IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 30 January 2024, accepted 17 February 2024, date of publication 20 February 2024, date of current version 27 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3367801

RESEARCH ARTICLE

ViRAL: Vision Transformer Based Accelerator for ReAL Time Lineage Assignment of Viral Pathogens

ZUHER JAHSHAN[1], ESTEBAN GARZÓN[2], (Senior Member, IEEE), AND LEONID YAVITS[1], (Member, IEEE)

[1]EnICS Laboratories, Faculty of Engineering, Bar-Ilan University, Ramat Gan 5290002, Israel
[2]Department of Computer Engineering, Modeling, Electronics and Systems, University of Calabria, 87036 Rende, Italy

Corresponding author: Esteban Garzón (esteban.garzon@unical.it)

This work was supported in part by the European Union's Horizon Europe Programme for Research and Innovation under Grant 101047160, in part by the Israeli Ministry of Science and Technology under Lise Meitner Grant for Israeli-Swedish Research Collaboration under Grant 1001569396, and in part by the Israeli Ministry of Science and Technology Grant for Groundbreaking Research under Grant 1001702600. The work of Esteban Garzón was supported by the Italian Ministry for Universities and Research (MUR) under the Call "Horizon Europe (2021–2027) Programme" under Grant H25F21001420001.

ABSTRACT Real-time genome detection, classification and lineage assignment are critical for efficient tracking of emerging mutations and variants during viral pandemics such as Covid-19. For genomic surveillance to work effectively, each new viral genome sequence must be quickly and accurately associated with an existing viral family (lineage). ViRAL is a hardware-accelerated platform for real-time viral genome lineage assignment based on minhashing and Vision Transformer. Minhashing is a locality sensitive hashing based technique for finding regions of similarity within sequenced genomes. Vision Transformer is a model for image classification that employs a Transformer-like architecture over patches of images. In ViRAL, such image patches are genome fragments extracted from the regions of high similarity. ViRAL is especially efficient in lineage assignment of extremely low quality (or highly ambiguous) genomic data, i.e. when a large fraction of DNA bases are missing in an assembled genome. We implement ViRAL on CPU, GPU and a custom-designed hardware accelerator denoted ACMI. ViRAL assigns newly sequenced SARS-CoV-2 genomes to existing lineages with the top-1 accuracy of 94.2%. The probability of the correct assignment to be found among the five most likely placements generated by ViRAL (top-5 accuracy) is 99.8%. Accelerated ViRAL outperforms the fastest state-of-the-art assignment tools by 69.4×. It also outperforms ViRAL GPU implementation by 19.5×. ViRAL strongly outperforms the state-of-the-art solutions in assigning highly-ambiguous genomes: while state-of-the-art tools fail to assign lineage to genomes with 50% ambiguity, ViRAL achieves 77.6% assignment accuracy. We make ViRAL available to the research community through GitHub.

INDEX TERMS Vision transformer, transformers, viral pathogens, SARS-CoV-2, genome, accelerator.

I. INTRODUCTION

One of the lessons of COVID-19 pandemic is that our computational infrastructure and bioinformatics tools are inefficient in tracking quickly mutating viral pathogens, spread across the entire world [1], [2], [3]. Without a timely and accurate understanding of the transmission dynamics of the pathogen, an efficient control of the pandemic is impossible. The foremost tasks of such pathogen tracking are pathogen detection in metagenomic samples (i.e., samples containing DNA of multiple organisms, such as wastewater samples), classification into one of the many existing variants (*lineage assignment*), and discovery of emerging mutations. Lineage assignment is different from genome classification,

The associate editor coordinating the review of this manuscript and approving it for publication was Gian Domenico Licciardo.

where an organism is classified into one of very few very distinct classes of species (such as human vs. virus [4]). Viral lineage assignment targets hundreds, potentially thousands of very closely related families, which makes the classification task much more complex.

Existing diagnostic tools, such as Polymerase Chain Reaction (PCR) tests, can neither detect new variants of a quickly mutating virus nor efficiently classify viral samples [5]. This is because PCR tests require custom (i.e., per-variant) ''primers'' that attach to and amplify specific regions of DNA/RNA in the target virus's genome. Each of such primers must be specifically designed for each new variant. Therefore, identifying a new viral sample with one out of hundreds of existing variants using PCR is infeasible. A practical solution requires DNA sequencing and digital genome analysis.

Unfortunately, an accurate lineage assignment of a quickly mutating viral pathogen such as SARS-CoV-2 may have to deal with hundreds or thousands, and potentially millions of previously sequenced viral genomes. Existing bioinformatics tools are either prohibitively slow [6], [7], [8] or use heuristics to trade accuracy for speed [9]. The underlying computing infrastructure is inefficient in dealing with problems of such complexity and scale. A new paradigm, combining new bioinformatics methods that can deal with the speed of pathogen evolution and abundance of variants, and new computing platforms to efficiently implement such new methods, is required to efficiently face the next pandemic.

Deep learning networks emerged as a very powerful infrastructure for genome and metagenomic classification tasks. State of the art DNN based solutions use Convolutional Neural Networks (CNN) [10], [11], [12], Recurrent Neural Networks (RNN) [13], a combination of CNNs and RNNs [4], [14], [15], a combination of CNN and LSTM [16], Multilayer Perceptron (MLP) [12], and Transformer networks [17] to identify and classify genomic samples of a wide spectrum, from humans to viruses.

Beyond metagenomic classification, DeepTE [18] classifies transposable elements which constitute a large portion of many known eukaryotic genomes using CNN. Basset [19] tool applies CNNs to learn the functional activity of DNA sequences from genomics data. BERTax [20] is a tool that uses a transformer network to perform taxonomy classification of DNA sequences. fDNN [21] integrates DNN with a supervised forest feature detector to extract feature representation for gene expression data classification.

However, unlike metagenomic classification, where species exhibit significant differences, lineage assignment of quickly mutating viral pathogens such as SARS-CoV-2 deals with very similar genomes, where the difference is often limited to several bases [22].

In this work, we make a case for an alternative deep learning approach to lineage assignment. We introduce ViRAL (Figure 1), a novel algorithm and accelerated system for quick and accurate identification, classification, and lineage assignment of viral genomes. ViRAL is based on a combination of Vision Transformer (ViT) [23] and MinHash [24], [25]-like locality sensitive hashing (LSH) technique. ViT is a deep neural network (DNN) for image classification, which we modify and amend to enable extremely fast and accurate lineage assignment of viral genomes. MinHash identifies the regions of high similarity in genomes. During the network training, these similarity regions are learned and embedded by the ViT. During inference, extracting regions of similarity from a genome assists the network in identifying such a genome with the correct lineage with high probability.

Specifically, ViRAL receives a newly sequenced SARS-CoV-2 genome and outputs a list of the most probable lineages it might belong to, ordered by their likelihood. One of the two most probable lineages identified by ViRAL is the correct one with a probability of 97.9%. Such probability grows to 99.8% for the five most probable lineages (i.e., the probability of the correct result being among these five most probable lineages is 99.8%).

Digital analysis of the sequenced SARS-CoV-2 material relies on high quality and high coverage[1] genome sequencing [26]. However, low-quality sequencing is ubiquitous in low-cost field settings, especially in low- and middle-income countries. As a result, a considerable amount of low-quality sequence data is found in SARS-CoV-2 databases [27]. Consequently, such low-quality data fragments are typically ignored or removed during analysis [28]. Such an inability to use low-quality sequenced data adversely affects the evolutionary tracking and transmission dynamics of the virus.

ViRAL maintains 94.2% and 77.6% lineage assignment accuracy when analyzing high quality and low quality (50% ambiguity[2]) genomic data, respectively. In that, it strongly outperforms state-of-the-art tools, which typically require high-quality ambiguity-free data to perform.

In addition to accurately assigning a lineage to a newly sequenced SARS-CoV-2 that belongs to one of the existing lineages, ViRAL can potentially detect a novel SARS-CoV-2 variant or lineage, and provide some initial evolutionary tracking hints, which may simplify the phylogenetic analysis of such a new variant or lineage.

A worldwide viral pandemic is not the only looming health disaster. *Antimicrobial resistance* is an escalating global crisis, where the spread of drug resistance is outpacing the development of new antimicrobials. It already causes at least 700,000 deaths a year and is expected to cause over 10 million deaths annually by 2050 [29]. To act against this global threat, fast and accurate tools for antimicrobial resistance diagnostics are required. DNA sequencing and computational analysis of bacteria are sufficiently scalable and enable the discovery of new genes and variants, but technical and computational difficulties complicate the use of

[1] Sequencing coverage refers to the average number of times each base in the genome is read during the sequencing process.

[2] Ambiguity is the percentage of genome sites not covered by any DNA read; we further define base ambiguity in Section VI-A1.

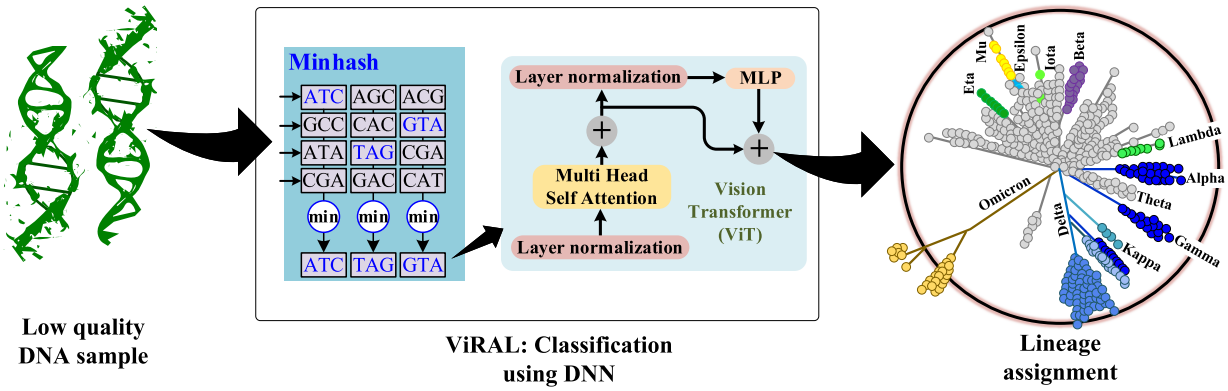


FIGURE 1. ViRAL: From low-quality DNA sample to high-quality lineage assignment, in real-time. .

genome sequencing and analysis-based approaches in clinical settings.

ViRAL aspires to enable a rapid and accurate interpretation of transmission and evolutionary dynamics of pathogens, presently limited by the inefficiencies and bottlenecks of von Neumann model [30]. The most efficient solution to the performance limitations of general-purpose computers is hardware acceleration [31]. ViRAL is designed to accelerate pathogen (virus or bacteria) classification and lineage assignment, such that a newly sequenced viral or bacterial genome can be accurately and rapidly identified and contextualized within its entire evolutionary history.

We develop three ViRAL solutions: The first is a baseline software-only CPU implementation. The second is a GPU-accelerated solution. It achieves 3.53× speedup over UShER [9], arguably the best performance state-of-the-art assignment tool, when executed on NVIDIA's GeForce RTX 2080 Ti GPU-powered computer. The third solution is a hardware-accelerated ViRAL implementation that uses a dedicated ASIC design and achieves 69.4× speedup over UShER.

