCGR` (see Figure 3), a convolutional neural network defined by stacking $ConvFCGR$ layers and whose output is an n-dimensional vector, the so-called *embedding*. Unlike traditional supervised learning approaches designed for mere classification into a fixed number of classes, metric learning focuses on learning distance metrics or similarity measures tailored to some specific tasks. The goal of metric learning is to guarantee that similar instances are closer in the embedding space, and dissimilar ones are further apart, clustering the input data in the n-dimensional space. This is achieved using ad-hoc loss functions, most notably contrastive loss [14], or triplet loss [41] by directly optimizing the embeddings.

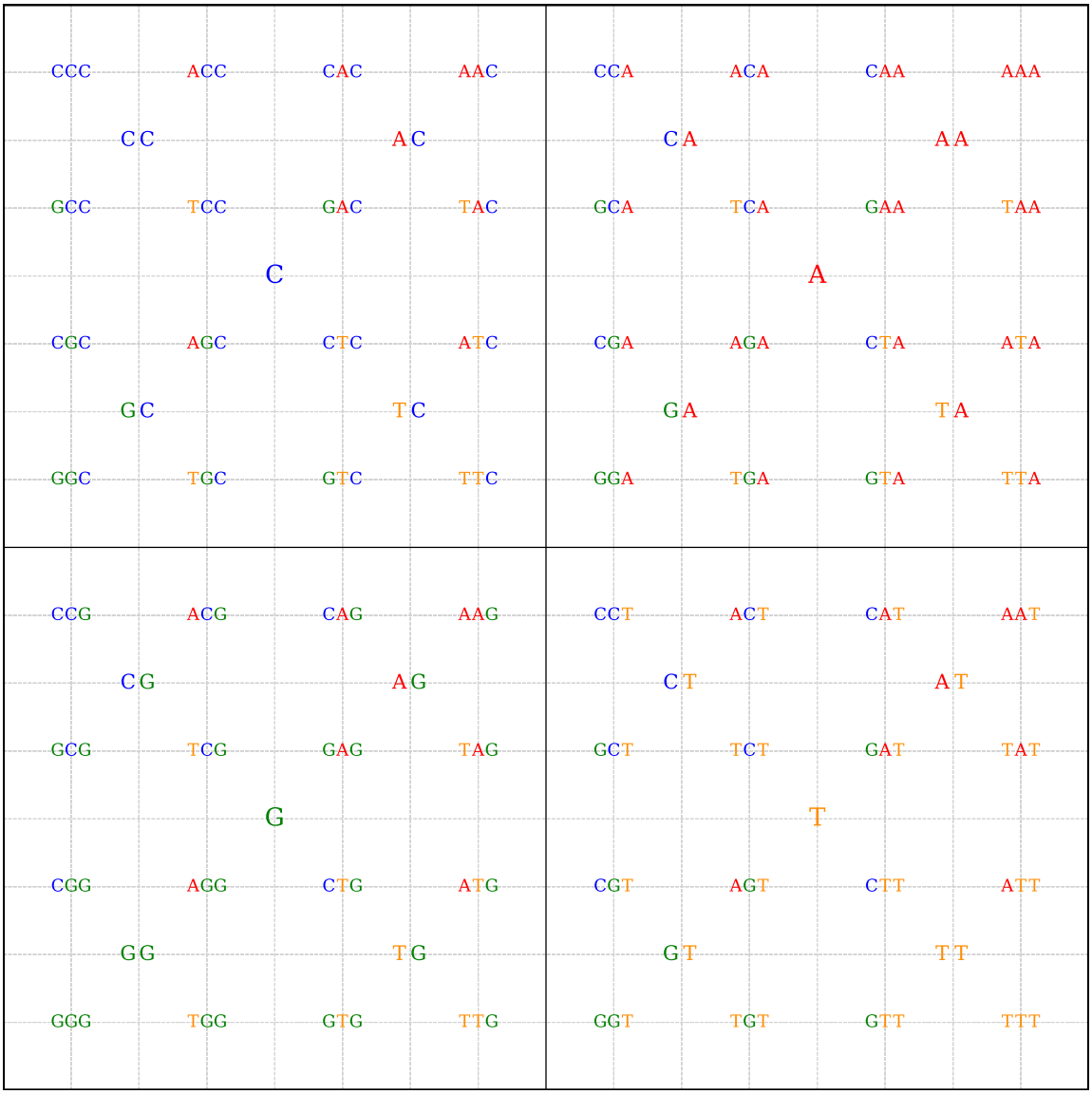


Figure 2: **Position of each k-mer** (for $k \in \{1, 2, 3\}$) in the square $[-1, 1]^2 \subset \mathbb{R}^2$ given by the CGR encoding. Each nucleotide is assigned to a vertex of the square to guide the encoding of k-mers as in Definition 1. We can observe that a subdivision of one level of a sub-square (*i.e.* from k to $k+1$) corresponds to adding the letters A, C, G, T as prefixes to the k-mer encoded in the center of such sub-square, following the orientation of the vertices.

2.2.1 ConvFCGR layer

The $ConvFCGR$ layer of level l corresponds to a convolutional layer with kernel size 2^l, stride 2^l, and no padding. Hence, for an input of size $2^k \times 2^k$, the $ConvFCGR$ of level l outputs a tensor of dimension $2^{(k-l)} \times 2^{(k-l)}$. Given l levels and f filters, the convolutional FCGR layer is denoted by $ConvFCGR(l, f)$. In Figure 4 are illustrated $ConvFCGR(1, 1)$ and $ConvFCGR(2, 1)$ on an input of dimension $2^3 \times 2^3$.

2.2.2 Encoder: `CNNFCGR` architecture

Given an input tensor of dimensions $(2^k \times 2^k \times 1)$ (FCGR from k-mers with depth channel included) the encoder is defined as a concatenation of $ConvFCGR(1, 4l)$ layers, $l \in \{k, k-1, \ldots, 4\}$ (in that order) each one followed by the ReLU activation function [12] and a batch normalization in the channel

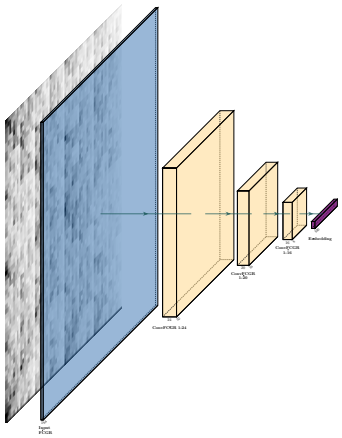
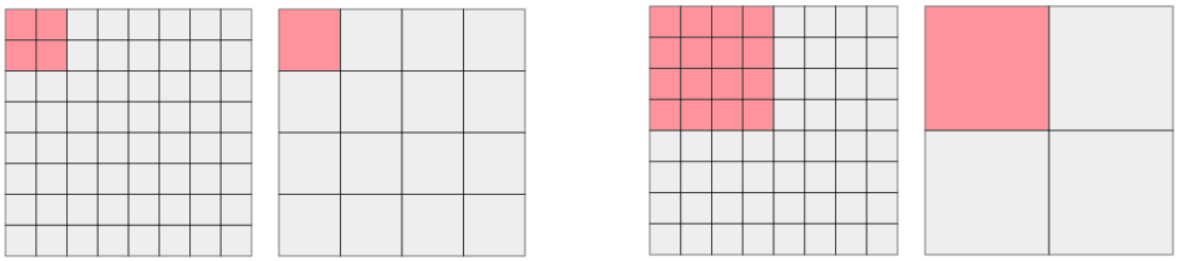


Figure 3: `CNNFCGR` architecture with input size $2^6 \times 2^6 \times 1$, representing the FCGR of a (draft) assembly. The architecture is a concatenation of three $ConvFCGR$ layers of level 1, and the number of filters $24, 20$, and 16, respectively. The output of the encoder is an L2 normalized n-dimensional vector (**Embedding**), represented by a dense layer with n neurons. The ReLU activation function follows all $ConvFCGR$ layers, and batch normalization in the filter axis.



