Machine Learning (2023) 112:3633–3658
https://doi.org/10.1007/s10994-023-06371-4

Spike2CGR: an efficient method for spike sequence classification using chaos game representation

Taslim Murad[1] · Sarwan Ali[1] · Imdadullah Khan[2] · Murray Patterson[1]

Received: 28 November 2022 / Revised: 12 July 2023 / Accepted: 17 July 2023 /
Published online: 28 August 2023

Abstract

Biological sequence classification is an essential task in many fields, such as machine learning, biology, and bioinformatics. Due to the spread of the coronavirus disease, numerous sequence data are available to researchers. As this virus has affected many hosts (e.g. bats, humans, chickens, etc.) and transformed into different lineages/variants (e.g., alpha, beta, gamma, and omicron, etc.), the biological sequence data for this virus is accompanied by the respective information about the affected host and lineage type of the sequences. Moreover, it is well known in the biology domain that many mutations (that happen disproportionally) related to the coronavirus are present in the spike protein region. Therefore, working with only spike sequences is usually sufficient rather than using a full-length genome for sequence analysis. For a spike sequence, this paper intends to design an image representation so that sophisticated image classification algorithms can be applied to perform the tasks of coronavirus lineages and host classifications. We propose a method based on the idea of chaos game representation (CGR), called Spike2CGR, which converts spike sequences into graphical form (images), and those images are used as input to deep learning (DL) models. We also use some domain knowledge from the biology field to design a few modified versions of Spike2CGR that are biologically meaningful and outperform the SOTA in terms of predictive performance. We use different DL models to perform coronavirus lineage and host classifications and report predictive results employing various evaluation metrics. Using two real-world datasets, we show that Spike2CGR outperforms the SOTA method in terms of predictive performance on spike sequences for variant and host classification.

Keywords Spike sequence · Deep learning · Sequence classification · COVID-19 · Chaos game representation · Image classification · Coronavirus host

Editors: Fabio Vitale, Tania Cerquitelli, Marcello Restelli, and Charalampos Tsourakakis.

Taslim Murad and Sarwan Ali have contributed equally.

Extended author information available on the last page of the article

1 Introduction

The coronavirus disease, which began in 2019 (also called COVID-19), impacted the whole world by causing a global health crisis (Majumder and Minko, 2021). Almost 2.94 million deaths and 136 million people infected, belonging to roughly 219 countries, have been reported by April 2021 (Uyangodage et al. 2021). Due to its rapidly increasing impacts and availability of data, it has received much attention from the research perspective. More work is being done to gain a deeper understanding of this virus, which can help in preventing its further spread and minimizing its existing effects (Ahmed and Jeon, 2021; Kuzmin, 2020). For instance, efforts are put into investigating its genetic diversity and dynamics (e.g., mutations, variations, hosts), etc. Analysis of bio-sequences can help researchers, governments, and medical experts to understand the genetic variations within a sequence (e.g., the SARS-CoV-2 sequence). Because of the recent COVID-19 pandemic, it becomes more crucial to analyze and understand the virus's origin, behavior, and structure. This type of analysis could help relevant authorities develop effective vaccines and antiviral drugs, and design efficient techniques to reduce their impact on society (e.g., designing an early warning system).

It is well-known that major mutations related to COVID-19 appear disproportionately in the spike region of the virus (Kuzmin, 2020; Ali and Patterson, 2021), possibly because of its role in attaching to the host membrane, e.g., many of the residues in the spike amino acid (protein) sequence are exposed compared to those in other protein sequences (Kuzmin, 2020). Therefore, we use only the spike protein sequences to further classify the sequences either variant-wise or infected host-wise, rather than using the full-length genome sequences. Figure 1 illustrates the structure of the full-length genome of the SARS-CoV-2 virus.

The traditional method, such as phylogenetics (Hadfield et al., 2018; Minh, 2020), is not a feasible/practical option for sequence analysis as they are computationally very expensive, hence are not very scalable. Phenetic methods may have some promise because they approximate phylogenetic relationships (to the advantage of being faster), however, they still incorporate some taxonomic information. On the other hand, embedding generation is a complete departure from any taxonomic relationships (it is a completely "flat" structuring of elements to just pairwise distances). The phylogenetics and other traditional methods for SARS-CoV-2 classification have some other limitations too, such as the requirement of extensive domain knowledge, long computational times, and potential inaccuracies due to a lack of information or incomplete data. The use of image recognition and DL computation is justified by the potential benefits such as the ability to handle large-scale data and classify sequences in an automated and fast manner, without the need for extensive domain knowledge.

Due to the availability of large-scale bio-sequence data on databases like GISAID (2021), machine learning (ML)/DL-based sequence analysis has become a very

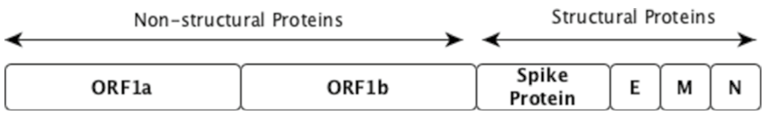


Fig. 1 The full-length genome of the coronavirus (SARS-CoV-2) is roughly 30kb in length, which includes structural and non-structural proteins. The ORF1a and ORF1b encode the non-structural proteins. The structural proteins are encoded by the corresponding spike, envelope, membrane, and nucleocapsid regions. The resulting encoded spike protein sequence is composed of approximately 1300 amino acids

attractive alternative. However, these analytical approaches require numerical data to operate, therefore many researchers have proposed methods to convert the biosequences into numerical forms. For example, OHE (Kuzmin, 2020) came up with an alignment-based technique to get a binary representation of the sequence, but since multiple sequence alignment is an expensive operation (Chowdhury and Garai, 2017), it is not desirable to use the alignment-based methods in real-world scenarios. Likewise, the alignment-free *k*-mers frequency-based method (Ali and Patterson, 2021) is also put forward, however, it generates sparse and high dimensional numerical embeddings. Moreover, PWKmer (Ma et al., 2020) is built upon the concept of position distribution information and *k*-mers to map the sequences to numerical representations, but it is computationally expensive and also undergoes the curse of dimensionality challenge. Although these sequence-to-vector-based methods show promising predictive performance, since they involve feature engineering in the embedding process, a step that is hand-crafted towards a specific task, it is not always guaranteed to preserve all information in the resultant vectors. An alternative is to convert the sequences into images while preserving all information and then use DL models to perform image-based classification for sequence analysis.

Image classification in the natural language processing (NLP) domain is a well-studied problem. Many sophisticated DL models have been developed in past years to achieve state-of-the-art performance in terms of image classification using DL architectures such as convolutional neural networks (CNN). Our idea in this paper is to use the power of those image classification algorithms for the classification of coronavirus variants and hosts based on spike sequence content. More specifically, given a spike sequence, we propose a method called Spike2CGR, which converts the sequence into a graphical form that can be used as input to DL algorithms such as CNN for image classification. One advantage of converting sequences to images is that this approach does not depend on sequence length or alignment — the size of the image remains fixed.