We make ViRAL available to the research community through GitHub [32], as well as at Pango cov-lineages portal.[3]

This work makes the following contributions:

1) We propose ViRAL, a new DNN-based pathogen classification and lineage assignment tool. This is the first time a Vision Transformer has been applied to genome identification, specifically to classification of genomes of the same species, into hundreds of closely related lineages. It is sufficiently sensitive and accurate to differentiate between genomes with very little variations among them.
2) We combine ViT with MinHash, whose role is the identification of the regions of high similarity within individual genomes. Those regions are used for training the network as well as during inference.
3) ViRAL achieves high accuracy (above 77%) when classifying low-quality and ambiguous genomic data (ambiguity of 50%).
4) We develop a hardware accelerator that enables very significant speedup over state-of-the-art assignment tools, and benefits a wide range of bioinformatics applications, ranging from genome classification to genome assembly.

[3]https://cov-lineages.org/resources.html

II. BACKGROUND AND RELATED WORK

A. DNA SEQUENCING AND GENOME ANALYSIS

DNA is composed of four nucleotides: Adenine (A), Guanine (G), Cytosine (C), and Thymine (T), which are frequently referred to as DNA basepairs, bases, or bps. Accordingly, a DNA data element is a DNA base that can have one of four values (A, G, C, and T). DNA sequencing is the process of determining the bases of a DNA chain in a given biological sample. Contemporary high-throughput DNA sequencers can sequence multiple biological samples in parallel [33]. A DNA sequencing process, along with genome analysis, is carried out in several steps [34]: (1) sample preparation; (2) DNA sequencing that generates multiple DNA reads (i.e., sub-sequences of the DNA sample); and (3) DNA classification, DNA read alignment, genome assembly, variant analysis, etc.

While variant refers to a single change in a genome (i.e., a single base change mutation), **lineage** is a collection of variants that help define a specific line of an organism.

B. VISION TRANSFORMER

Transformer is a DNN model proposed by Vaswani et al. [35]. It became state-of-the-art in many deep learning applications, such as natural language processing. Vision Transformer (ViT) is an image classification DNN based on the transformer model [23]. ViT architecture is presented in Figure 2. It comprises an encoder and a multilayer perceptron (MLP) layer with a softmax activation function to perform predictions (in this paper we will refer to this MLP layer as MLP head). An encoder receives a sequence of

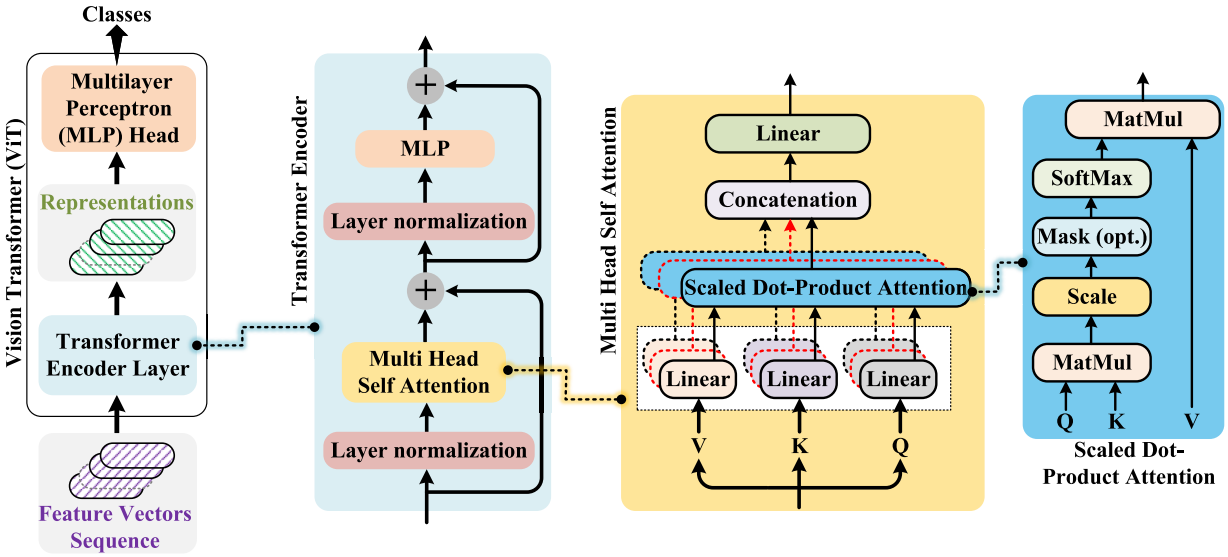


FIGURE 2. The ViT architecture. The encoder is a stack of alternating L multi-head self-attention (MHSA) layers and L MLP blocks. Layer normalization is applied before every block and a residual connection is employed around each of the sub-layers (i.e., MHSA and MLP).

representations, each of them is of dimensionality d_{model} (i.e., sequence of vectors consisting d_{model} elements each), and outputs a sequence of learned representations of the same dimensionality. For the sake of clarity, we refer to the sequence of input representations as "feature vectors sequence" throughout this section. The transformer encoder layer is composed of two main sub-layers, namely the Multi Head Self Attention (MHSA) and the piece-wise MLP. In the following subsections, we will present the structure and the purpose of those sub-layers.

1) MULTI HEAD SELF ATTENTION

This sub-layer, depicted in Figure 2 (right) receives as an input a sequence of feature vectors $f_1, f_2, \ldots, f_n$, each of dimensionality d_{model}. The sub-layer outputs a sequence of learned representations $r_1, r_2, \ldots, r_n$, a representation per each feature vector. The output learned representations are of the same dimensionality as the input.

The following presents the MHSA data flow. First, each feature vector f_j is passed through $3h$ linear layers $\{q_l, k_l, v_l\}_{l=1}^{h}$, where h denotes the number of heads, and 3 denotes the three types of different representations (i.e., query, key, and value). The purpose of those linear layers is to learn h query representations for f_j, h key representations for f_j, and h value representations of f_j. Each linear layer defines a weight matrix as follows: $W_l^q \in \mathbb{R}^{d_{model} \times d_q}$; $W_l^k \in \mathbb{R}^{d_{model} \times d_k}$; $W_l^v \in \mathbb{R}^{d_{model} \times d_v}$. They are used to calculate their respective representations: $q_l(f_j) = f_j^T \cdot W_l^q$; $k_l(f_j) = f_j^T \cdot W_l^k$; $v_l(f_j) = f_j^T \cdot W_l^v$.

The query and key representations are used to calculate the degree of attention that the learned output representation, r_j, will give to other feature vectors. The value representations are used to represent learned properties of the feature vector that we intend to pass through the network. Formally, this first step of the MHSA layer is defined as follows:

$$A_j^l = \sum_{i=1}^{n} similarity(q_l(f_j), k_l(f_i)) \cdot v_l(f_i)$$

where

$$similarity(q_l(f_j), k_l(f_i)) = softmax_i(\frac{q_l(f_j)^T k_l(f_1)}{\sqrt{d_k}}, \ldots, \frac{q_l(f_j)^T k_l(f_n)}{\sqrt{d_k}})$$

where l refers to the l'th head, and $softmax_i$ is the i'th element of the softmax function, defined as

$$softmax(x_1, x_2, \ldots, x_n) = \frac{1}{\sum_{i=1}^{n} e^{x_i}}(e^{x_1}, e^{x_2}, \ldots, e^{x_n}).$$

Note that for a given head l, feature vectors with higher similarity to f_j will have a larger dot product with $q_l(f_j)$ compared to other feature vectors. This drives the network to learn, for each representation, r_j, on which feature vectors of the input sequence it should focus its attention to learn the representations of value. To recap, representations $A_j^1, A_j^2 \ldots, A_j^h$ depend on feature vectors that receive a high degree of attention.

Next, these representations are concatenated to a form $A_j = A_j^1 \circ A_j^2 \circ \cdots \circ A_j^h$.

The last step of the MHSA is applying to A_j a linear layer, to generate the MHSA output representation.

To further explore the design space, we experiment with a different activation function. Specifically, we replace softmax by rectified linear unit (ReLU) activation function,

as suggested in [36], as follows:

$$similarity(q_l(f_j), k_l(f_i)) = \frac{ReLU(\frac{q_l(f_j)^T k_l(f_1)}{\sqrt{d_k}}, \ldots, \frac{q_l(f_j)^T k_l(f_n)}{\sqrt{d_k}})}{\gamma\sqrt{\frac{n}{2}}}$$

where n is the number of extracted fragments and γ is a hyper parameter.

2) TRAINABLE WEIGHTS AND HYPER-PARAMETERS

For each head, $l \in [1, h]$, we define three weight matrices W_l^q, W_l^k, W_l^v. Additional weight matrix $W^o \in \mathbb{R}^{h \cdot d_v \times d_{model}}$ is reserved for the linear layer that processes A_j.

The MHSA hyper-parameters include the number of heads h, the dimensionality of the query and key representation $d_q = d_k$, and the dimensionality of the value representation d_v.

3) PIECE-WISE MULTI LAYER PERCEPTRON

The piece-wise MLP receives as an input a sequence of representations (feature vectors). For each feature vector, it learns an output representation. Unlike the MHSA, in the piece-wise MLP layer each learned representation depends only on its input feature vector and does not depend on other feature vectors in the sequence. This layer transforms each input feature vector into an intermediate representation of dimensionality d_{ff}, and learns the output representation which is of dimensionality d_{model}. Formally, to perform this calculation, we define two weight matrices $W^{ff} \in \mathbb{R}^{d_{model} \times d_{ff}}$ and $W^m \in \mathbb{R}^{d_{ff} \times d_{model}}$. and calculate the representation of f as follows:

$$MLP(f) = (ReLU(f^T \cdot W^{ff}))^T \cdot W^m$$

4) TRAINABLE WEIGHTS AND HYPER-PARAMETERS

There are two weight matrices W^{ff} and W^m as defined above. The only hyper-parameter is d_{ff}.

5) LAYER NORMALIZATION

Layer normalization [37] is a method used to normalize the activities of the neurons of a layer improving the training speed for neural network models. It directly estimates the normalization statistics from the summed inputs to the neurons within a hidden layer.

6) THE TRANSFORMER ENCODER

The Transformer Encoder (TE) comprises a stack of L identical encoder layers, each comprising an MHSA sub-layer followed by a piece-wise MLP sub-layer. Layer normalization is applied before each sub-layer, and a residual connection is employed around each of the two sub-layers.

7) HYPER-PARAMETERS OF THE TE

The number of encoder layers - L, the dimensionality of the query and key representations - $d_q = q_k$, the dimensionality of the value representation - d_v, and the dimensionality of the intermediate representation of the piece-wise MLP - d_{ff}.

C. GREEDY LAYER-WISE PRETRAINING

Training DNN is challenging due to the vanishing gradient problem (i.e., the weights in layers close to the input layer are not updated in response to errors calculated on the training dataset). The Greedy Layer-Wise Pretraining allows DNN to be successfully trained, achieving state-of-the-art performance. Pretraining involves adding a new hidden layer to a model and refitting, allowing the new model to learn the inputs from the existing hidden layers, while keeping the weights for the existing hidden layers fixed and not trainable. This technique is based on the observation that training a shallow network is easier than training a deep one [38].