(a) (b)

Figure 4: **ConvFCGR layer**. Gray grids correspond to the input and output of the layers. Red grids represent the position where a filter of a layer is applied. In (a) a $ConvFCGR(1,1)$ is applied to the input (FCGR) tensor of size $2^3 \times 2^3$, level $l = 1$ correspond to a layer with stride and kernel equal to 2, *i.e.* the output is a tensor of size $2^2 \times 2^2$. In (b) $ConvFCGR(2,1)$ is applied to the same input. This time, since $l = 2$, the layer uses 16 positions of the tensor, and outputs 1.

axis. The number of layers is equal to $k - 3$, so given an input of dimension $(2^k \times 2^k \times 1)$, it outputs a $(2^4 \times 2^4 \times 16)$, *i.e.* 1024 features. This is then flattened and fed to a dense layer of n neurons, with an L2 normalization, so embeddings have a norm equal to 1, and Euclidean distances between embeddings lie in $[0, 2]$. Notice that the choice of stopping at $k = 4$ is to consistently decrease the number of dimensions of the feature space up to the embedding layer so that we can manage values of n up to 1000. We choose $n \in \{128, 256, 512\}$ for our experiments. We show in Figure 3 the `CNNFCGR` architecture for 6-mers.

2.3 Learning embedding representation of assemblies

In what follows, we describe how the `CNNFCGR` architecture is trained. It receive as input the FCGR matrix with rescaled, and use the Rectified Adam optimizer [30] with a learning rate of 0.001, combined with the Lookahead optimizer algorithm, which improves the learning stability and lowers the variance of its inner optimizer with negligible running time and memory usage [56].

2.3.1 Preprocessing of the FCGR matrix

Each FCGR is scaled to the interval $[0, 1]$ before being fed to the neural networks, by dividing each k-mer ocurrence value by the maximum of them.

2.3.2 Metric Learning

In metric learning, in the context of supervised learning, the goal is to embed representations of the input data learned such that embeddings sharing a label are closer in Euclidean distance than embeddings with different labels. The `CNNFCGR` (described in Section 2.2.2) is designed to create such embeddings from the FCGR of assemblies based on their species labels. It receives as input the FCGR matrix and outputs a n-dimensional vector. We use the triplet loss function [41] to learn such embedding representations.

Embedding assemblies with the triplet loss
Given a triplet (x_i^a, x_i^p, x_i^n), the triplet loss is defined as follows,

$$\mathcal{L}_{\text{triplet}} = \min\left(0, \|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha\right) \tag{5}$$

where x_i^a is the anchor sample, x_i^p is the positive sample (same class as the anchor), x_i^n is the negative sample (different class from the anchor), $f(\cdot)$ is the embedding function that maps inputs into the feature space, $\|\cdot\|_2$ is the Euclidean (L2) distance, α is the margin enforced between positive and negative pairs, usually defined in the interval $[0.1, 0.5]$. In our case, $f(\cdot)$ denotes the encoder, and the pair (x_i^a, x_i^p) are two FCGRs from the same species, while x_i^n corresponds to an FCGR of a different species.

Notice that by the definition of $\mathcal{L}_{\text{triplet}}$ not all triplets might contribute to the learning process. In fact, those triplets (x_i^a, x_i^p, x_i^n) such that

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha \geq \|f(x_i^a) - f(x_i^n)\|_2^2 \tag{6}$$

already satisfy the desired separability condition, hence $\mathcal{L}_{\text{triplet}} = 0$. In this case, we the computation of the gradient is meaningless, therefore it does not influence the update of weights in the neural network.

As explained in [41], choosing triplets is crucial to use the triplet loss effectively, and a triplet of interest (for which $\mathcal{L}_{\text{triplet}} \neq 0$) can be categorized into *semi-hard* and *hard* triplets.

In this context, given an anchor x_i^a, a hard-positive is computed as the $\text{argmax}_{x_i^p} \|x_i^a - x_i^p\|$, while a hard-negative is computed as the $\text{argmin}_{x_i^n} \|x_i^a - x_i^n\|$. These examples are the most difficult to separate among those that violate the condition from Equation (6). Since computing the argmax and argmin is computationally expensive over the entire dataset (*e.g.* for a dataset of N elements, the number of possible triplets is $O(N^3)$), in practice, these *hard* triplets are chosen from a mini-batch (the subset of data used in each iteration of an epoch). A drawback of selecting the hardest negatives is that they can lead to bad local minima early on in training [41]. On the other hand, a triplet (x_i^a, x_i^p, x_i^n) is called *semi-hard* when

$$\|f(x_i^a) - f(x_i^p)\|_2^2 < \|f(x_i^a) - f(x_i^n)\|_2^2 < \alpha \tag{7}$$

so even when x_i^p is closer to the anchor than x_i^n, it still lies inside the margin α.

The suggested procedure in [41] when using the triplet loss is to start training with semi-hard triplets, and then focus on hard triplets. In our experiments we saw that the use of hard triplets (after training with semi-hard triplets) worsen the results, so we considered only semi-hard triplets for our trainings.

2.4 Large bacterial data sets

Our goal is to index and query large bacterial datasets. In [17] the collection of all bacterial data in the European Nucleotide Archive (ENA) available up to 2018 was consolidated into a data set of around 661K (draft) assemblies. Then it was updated with all the bacterial assemblies up to 2023, and named as AllTheBacteria [17] data set.

The assemblies in these two data sets were assembled with the same pipeline using short reads and are isolated assemblies, *i.e.* they contain a unique species. The main differences between these data sets

Dataset	High quality		All	
	Assemblies	Species	Assemblies	Species
661K	639 981	2336	661 405	6997
AllTheBacteria	1 857 792	10 542	1 932 812	11 445

Table 2: **Bacterial datasets.**

The total number of species in the AllTheBacteria database was obtained by counting all unique values in the column label of the file SAMPLE2SPECIES2FILE.TSV.GZ from https://ftp.ebi.ac.uk/pub/databases/AllTheBacteria/Releases/0.2/metadata/, since only the number of species of high-quality assemblies was reported in the manuscript.

are (1) how a *high-quality* assembly is defined and (2) how the assemblies are assigned a unique species label. In the 661K dataset, the abundance of species in the sample is obtained with a Kraken [53]-Bracken [31] pipeline, and the label of an isolated assembly is defined as the most abundant species in the sample if its abundance exceeds 90%. In contrast, in the AllTheBacteria data set, the assignment of a label consists of estimating the abundance of species directly from the isolated assembly using sylph [42], where the most abundant species in the assembly is assigned as its label if the abundance is greater than 99%. The number of assemblies and species for both datasets is reported in Table 2.

High-quality assemblies are defined by several criteria, based on the length of the assembly, number of contigs, N50, and the most abundant species. Under these high-quality criteria, the 96.76% and 96.12% of the assemblies are considered high-quality in the 661K and AllTheBacteria data sets, respectively.