Although deterministic bioinformatics algorithms can be used to analyze biological sequence data, there are several advantages to formulating the problem as a prediction task. First, by leveraging big data, prediction algorithms can handle (learn from) noisy and incomplete data along with variable length sequences (alignment-free property) more effectively than deterministic algorithms, which are engineered ahead of seeing the data. This is particularly relevant in the case of novel viral sequencing data, which can be noisy and incomplete due to various factors such as sequencing errors, low coverage, and viral diversity. Second, prediction algorithms can learn complex relationships between sequence features and phenotype, which may not be immediately apparent to human experts. This can lead to new insights and discoveries in the field of virology. Finally, prediction algorithms can be trained on large-scale data sets, allowing for the identification of subtle patterns and trends that may be missed by manual inspection or deterministic algorithms. We believe that our proposed method, Spike2CGR, which converts viral spike sequences into graphical form for input to deep learning models, addresses these advantages of prediction algorithms. Our method can effectively handle noisy and incomplete data, learn complex relationships between sequence features and phenotype, and can be trained on large-scale data sets. Furthermore, the proposed Spike2CGR incorporates domain knowledge (concepts of minimizers) from the biology field to improve predictive performance, and we show in our experiments that Spike2CGR outperforms the state-of-the-art method in terms of predictive performance on spike sequences for variant and host classification.

In this paper, our contributions are the following:

1. We propose an efficient way to convert sequences into graphical form in order to apply well-established image classification algorithms from the NLP domain to biological sequence classification.
2. Our proposed embedding method is alignment-free and could improve the "*area of interest*" within the image by performing biologically meaningful manipulation of a sequence first and then mapping the manipulated sequence into an image, which could help the underlying DL model in terms of efficient classification.
3. The Spike2CGR-based embedding employs concepts (minimizers) from the domain of metagenomics, which adds biological relevance to our proposed embedding.
4. We show from the results that the proposed Spike2CGR embedding is better than the current state-of-the-art (SOTA) approaches in terms of predictive performance.
5. We also propose a *k*-mers, minimizer, and position weight matrix-based variations of Spike2CGR that outperforms the SOTA approach in terms of predictive performance.

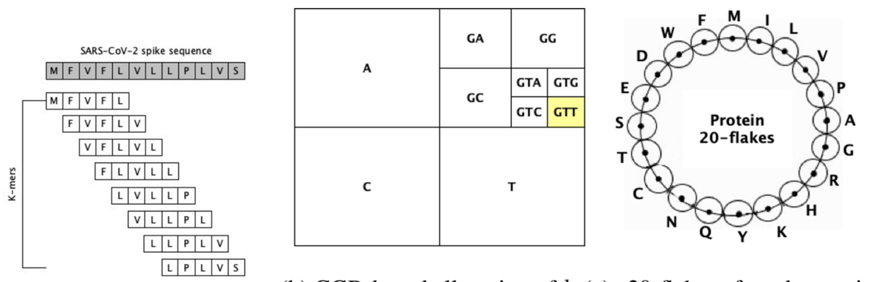
Our manuscript is organized as follows: Sect. 2 contains previous work related to our method. Our proposed model is introduced in Sect. 3. Details related to the experimental setup and dataset statistics are given in Sect. 4. The proposed method's results and comparison with the SOTA methods are reported in Sect. 5. Finally, we conclude the paper in Sect. 6.

2 Related work

Various efforts are put into transforming protein sequences into machine-readable forms. Many such encoding schemes exist, broadly categorized into three groups: sequence-based encoding, structural-based encoding, and alternative encoding. A comprehensive review of these encodings is given in Spänig and Heider (2019).

Some of the sequence-based encoding (e.g., feature engineering-based methods) are Sparse (Hirst and Sternberg, 1992), Amino Acid Composition (Matsuda et al., 2005), n-gram based vector representation (Ali et al., 2021, 2021), and Physicochemical Properties (Deber et al., 2001). In Sparse, a one-hot binary vector (length 20) is used to represent each amino acid of the protein sequence. But it is an inefficient and redundant representation due to being high-dimensional and very sparse. Amino Acid Composition presents another protein representation scheme by considering the amino acid's local compositions and twin amino acids. However, it lacks taking the sequence order into account. Physicochemical Properties consider the physicochemical properties of the molecular components of the amino acids and use them as features to predict the protein structure and function. However, the unknown physicochemical properties related to protein folding hinder the path of formation of effective physicochemical properties encoding. Overall, feature engineering-based methods are domain-specific and may not be able to generalize to different types of data.

The structural-based encoding comprises Quantitative Structure-Activity Relationship (QSAR) (Cherkasov, 2014), and General Structure (Cui et al., 2008) etc. QSAR uses the chemical properties to describe the amino acids of the sequence but unlike the physicochemical properties method, which encodes the whole residue, QSAR only focuses on the molecules. However, it is prone to biological data experimentation errors causing false correlations. General Structure encoding deals with mapping the structural information (like residue depth, 3D shape, secondary structure, etc.) of the protein



(b) CGR-based allocation of k-(c) 20-flakes for the amino
(a) 5-mers for a spike sequencemer in an image acids of a protein seq.

Fig. 2 Example of (a) k-mers, (b) determination of the location (in yellow) of the 3-mer "GTT" in the image using CGR method, and (c) 20-flakes image based on Chaos/FCGR method

sequence to a numerical form. But it has limited performance due to the limited number of known protein structures.

The alternative encoding includes Chaos Game Representation (CGR) (Jeffrey, 1990). CGR focused on the visual encoding of sequences based on generating fractals. Although CGR builds a visual representation of a given sequence, it was originally designed for DNA (nucleotide) sequences (Jeffrey, 1990; Hoang et al., 2016; Rizzo et al., 2016). Authors of Löchel et al. (2020) proposed modifications to CGR known as FCGR (Frequency Chaos Game Representation) to handle protein sequences. However, they consider the amino acids of a sequence one by one (rather than considering substrings using n-gram (Ali and Patterson, 2021, Ali et al., 2021) and assign equal weight to every amino acid (i.e., every pixel in the image corresponding to an amino acid will have a value 1 rather than weights based on their positions in the sequence) in their respective graphical representation. Moreover, the existing CGR-based approaches operate on k-mer (for $k > 1$) for nucleotide only and only 1-mers in the case of protein sequences (also known as FCGR). However, better underlying sequence representations, such as Minimizers, can be used rather than k-mers. Similarly, using $k > 1$ for FCGR could also produce better results.