D. REGULARIZATION

To generalize the results of the training set better, we use several regularization techniques in parallel. We employ a dropout after each MHSA and MLP sub-layers in the network. Moreover, we use L2 regularization [39] on the trainable weights of the network (not including the biases).

E. THE MINHASH SCHEME

A modified MinHash [24], [25] is the second processing component of ViRAL, whose role is extracting the informative feature vectors from the genome. MinHash, or the min-wise independent permutations scheme is a technique for similarity estimation. MinHash is used in many tasks in computational biology, including genome assembly [40], [41], metagenomic gene clustering [42], [43] and genomic distance estimation [44]. MinHash implements the following sketch function:

Given a set of characters A, a compression factor n, and a hash function h, the elements of the set A are hashed using function h to generate the set $H(A)$. Then the elements of $H(A)$ are sorted and the inputs to the smallest n elements are returned as described in Algorithm 1.

Algorithm 1 The Sketch Algorithm $sketch(A)$

Require: set $A = \{a_1, \ldots, a_{|A|}\}$, compression parameter n and a hash function h
$sketch \leftarrow \emptyset$
$H(A) \leftarrow (h(a_1), \ldots h(a_{|A|}))$
Sort H(A) to get $(h(a_{i_1}), h(a_{i_2}), \ldots, h(a_{i_{|A|}}))$
return $(a_{i_1}, a_{i_2}, \ldots, a_{i_n})$[4]

The subset obtained by applying the sketch algorithm provides a good estimate for the Jaccard index [45], defined as follows: Given two sets A, B the Jaccard index is

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

[4]These are the genome representative kmers referred to in Section III-A.

Formally, given two sets, A and B,

$$\frac{|sketch(A) \cap sketch(B) \cap sketch(A \cup B)|}{|sketch(A \cup B)|} \approx J(A, B).$$

We use the sketch algorithm to extract feature vectors from a genome, as presented further.

F. RELATED WORK

1) DNA CLASSIFICATION

The goal of a DNA classifier is to find which organism(s) a biological sample contains. Several probabilistic classifiers have been proposed, such as interpolated Markov model-based Phymm and PhymmBL [46], BLAST-based models [47], MEGAN [48] and METAPHYLER [49], naive Bayesian classifier NBC [50], and others. To speed the DNA classification up, exact pattern matching classifiers were developed, including CLARK [51], CLARK-S [52], Kraken [53] and Kraken2 [54].

Several tools apply deep learning techniques to classification of biological samples. In [55], a convolutional neural network (CNN) is applied to the classification of bacterial DNA, focusing on 16S rRNA (a gene that appears in bacteria and provides species-specific signature) recognition. In [56], a CNN and a recurrent neural network (RNN) based solutions for several special cases of bacterial DNA classification are developed and compared. PACIFIC [4], a CNN/RNN based solution has been proposed to detect the presence of SARS-CoV-2 and other common respiratory RNA viruses in RNA-seq data (RNA-seq is a sequencing technique to detect and quantify RNA in a biological sample). Transformer-based tools have been proposed for gene expression prediction [57], [58].

While state-of-the-art classifiers offer a solution for detection and classification of species (e.g., detecting SARS-CoV-2 DNA in a biological sample containing DNA of several different organisms, or classifying different bacteria), they typically lack the sensitivity to classify variants, or mutations of the same organism (e.g., Omicron BA.4 and Omicron BA.5 variants of SARS-CoV-2 virus). This is due to high similarity between variants of the same organism.

Several recently developed tools aim at resolving the much more complex problem of lineage and phylogenetic assignment (i.e., placement of an accession genome on a phylogenetic tree). EPA-ng [7] computes the optimal insertion position for an accession in a given reference phylogenetic tree with respect to its maximum likelihood. IQ-TREE [6] uses stochastic algorithms for estimating maximum likelihood phylogeny. UShER [9] uses a maximum parsimony approach by searching for an assignment that requires the fewest additional mutations. An apparent limitation of UShER is that it only supports certain types of mutations.

State-of-the-art classifiers include hardware-accelerated solutions as well. SquiggleFilter [5] is a virus detection framework that analyzes the raw output (raw squiggles) of the ONT MinION sequencer [59] and filters out all but the target virus DNA reads. EDAM [60] is an edit distance-tolerant content addressable memory for approximate search, used for the DNA detection and classification tasks. GenSLMs [61] applies large language models to the identification and classification of viral variants using supercomputers such as Polaris at the Argonne Leadership Computing Facility and Selene at NVIDIA, as well as Cerebras CS-2 wafer-scale cluster.

2) MINHASH ACCELERATORS

Several Minhash acceleration solutions have been proposed. MetaCache GPU [62] uses MinHash for metagenomic classification and implements it on GPU. MSIM [63] is a near-memory MinHash accelerator with a limited energy consumption reduction (26.4× vs. high-performance GPU), targeted for high-performance applications. In [64], MinHash is used for kmer clustering. An FPGA-based solution is limited to parallel calculation of 15 hash functions (whereas ViRAL calculates in parallel at least 256 hash functions). JACK-FPGA [65] uses a cloud FPGA to accelerate MinHash. Scotch [66] is an FPGA-based locality-sensitive hashing accelerator with a limited energy consumption gain (5× over high-performance GPU).

The purpose of MinHash acceleration in ViRAL framework is to balance the MinHash and Vision Transformer calculations while providing the highest energy efficiency, preferably sufficient for portable applications.

III. ViRAL ALGORITHM

ViRAL is the platform for a quick and accurate identification, classification, and lineage assignment of SARS-CoV-2 variants. It accepts an assembled genome and outputs the ordered list of most probable lineages such genome belongs to. If the top probabilities output by ViRAL are similar, this might indicate that the new genome belongs to an unknown lineage, creating a new node on the phylogenetic tree.

The top-level overview of ViRAL algorithm is presented in Figure 3. The input to ViRAL is an assembled genome in FASTA file format. To convert the genome into a sequence of feature vectors, we build a pipeline of preprocessing steps, comprising the feature extractor and the embedding layer. The feature extractor chooses a set of fragments (i.e., genome subsequences of fixed length), represents each fragment as a two-dimensional matrix, and outputs those genome fragment matrices (GFMs). Each GFM is fed into an embedding layer that consists of one neuron. This layer converts the GFM into a genome fragment vector, designated the feature vector. The feature vectors are fed into the ViT, which outputs the most likely assignment candidates (by attaching probabilities to each lineage).

The feature extractor together with the embedding layer transforms a genome into a sequence of representative feature vectors, which are the numerical representations of genome fragments. We implement the feature extractor using MinHash to preserve similarities in the genome. It allows the feature extractor to find features in the query genome

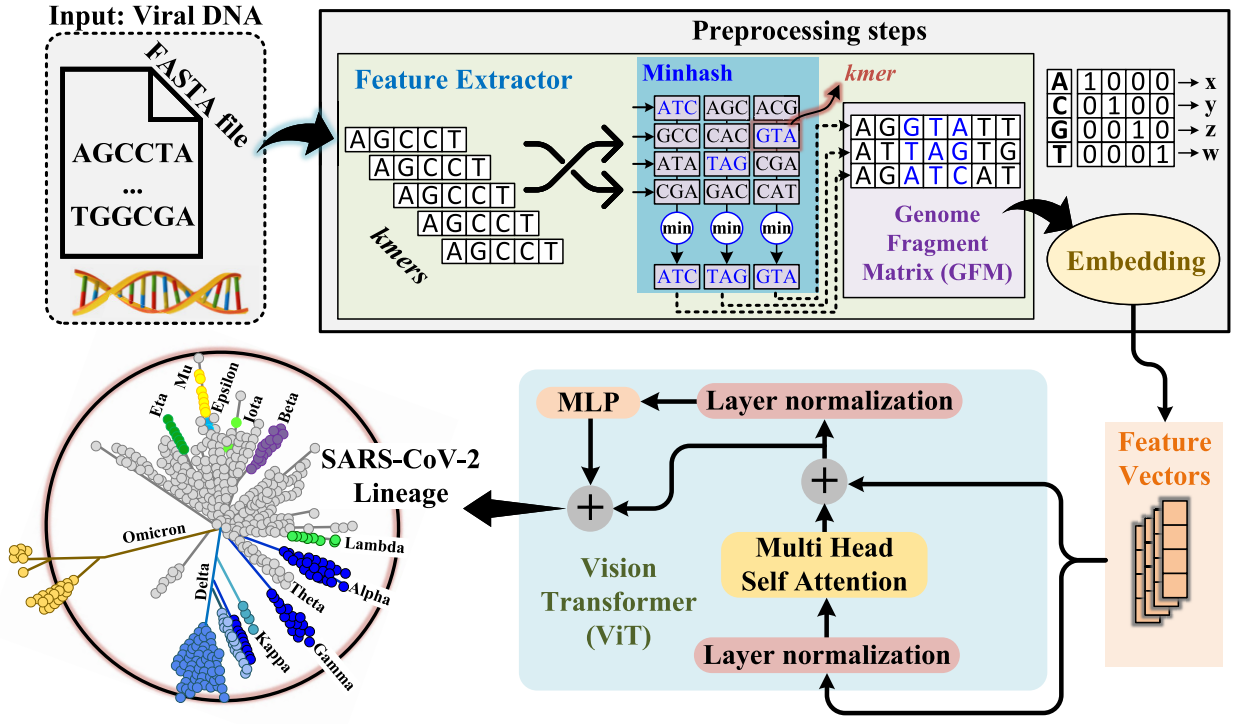


FIGURE 3. A general overview of the ViRAL algorithm.

that are shared by the known lineages. The embedding layer is used as a dimensionality reduction step. It allows for the reduction of ViRAL latency without affecting its assignment accuracy.

A. FEATURE EXTRACTOR

The first part of the ViRAL pipeline receives the assembled genome and outputs matrices that represent genome fragments. Feature extractor employs MinHash to find similar fragments (i.e., features) in different genomes.

Algorithm 2 The Feature Extractor

Require: genome g, compression parameter n, fragment size f, kmer size k, hash function h

$G \leftarrow \{g_i \equiv g[i : i+k]\}\ s.t.\ i \in [0, n-k+1]$

$kmers \leftarrow sketch(G, n, h)$

$left \leftarrow floor(\frac{f-k}{2})$

$right \leftarrow ceil(\frac{f-k}{2})$

$fragments \leftarrow \{g[i-left, i+right] : g_i \in kmers\}$

return $one-hot-encode(fragments)$

One hot encoding of a fragment refers to the operation of transforming each base in the fragment as follows:

$A \rightarrow [1,0,0,0]$; $C \rightarrow [0,1,0,0]$; $G \rightarrow [0,0,1,0]$; $T \rightarrow [0,0,0,1]$; $N \rightarrow [0,0,0,0]$.

For the example please refer to Figure 4(b).