2.4.1 Mini-batch generation and training configuration

A mini-batch consists of a subset of the data set randomly sampled to be fed to a network in the training phase. For example, in the 661k bacterial data set, we find 2336 species belonging to high-quality assemblies, while in the AllTheBacteria bacteria data set, the number of species reported is 10542 (see Table 2), with most of the assemblies belonging to only a small portion of the available species, *e.g.* the top 10 species constitute the 75% of the high-quality data set. When dealing with a highly imbalanced data set and a large number of labels, the generation of mini-batches requires special considerations, since a random selection will most likely lead to discard the underrepresented species. With the triplet loss, we must provide a way to subsample the data set for creating (either semi-hard or hard) triplets. For this purpose, we use a mini-batch generator that first randomly selects a portion of the labels (species), and then selects a fixed number of FCGRs for each label. Thus, each time we train the `CNNFCGR`, the triplets are created from a balanced data set with a restricted number of possible anchors.

We selected a mini-batch size of 512 assemblies, corresponding to (randomly selected) 64 species, and multiplied by a factor of 5 the number of mini-batches generated for training and validation steps, *i.e.* the number of mini-batches corresponds to 5× the ratio between the FCGRs in the training set (as well in the validation set) and the mini-batch size, thus increasing the number of triplets seen during each training epoch.

We set the training of the `CNNFCGR` to 100 epochs, with a patience early stopping of 30 epochs, and a patience learning rate of 15 epochs with a factor of 0.1, both monitoring the validation loss.

Training, validation, and test sets were built as follows, for each species with more than 12 samples in the dataset, we randomly split them into training, validation, and test sets in proportion 80 : 10 : 10. We consider only species with this representativity since in our experiments we want to query the index (built from training and validation sets) and retrieve up to 11 neighbors, so we can be sure that there are 11 assemblies for each species in the index.

2.5 Creating the Index

We recall that our objective is to learn a function $f : \mathcal{S}_k \mapsto \mathbb{R}^n$ (for $n \ll 4^k$) that reduces the dimensionality of the k-mer occurrences of an assembly by encoding some relevant information in an n-dimensional vector (embedding) for its classification, where $\mathcal{S}_k$ represents the space of k-mer

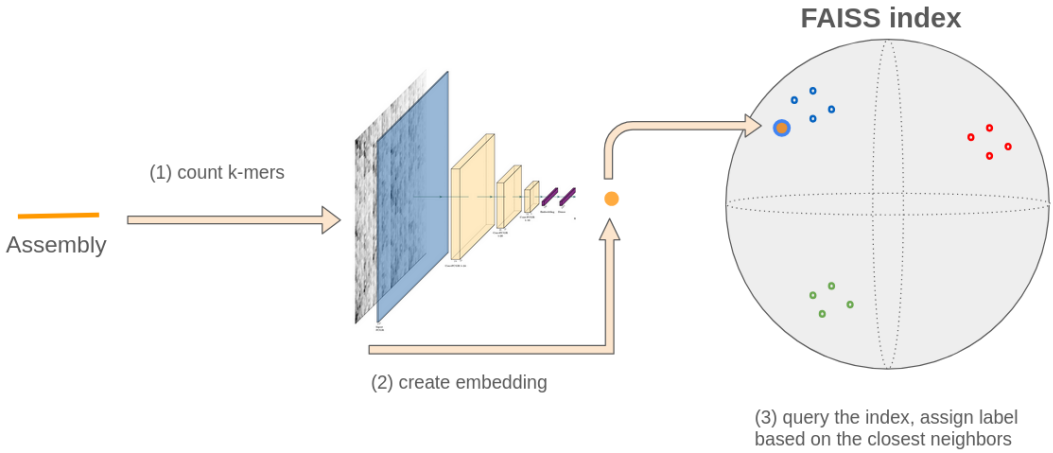


Figure 5: **Query index**. Given a query (draft) assembly, we start by (1) counting k-mers and creating its FCGR representation. Then (2) the FCGR is fed to the Encoder to produce the embedding representation. Finally, (3) we query the FAISS index to retrieve its K-closest neighbors and assign a label to the query assembly based on the majority of species in the query result.

occurrences for the set $\mathcal{S}$ of assemblies. We can approximate f by choosing a mapping $E : \mathcal{S}_k \mapsto \mathbb{R}^n$, from now on called the *encoder*, corresponding to the trained `CNNFCGR` (see Figure 3).

The index $\mathcal{I}$ has three components: (1) a FAISS index $\mathcal{F}$ [20] storing the embeddings of each assembly in $\mathcal{S}$, (2) the encoder E, and (3) the labeling function λ.

With the encoder E, we create the embedding representation for each assembly $s \in \mathcal{S}$. These embeddings are indexed using FAISS [20], an easy-to-use index, extremely compressed (less than 1GB for $\sim$1.6 million embeddings of dimension 128), and fast to query. To facilitate the association of embeddings in $\mathcal{F}$ to labels (species) by the λ function, we store a list with labels for each embedding in $\mathcal{F}$. To query other assemblies against the set $\mathcal{S}$, we need to map them to the embedding space, so we need to preserve the mapping E as part of the index $\mathcal{I}$.

2.6 Querying the Index

The process of going from an (possibly draft) assembly to the embedding space where the indexed assemblies lie involves (1) counting k-mers of the assembly, (2) creating the FCGR matrix, and (3) using the encoder E to get an embedding in $\mathbb{R}^n$, as illustrated in Figure 5.

A query to the index $\mathcal{I}$ is defined as a (draft) assembly s_q (in fasta format, and which might not belong to $\mathcal{S}$), and the query results correspond to a list with the K-closests (a user defined parameter) embeddings in the FAISS index $\mathcal{F}$ and their labels given by the labeling function λ. We assign the label L to s_q, where L corresponds to the most common label in the query result.

3 Experiments

We have two main goals in these experiments. The first is to measure how efficiently we index the largest bacterial data set available in the literature, the AllTheBacteria data set (see Table 2). The second is to evaluate how accurately we can classify assemblies with our method compared to the state-of-the-art tool `GSearch`, and the computational resources required for queries.

We provide experiments using the `CNNFCGR` architecture with 6-mers, 7-mers and 8-mers, and include experiments using the `ResNet50` [16] architecture, a well-known model for image classification in deep learning, as a baseline for our `CNNFCGR` architecture.

Additionally, we provide an analysis of ANI values between assemblies and their query results in retrieved by `PANSPACE` and `GSearch` indexes.

3.1 Indexing the bacterial data set

We recall that the index produced with PANSPACE is composed of (1) the FAISS index, which stores all embeddings of the indexed assemblies, (2) the trained model, namely the *Encoder*, which serves to map assemblies to the embedding space, and (3) the list of labels for each assembly in the index.

We report in Table 3 the size of the index and encoder for PANSPACE (list of labels requires 60MB), the maximum RAM usage and elapsed time to create the indexes. PANSPACE index is 45× smaller than GSearch index, and requires 22× less RAM for its construction (for embedding size of 256, our best results for queries), and is also faster to construct.

To create the indexes for the AllTheBacteria data set, in the case of PANSPACE, we require to store the FCGR representations of assemblies. For 6-mers we require 31GB of storage, for 7-mers 131GB, and for 8-mers we require 480GB. In the case of GSearch, its index construction requires the collection of assemblies (as fa.gz), which require 4.1TB of storage.