3 Proposed approach

This section discusses the proposed approaches to generate images from SARS-CoV-2 spike sequences. The images are further used for coronavirus variant and host classifications using DL models.

A popular approach used for encoding biological sequences into images is Chaos Game Representation (CGR) (Barnsley, 2012; Jeffrey, 1990). It is a technique to transform sequences into graphical form. It starts by computing k-mers (also called n-gram) of a given sequence.

Definition 1 (k-mers) A k-mer is a consecutive sub-string of length k extracted from a sequence (see Fig. 2(a)). For any sequence of length N, the total number of k-mers are $N - k + 1$.

After the k-mers computation, for each k-mer, its respective nucleotides are utilized to allocate a position to the k-mer in the image. For example, in the case of a DNA genome sequence, an empty image is divided into 4 quadrants, each one representing a unique nucleotide, i.e., A “upper left", C “lower left", G “upper right", and T “lower right". Each of these 4 quadrants is further subdivided, recursively, up until the length k of the k-mer. Based on the nucleotides of a given k-mer, the appropriate position of the k-mer in the image is then detected in this recursive manner, the respective pixel value incremented by 1. This process is visually shown in Fig. 2(b).

Although the above-mentioned method works perfectly fine for DNA (genome) sequences (with the number of unique nucleotides $n \leq 4$), it poses the challenge of overlapping in an image for $n > 4$, e.g., protein sequences with twenty proteinogenic amino acids. Therefore to eliminate the overlapping, authors in Löchel et al. (2020) proposed a frequency matrix-based CGR, known as FCGR (for reference, we call this method “Chaos" in the rest of the paper). The Chaos produces an n-flakes (also referred to as poly-flake) Tzanov (2015) based graphical representation (where n is the number of amino acids), which consists of an image with multiple icosagons (see Fig. 2(c) for an example).

Definition 2 (Icosagons) In geometry, an icosagon or 20-gon is a twenty-sided polygon (see Fig. 2(c)).

An n-flake or poly-flake follows an iterative mechanism to construct a fractal starting from an n-gon. For example, given a spike sequence with 20 amino acids, the Chaos graphical representation generates twenty edges (one for each amino acid) and then generates twenty inner icosagons within the larger icosagon (see Fig. 2(c)).

Given a sequence, the fractal is then generated via an iterative process, starting at the center of the image. Each subsequent step is governed by the following process for determining the location of the next pixel in the image based on the previous: First, we calculate the contraction ratio r between the outer and inner polygon. For this purpose, we use the following expression (as proposed in Löchel et al. (2020); Strichartz (2000)):

$$r = \frac{sin\left(\frac{\pi}{n}\right)}{sin\left(\frac{\pi}{n}\right) + sin\left(\frac{\pi}{n} + \frac{2\pi m}{n}\right)}, \text{for m} = \left\lfloor \frac{n}{4} \right\rfloor \tag{1}$$

where $n = 20$ for twenty amino acids in the protein/spike sequence. Then we define a scaling factor (SF), which is the ratio of the distance between the current location and the target edge (amino acid). The SF is computed using the following expression (as proposed in Löchel et al. (2020)):

$$\text{sf} = 1 - r \tag{2}$$

Finally, we calculate the coordinates x and y of this next pixel using the following expression (as proposed in Löchel et al. (2020)).

$$x[i] = r \cdot sin\left(\frac{2\pi i}{n} + \theta\right) \tag{3}$$

$$y[i] = r \cdot cos\left(\frac{2\pi i}{n} + \theta\right) \tag{4}$$

where $i \in \{1, 2, \dots, |S|\}$ (where S is a spike sequence and $s[i] \in S$ corresponds to a single amino acid), n is the total number of amino acids, and θ is the angle of orientation. The Chaos method takes a protein sequence as input and yields an image as output by considering the amino acids of the sequence one by one, following the Eqs. 3 and 4 to get the coordinates of every amino acid in the image. See Fig. 7(a) for an image generated using the Chaos method.

Remark 1 The FCGR generates a greyscale image of a protein sequence based on the coordinates of each amino acid in the spike sequence computed using Eqs. 3 and 4.

Remark 2 Note that the FCGR allocates pixel value 1 for each amino acid in the spike sequence for which x-y coordinates within the image are computed using Eqs. 3 and 4.

In summary, the Chaos method iteration is as follows, starting from point (0,0):

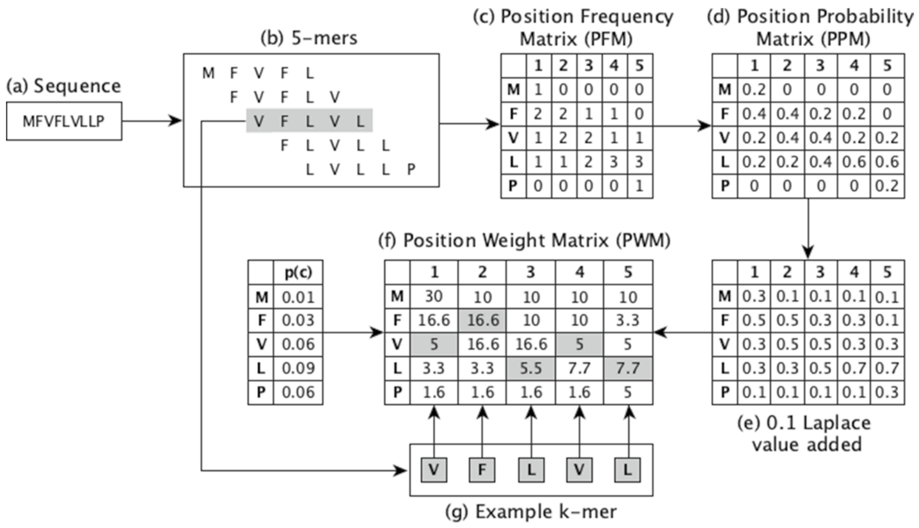
1. Check the next character (amino acid) of the sequence.
2. Go a fraction of the way to the corresponding amino acid (base), according to the scaling factor.
3. Save coordinates x, y of this next point (and draw a pixel at those coordinates) and repeat (from bullet point 1.).

3.1 Spike2Vec (Ali and Patterson 2021)

For a given sequence, the original FCGR works by doing one-to-one mapping of the amino acids to their respective pixels, which means that each amino acid will be represented by a single pixel in the corresponding image. Since the original FCGR includes a single pixel value for each amino acid, it may not represent the spike sequence very effectively. For this purpose, we use a recently proposed method called Spike2Vec (Ali and Patterson, 2021). The Spike2Vec uses the idea of k-mers to generate the feature embeddings. In this paper, we use the same k-mers idea to generate the images from spike sequences by considering a set of amino acids (rather than single amino acid at a time as done by the Chaos approach) with a sliding window (of increment 1) to draw pixels within images. After generating all the k-mers for a given sequence, we concatenate them to make a single (new) sequence, which is used as input to the Chaos method for image generation (using Eqs. 3 and 4). In our experiments, the images generated for Spike2Vec use 9-mers (selected using standard validation set approach (Devijver and Kittler, 1982)). Spike2Vec-based image encoding has enabled the mapping of each amino acid to multiple pixels in the corresponding generated image due to the usage of k-mers, therefore it can capture more information about the sequence as compared to the Chaos method.