An example is illustrated in Figure 4(a). Input is a genome sequence g of length $N = 19$. We generate all possible kmers (genome sub-sequences of length k) G (where $k = 4$), and extract ($n = 3$) 4mers from the genome using the MinHash scheme (i.e., $sketch(G, n = 3, h)$). The next step is to generate fragments of length $f = 8$ (by extending each of three 4mers by $\frac{f-k}{2} = \frac{8-4}{2} = 2$ bases in each direction). The last step of this workflow is to encode each basepair of the fragments using one-hot encoding.

For the genome sequence g of length N, the extractor first generates a set G of all possible kmers. It then sketches (i.e., applies MinHash *sketch* function to) the set of kmers, G, to extract n representative kmers of the genome (Algorithm 1). Those kmers are used as anchors to be expanded to generate fragments. The last part is transforming a fragment into a numerical matrix where the genome basepairs A, G, C, and T are encoded using one-hot encoding as described in Algorithm 2.

B. EMBEDDING LAYER

The input of the embedding layer is a genome fragment matrix of dimensions $f \times 4$ (i.e., f represents the fragment length and 4 is the dimensionality of the basepair one-hot encoding). It outputs a feature vector of size f. The embedding layer consists of one neuron which performs a base-wise linear transformation. Specifically, the embedding layer defines 5 learnable parameters, 4 weights w_A, w_C, w_G, w_T, and one bias term b_N. Given a genome fragment matrix, we transform each base (represented in one-hot encoding) to a numerical token as presented in Figure 4(b), which also contains an example in which we embed a fragmented matrix of dimensions 8×4.

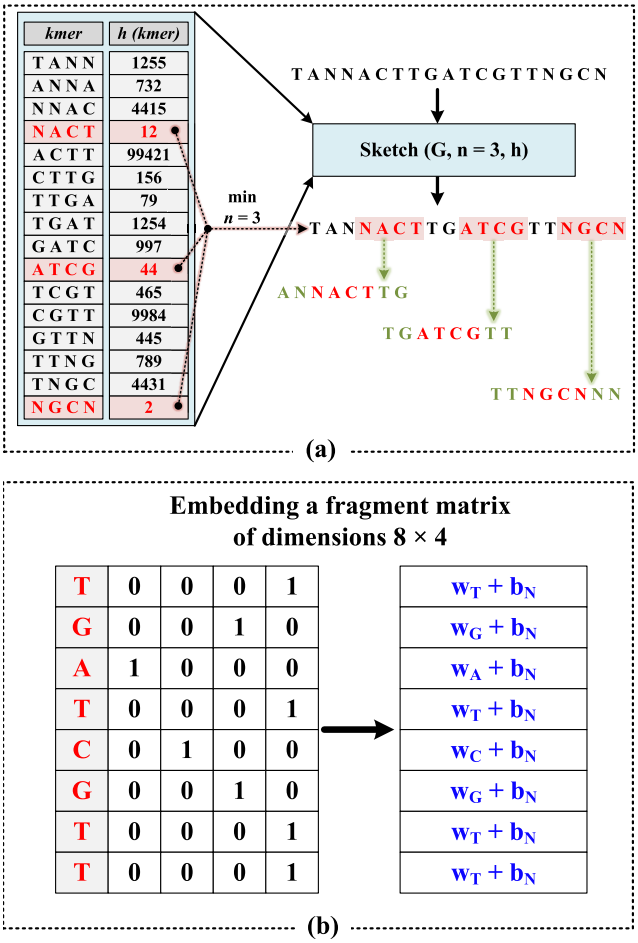


FIGURE 4. (a): Feature extractor workflow example. (b): Embedding a fragment matrix of dimensions 8 × 4.

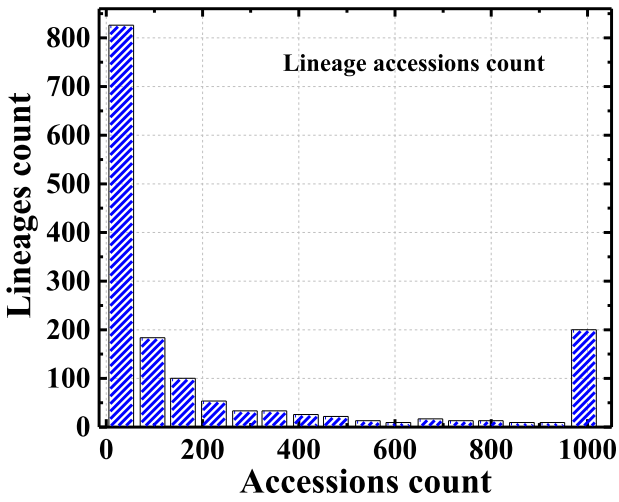


FIGURE 5. The number of sequenced genomes per lineage histogram. Most of the lineages (more than 800 out of 1,536) have less than 64 accessions available. The peak in the last bin accounts for all lineages with more than 960 accessions.

IV. VIRAL BASELINE AND GPU – ACCELERATED IMPLEMENTATIONS

The purpose of these designs is to implement ViRAL in conventional server infrastructure. While the preferable application of ViRAL is in portable field settings or at clinical points of care, it can also provide a fast and accurate software solution for pathogen tracking and diagnostics. We use an 8-core Intel i7-9700K desktop computer, operating at 3.60GHz, with 32GB of 2,666 MT/s DDR4 RAM, equipped with NVIDIA's GeForce RTX 2080 Ti GPU with an 11GB / 14Gbps GDDR6 frame buffer, running at 1.545GHz. The latter is used in GPU-accelerated ViRAL.

In this section, we present the dataset used to train, validate and test the model. We also describe the training process. In the end, we present ViRAL profiling results which provide addition motivation for its hardware acceleration.

A. DATASET

We use the COVID-19 Data Portal [67] as a source of sequenced SARS-CoV-2 genomes. At the time of ViRAL development, it contained 4,470,553 assembled SARS-CoV-2 genomes classified into 1,536 distinct lineages. The distribution of genomes across the lineages is highly non-uniform. For example, 463,426 samples are categorized as B.1.1.7, while most lineages have only a small number of samples, as depicted in Figure 5. Such non-uniformity makes training the network challenging. To mitigate, we select at most 1024 samples from each lineage and discard the rest when constructing the dataset. This way we prevent any potential overfitting towards lineages like B.1.1.7. We also discard all samples that belong to lineages with fewer than 512 accessions, since such a small number of samples limits the model's ability to generalize. These steps form our dataset, which contains 240,734 samples belonging to 270 different lineage classes.

We divide our dataset into three non-overlapping subsets: the training set, the validation set, and the test set (which we designated the *existing-lineage* test set). The training and validation sets are used to train and optimize the network. The test set is applied to evaluate ViRAL's assignment accuracy and to compare it with state-of-the-art solutions. Both the validation and test sets consist of 13,500 genomes randomly selected from the dataset. The rest of the samples comprise the training set.

We also define two additional test sets, the *novel-lineage* and the *non-SARS-CoV-2*, as further detailed in Section VI-A1. The purpose of the former is to evaluate ViRAL on novel variants of SARS-CoV-2, which presumably did not exist at the time of (and hence were not used in) training the network. The purpose of the latter is to evaluate ViRAL's on genome material that comes from organisms other than SARS-CoV-2, for example, different viruses.

Typically SARS-CoV-2 samples that contain ambiguous data (base ambiguity is defined further in Section VI-A1) are excluded from genome analysis [68]. To the contrary, we do not filter out genomes that contain ambiguous bases.

TABLE 1. Network training configuration and network size.

$encoder\_repeats$	d_{model}	d_v	d_k	d_{ff}	h	network size [weights]	training time [h]
4	256	96	96	1536	18	5,359,485	46

On the contrary, we keep base ambiguity in both training and test sets, to ensure the robustness of our solution and enable accurate identification and assignment of SARS-CoV-2 genomes with a significant percentage of ambiguous bases.

B. NETWORK TRAINING

We train ViRAL using Adam [69], which is an optimized gradient descent technique, with $\beta_1 = 0.9$, $\beta_2 = 0.999$, batch size of 1024, and weight decay of 0.0001. We experimentally arrived at the following set of near-optimal hyper-parameters: $encoder\_repeats = 4, d_{model} = 256, d_v = 96, d_k = 96, d_{ff} = 1536, h = 18$. We also applied a dropout rate of 0.2 after each sub-layer. As summed up in table 1. The network was trained to minimize the categorical cross-entropy loss function, using the Greedy Layer-Wise Pretraining approach. In the variant of ViRAL where softmax is replaced by ReLU, we set the hyper parameter $\gamma = 10$. We started the pretraining process with a model consisting of only one TE layer, and gradually added several TE layers at each pertaining step, fixing the weights of the old layers, and training only the weights of the newly added TE layer. After constructing 4 TE layers, we allowed the training of the weights in all the layers and trained the model until the validation loss saturated. The validation loss saturated after 200 epochs which took 46 hours using GPU-accelerated framework. The resulting network size is 5,359,485 weights.

If the model needs to scale up to support newly discovered lineages, it does **not** need to be trained from scratch. The retraining is much shorter, depending on the number of newly added lineages. Due to our use of transfer learning, it takes only an hour to add 100 new lineages to the network.

C. PROFILING

To test the performance of our solution, we ran ViRAL in inference mode, classifying the test set. ViRAL achieved the speed of 55ms per inference, obtaining 3.53× speedup over UShER (we address this result further in Section VI-C). While classifying the test set, we measured the fractions of time spent executing different stages of the program. It was found that 95% of the run time is spent in the feature extraction stage. This presented a clear opportunity to significantly accelerate the ViRAL execution.

V. ACCELERATED ViRAL

We envision ViRAL as a standalone portable platform, to be used in conjunction with portable library preparation and sequencing kit (such as ONT's VolTRAX and MinION [59]), that will democratize pathogen identification and lineage assignment. A portable ViRAL can provide an instant real-time pathogen detection and identification solution at clinical points of care, or in a field environment during a viral pandemic. With a network model easily fitting into an on-chip SRAM, it does not only provide small clinics and labs with the ability to quickly detect a viral pathogen such as SARS-CoV-2, it can also accurately pinpoint its lineage out of hundreds of existing options. Such application is typically form-factor and energy budget limited and therefore can benefit from a dedicated hardware solution.

Furthermore, despite the relatively high performance of software-only (CPU-based) and GPU-accelerated solutions, accelerating ViRAL is advantageous in a server / data center environment as well, especially when facing a problem that requires a viral or bacterial phylogenetic analysis at scale. As presented above, MinHash is responsible for 95% of the execution time, mainly because the conventional computers are not optimized for operations required by LSH. By further zooming into MinHash, we identify several components, such as signature sorting, whose latency scales as $nlog(n)$ where n is the genome size, thus further exacerbating the problem for larger n. If ViRAL needs to be scaled and sped up, MinHash is an obvious target for acceleration.

GenSLMs [61] provides an additional motivation for accelerating ViRAL, by setting very challenging performance goals for viral lineage assignment, targeting supercomputers and wafer scale integration.

A. ADDITIONAL USE OF THE ACCELERATED SOLUTION

In addition to applications briefly listed in Section II-E, MinHash is widely used elsewhere in bioinformatics. MASH [44] uses the MinHash scheme to implement genomic distance estimation (i.e., given two genomes, MASH evaluates their proximity to each other). MetaCache [43] is a metagenomic classification tool that uses MinHash to classify metagenomic samples and identify organisms present in a given sample. MHAP [40], Shasta [41] and other de-novo genome assembly algorithms apply MinHash to aligning DNA reads.