We point out that the ResNet50 encoder is 84× larger than the CNNFCGR encoder, which affect the training time, and as we will see in the next section, it does not lead to better results.

	Index (GB)	Encoder (MB)	Max RAM (GB)	Elapsed time (hrs)
CNNFCGR-128	0.8	0.7	4.4	8:04
CNNFCGR-256	1.6	1.1	6.9	14:29
CNNFCGR-512	3.2	2.1	11.0	16:16
ResNet50-256	1.6	93	7	114:40
GSearch	75	-	156.8	24:52

Table 3: **Index size and construction**. The index size for the 90% of the high-quality assemblies (the set of species with at least 13 assemblies in the data set) of the AllTheBacteria data set used to train the models, comprising 1656597 assemblies. The Encoder column correspond to the disk space used by each trained model. The difference of encoder size are negligible for the different k-mer sizes, so we only report it once for each architecture indicating the embedding size next to them (128, 256 and 512). The index size does not depend on the k-mer size, neither the architecture, just on the embedding size, that is why CNNFCGR-256 and ResNet50-256 share the same index size. Max RAM and Elapsed time reported correspond to the models trained with 7-mers. GSearch index was built on top of the same dataset with its default parameters.

3.2 Quality of queries to the index

This experiment aims to assess the quality of queries against the index that we have built on top of the train and validation sets from the AllTheBacteria data set. The data set was divided into train, validation, and test sets, in a proportion of 80:10:10. The test set is used to query the index built on top of the 90% of the data set used in the training phase. We consider only high-quality assemblies, and the subset of species with at least 13 assemblies — so that at least 1 assembly can be part of the test set using the triplet loss, and there are at least 11 assemblies for each species in the index.

Given a query assembly, we retrieve the K-closest (based on the Euclidean distance between their embeddings) assemblies and their labels, using $K \in \{1, 5, 11\}$. The label assigned to the query is the most common label among the labels in the query result.

We report results for our encoders based on CNNFCGR with embedding size $128, 256$ and 512 for 6-mers, 7-mers and 8-mers, for ResNet50 with embedding size 256 using 7-mers (parameters with the best results for CNNFCGR), and for GSearch with default parameters.

3.2.1 Computational resources for query

To assess the computational resources required to query the indexes built with PANSPACE and GSearch, we randomly selected 1000 draft assemblies from the AllTheBacteria data set. These assemblies are from the test set; therefore, they are not part of the index.

For the construction of the FCGR representations of assemblies we use KMC3 [23] to count k-mers. In this experiment we only report results using 7-mers. GSearch was run with default parameters, using 24 threads and CPU (as reported in [57]), and we use the same configuration to run PANSPACE queries.

In Table 4 we report the maximum RAM usage and time required to query the 1000 assemblies for each tool, for PANSPACE, we report results using the CNNFCGR architecture with embedding size 128, 256 and 512, and also add the ResNet50 architecture with embedding size 256.

In PANSPACE, most of the time is spent in the FCGR generation. The creation of embeddings for the 1000 assemblies takes roughly 1 second in total with the CNNFCGR architecture (all embedding sizes) and 22 seconds with the ResNet50 architecture.

PANSPACE with the CNNFCGR architecture requires 15× less RAM to perform the queries (5.4GB vs 83.9GB), and around half the time (2:39 min vs 4:57 min) than GSearch. These experiments were run using 24 threads and only CPU.

	Max RAM (GB)	Elapsed time (mm:ss)
CNNFCGR-128	2.9	2:03
CNNFCGR-256	3.7	2:13
CNNFCGR-512	5.4	2:39
ResNet50-256	3.8	2:50
GSearch	83.9	4:57

Table 4: **Computational resources to query**. We randomly selected 1000 draft assemblies from the AllTheBacteria data set (from the 10% of the data set defined as the test set) and query the indexes built with PANSPACE and GSearch (built with the 90% of the data set). For PANSPACE the reported results were computed using 7-mers. In all cases we used 24 threads and only CPU.

3.2.2 Evaluation Metrics

We evaluate the performance of our model in the classification of assemblies at two taxonomy levels: species and genus.

A positive case (P) is when the true species (genus) label of the assembly is the same as the species (genus) label given by the query result, and a negative case (N) is when the true label of the assembly is different from the label of the query result. We consider three key metrics: precision, recall, and the F1-score.

$$\text{Precision} = \frac{TP}{TP + FP}; \text{Recall} = \frac{TP}{TP + FN}; \text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

where True Positives (TP) represent instances where the model correctly identifies positive cases, False Positives (FP) occur when the model incorrectly classifies negative instances as positive, and False Negatives (FN) arise when the model fails to identify positive instances.

Since we are dealing with multi-class classification, we report both the *macro average* and the *weighted average* of precision, recall, and F1 score, where,

- **Macro average**: unweighted mean of all per-class scores.
- **Weighted average**. Mean of all per-class scores, weighted by sample sizes for each class.

The macro average and weighted average for precision, recall, and F1-score are reported at the species and genus level in Table 5 and Table 6 respectively.

The optimal results for both GSearch and PANSPACE are achieved when utilizing a single neighbor, whereas increasing the number of neighbors leads to a decline in performance metrics. At the species level, GSearch demonstrates the highest effectiveness, with PANSPACE closely following in second place, with the best results obtained using 7-mers and embedding size 256. In contrast, at the genus level, PANSPACE outperforms GSearch, yielding superior results; this is mostly explained due to the fact that in some cases GSearch does not output a query result (in which case we define the label as 'not-trusted'). PANSPACE in the other hand, always returns a query result.

There were 140 and 208 assemblies misclassified at species level by GSearch and PANSPACE (using CNNFCGR-256[7]), respectively.

For GSearch, out of the 140 misclassified assemblies, 60 correspond to cases where there was no query result, 23 were classified into a different genus, and the remaining 57 were classified into the

correct genus but wrong species, where most (34) correspond to species of the genus Neisseria that were predicted as Neisseria meningitidis.

For PANSPACE, out of the 208 misclassified assemblies, 16 correspond to assemblies classified into the wrong genus (spanning 15 species, with predictions in 14 different species), and the remaining 192 assemblies were classified into the correct genus.