3.2 PWM2Vec (Ali et al. 2022)

Both Chaos and Spike2Vec assign an equal weight of 1 to each amino acid. However, this uniform value may not be the most effective way to come up with a graphical representation of a given sequence. To assign weight to each amino acid within k-mers, we use a recently proposed method, called PWM2Vec (Ali et al., 2022). The PWM2Vec technique is also driven by the notion of k-mers, however, instead of using a constant frequency value, it uses weighted values computed from the Position Weight Matrix (PWM) corresponding


(a) Sequence
MFVFLVLLP
(b) 5-mers
M F V F L
F V F L V
V F L V L
F L V L L
L V L L P
(c) Position Frequency Matrix (PFM)
(d) Position Probability Matrix (PPM)
(e) 0.1 Laplace value added
(f) Position Weight Matrix (PWM)
p(c)
(g) Example k-mer
V F L V L

	1	2	3	4	5
M	1	0	0	0	0
F	2	2	1	1	0
V	1	2	2	1	1
L	1	1	2	3	3
P	0	0	0	0	1

	1	2	3	4	5
M	0.2	0	0	0	0
F	0.4	0.4	0.2	0.2	0
V	0.2	0.4	0.4	0.2	0.2
L	0.2	0.2	0.4	0.6	0.6
P	0	0	0	0	0.2

	1	2	3	4	5
M	0.3	0.1	0.1	0.1	0.1
F	0.5	0.5	0.3	0.3	0.1
V	0.3	0.5	0.5	0.3	0.3
L	0.3	0.3	0.5	0.7	0.7
P	0.1	0.1	0.1	0.1	0.3

	p(c)
M	0.01
F	0.03
V	0.06
L	0.09
P	0.06

	1	2	3	4	5
M	30	10	10	10	10
F	16.6	16.6	10	10	3.3
V	5	16.6	16.6	5	5
L	3.3	3.3	5.5	7.7	7.7
P	1.6	1.6	1.6	1.6	5

Fig. 3 Workflow of PWM for a given sequence

to alphabets in a sequence. Like Spike2Vec, PWM2Vec enables capturing of locality information due to *k*-mers usage, but it also considers the importance of relative positions of amino acids in the sequence.

Definition 3 (Position Weight Matrix (PWM)) It is a representation method for motifs (patterns) in biological sequences. A PWM comprises information about the count of amino acids for each position in the form of weights (log-likelihood).

Figure 3 illustrates the steps of calculating PWM2Vec (as described in Ali et al. (2022)) for a given sequence based on 5-mers. The steps from (a) to (f) are followed to get a PWM for any sequence, and the steps are as follows. Step (a) shows the input spike sequence and (b) deals with the extraction of *k*-mers from the sequence. Further, a position frequency matrix (PFM) is created based on these *k*-mers by counting the frequencies of each character with respect to their positions in the *k*-mers in step (c). PFM is converted into a position probability matrix (PPM) in (d) by computing column-wise probabilities using the formula:

$$PPM = \frac{\text{alphabet count in the column}}{\text{column sum}} \tag{5}$$

A Laplace value (0.1) is added to every element of PPM in (e) to ensure they are always non-zero. Lastly, in step (f) a PWM is created by calculating the log-likelihood of each alphabet $c \in \Sigma$ at a position i with the following formula:

$$PWM_{c,i} = \log_2 \frac{p(c,i)}{p(c)} \tag{6}$$

where $p(c) = \frac{n(c)}{61}$ and $n(c)$ is the number of codons (as described in Ali et al. (2022)) for each amino acid $c \in \Sigma$ and 61 is the number of sense codons. After assigning weight to

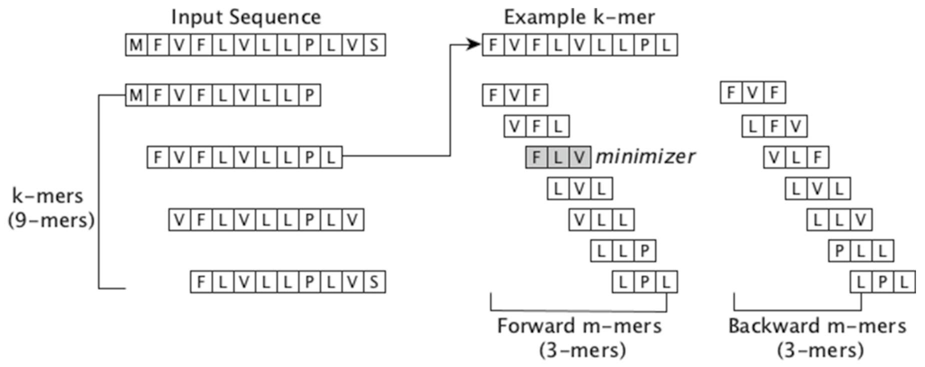


Fig. 4 Workflow of Minimizer. Firstly, the *k*-mers (9-mers in this case) are extracted from the sequence. Then for every *k*-mer, its corresponding minimizer is computed by finding the lexicographically smallest one among the forward and backward *m*-mers (3-mers in this case)

each amino acid within the *k*-mers of a sequence using PWM2Vec, we use those weights as corresponding pixel values (rather than assigning the pixel value 1 to each amino acid in the sequence). Similar to Spike2Vec, the procedure of PWM2Vec image encoding for a given spike sequence computes *k*-mers and starts off with a blank image but it includes an additional step of calculating the PWM of the sequence and updating the pixel values corresponding to the amino acids of a *k*-mer based on PWM values rather than a constant value 1. Now for each *k*-mer, once the respective coordinates of every amino acid are determined in the image using Eq. 3 and Eq. 4, the pixel values representing each amino acid are updated by adding the respective PWM value. Yet again, the task of coordinates finding and pixel value update is repeated for all *k*-mers of the given sequence and it results in image generation.

3.3 Minimizer

In this technique, rather than utilizing k-mers for image construction, we use another popular approach in bioinformatics, called minimizers (Roberts et al., 2004).

Definition 4 (Minimizer) For a given *k*-mer, a minimizer (also called m-mer) is a substring of consecutive characters (amino acids) of length m from the *k*-mer, which is lexicographically smallest one in both forward and backward order of the *k*-mer, where $m < k$ and is fixed.