Accelerating MinHash using a standalone accelerator potentially benefits these and other bioinformatics applications that extensively use the MinHash scheme.

B. SYSTEM ARCHITECTURE

Figure 6 illustrates the system view of the ViRAL platform. DNA samples are prepared and sequenced. The assembled genome material is fed into ViRAL, whose main components are MinHash and ViT accelerators, implemented by ACMI,[5] a standalone ACcelerated MInHash ASIC, and NVidia GPU, respectively.

[5]Pronounced "akmee".

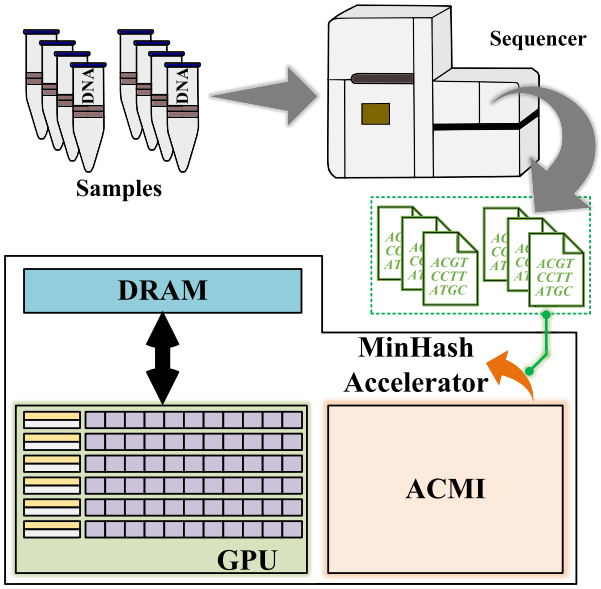


FIGURE 6. System view of the accelerated ViRAL platform featuring MinHash accelerator ACMI, combined with NVidia GPU.

ACMI is a throughput oriented streaming platform. ACMI (1) inputs a genome $i+1$, (2) processes both the previously received genome i and the genome $i+1$, and (3) outputs the genome i related results, in parallel. ACMI inputs the accession genome one DNA base per cycle (on average). In the case of SARS-CoV-2 virus whose size is approximately 30K DNA bases, it takes approximately 30K cycles to upload a single genome. The result of ACMI is a Genome Fragment Matrix (GFM). Both the compression factor and the *fragment_size* are 256, hence the GFM is a 256×256 matrix of 4-byte elements. To balance the pipeline, ACMI is designed to output 8B/cycle, thus taking approximately $256\times256\times4\,/\,8 = 32$K cycles to output the GFMs.

Since ACMI operating at 500 MHz requires I/O throughput of less than 5 GB/ s to achieve its optimal performance, data connectivity can be implemented by PCIe 3.0 or higher.

The ACMI architecture is presented in Figure 7. ACMI comprises the Fragment Memory (FM), the kmer buffer, the kmer index counter, Hasher, Sorter, and Extender units. Hasher performs hashing of the input kmer sets using MurmurHash3, which is a non-cryptographic hash function suitable for general hash-based lookup. Sorter sorts the hashed kmers (kmer signatures) and outputs 256 smallest signatures. Extender extends the signatures into larger DNA fragments and generates the GFMs.

While Hasher and Sorter process kmers in a pipelined fashion, "on the fly", the Extender requires access to the entire genome. Therefore, the Fragment Memory (FM) unit is organized as a double buffer, comprising buffers FM1 and FM2, operating in two phases: in phase 1, FM1 receives genome i, which is subsequently hashed and sorted. At the end of the phase, FM1 and FM2 are logically swapped. In phase 2, FM2 receives genome $i+1$, which is being

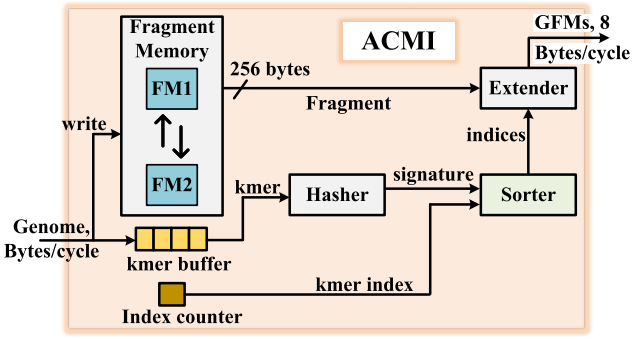


FIGURE 7. The top-level architecture.

processed by the Hasher and Sorter. At the same time, FM1 serves the Extender, which generates the GFMs of the genome i. At the end of the phase, the buffers are swapped again, and so on. ACMI pipeline timing is shown in Figure 8.

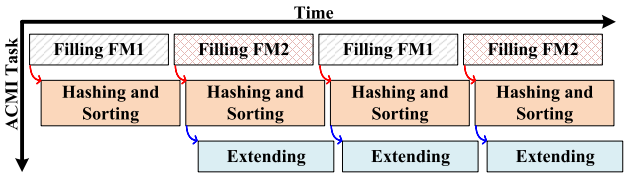


FIGURE 8. ACMI pipeline timing.

C. KMER BUFFER AND INDEX COUNTER

The kmer buffer converts the DNA base stream received by ACMI into the stream of kmers (16mers). The index counter keeps track of the kmer index (the position of the kmer on the genome sequence).

D. FRAGMENT MEMORY

The Fragment Memory buffers enable parallel write and read. The write operation occurs at a single-byte granularity. The read occurs in chunks of 256 bytes. As depicted in Figure 9, each FM buffer comprises eight 128 × 32 byte SRAM modules and can store up to 32KB of genomic information. FM address is composed of the row pointer (7 MS-bits), the SRAM module pointer (the following 3 bits), and the byte offset (the byte address within the row, 5 LS-bits). Since the

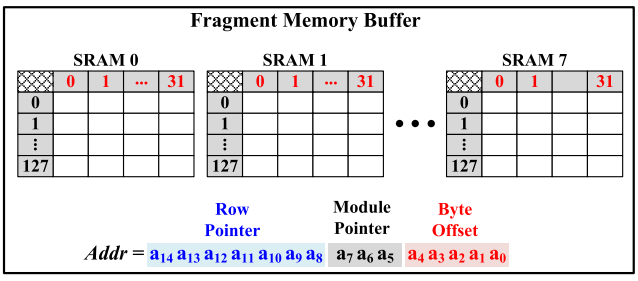


FIGURE 9. A fragment memory buffer comprising 8 128 × 32 bytes SRAM modules.

read data granularity is 256 bytes, only the 7 MS-bits of the address are used in read access.

E. SORTER

The Sorter maintains a list of compression factor = 256 lexicographically smallest kmer signatures. It receives the signatures from the Hasher, along with their indices. After all kmers of the genome are hashed and sorted, Sorter transfers the entire list of 256 indices of the smallest signatures to the Extender (in parallel). At this point, the FM buffers are swapped, the Extender begins processing the index data of the recently hashed and sorted genome, while a new genome is being written to the other FM buffer, and hashed and sorted by ACMI in a pipeline fashion.

The hashed kmers (signatures) are transferred sequentially from the Hasher to the Sorter. In each cycle, each comparator in the chain receives from its left neighbor a tuple comprising a kmer signature and its index, and compares such signature with the minimal signature value the comparator stores. If the new signature is smaller (lexicographically) than the stored one, the comparator saves the new tuple and outputs the old tuple, otherwise, it outputs the new tuple. After all, signatures are processed, the comparator chain retains the 256 smallest signatures and their indices (which are afterwards transferred to the Extender).

F. EXTENDER

Extender converts each of the 256 kmers with the smallest signatures into a 256-base wide DNA fragment, by extending such kmer left and right. To accomplish that, Extender reads the DNA fragments directly from one of the FM buffers, using the index it receives from the Sorter to calculate the FM row pointer, as follows.

1) Calculate the fragment position on the genome sequence $frag\_idx = \lfloor \frac{(kmer\_idx - \lfloor \frac{frag\_len - kmer\_len}{2} \rfloor)}{frag\_len} \rfloor$.
2) Read the DNA fragment pointed to by $frag\_idx$.
3) Read the DNA fragment from the next FM buffer row, addressed by $frag\_idx + 1$.
4) Render the extended DNA fragment by concatenating the relevant parts of those two consecutive DNA fragments.

The 256 extended DNA fragments comprise the GFMs that become the output of ACMI.

An example of extension is shown in Figure 10. Suppose the kmer length is 4 (a 4-mer), the fragment length is 8, and the size of the genome is 32. The Extender has to extend the 4-mer with index 9 (i.e., the 4-mer TAAG marked in red in Figure 10). In this case,

$$frag\_idx = \lfloor \frac{(kmer\_idx - \frac{frag\_len - kmer\_len}{2})}{8} \rfloor$$
$$= \lfloor \frac{(9 - \frac{8-4}{2})}{8} \rfloor \lfloor \frac{7}{8} \rfloor = 0.$$

Hence the Extender will read from FM buffer fragments 0 and 1. It will then concatenate the relevant bases using shift operations, render a single 8-base wide DNA fragment, encode the bases using one-hot encoding, and append the encoded extended DNA fragment to the GFMs.

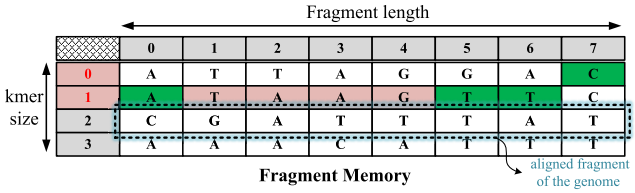


FIGURE 10. Example of extension. The fragment length is 8, the kmer size is 4. Each line represents an aligned fragment of the genome. The bases in pink are the bases of the 4mer at position 9. The extended fragment includes two bases (marked in green) to the left and to the right of the original 4mer.

VI. EVALUATION

In this section, we define the figures of merit used to evaluate ViRAL speed and accuracy, and quantitatively compare ViRAL with state-of-the-art phylogenetic assignment tool UShER [9], classification tool Kraken2 [54], as well as popular classification neural networks ResNet50 [70] and EfficientNet [71]. We also present the results obtained by a variant of ViRAL where softmax is replaced by RELU activation function.

Furthermore, we present the results of ACMI synthesis, and place and route, including the operating frequency, area, and power consumption of ACMI.

A. SETUP

1) LINEAGE ASSIGNMENT ACCURACY AND ASSIGNMENT RATE CRITERIA

ViRAL's ViT is a classification network, which outputs a vector of probabilities, one per each lineage the network is trained on. We use the *top-n accuracy* as ViRAL, as well as ResNet50 and EfficientNet based classifiers' assignment accuracy criterion. Top-n accuracy quantifies the likelihood of the *correct* assignment (i.e., the lineage to which the query genome truly belongs to[6]) appearing among the n most probable results (i.e., n lineages output by the network at the top of the descending-order probability list). Specifically, we calculate and present ViRAL's top-1, top-2 and top-5 accuracy. Such criteria make the comparison to the state-of-the-art straightforward.