		Macro average					
		Species			Genus		
Neighbors	Model	Precision	Recall	F1-score	Precision	Recall	F1-score
1	CNNFCGR-128^{6}	0.635	0.627	0.619	0.985	0.964	0.970
	CNNFCGR-256^{6}	0.832	0.811	0.811	0.998	0.996	0.996
	CNNFCGR-512^{6}	0.935	0.914	0.916	**0.999**	**0.999**	**0.999**
	CNNFCGR-128^{7}	0.945	0.927	0.930	0.998	0.998	0.998
	CNNFCGR-256^{7}	0.982	0.967	0.971	0.997	0.998	0.998
	CNNFCGR-512^{7}	0.975	0.967	0.967	0.996	**0.999**	0.997
	CNNFCGR-128^{8}	0.939	0.910	0.916	0.996	0.995	0.995
	CNNFCGR-256^{8}	0.978	0.965	0.967	0.998	0.995	0.996
	CNNFCGR-512^{8}	0.965	0.947	0.951	0.998	**0.999**	0.998
	ResNet50-256^{7}	0.528	0.472	0.475	0.880	0.771	0.795
	GSearch	**0.992**	**0.981**	**0.984**	0.992	0.984	0.987
5	CNNFCGR-128^{6}	0.639	0.635	0.625	0.980	0.965	0.967
	CNNFCGR-256^{6}	0.837	0.813	0.814	**0.994**	0.990	0.991
	CNNFCGR-512^{6}	0.928	0.897	0.903	**0.994**	0.992	**0.993**
	CNNFCGR-128^{7}	0.938	0.905	0.914	0.993	0.991	0.992
	CNNFCGR-256^{7}	0.966	0.946	0.951	0.993	**0.994**	**0.993**
	CNNFCGR-512^{7}	0.950	0.936	0.939	0.991	0.993	0.992
	CNNFCGR-128^{8}	0.923	0.885	0.885	0.991	0.986	0.988
	CNNFCGR-256^{8}	0.953	0.933	0.937	0.992	0.987	0.989
	CNNFCGR-512^{8}	0.957	0.922	0.932	0.993	0.990	0.991
	ResNet50-256^{7}	0.547	0.495	0.496	0.887	0.793	0.816
	GSearch	**0.989**	**0.978**	**0.981**	0.986	0.978	0.981
11	CNNFCGR-128^{6}	0.643	0.637	0.626	0.980	0.968	0.970
	CNNFCGR-256^{6}	0.828	0.803	0.802	**0.994**	0.990	0.991
	CNNFCGR-512^{6}	0.901	0.872	0.875	0.992	**0.992**	0.991
	CNNFCGR-128^{7}	0.898	0.866	0.873	0.993	0.986	0.989
	CNNFCGR-256^{7}	0.943	0.926	0.928	0.992	**0.992**	**0.992**
	CNNFCGR-512^{7}	0.934	0.915	0.918	0.990	0.990	0.989
	CNNFCGR-128^{8}	0.894	0.858	0.865	0.987	0.980	0.982
	CNNFCGR-256^{8}	0.922	0.905	0.907	0.987	0.984	0.984
	CNNFCGR-512^{8}	0.922	0.889	0.897	0.991	0.985	0.987
	ResNet50-256^{7}	0.550	0.496	0.499	0.880	0.784	0.808
	GSearch	**0.982**	**0.974**	**0.975**	0.983	0.975	0.978

Table 5: **Macro average Precision, Recall and F1-score** at species and genus levels. The embedding size is next to each model. The superscript represent the k-mer size (except for GSearch used with default parameters). The first column (Neighbors)correspond to the number of neighbors used to compute the metrics. The first and second best results are highlighted in **bold** and with an underline, respectively.

3.2.3 Embedding proximity reflects high ANI values between assemblies

In this experiment, we show that assemblies that are close w.r.t. the Euclidean distance between their embeddings obtained with PANSPACE have a large ANI value.

We randomly selected 1000 assemblies (from the test set) from each of the 30 most represented species in the AllTheBacteria data set. First, we query PANSPACE index (created with CNNFCGR, 7-mers

		Weighted average					
		Species			Genus		
Neighbors	Model	Precision	Recall	F1-score	Precision	Recall	F1-score
1	CNNFCGR-128^{6}	0.897	0.926	0.905	0.986	0.985	0.985
	CNNFCGR-256^{6}	0.952	0.967	0.954	1	1	1
	CNNFCGR-512^{6}	0.985	0.981	0.976	1	1	1
	CNNFCGR-128^{7}	0.996	0.996	0.996	1	1	1
	CNNFCGR-256^{7}	**0.999**	**0.999**	**0.999**	1	1	1
	CNNFCGR-512^{7}	**0.999**	**0.999**	**0.999**	1	1	1
	CNNFCGR-128^{8}	0.984	0.980	0.973	1	1	1
	CNNFCGR-256^{8}	**0.999**	**0.999**	**0.999**	1	1	1
	CNNFCGR-512^{8}	0.997	0.997	0.997	1	1	1
	ResNet50-256^{7}	0.844	0.871	0.837	0.939	0.939	0.931
	GSearch	**0.999**	**0.999**	**0.999**	1	1	1
5	CNNFCGR-128^{6}	0.912	0.941	0.921	0.999	0.998	0.998
	CNNFCGR-256^{6}	0.953	0.968	0.955	1	1	1
	CNNFCGR-512^{6}	0.985	0.982	0.977	1	0.999	1
	CNNFCGR-128^{7}	0.996	0.996	0.996	1	0.999	1
	CNNFCGR-256^{7}	**0.999**	**0.999**	0.998	1	1	1
	CNNFCGR-512^{7}	0.998	0.998	0.998	1	1	1
	CNNFCGR-128^{8}	0.984	0.980	0.973	1	0.999	0.999
	CNNFCGR-256^{8}	0.998	0.998	0.998	1	1	1
	CNNFCGR-512^{8}	0.997	0.996	0.996	1	0.999	1
	ResNet50-256^{7}	0.878	0.907	0.875	0.973	0.975	0.969
	GSearch	**0.999**	**0.999**	**0.999**	1	0.999	1
11	CNNFCGR-128^{6}	0.913	0.942	0.922	0.999	0.999	0.999
	CNNFCGR-256^{6}	0.953	0.968	0.955	1	1	1
	CNNFCGR-512^{6}	0.984	0.981	0.976	1	1	1
	CNNFCGR-128^{7}	0.995	0.995	0.995	1	0.999	1
	CNNFCGR-256^{7}	0.998	0.998	0.998	1	1	1
	CNNFCGR-512^{7}	0.998	0.998	0.998	1	1	1
	CNNFCGR-128^{8}	0.983	0.979	0.972	1	0.999	0.999
	CNNFCGR-256^{8}	0.997	0.998	0.997	1	1	1
	CNNFCGR-512^{8}	0.995	0.996	0.995	1	0.999	1
	ResNet50-256^{6}	0.881	0.911	0.878	0.975	0.978	0.972
	GSearch	**0.999**	**0.999**	**0.999**	1	0.999	1

Table 6: **Weighted average Precision, Recall and F1-score** at species and genus levels. The embedding size is next to each model. The superscript represent the k-mer size (except for `GSearch` used with default parameters). The first column (Neighbors) correspond to the number of neighbors used to compute the metrics. The weight correspond to the fraction of assemblies of each class over the size of the dataset. The first and second best results are highlighted in **bold** and with an underline, respectively.

and embedding size 256) with these 1000 assemblies, and compute the ANI value between the query assembly and each of their 11 closests neighbors (the query result) using `fastANI` v1.34 with default parameters. We perform the same query with `GSearch`. Results of ANI values for both `PANSPACE` and `GSearch` are shown in Figure 6, where we observe that `GSearch` concentrates most of their close to ANI values close to 1, which is expected since this index relies explicitly on ANI values between assemblies for its construction. On the other hand, `PANSPACE` also show large ANI values for the majority of the queries, but more importantly the query results (in both cases) are above the ANI threshold 95, which determines that two assemblies belong to the same species.