One of the main problems with *k*-mers is that they introduce redundancy (repeated *k*-mers) in long sequences and hence increase the computation and storage cost. This redundancy could be eliminated using minimizers. The pseudocode of calculating m-mers for any given sequence is shown in Algorithm 1. Its workflow is given in Fig. 4 and it illustrates that for a given sequence the *k*-mers of the sequence are extracted (9-mers in the figure example). Then for each *k*-mer, a minimizer (3-mer in figure example) is computed by getting the lexicographically smallest one from both forward and backward *m*-mers of the *k*-mer. The minimizer takes two parameters (k, m). k is the length of *k*-mer (where $k = 9$ in our case) while m indicates the size of *m*-mer (where $m = 3$ in our case). Given a spike sequence, it extracts k-mers of the sequence. Then for each k-mer, it computes a corresponding m-mer (minimizer). After computing minimizers, we concatenate all *m*-mers to make a new sequence and use it as input to the Chaos method for image generation (by computing x and y coordinates of each amino acid in this new sequence using Eqs. 3 and 4 and increment the corresponding pixel values of the amino acids by 1).

Algorithm 1 Minimizer Computation

```
1:  Input: Sequence $seq$ and integer $kSize$ and $mSize$
2:  Output: Set of Minimizers
3:  minimizersList = $\emptyset$
4:  q = []                                          ▷ maintain queue of all m-mers
5:  index = 0                                       ▷ index of the current minimizer
6:  for $i \leftarrow 1$ to seq $- kSize + 1$ do
7:      kmer = $seq[i : i + kSize]$
8:      if index $> 1$ then
9:          q.dequeue
10:         mmer = $seq[i + kSize - mSize : i + kSize]$      ▷ new m-mer
11:         index $\leftarrow$ index $-1$                   ▷ shift index of current minimizer
12:         mmer = min(mmer, reverse(mmer))                 ▷
    lexicographically smallest forw./rever.
13:         q.enqueue(mmer)
14:         if mmer $<$ q[index] then
15:             index = $kSize - mSize$             ▷ update minimizer with new m-mer
16:         end if
17:     else
18:         q = []                                  ▷ reset the queue
19:         index = 0
20:         for $j \leftarrow 1$ to $kSize - mSize + 1$ do
21:             mmer = kmer$[j : j + mSize]$        ▷ compute each m-mer
22:             mmer = min(mmer, reverse(mmer))
23:             q.enqueue(mmer)
24:             if mmer $<$ q[index] then
25:                 index = $j$                     ▷ index of current minimizer
26:             end if
27:         end for
28:     end if
29:     minimizersList $\leftarrow$ minimizersList $\cup$ q[index]     ▷ add current minimizer
30: end for
31: return(minimizersList)
```

Remark 3 Note that methods like Spike2Vec and PWM2Vec are originally designed to generate numerical vectors. We transform those methods to generate the 2-D visual representations so that we can apply image classification models for supervised analysis.

3.4 Spike2CGR

The main idea of Spike2CGR is the same as the minimizer computation as given in Algorithm 1. The main difference in the case of Spike2CGR is the preservation of the order of amino acids in m-mers (minimizers). More formally, given a sequence s, we first compute the minimizers using Algorithm 1. Now, we combine all minimizers to make a single sequence s' (see Fig. 5). Note that s' will be a different sequence from s as it only contains the amino acids within the minimizers. We then repeat the process of computing k-mers (similar to Spike2Vec) from s'. This will give us a list of k-mers computed from s'. We then concatenate all those k-mers to make a new sequence s''. This new sequence s'' is then used as input to the chaos method for image generation (by computing the x and y coordinates of each amino acid in s'' using Eqs. 3 and 4 and increment the corresponding pixel values of the amino acids by 1).

We propose an image encoding technique, named Spike2CGR, to transform the protein sequences into images. Although it follows the Chaos procedure to create the images, it manipulates the sequences in a (biologically meaningful) way that the corresponding generated images can capture more information about the sequences which can improve the generated image-based classification predictive performance. Our suggested sequence manipulation is a novel technique and it has been shown to outperform the Chaos method in the experiments indicating that Spike2CGR can capture more relevant information about a sequence in its respective generated image in terms of classification performance. Similarly, another novel sequence manipulation strategy put forward, by us to get better images, is the Minimizer-based encoding method. Moreover, we also investigated some of the popular numerical embedding generation methods (Spike2Vec, PWM2Vec) for protein sequences to create images. As these approaches are originally proposed to compute numerical feature vectors for protein sequences, but we combine them with the Chaos concept to create images of the protein sequences instead of numerical embeddings. We aim to explore the domain of image-based classification for protein sequences, so coming up with effective and efficient mapping strategies from protein sequences to images is the first essential step in that regard.

3.5 Deep learning models architectures

We have used various DL models to perform the classification of our datasets. We trained 4 different convolutional neural networks (CNN) models, with a varying number of hidden blocks, from scratch to view the impact of an increasing number of layers on the classification accuracy. The basic block is named Block A and it consists of a 2D Convolution (Conv2D) layer followed by a ReLu and MaxPooling (Max-Pool) layer (see Fig. 6). In each of these models, the final Block A module is followed by 2 fully connected (FC) layers and a softmax classification layer. In the experiments, these 4 models are referred to as 1-layer CNN, 2-layer CNN, 3-layer CNN, and 4-layer CNN models depending on the number of "Block A" modules used respectively. The architecture of the 4-layer CNN model is illustrated in Fig. 6. It consists of 4 Block A modules with two FC layers and a softmax classifier to perform the classification of K classes. For all datasets, the input image size is 480x480, and FC1 outputs 500 features while FC2 gives K out features. Moreover, the impact of transfer learning is also investigated by using pre-trained RESNET-50 (He et al., 2016) and VGG19 (Simonyan and Zisserman, 2015) models, as they are considered state-of-art models for image classification.

For a given SARS-CoV-2 spike sequence encoded image as input, the given model will yield either the variant or the host of the respective sequence as output depending on the classification task. The overview of images from the dataset against their respective encoding method is shown in Fig. 7. The images are marked with red boxes to indicate some of the areas of interest (major changes) among them. The differences among these images are subtle, therefore they are not easily identifiable by the naked eye. For instance, the number of pixels in each image is different. Similarly, the orientation of the circles also differs in some cases. The differences can be spotted by looking closely at the pixels (dots) in the area of interest regions (highlighted using red boxes). As we are performing multi-class classification tasks, we use the negative log-likelihood (Yao et al., 2019) (NLL) loss function to train the DL models. NLL is also known as the cross-entropy loss function for multi-class problems (Bishop and Nasrabadi, 2006). Its formula for k classes, y target output, and a^L predicted output is following:

$$C_{LL} = -\frac{1}{n}\Sigma_x \Sigma_{k=1}^{K} y_k ln(a_k^L) \tag{7}$$

Furthermore, two tabular CNN models are used to perform classifications of the feature vector-based data gained from the OHE and WDGRL baselines. These models take tabular data as input and are referred to as 3-layer Tab CNN and 4-layer Tab CNN in experiments due to having 3 and 4 layers respectively. Each hidden layer in both models consists of a linear operation followed by batch normalization and the ReLu function. The models also utilized a dropout operation with a 0.2 parameter value before the decision layer. The decision layer is a linear operation that maps the output from previous layers to the variant/host class value. The NLL loss function is used for training these models, as we are dealing with multi-class problems.