Since the *existing-lineage* test set includes only genomes that belong to the lineages the network is trained on, we calculate the top-1, top-2, and top-5 accuracy as the percentage of appearances of the correct assignments among top 1, 2, and 5 highest probability assignments.

The *novel-lineage* and the *non-SARS-CoV-2* test sets are used to evaluate the ViRAL accuracy in classifying unknown genome samples, for example, a novel SARS-CoV-2 variant, and genome material different from SARS-CoV-2,

[6]Since the COVID-19 Data Portal assigns all sequenced genomes in our dataset to their respective lineages, the correct assignments are known.

respectively. The former contains 100 SARS-CoV-2 genomes belonging to new lineages **not** used in training. The latter contains viral genomes different from SARS-CoV-2, such as Lassa virus.

Since the genome material of the novel-lineage and the non-SARS-CoV-2 sets did not take part in training, there are no *correct results* as in the case of the existing-lineage test set. We however can study the output probability distribution to generate several useful insights, as presented in Section VI-B1.

We define the fraction of ambiguous bases (i.e., R, Y, K, N, etc.) appearing in an assembled SARS-CoV-2 genomes as *base ambiguity*. Typically, base ambiguity is caused by incomplete genome coverage, which might be the result of low-quality library preparation and sequencing settings. N marks the worst ambiguity because it can mean any base (A, G, C, or T). We further evaluate the accuracy of ViRAL assignment under different levels of base ambiguity. In our study, we conservatively use only the worst-case (i.e., N) ambiguity.

2) LINEAGE ASSIGNMENT RATE

Another criterion we apply in our analysis is the *lineage assignment rate*, a fraction of genomes that ViRAL is able to assign to a lineage (correctly or incorrectly). The reasons the assignment rate does not reach 100% are base ambiguity and sequencing errors.

B. VIRAL LINEAGE ASSIGNMENT ACCURACY

1) LINEAGE ASSIGNMENT

UShER uses a maximum parsimony approach where it searches for an assignment that requires the fewest additional mutations. UShER, therefore, requires variant information in variant call format (VCF) stored along with the phylogenetic tree to track mutation data. In our comparative evaluation, we use UShER version v0.5.6 and execute UShER software on the same local computing platform used to implement ViRAL and the rest of the reference classifiers. The computing platform comprises Intel i7-9700K 8-core CPU and NVIDIA's GeForce RTX 2080 Ti GPU. The same test set we use for ViRAL, comprising 13,500 genome samples evenly distributed between the SARS-CoV-2 lineages, is also used for UShER and the rest of the reference classifiers.

Kraken2 operates directly on the sequenced genome. ResNet50 and EfficientNet receive the same input as ViRAL. We use Memory Profiler [72] to measure the memory consumption and TimeIt [73] to measure the execution time of the classifiers under evaluation.

UShER achieves 92.1% top-1 accuracy over the test set, while ViRAL presents a 2.1% increment over UShER in top-1 accuracy performance, achieving 94.2%. Moreover, ViRAL achieves a top-2 accuracy of 97.9% and top-5 accuracy of 99.8%. ResNet50 and EfficientNet achieve top-1 accuracy of 90.1% and 88.4% respectively, while Kraken2 achieves the placement accuracy of 51.3%, as summarized in Table 2.

TABLE 2. Accuracy comparison, existing-lineage test set (all tested genomes come from lineages the network was trained on).

Program	top-1 accuracy	top-2 accuracy	top-5 accuracy
ViRAL	94.2%	97.9%	99.8%
ViRAL with ReLU	87.6%	92.2%	96.9%
ResNet50	90.1%	95.4%	97.6%
EfficientNet	88.4%	92.4%	96.0%
UShER	92.1%	97.2%*	-
Kraken2	51.3%	-	-

* UShER uses accuracy criterion called *sister node placement*. This criterion is very similar, although not strictly identical with the top-2 accuracy.

Applying ViRAL to the novel-lineage and the non-SARS-CoV-2 test sets (containing SARS-CoV-2 genomes of lineages not used in the network training, and non-SARS-CoV-2 viral DNA, respectively), we obtain the following results and conclude as follows:

- The average highest probability output by the network when classifying data of the non-SARS-CoV-2 set is 0.02 (compared to the average highest probability when classifying the data of the existing-lineage test set, 0.63). We therefore can set a probability threshold (for example at 0.1-0.2), which will allow differentiating between SARS-CoV-2 and other unrelated organisms.
- The average highest probability output by the network when classifying the novel-lineage test data is 0.45. While being lower than the highest probability of the existing-lineage test data, it still places the genomes of the novel-lineage test set well above the probability threshold discussed above.
- The top (i.e. the highest probability) results generated by the classification of the novel-lineage test set samples typically include the closest relatives of the genome we attempt to classify. By accurately and rapidly identifying such closest relatives, we potentially make the evolutionary tracking of virus mutation faster and more efficient.

2) ROBUSTNESS TO BASE AMBIGUITY

Some low-quality SARS-CoV-2 genomes may contain ambiguous bases, which might be a result of low-quality sample, low-quality or incomplete sequencing, or low-quality assembly. It is obviously highly desirable that such genomes be correctly identified and placed even when the level of ambiguity is significant. To test the robustness of ViRAL to base ambiguity, we mask random sites (i.e., replace basepairs by N) in each SARS-CoV-2 genome of the test set, with the percentage of ambiguous bases ranging from 0 to 90%. We then compare ViRAL accuracy and lineage assignment rate on this *ambiguous* test set (13,500 genome accessions with randomly masked sites) against the lineage assignment rate and accuracy of state of the art reference solutions. Figure 11 presents the results. At 30% ambiguity, UShER's lineage assignment rate is 0.5%, which means UShER is able to place only 67 accessions out of 13,500. For the rest of

TABLE 3. Run-time and memory requirements comparison.

Program	Avg runtime (per sample) [ms]	Peak memory usage [GB]	speedup over UShER
ACMI-accelerated ViRAL	**2.81**	N/A	**69.39**
GPU-accelerated ViRAL	55	5.02	3.53
Software only ViRAL	94	3.74	2.06
UShER v0.5.6	195	**0.49**	1
Kraken2	104	3.7	1.8

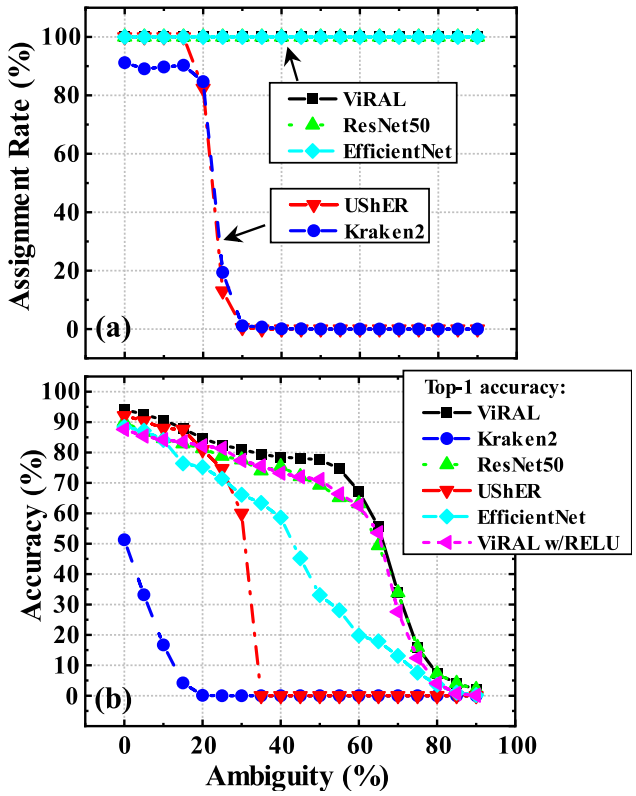


FIGURE 11. (a) Assignment rate and (b) accuracy as a function of the genome ambiguity, i.e., the fraction of the ambiguous bases in the input genome. .

accessions, UShER returns "failed to map" message. For those accessions UShER is able to map, it reaches the top-1 accuracy of 60.5%.

Somewhat similarly, the recently unveiled viral genome identification accelerator GenSLMs [61] removes the SARS-CoV-2 genomes with more than 1% base ambiguity from the datasets.

In contrast, ViRAL maintains a constant 100% lineage assignment rate throughout this experiment. ResNet50 and EfficientNet maintain 100% lineage assignment rate as well.

At 30% ambiguity, ViRAL reaches the top-1 accuracy of 81.1%, while ResNet50's and EfficientNet top-1 accuracy is 77.6% and 66.1% respectively. At 50% ambiguity ViRAL's top-1 accuracy stands at 77.6% while ResNet50's and EfficientNet top-1 accuracy is limited to 69.3% and 33.1% respectively. ViRAL exhibits such tolerance to base ambiguity due to its general resilience to errors and low-quality data.

TABLE 4. ACMI synthesis results.

ASIC Element	Area (mm^2)	Power (W)
Memory Unit	0.133	0.016
Hasher	0.003	0.004
Sorter	0.003	0.001
Extender	0.027	0.014
Total	0.168	0.035

C. VIRAL PERFORMANCE

We compare the speed and memory usage for three ViRAL configurations: ACMI-accelerated ViRAL, GPU-accelerated ViRAL running on NVIDIA's GeForce RTX 2080 Ti, and software-only ViRAL running on Intel i7-9700K 8 core CPU, with those of UShER while placing SARS-CoV-2 genomes of the test set. The results are presented in Table 3.

We also compare ACMI standalone performance with a popular GPU-optimized LSH package Faiss [74]. The ACMI-accelerated feature extraction time for a single SARS-CoV-2 genome is 0.12 ms, achieving the speedup of 866× over the software-only feature extraction. ACMI outperforms Faiss's LSH executed on NVIDIA's GeForce RTX 2080 Ti by 25×.

GPU-accelerated ViRAL outperforms UShER by 3.53×, while CPU-based software-only ViRAL outperforms UShER by 2.06×.

Overall, the ACMI-accelerated ViRAL takes 2.81 ms to place a single SARS-CoV-2 genome, outperforming UShER by 69.4× and ViRAL GPU accelerated implementation by 19.5×, respectively.

The memory usage of GPU-accelerated ViRAL is almost an order of magnitude higher than that of UShER, however, it is still reasonably modest (around 5 GB).

D. ACMI AREA AND POWER

We synthesized ACMI using Cadence Genus 21.12 in a commercial 16nm process using FinFET and Connected Poly on gate Oxide and Diffusion Edge technologies. A mix of standard and low-voltage cell libraries was used. Table 4 shows ACMI synthesized to a 0.168 mm^2, consuming 35mW, operated at 500MHz.

With 25× higher throughput, ACMI achieves 167, 800× better power efficiency than NVIDIA's GeForce RTX 2080 Ti GPU. It also achieves 42, 800× better power efficiency compared to NVIDIA's edge GPU Jetson TX2.