Secondly, we explore how the assemblies cluster themselves by using `PANSPACE` distances. We randomly selected 100 assemblies for each of the 30 most represented species (from the test set), and

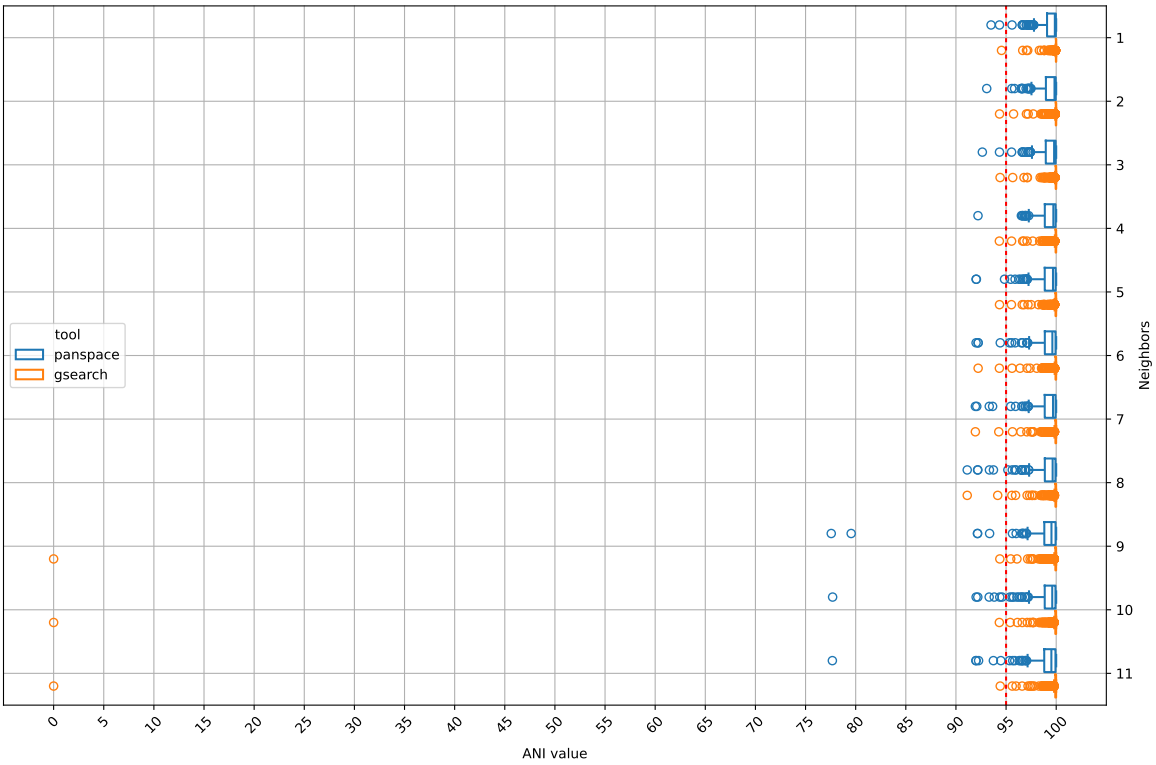


Figure 6: **Boxplot of ANI values** between query sequences and their 11 closests neighbors in indexes built with `PANSPACE` (blue) and `GSearch` (orange). We randomly selected 1000 draft assemblies from the AllTheBacteria data set (from the 10% of the data set defined as the test set) and query the indexes built with the 90% of the data set. We compute the ANI similarity (x-axis) between each query assembly and its 11 closests assemblies (y-axis) in the indexes. A vertical red dashed line indicates the 95 ANI value threshold for determining that two assemblies belong to the same species. ANI value was computed using `fastANI` v1.34 with default parameters. ANI value equal 0 correspond to empty query results, *i.e.* where no assembly was retrieved by `GSearch`.

created a UMAP reduction of them [35], shown in Figure 7[3]. We observe distinct, well-separated clusters corresponding to different species, with embeddings from the Salmonella genus appearing in closer proximity to one another.

3.3 Experimental Setup

All experiments for `PANSPACE` were conducted on a desktop computer with the following characteristics: AMD Ryzen 7 16 cores, with 1TB of disk space, 16 GB of RAM, NVIDIA GeForce RTX 4080, using CUDA Version: 12.1. The server runs Linux as the operating system.

Experiments regarding `GSearch` were conducted on a server equipped with 64 CPU cores (AMD EPYC 7301, 16-core, 2.2 GHz), 256 GB of RAM, and 9.1 TB of storage. The server runs Linux as the operating system.

The Python environment setup for training, creating, and querying the index is defined by the following libraries and versions: tensorflow v2.15.0, faiss-gpu v1.7.2, scikit-learn v1.3.2, complexcgr v0.8.0, and Python v3.10.13.

3.4 Data Availability

The AllTheBacteria dataset was downloaded from https://ftp.ebi.ac.uk/pub/databases/AllTheBacteria/Releases/0.2/.