Note that we also computed results using some of the classical ML classifiers (including SVM, RF, and XGBoost) and they were not very encouraging, even as a baseline. Hence we opted to only work with deep learning models. We believe that utilizing deep learning models instead of standard machine learning methods was a justified choice for our research. With the complex nature of the tabular baseline representations, the need to capture non-linear relationships, and the desire to leverage the expressiveness and adaptability of deep learning models were the driving factors behind this decision.

4 Experimental setup

In this section, we discuss the details related to experimentation and dataset statistics. All experiments are conducted using a server having Intel(R) Xeon(R) CPU E7-4850 v4 @ 2.40GHz with Ubuntu 64 bit OS (16.04.7 LTS Xenial Xerus) having 3023 GB memory.

The image-based DL models are trained using 64 as batch size, ADAM optimizer, 10 epochs, and 0.003 learning rate. The input of these models is a spike image of 480x480 size. The tuning of hyperparameters is done with cross-validation using the validation dataset. The DL models are developed using the PyTorch framework. For a model, we used almost 80% of data for training and almost 20% for testing. Our Code and data are available online .[1] For encoding sequences to images, the image resolution used is 100 and the

[1] https://github.com/sarwanpasha/Spike2CGR/tree/main

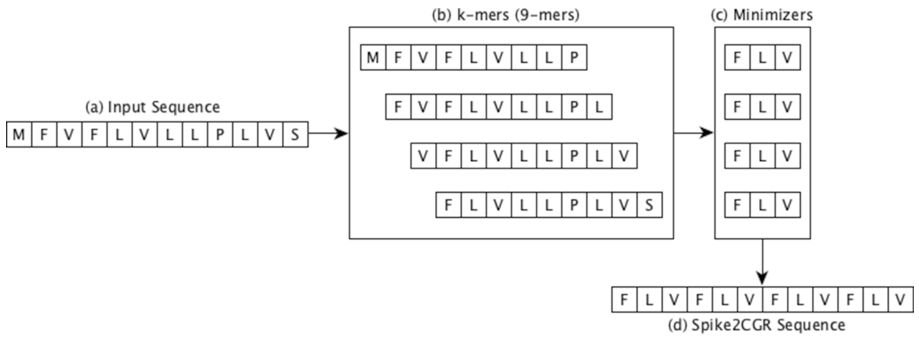


Fig. 5 Workflow of Spike2CGR for a given sequence. For a given spike sequence, steps from (a) to (d) are followed to generate the corresponding Spike2CGR sequence

conversion of sequences to images is done in RStudio. For Spike2Vec- and PWM2Vec-based encodings, we use $k = 9$ (for k-mers) to create the images. Similarly, for minimizer and Spike2CGR, we use $k = 9$ and $m = 3$ (for k-mers and m-mers).

The tabular DL models (based on data from OHE and WDGRL) are also trained for 64 batch size, ADAM optimizer, 10 epochs, and 0.003 learning rate after tuning the hyperparameters using a cross-validation technique. A typical input for these models is a vector of 27817 in size. The feature vectors generated by OHE are preprocessed to make them aligned by using the zero padding technique to make them compatible with the CNN models. WDGRL method also receives the feature vectors generated by OHE as input and transforms them into low-dimensional vectors. In our experiments, it transforms a 27817 size feature vector into a 10 dimensional vector.

To evaluate the performance of the DL models, we report average accuracy, precision, recall, F1 (weighted), F1 (macro), ROC-AUC, and training runtime. To use binary classification-based evaluation metrics for multi-class classification, we use the one-vs-rest approach. The performance (improvement in %) gain by Spike2CGR for all evaluation metrics as compared to the Chaos method is also reported. This improvement is computed by multiplying the Spike2CGR and Chaos values difference with 100 for all the evaluation metrics except train time (because of the known range of percentage). The train time improvement is calculated using the formula:

$$\%improvement = \frac{Spike2CGRValue - ChaosValue}{ChaosValue} * 100 \tag{8}$$

Remark 4 Note that we use the original Chaos method (Löchel et al., 2020) as the SOTA approach.

4.1 Dataset statistics

We used two datasets (SARS-CoV-2 and coronavirus host) to perform the evaluation.

SARS-CoV-2 dataset: This dataset consists of 32, 738 spike sequences, having 13 unique coronavirus variants, extracted from GISAID (2021) database (in January 2022). Originally, we had extracted almost 4 million sequences but we exclude the ones with

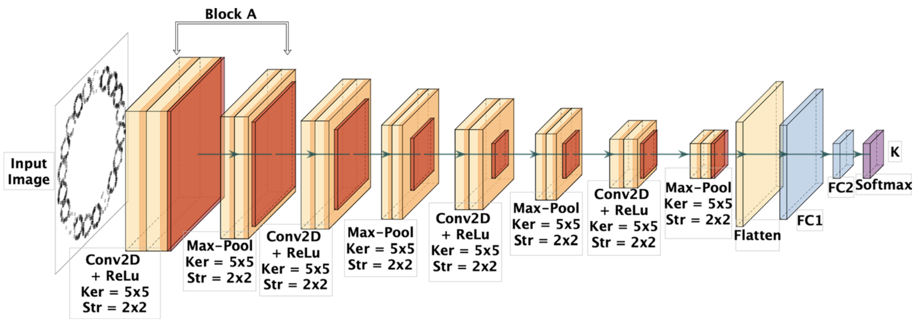

Block A
Input Image
Conv2D + ReLu Ker = 5x5 Str = 2x2
Max-Pool Ker = 5x5 Str = 2x2
Conv2D + ReLu Ker = 5x5 Str = 2x2
Max-Pool Ker = 5x5 Str = 2x2
Conv2D + ReLu Ker = 5x5 Str = 2x2
Max-Pool Ker = 5x5 Str = 2x2
Conv2D + ReLu Ker = 5x5 Str = 2x2
Max-Pool Ker = 5x5 Str = 2x2
Flatten
FC1
FC2
Softmax
K

Fig. 6 The architectures of the 4-layer CNN model, which is used to classify 'K' classes. Here ker represents kernel and str represents stride filter size