Placement and Routing were carried out by Cadence Innovus 21.11, using seven metal layers. Figure 12 shows the ACMI layout. Approximately 80% of ACMI area is occupied

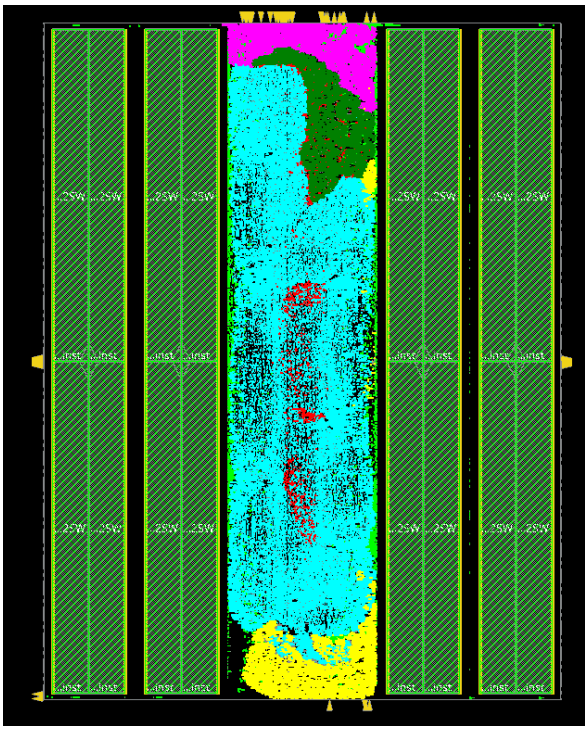


FIGURE 12. ACMI layout. FM is light green. The Hasher logic is magenta. The Sorter logic is yellow. The extender logic is deep green. Light blue is a register file (shared by the Sorter and the extender).

by the FM buffers. Most of the remaining space is occupied by flip-flops belonging to Extender and Sorter register files.

VII. CONCLUSION

ViRAL is inspired by the enormous success of neural networks in implementing classification functions. For the first time, a combination of Vision Transformer, a classification neural network, and MinHash, a locality-sensitive tool, is applied to accurately assign newly sequenced viral pathogen genomes to viral lineages in real-time. ViRAL not only outperforms state-of-the-art tools in accuracy and latency, by identifying and classifying a SARS-CoV-2 genome in 2.81 ms with 94.2% accuracy. It also shows a strong resilience to extremely low quality and low coverage genome data (i.e., high ambiguity of the newly sequenced genomes). ViRAL is able to assign SARS-CoV-2 genomes with 50% ambiguity with 77.6% accuracy. Furthermore, ViRAL can be used to help detect novel viral mutations and variants, by analyzing the distribution of probabilities generated by the network.

ViRAL is envisioned as a portable real-time classification and lineage assignment platform, to be deployed in a field environment or at clinical points of care, to provide a quick and accurate diagnostic of pathogen in newly collected test samples. However, it can also be applied in a data center infrastructure, to provide high-speed pathogen classification as a service.

ViRAL applications are not limited to viral genome classification and lineage assignment. It can also be efficiently applied to classification and identification of microbial pathogens, with the goal of personalizing antibiotic treatment and reducing treatment failures, opportunistic infections and the emergence of drug-resistant organisms.

REFERENCES

[1] E. B. Hodcroft, N. De Maio, R. Lanfear, D. R. MacCannell, B. Q. Minh, H. A. Schmidt, A. Stamatakis, N. Goldman, and C. Dessimoz, "Want to track pandemic variants faster? Fix the bioinformatics bottleneck," *Nature*, vol. 591, no. 7848, pp. 30–33, Mar. 2021.

[2] S. Knyazev, K. Chhugani, V. Sarwal, R. Ayyala, H. Singh, S. Karthikeyan, D. Deshpande, P. I. Baykal, Z. Comarova, and A. Lu, "Unlocking capacities of genomics for the COVID-19 response and future pandemics," *Nature Methods*, vol. 19, no. 4, pp. 374–380, 2022.

[3] L. Ma, H. Li, J. Lan, X. Hao, H. Liu, X. Wang, and Y. Huang, "Comprehensive analyses of bioinformatics applications in the fight against COVID-19 pandemic," *Comput. Biol. Chem.*, vol. 95, Dec. 2021, Art. no. 107599.

[4] P. Acera Mateos, R. F. Balboa, S. Easteal, E. Eyras, and H. R. Patel, "PACIFIC: A lightweight deep-learning classifier of SARS-CoV-2 and co-infecting RNA viruses," *Sci. Rep.*, vol. 11, no. 1, pp. 1–14, Feb. 2021.

[5] T. Dunn, H. Sadasivan, J. Wadden, K. Goliya, K.-Y. Chen, D. Blaauw, R. Das, and S. Narayanasamy, "SquiggleFilter: An accelerator for portable virus detection," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2021, pp. 535–549.

[6] B. Q. Minh, H. A. Schmidt, O. Chernomor, D. Schrempf, M. D. Woodhams, A. von Haeseler, and R. Lanfear, "IQ-TREE 2: New models and efficient methods for phylogenetic inference in the genomic era," *Mol. Biol. Evol.*, vol. 37, no. 5, pp. 1530–1534, May 2020.

[7] P. Barbera, A. M. Kozlov, L. Czech, B. Morel, D. Darriba, T. Flouri, and A. Stamatakis, "EPA-NG: Massively parallel evolutionary placement of genetic sequences," *Systematic Biol.*, vol. 68, no. 2, pp. 365–369, Mar. 2019.

[8] A. Löytynoja, A. J. Vilella, and N. Goldman, "Accurate extension of multiple sequence alignments using a phylogeny-aware graph algorithm," *Bioinformatics*, vol. 28, no. 13, pp. 1684–1691, Jul. 2012.

[9] Y. Turakhia, B. Thornlow, A. S. Hinrichs, N. De Maio, L. Gozashti, R. Lanfear, D. Haussler, and R. Corbett-Detig, "Ultrafast sample placement on existing tRees (UShER) enables real-time phylogenetics for the SARS-CoV-2 pandemic," *Nature Genet.*, vol. 53, no. 6, pp. 809–816, Jun. 2021.

[10] N. A. Kassim and A. Abdullah, "Classification of DNA sequences using convolutional neural network approach," in *UTM Computing Proceedings Innovations in Computing Technology and Applications*, vol. 2, 2017, pp. 1–6. [Online]. Available: https://www.semanticscholar.org/paper/Classification-of-DNA-Sequences-Using-Convolutional-Kassim-Abdullah/5b080c8a7d67fd23c3230dc420452ac2ee34967a#citing-papers

[11] A. Tampuu, Z. Bzhalava, J. Dillner, and R. Vicente, "ViraMiner: Deep learning on raw DNA sequences for identifying viral genomes in human samples," *PLoS ONE*, vol. 14, no. 9, Sep. 2019, Art. no. e0222271.

[12] M. A. B. Mahmoud and P. Guo, "DNA sequence classification based on MLP with PILAE algorithm," *Soft Comput.*, vol. 25, no. 5, pp. 4003–4014, Mar. 2021.

[13] M. A. Deif, A. A. A. Solyman, M. A. Kamarposhti, S. S. Band, and R. E. Hammam, "A deep bidirectional recurrent neural network for identification of SARS-CoV-2 from viral genome sequences," *Math. Biosci. Eng.*, vol. 18, no. 6, pp. 8933–8950, 2021.

[14] J. M. Bartoszewicz, A. Seidel, R. Rentzsch, and B. Y. Renard, "DeePaC: Predicting pathogenic potential of novel DNA with reverse-complement neural networks," *Bioinformatics*, vol. 36, no. 1, pp. 81–89, Jan. 2020.

[15] A. Whata and C. Chimedza, "Deep learning for SARS COV-2 genome sequences," *IEEE Access*, vol. 9, pp. 59597–59611, 2021.

[16] C. M. Dasari and R. Bhukya, "Explainable deep neural networks for novel viral genome prediction," *Int. J. Speech Technol.*, vol. 52, no. 3, pp. 3002–3017, Feb. 2022.

[17] N. Q. K. Le and Q.-T. Ho, "Deep transformers and convolutional neural network in identifying DNA N6-methyladenine sites in cross-species genomes," *Methods*, vol. 204, pp. 199–206, Aug. 2022.

[18] H. Yan, A. Bombarely, and S. Li, "DeepTE: A computational method for de novo classification of transposons with convolutional neural network," *Bioinformatics*, vol. 36, no. 15, pp. 4269–4275, Aug. 2020.

[19] D. R. Kelley, J. Snoek, and J. L. Rinn, "Basset: Learning the regulatory code of the accessible genome with deep convolutional neural networks," *Genome Res.*, vol. 26, no. 7, pp. 990–999, Jul. 2016.

[20] F. Mock, F. Kretschmer, A. Kriese, S. Böcker, and M. Marz, "BERTax: Taxonomic classification of DNA sequences with deep neural networks," *Proc. Nat. Acad. Sci. USA*, vol. 119, no. 35, 2022, Art. no. e2122636119. [Online]. Available: https://www.biorxiv.org/content/10.1101/2021.07.09.451778v1.article-info and https://www.pnas.org/doi/full/10.1073/pnas.2122636119

[21] Y. Kong and T. Yu, "A deep neural network model using random forest to extract feature representation for gene expression data classification," *Sci. Rep.*, vol. 8, no. 1, p. 16477, Nov. 2018.

[22] K. Gayvert, S. McKay, W. K. Lim, A. Baum, C. Kyratsous, R. Copin, and G. S. Atwal, "Evolutionary trajectory of SARS-CoV-2 genome shifts during widespread vaccination and emergence of omicron variant," *NPJ Viruses*, vol. 1, no. 1, p. 5, Nov. 2023.

[23] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16×16 words: Transformers for image recognition at scale," 2020, *arXiv:2010.11929*.

[24] A. Z. Broder, "On the resemblance and containment of documents," in *Proc. Compress. Complex. SEQUENCES*, Jun. 1997, pp. 21–29.

[25] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *J. Comput. Syst. Sci.*, vol. 60, no. 3, pp. 630–659, Jun. 2000.

[26] *Sequencing of SARS-CoV-2-First Update*, Eur. Centre for Disease Prevention Control, Stockholm, 2021.

[27] Y. Furuse, "Genomic sequencing effort for SARS-CoV-2 by country during the pandemic," *Int. J. Infectious Diseases*, vol. 103, pp. 305–307, Feb. 2021.

[28] N. De Maio, C. Walker, R. Borges, L. Weilguny, G. Slodkowicz, and N. Goldman, "Issues with SARS-CoV-2 sequencing data," *Image*, vol. 1119, no. 869, pp. 6–24, 2020.

[29] I. C. Group, "No time to wait: Securing the future from drug-resistant infections," Secretary Gen. United Nations, New York, NY, USA, Tech. Rep., 2019. [Online]. Available: https://www.who.int/publications/i/item/no-time-to-wait-securing-the-future-from-drug-resistant-infections

[30] J. von Neumann, "First draft of a report on the EDVAC," *IEEE Ann. Hist. Comput.*, vol. 15, no. 4, pp. 27–75, Feb. 1993.