[3]Image created with https://datamapplot.readthedocs.io

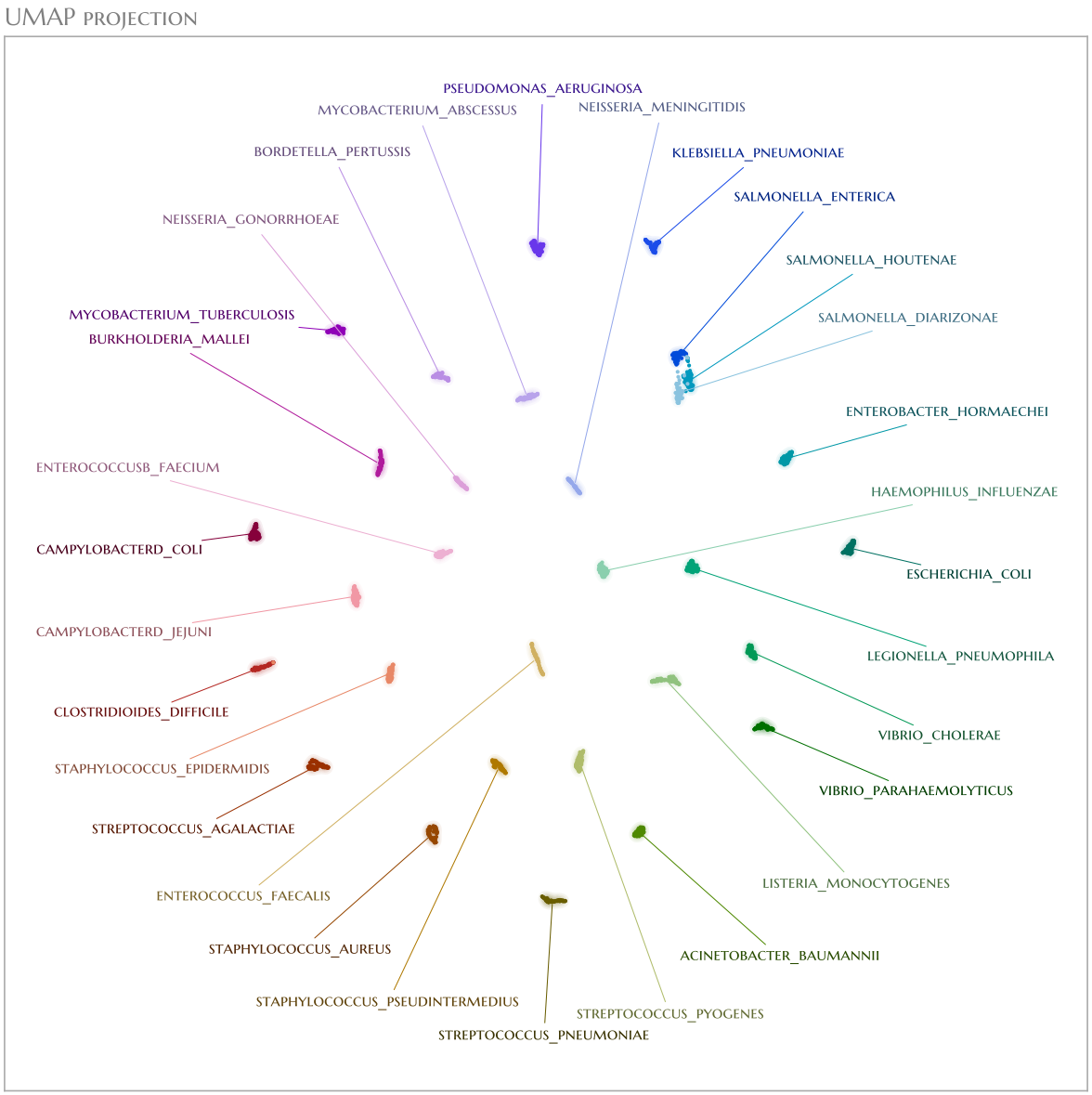


Figure 7: **UMAP** `PANSPACE`. We randomly selected 100 assemblies for each of the 30 species more represented in the AllTheBacteria data set (all assemblies were selected from the test set), then using the the `CNNFCGR` architecture for 7-mers and embedding size 256, we created the embedding representation of each assembly, and then used it to create a UMAP projection in 2-dimensions using default parameters.

Experiment results reported in this manuscript can be found in Zenodo at the following link https://zenodo.org/records/14936601.
Code for experiments are available in github at https://github.com/pg-space/panspace-paper.
`PANSPACE` is avaialable under the GNU 3.0 license at https://github.com/pg-space/panspace.

4 Discussion

In this work, we have presented `PANSPACE`, a deep-learning tool that is able to (1) index a massive database of bacterial genome assemblies and (2) to query such index to find the origin species of a generic genomic sequence. Most notably, we have indexed the AllTheBacteria database and the resulting index uses less than 4 GB of disk space. The construction of the index is also very efficient,

since the peak RAM used is 11 GB using an embedding representation of 512 and 7 GB using 256 (with FCGR from 7-mers). For a comparison, `GSearch` (which is the state of the art) needs 75 GB to store the index and more than 150 GB of RAM to build it. While those requirements might not look excessive, the AllTheBacteria database is including more and more assemblies. Our index will be able to manage all foreseeable expansions of AllTheBacteria.

We have compared our tool, `PANSPACE`, with `GSearch` also on the quality of the query results, where we ask for the origin species of a genomic sequence. We have also added to the comparison a baseline deep-learning approach based on `ResNet50`, showing that not only `CNNFCGR` is more efficient for queries and construction, but also better at processing the FCGR leading to considerably better results. The accuracy of the query results is assessed by considering the predicted origin species and its genus. `PANSPACE` has obtained almost the same accuracy as `GSearch` when considering the predicted species. More precisely, the accuracy is the same (F1-score 0.999) on the weighted average and slightly worse (F1-score 0.971 vs 0.984) on the macro average.When considering the predicted genus, the results are reversed. In fact, in this case the accuracy is still the same (F1-score 0.999) on the weighted average, but `PANSPACE` has a slightly better F1-score (0.998 vs 0.987) on the macro average. On all those cases, the `ResNet50` approach has obtained worse results (e.g. F1-score 0.475 for macro average on species).

Finally, we test another property of our index. The index is based on the notion of embedding, that we map each genome assembly to a metric space which allows to quickly compute a distance between aseemblies. We have shown how such a distance confirms the established classification into genus.

5 Competing interests

No competing interest is declared.

6 Acknowledgments

This research work has received funding from the European Union's Horizon 2020 Research and Innovation Staff Exchange programme under the Marie Skłodowska-Curie grant agreement No. 872539 (Pangaia) and from the research and innovation programme under the Marie Skłodowska-Curie grant agreement No 956229 (Alpaca). This research work is also supported by the grant MIUR 2022YRB97K, Pangenome Informatics: from Theory to Applications (PINC).

References

[1] Computational pan-genomics: status, promises and challenges. *Briefings in bioinformatics*, 19(1):118–135, 2018.

[2] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[3] Dennis A Benson, Mark Cavanaugh, Karen Clark, Ilene Karsch-Mizrachi, James Ostell, Kim D Pruitt, and Eric W Sayers. Genbank. *Nucleic acids research*, 46(Database issue):D41, 2018.

[4] Timo Bingmann, Phelim Bradley, Florian Gauger, and Zamin Iqbal. Cobs: a compact bit-sliced signature index. In *String Processing and Information Retrieval: 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7–9, 2019, Proceedings 26*, pages 285–303. Springer, 2019.

[5] Karel Břinda, Leandro Lima, Simone Pignotti, Natalia Quinones-Olvera, Kamil Salikhov, Rayan Chikhi, Gregory Kucherov, Zamin Iqbal, and Michael Baym. Efficient and robust search of microbial genomes via phylogenetic compression. *bioRxiv*, pages 2023–04, 2023.

[6] Andrei Z Broder. Proceedings. compression and complexity of sequences 1997 (cat. no. 97tb100171). *IEEE*, pages 21–29, 1997.

[7] Jose Luis Bellod Cineros and Ole Lund. Kmerfinderjs: a client-server method for fast species typing of bacteria over slow internet connections. *BioRxiv*, page 145284, 2017.

[8] Karen Clark, Ilene Karsch-Mizrachi, David J Lipman, James Ostell, and Eric W Sayers. Genbank. *Nucleic acids research*, 44(D1):D67–D72, 2016.

[9] Rachel M Colquhoun, Michael B Hall, Leandro Lima, Leah W Roberts, Kerri M Malone, Martin Hunt, Brice Letcher, Jane Hawkey, Sophie George, Louise Pankhurst, et al. Pandora: nucleotide-resolution bacterial pan-genomics with reference graphs. *Genome biology*, 22:1–30, 2021.

[10] Patrick J Deschavanne, Alain Giron, Joseph Vilain, Guillaume Fagot, and Bernard Fertil. Genomic signature: characterization and classification of species assessed by chaos game representation of sequences. *Molecular biology and evolution*, 16(10):1391–1399, 1999.

[11] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete mathematics & theoretical computer science*, (Proceedings), 2007.

[12] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[13] Johan Goris, Konstantinos T Konstantinidis, Joel A Klappenbach, Tom Coenye, Peter Vandamme, and James M Tiedje. Dna–dna hybridization values and their relationship to whole-genome sequence similarities. *International journal of systematic and evolutionary microbiology*, 57(1):81–91, 2007.

[14] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[15] Daniel H Haft, Azat Badretdin, George Coulouris, Michael DiCuccio, A Scott Durkin, Eric Jovenitti, Wenjun Li, Megdelawit Mersha, Kathleen R O'Neill, Joel Virothaisakun, et al. Refseq and the prokaryotic genome annotation pipeline in the age of metagenomes. *Nucleic Acids Research*, 52(D1):D762–D769, 2024.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] Martin Hunt, Leandro Lima, Wei Shen, John Lees, and Zamin Iqbal. Allthebacteria-all bacterial genomes assembled, available and searchable. *bioRxiv*, pages 2024–03, 2024.

[18] Chirag Jain, Luis M Rodriguez-R, Adam M Phillippy, Konstantinos T Konstantinidis, and Srinivas Aluru. High throughput ani analysis of 90k prokaryotic genomes reveals clear species boundaries. *Nature communications*, 9(1):5114, 2018.

[19] H Joel Jeffrey. Chaos game representation of gene structure. *Nucleic acids research*, 18(8):2163–2170, 1990.