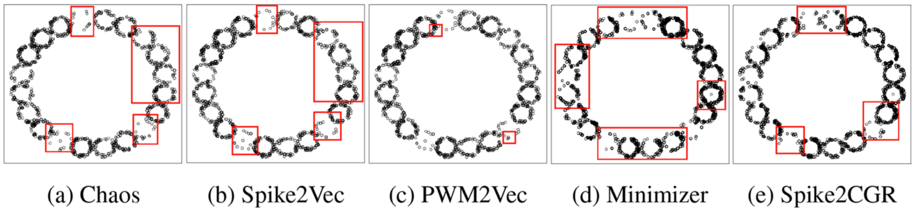


Fig. 7 Graphical representation of a spike sequence of B.1.351 variant (from SARS-CoV-2 dataset) using different methods. Some of the major changes in the images (area of interest) are highlighted using the red boxes

missing lineage (variant) information. We also filter the variants corresponding to a small number of sequences. Since we did not have computing resources to conduct experiments on all 4 million sequences, we use stratified sampling to get a subset of data (approximately 1% of the data) while preserving the distribution of lineages from the original data. We then selected the top 13 lineages that had the most number of sequences available, which ends up in getting 32738 sequences. Moreover, the processed data does not involve any filters regarding geographical locations and is only based on lineage information. Furthermore, the quality of the sequences is ensured by GISAID by accepting only the consensus sequences from high-coverage (200/300X) Illumina sequencing, which will have less than 0.01% errors — hence we relied on this for quality. Additionally, due to the occurrence of the majority of the mutations regarding coronavirus in its spike protein region (Kuzmin, 2020), we employ only this region to perform our analysis rather than using the full-length genome. In this way, we can compute higher predictive performance in less computational time as spike proteins are quite compact and contain many variations compared to other parts of the DNA. Moreover, analyzing full-length sequences could result in increasing computational time while not improving predictive performance significantly. The variants information and their training, validation, and testing set distributions used to perform classification are given in Table 1. The first column corresponds to the variant name (Pangolin, 2022) which is used as a class label for the variant classification task. The second column represents the regions where the variants were discovered initially. The third column has the label for the coronavirus variant, which is assigned by the world health organization. The fourth column contains the mutation count for the variant in spike region (S) versus the full-length genome sequence. The last three columns contain the frequency of each variant in the training, validation, and testing set, respectively.

Coronavirus Host dataset: This dataset contains spike sequences from different clades of the Coronaviridae family. These sequences are accompanied by the respective infected host information. This data has 5558 total sequences extracted from GISAID (2021) and ViPR (Pickett et al., 2012; Ali et al., 2022) having the information about 21 unique hosts (Ali et al., 2022). The training, validation, and testing set distribution of host data is shown in Table 2. For performing host classification, we use the respective host's name (host column from Table 2) as a class label for a given spike sequence as input.

4.2 Feature engineering baselines

Apart from the Chaos method (Löchel et al., 2020) (that is used as the SOTA approach for comparison), we use two numerical feature vector-based sequence embedding generation methods as baselines. The detail of both baselines is given below.

4.2.1 One-hot encoding (OHE) (Kuzmin, 2020)

OHE is an algorithm to generate a numerical representation of a sequence. A binary feature vector is created against each character of the sequence, and this set of binary vectors is concatenated to represent the entire sequence. Although it is a simple and straightforward method, the resultant vectors are very sparse and suffer from the curse of dimensionality issues.

4.2.2 Wasserstein distance guided representation learning (WDGRL) (Shen et al., 2018)

WDGRL, being an unsupervised domain adoption approach, transforms high-dimensional vectors into low-dimensional ones. This technique determines the Wasserstein distance (WD) with the help of the source and target encoded distributions to get the input data features by utilizing neural networks. Its goal is the optimization of the feature extractor network and the minimization of the estimated WD for getting the representation. WDGRL operates on the feature vectors generated by OHE. Since WDGRL employs a neural network, its need for training data is an expensive requirement.

5 Results and discussion

In this section, we report the variant and host classification results for different embeddings and DL-based methods using multiple evaluation metrics.

5.1 SARS-CoV-2 dataset results

The variant classification results for the SARS-CoV-2 dataset are reported in Table 3. We can observe that the CNN model with 4 layers (using Spike2CGR-based embedding) outperforms all other methods in terms of average accuracy and recall (best values are shown in bold). For the F1 (macro) score and precision, the 4-layer CNN with PWM2Vec-based embedding outperforms all other embeddings and DL models. However, for ROC-AUC, the 3-layer CNN model gives the best performance by using minimizer-based embedding. Moreover, the minimum training runtime is gained by the WDGRL baseline against the 3-layer Tab CNN model. These results illustrate that the performance of the spike to image-based models is higher than the feature vector-based (baselines) models (OHE and WDGRL), hence indicating that the graphical representation retains more meaningful information about the input (spike) sequence as compared to the baseline numerical representations for doing classification. We have also reported the performance improvement of the Spike2CGR method as compared to the SOTA Chaos method, and the results show that Spike2CGR has achieved up to 16.7% improvement in terms of ROC-AUC compared with the Chaos method. Overall, Spike2CGR outperforms the Chaos method in most of the evaluation metrics and DL approaches. An important point to note here is that the pre-trained models (RESNET50 and VGG19) are not performing better than the customized CNN models. A possible reason for this behavior is that those models are originally trained on different types of images that have different scales, backgrounds, and foreground information. Hence they fail to generalize on the Chaos-based images.

Remark 5 Note that although the performance improvement (for SARS-CoV-2 data) is not very high in some cases compared to Chaos, Spike2CGR can assist doctors, biologists, and relevant government authorities better in taking efficient decisions to minimize the effect and spreading of the coronavirus. Since human health is the most important factor when we talk about fighting a pandemic, a small improvement in the predictive model can help in taking impactful timely decisions.

5.2 Host dataset results

The host classification results are shown in Table 4. Unlike variant classification, it portrays the best performance for the 1 layer CNN model, and it is because the dataset is small. Hence increasing the model's layers could lead to over-fitting. For the 1 layer CNN model, the Spike2CGR-based embedding illustrates the best performance in terms of precision, F1 macro, and ROC AUC scores compared to other methods. However, the Minimizer-based embedding has maximum accuracy, recall, and F1 weighted scores but the difference between Minimizer and Spike2CGR for these metrics is small. We can also observe that the image-based host classification outperforms the feature vector-based methods (OHE and WDGRL). Furthermore, similar to lineage classification, the pre-trained models (RESNET50 and VGG19) show bad performance for the host classification task. The performance improvement of Spike2CGR compared to Chaos yet again illustrates the performance gain by our method as Spike2CGR gains up to 7.2% improvement in accuracy compared to the Chaos method.