[31] *Pioneers of Modern Computer Architecture Receive ACM A.M. Turing Award*. Accessed: Jul. 11, 2023. [Online]. Available: https://www.acm.org/media-center/2018/march/turing-award-2017

[32] (2024). *CoViT: Real-time Phylogenetics for the SARS-CoV-2 Pandemic Using Vision Transformers*. [Online]. Available: https://github.com/zuherJahshan/viral

[33] Illumina. (2021). *Illumina—DNA Sequencing*. [Online]. Available: https://www.illumina.com/techniques/sequencing/dna-sequencing.html

[34] J. S. Kim, D. Senol Cali, H. Xin, D. Lee, S. Ghose, M. Alser, H. Hassan, O. Ergin, C. Alkan, and O. Mutlu, "GRIM-filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies," *BMC Genomics*, vol. 19, no. S2, pp. 23–40, May 2018.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 5998–6008.

[36] K. Shen, J. Guo, X. Tan, S. Tang, R. Wang, and J. Bian, "A study on ReLU and softmax in transformer," 2023, *arXiv:2302.06461*.

[37] J. Lei Ba, J. Ryan Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*.

[38] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 19, 2006, pp. 153–160.

[39] A. Krogh and J. Hertz, "A simple weight decay can improve generalization," in *Proc. Adv. Neural Inf. Process. Syst*, vol. 4, 1991, pp. 950–957.

[40] K. Berlin, S. Koren, C.-S. Chin, J. P. Drake, J. M. Landolin, and A. M. Phillippy, "Assembling large genomes with single-molecule sequencing and locality-sensitive hashing," *Nature Biotechnol.*, vol. 33, no. 6, pp. 623–630, Jun. 2015.

[41] K. Shafin, T. Pesout, R. Lorig-Roach, M. Haukness, H. E. Olsen, C. Bosworth, J. Armstrong, K. Tigyi, N. Maurer, and S. Koren, "Nanopore sequencing and the Shasta toolkit enable efficient de novo assembly of eleven human genomes," *Nature Biotechnol.*, vol. 38, no. 9, pp. 1044–1053, Sep. 2020.

[42] Z. Rasheed and H. Rangwala, "A map-reduce framework for clustering metagenomes," in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops Phd Forum*, May 2013, pp. 549–558.

[43] A. Müller, C. Hundt, A. Hildebrandt, T. Hankeln, and B. Schmidt, "MetaCache: Context-aware classification of metagenomic reads using minhashing," *Bioinformatics*, vol. 33, no. 23, pp. 3740–3748, Dec. 2017.

[44] B. D. Ondov, T. J. Treangen, P. Melsted, A. B. Mallonee, N. H. Bergman, S. Koren, and A. M. Phillippy, "Mash: Fast genome and metagenome distance estimation using MinHash," *Genome Biol.*, vol. 17, no. 1, pp. 1–14, Dec. 2016.

[45] J. M. Hancock, "Jaccard distance (Jaccard index, Jaccard similarity coefficient)," in *Dictionary of Bioinformatics and Computational Biology*. Chichester, U.K.: Wiley, 2004. [Online]. Available: https://www.semanticscholar.org/paper/Classification-of-DNA-Sequences-Using-Convolutional-Kassim-Abdullah/5b080c8a7d67fd23c3230dc420452ac2ee34967a#citing-papers

[46] A. Brady and S. L. Salzberg, "Phymm and PhymmBL: Metagenomic phylogenetic classification with interpolated Markov models," *Nature Methods*, vol. 6, no. 9, pp. 673–676, Sep. 2009.

[47] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *J. Mol. Biol.*, vol. 215, no. 3, pp. 403–410, Oct. 1990.

[48] M. G. Dumont, C. Lüke, Y. Deng, and P. Frenzel, "Classification of pmoA amplicon pyrosequences using BLAST and the lowest common ancestor method in MEGAN," *Frontiers Microbiol.*, vol. 5, p. 34, Jan. 2014.

[49] B. Liu, T. Gibbons, M. Ghodsi, T. Treangen, and M. Pop, "Accurate and fast estimation of taxonomic profiles from metagenomic shotgun sequences," *Genome Biol.*, vol. 12, no. S1, pp. 1–27, Sep. 2011.

[50] G. Rosen, E. Garbarine, D. Caseiro, R. Polikar, and B. Sokhansanj, "Metagenome fragment classification using -Mer frequency profiles," *Adv. Bioinf.*, vol. 2008, pp. 1–12, Nov. 2008.

[51] R. Ounit, S. Wanamaker, T. J. Close, and S. Lonardi, "CLARK: Fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers," *BMC Genomics*, vol. 16, no. 1, pp. 1–13, Dec. 2015.

[52] R. Ounit and S. Lonardi, "Higher classification sensitivity of short metagenomic reads with CLARK-S," *Bioinformatics*, vol. 32, no. 24, pp. 3823–3825, Dec. 2016.

[53] D. E. Wood and S. L. Salzberg, "Kraken: Ultrafast metagenomic sequence classification using exact alignments," *Genome Biol.*, vol. 15, no. 3, pp. 1–12, Mar. 2014.

[54] D. E. Wood, J. Lu, and B. Langmead, "Improved metagenomic analysis with Kraken 2," *Genome Biol.*, vol. 20, no. 1, pp. 1–13, Nov. 2019.

[55] R. Rizzo, A. Fiannaca, M. La Rosa, and A. Urso, "A deep learning approach to DNA sequence classification," in *Proc. Int. Meeting Comput. Intell. Methods Bioinf. Biostatistics*. Cham, Switzerland: Springer, 2015, pp. 129–140.

[56] G. Lo Bosco and M. A. Di Gangi, "Deep learning architectures for DNA sequence classification," in *Proc. Int. Workshop Fuzzy Log. Appl.* Cham, Switzerland: Springer, 2016, pp. 162–171.

[57] Y. Ji, Z. Zhou, H. Liu, and R. V. Davuluri, "DNABERT: Pre-trained bidirectional encoder representations from transformers model for DNA-language in genome," *Bioinformatics*, vol. 37, no. 15, pp. 2112–2120, Aug. 2021.

[58] Ž. Avsec, V. Agarwal, D. Visentin, J. R. Ledsam, A. Grabska-Barwinska, K. R. Taylor, Y. Assael, J. Jumper, P. Kohli, and D. R. Kelley, "Effective gene expression prediction from sequence by integrating long-range interactions," *Nature Methods*, vol. 18, no. 10, pp. 1196–1203, Oct. 2021.

[59] ONT. (2021). *MinION—Portable Real-Time Devices for DNA and RNA Sequencing*. [Online]. Available: https://nanoporetech.com/products/minion

[60] R. Hanhan, E. Garzón, Z. Jahshan, A. Teman, M. Lanuzza, and L. Yavits, "EDAM: Edit distance tolerant approximate matching content addressable memory," in *Proc. 49th Annu. Int. Symp. Comput. Archit.*, Jun. 2022, pp. 495–507.

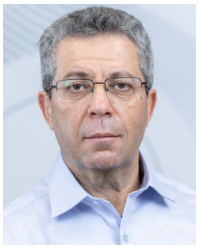
[61] M. T. Zvyagin, A. Brace, K. Hippe, Y. Deng, B. Zhang, C. O. Bohorquez, A. Clyde, B. Kale, D. Perez-Rivera, and H. Ma, "GenSLMs: Genome-scale language models reveal SARS-CoV-2 evolutionary dynamics," *Int. J. High Perform. Comput. Appl.*, vol. 37, no. 6, pp. 683–705, 2023. [Online]. Available: https://www.biorxiv.org/content/10.1101/2022.10.10.511571v1 and https://journals.sagepub.com/doi/10.1177/10943420231201154

[62] R. Kobus, A. Müller, D. Jünger, C. Hundt, and B. Schmidt, "MetaCache-GPU: Ultra-fast metagenomic classification," in *Proc. 50th Int. Conf. Parallel Process.*, Aug. 2021, pp. 1–11.

[63] A. Sinha, J.-Y. Mai, and B.-C. Lai, "MSIM: A highly parallel near-memory accelerator for MinHash sketch," in *Proc. IEEE 35th Int. Syst.-Chip Conf. (SOCC)*, Sep. 2022, pp. 1–6.
[64] J. E. Soto, T. Krohmer, C. Hernández, and M. Figueroa, "Hardware acceleration of k-Mer clustering using locality-sensitive hashing," in *Proc. 22nd Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2019, pp. 659–662.
[65] J. E. Soto, C. Hernández, and M. Figueroa, "JACC-FPGA: A hardware accelerator for Jaccard similarity estimation using FPGAs in the cloud," *Future Gener. Comput. Syst.*, vol. 138, pp. 26–42, Jan. 2023.
[66] M. Kiefer, I. Poulakis, S. Breß, and V. Markl, "Scotch: Generating FPGA-accelerators for sketching at line rate," *Proc. VLDB Endowment*, vol. 14, no. 3, pp. 281–293, Nov. 2020.
[67] *COVID19 Data Portal*. Accessed: Aug. 8, 2023. [Online]. Available: https://www.covid19dataportal.org/sequences/
[68] C. Goswami, M. Sheldon, C. Bixby, M. Keddache, A. Bogdanowicz, Y. Wang, J. Schultz, J. McDevitt, J. LaPorta, E. Kwon, S. Buyske, D. Garbolino, G. Biloholowski, A. Pastuszak, M. Storella, A. Bhalla, F. Charlier-Rodriguez, R. Hager, R. Grimwood, and S. A. Nahas, "Identification of SARS-CoV-2 variants using viral sequencing for the centers for disease control and prevention genomic surveillance program," *BMC Infectious Diseases*, vol. 22, no. 1, p. 404, Dec. 2022.
[69] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[70] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[71] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.
[72] (2024). *Memory Profiler*. [Online]. Available: https://pypi.org/project/memory-profiler/
[73] (2024). *TimeIt*. [Online]. Available: https://docs.python.org/3/library/timeit.html
[74] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, Jul. 2021.

ZUHER JAHSHAN received the B.Sc. and M.Sc. degrees from the Faculty of Electrical and Computer Engineering, Technion, Israel. He is currently pursuing the Ph.D. degree with the Faculty of Engineering, Bar-Ilan University, Israel. His research interest includes improving the efficiency of bioinformatic applications. This is achieved primarily through the development of innovative algorithms and the exploration of new computer architectures that can facilitate faster data processing and analysis.

ESTEBAN GARZÓN (Senior Member, IEEE) received the Ph.D. degree in electronics engineering from the University of Calabria (UNICAL), Italy, in 2022. In 2022, he won a Highly Competitive Research Fellowship funded by the Italian Ministry for Universities and Research (MUR), under the call "Horizon Europe (2021–2027) Programme." He is currently a Research Fellow with the Department of Computer Engineering, Modeling, Electronics, and Systems Engineering, UNICAL. He has authored/coauthored more than 45 scientific papers in international peer-reviewed journals and conferences and has participated in several IC tapeouts. His research interests include domain-specific hardware accelerators, spintronics, cryogenic embedded memories, standard and emerging technologies for logic and memory, and low-power applications.



LEONID YAVITS (Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical engineering from Technion, Israel. He is currently with the Faculty of Engineering, Bar-Ilan University, Israel. He is also a Serial Entrepreneur who was involved in co-founding and successful management (from a concept to M&A) of several start-ups in the field of ASICs. His research interests include bioinformatics, domain specific accelerators, and processing in memory.

Open Access funding provided by 'Università della Calabria' within the CRUI CARE Agreement