[20] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[21] Michał Karlicki, Stanisław Antonowicz, and Anna Karnkowska. Tiara: deep learning-based classification system for eukaryotic sequences. *Bioinformatics*, 38(2):344–350, 2022.

[22] Daehwan Kim, Li Song, Florian P Breitwieser, and Steven L Salzberg. Centrifuge: rapid and sensitive classification of metagenomic sequences. *Genome research*, 26(12):1721–1729, 2016.

[23] Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. Kmc 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 2017.

[24] Svetlana Kutuzova, Mads Nielsen, Pau Piera, Jakob Nybo Nissen, and Simon Rasmussen. Taxometer: Improving taxonomic classification of metagenomics contigs. *Nature Communications*, 15(1):8357, 2024.

[25] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[26] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.

[27] Qian Liang, Chengzhi Liu, Rong Xu, Minghui Song, Zhihui Zhou, Hong Li, Weiyou Dai, Meicheng Yang, Yunsong Yu, and Huan Chen. fidbac: a platform for fast bacterial genome identification and typing. *Frontiers in Microbiology*, 12:723577, 2021.

[28] Qiaoxing Liang, Paul W Bible, Yu Liu, Bin Zou, and Lai Wei. Deepmicrobes: taxonomic classification for metagenomics with deep learning. *NAR Genomics and Bioinformatics*, 2(1):lqaa009, 2020.

[29] Rebecca L Lindsey, Lori M Gladney, Andrew D Huang, Taylor Griswold, Lee S Katz, Blake A Dinsmore, Monica S Im, Zuzana Kucerova, Peyton A Smith, Charlotte Lane, et al. Rapid identification of enteric bacteria from whole genome sequences using average nucleotide identity metrics. *Frontiers in Microbiology*, 14, 2023.

[30] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[31] Jennifer Lu, Florian P Breitwieser, Peter Thielen, and Steven L Salzberg. Bracken: estimating species abundance in metagenomics data. *PeerJ Computer Science*, 3:e104, 2017.

[32] Jared Lumpe, Lynette Gumbleton, Andrew Gorzalski, Kevin Libuit, Vici Varghese, Tyler Lloyd, Farid Tadros, Tyler Arsimendi, Eileen Wagner, Craig Stephens, et al. Gambit (genomic approximation method for bacterial identification and tracking): A methodology to rapidly leverage whole genome sequencing of bacterial isolates for clinical identification. *Plos one*, 18(2):e0277575, 2023.

[33] Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

[34] Santiago Marco-Sola, Juan Carlos Moure, Miquel Moreto, and Antonio Espinosa. Fast gap-affine pairwise alignment using the wavefront algorithm. *Bioinformatics*, 37(4):456–463, 2021.

[35] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[36] Jan P Meier-Kolthoff and Markus Göker. Tygs is an automated high-throughput platform for state-of-the-art genome-based taxonomy. *Nature communications*, 10(1):2182, 2019.

[37] Donovan H Parks, Michael Imelfort, Connor T Skennerton, Philip Hugenholtz, and Gene W Tyson. Checkm: assessing the quality of microbial genomes recovered from isolates, single cells, and metagenomes. *Genome research*, 25(7):1043–1055, 2015.

[38] Michael Richter and Ramon Rosselló-Móra. Shifting the genomic gold standard for the prokaryotic species definition. *Proceedings of the National Academy of Sciences*, 106(45):19126–19131, 2009.

[39] Luuk Romeijn, Andrius Bernatavicius, and Duong Vu. Mycoai: Fast and accurate taxonomic classification for fungal its sequences. *Molecular Ecology Resources*, page e14006, 2024.

[40] Eric W Sayers, Mark Cavanaugh, Karen Clark, Kim D Pruitt, Stephen T Sherry, Linda Yankie, and Ilene Karsch-Mizrachi. Genbank 2024 update. *Nucleic Acids Research*, 52(D1):D134–D137, 2024.

[41] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[42] Jim Shaw and Yun William Yu. Rapid species-level metagenome profiling and containment estimation with sylph. *Nature Biotechnology*, pages 1–12, 2024.

[43] Wei Shen and Zamin Iqbal. Lexicmap: efficient sequence alignment against millions of prokaryotic genomes. *bioRxiv*, pages 2024–08, 2024.

[44] Li Song and Ben Langmead. Centrifuger: lossless compression of microbial genomes for efficient and accurate metagenomic sequence classification. *Genome Biology*, 25(1):106, 2024.

[45] Ramya Srinivasan, Ulas Karaoz, Marina Volegova, Joanna MacKichan, Midori Kato-Maeda, Steve Miller, Rohan Nadarajan, Eoin L Brodie, and Susan V Lynch. Use of 16s rrna gene for identification of a broad range of clinically relevant bacterial pathogens. *PloS one*, 10(2):e0117617, 2015.

[46] Jens-Uwe Ulrich and Bernhard Y Renard. Fast and space-efficient taxonomic classification of long reads with hierarchical interleaved xor filters. *Genome Research*, pages gr–278623, 2024.

[47] Bhavish Verma and John Parkinson. Hitaxon: a hierarchical ensemble framework for taxonomic classification of short reads. *Bioinformatics Advances*, 4(1):vbae016, 2024.

[48] Susana Vinga, Alexandra M Carvalho, Alexandre P Francisco, Luís MS Russo, and Jonas S Almeida. Pattern matching through chaos game representation: bridging numerical and discrete data structures for biological sequence analysis. *Algorithms for Molecular Biology*, 7:1–12, 2012.

[49] LG Wayne, DJ Brenner, RR Colwell, PAD Grimont, O Kandler, MI Krichevsky, LH Moore, WEC Moore, RGEea Murray, ESMP Stackebrandt, et al. Report of the ad hoc committee on reconciliation of approaches to bacterial systematics. *International Journal of Systematic and Evolutionary Microbiology*, 37(4):463–464, 1987.

[50] Patrick T West, Alexander J Probst, Igor V Grigoriev, Brian C Thomas, and Jillian F Banfield. Genome-reconstruction for eukaryotes from complex natural microbial communities. *Genome research*, 28(4):569–580, 2018.

[51] Carl R Woese. Bacterial evolution. *Microbiological reviews*, 51(2):221–271, 1987.

[52] Carl R Woese and George E Fox. Phylogenetic structure of the prokaryotic domain: the primary kingdoms. *Proceedings of the National Academy of Sciences*, 74(11):5088–5090, 1977.

[53] Derrick E Wood, Jennifer Lu, and Ben Langmead. Improved metagenomic analysis with kraken 2. *Genome biology*, 20:1–13, 2019.

[54] Derrick E Wood and Steven L Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome biology*, 15:1–12, 2014.

[55] Pelin Yilmaz, Laura Wegener Parfrey, Pablo Yarza, Jan Gerken, Elmar Pruesse, Christian Quast, Timmy Schweer, Jörg Peplies, Wolfgang Ludwig, and Frank Oliver Glöckner. The silva and "all-species living tree project (ltp)" taxonomic frameworks. *Nucleic acids research*, 42(D1):D643–D648, 2014.

[56] Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. *Advances in neural information processing systems*, 32, 2019.

[57] Jianshu Zhao, Jean Pierre Both, Luis M Rodriguez-R, and Konstantinos T Konstantinidis. Gsearch: ultra-fast and scalable genome search by combining k-mer hashing with hierarchical navigable small world graphs. *Nucleic Acids Research*, 52(16):e74–e74, 2024.