5.3 Confusion matrices

We also investigated the confusion matrices of the best-performing model (4-layer CNN) for Chaos (the SOTA approach (Löchel et al., 2020)) and Spike2CGR-based embedding (see Fig. 8) for variant classification. We can observe that although for the Alpha variant (B.1.1.7), Chaos-based embedding has the highest true positive count. However, for all other classes, Spike2CGR-based embedding performs better. This behavior shows that

Table 1 Dataset statistics for different coronavirus variants (32738 in total)

Lineage	Region	Labels	No. Mut. S/Gen.	No. of sequences		
				Training	Validation	Testing
B.1.1.7	UK (Galloway, 2021)	Alpha	8/17	9930	2527	3146
B.1.617.2	India (Yadav et al., 2021)	Delta	8/17	1877	450	456
P.2	Brazil (WHO, 2021)	Zeta	3/7	1780	432	533
B.1.429	California	Epsilon	3/5	1079	256	326
P.1	Brazil (Naveca et al., 2021)	Gamma	10/21	994	245	306
B.1.526	New York (West et al., 2021)	Iota	6/16	847	219	255
B.1.351	South Africa (Galloway, 2021)	Beta	9/21	837	221	258
B.1.427	California (Zhang, 2021)	Epsilon	3/5	835	218	268
B.1.1.529	South Africa	Omicron	34/53	747	178	253
C.37	Peru (WHO, 2021)	Lambda	8/21	732	169	228
B.1.621	Colombia (WHO, 2021)	Mu	9/21	717	168	219
B.1.525	UK and Nigeria	Eta	8/16	714	187	224
P.3	Philippines (WHO, 2021)	Theta	8/17	111	30	34
Total	_	_	_	21,200	5300	6238

Chaos tends to focus more on the label with high frequency in the data (a typical class imbalance problem) since the Alpha variant has the highest count in the dataset as given in Table 1. As the lineages B.1.429 and B.1.427 belong to the Epsilon category and Chaos misclassifies 84 instances of B.1.429 as B.1.427 while Spike2CGR misclassifies only 2 such instances (note that the converse scenario does not happen for either method). This indicates that Spike2CGR is able to distinguish between two similar lineages more accurately than Chaos.

On the other hand, the confusion matrices for host classification using Chaos and Spike2CGR embeddings for the best performing model (1 layer CNN) are given in Fig. 9. We can observe that Spike2CGR is performing better than Chaos for most of the labels, even for the most frequent ones.

5.4 Comparison of different embeddings

After the analysis of classification results, a natural question arises why results for different embedding methods are not similar? A few fundamental facts for this behavior are the following:

1. The i-th amino acid (pixel) drawn using the CGR for a given sequence corresponds to a specific position in the spike sequence (and it holds some local meaning). Therefore, no other sub-sequence within that spike sequence may have similar information/pattern (up to the resolution of the screen). Hence, there is a one-to-one mapping between the sub-sequence patterns of a given spike sequence and pixels of the CGR. If there is a mutation in the sequence, that mutation should be highlighted clearly in the resultant visual representation using CGR.
2. If we manipulate the sequence (in a biologically meaningful way) using different embedding tricks such as minimizers, the amino acid positions should be disturbed and will no

Table 2 Dataset statistics for different coronavirus infected hosts (5558 in total)

Host	No. of sequences		
	Training	Validation	Testing
Bat	96	23	34
Bird	242	61	71
Bovine	55	12	21
Camel	186	45	66
Canis	26	5	9
Cat	72	18	33
Cattle	1	0	0
Dolphin	4	1	2
Environment	682	162	190
Equine	3	0	2
Fish	1	0	1
Hedgehog	10	3	2
Human	1159	292	362
Monkey	1	0	1
Pangolin	13	2	6
Python	1	0	1
Rat	22	2	2
Swine	344	104	110
Turtle	1	0	0
Unknown	1	0	1
Weasel	629	160	205
Total	3549	890	1119

longer remain similar to the original spike sequence. This disturbance of the sequence may affect the performance of classifiers positively or negatively.

3. Because of this manipulation of the protein sequence, the "*area of interest*" within the CGR image could be different for different embeddings.
4. Assigning a certain weight to the pixels (as done using PWM2Vec) is also an important factor for classification as it could affect the resolution of the image.

In our study, we have extensively evaluated the performance of our Spike2CGR approach using different deep-learning models and compared it with state-of-the-art methods for virus lineage and host classification. Our results demonstrate that Spike2CGR outperforms the existing methods in terms of predictive performance, even when using only spike sequences instead of the full-length genome. Therefore, we believe that our approach can be a valuable addition to the existing methods for analyzing and classifying SARS-CoV-2 sequences, especially in cases where deep learning algorithms can reveal hidden patterns that other methods may miss. Moreover, our aim with the use of embeddings is to leverage the power of deep learning algorithms that can learn abstract representations of the sequence data, which can capture important features that may not be apparent from the raw sequence itself. We believe that this approach can provide complementary information to other methods such as phylogenetics and epidemiological studies.

Table 3 Classification results for different models and algorithms for variant classification using **SARS-CoV-2 dataset**. The ↑ and ↓ mean higher and lower values are better, respectively

DL Model	Method	Acc. ↑	Prec. ↑	Recall ↑	F1 (Weig.) ↑	F1 (Macro) ↑	ROC AUC ↑	Train Time (hrs.) ↓
3-Layer Tab CNN	OHE (Kuzmin, 2020)	0.472	0.301	0.472	0.368	0.060	0.552	0.594
	WDGRL (Shen et al., 2018)	0.636	0.457	0.636	0.523	0.263	0.594	**0.380**
4-Layer Tab CNN	OHE (Kuzmin, 2020)	0.637	0.469	0.637	0.528	0.157	0.511	0.977
	WDGRL (Shen et al., 2018)	0.688	0.517	0.688	0.582	0.227	0.637	0.866
1-Layer CNN	Chaos (Löchel et al., 2020)	0.700	0.680	0.696	0.651	0.563	0.673	8.195
	Spike2Vec (Ali and Patterson, 2021)	0.733	0.690	0.733	0.679	0.679	0.850	7.779
	PWM2Vec (Ali et al., 2022)	0.734	0.676	0.734	0.691	0.697	0.844	5.744
	Minimizer	0.743	0.707	0.743	0.709	0.709	0.832	6.171
	Spike2CGR	0.719	0.730	0.766	**0.739**	0.717	0.840	4.992
% improv. of Spike2CGR from SOTA Chaos (Löchel et al., 2020)		1.9	5	7	8.8	15.8	16.7	39.08
2-Layer CNN	Chaos (Löchel et al., 2020)	0.700	0.669	0.697	0.652	0.564	0.645	6.394
	Spike2Vec (Ali and Patterson, 2021)	0.740	0.730	0.744	0.729	0.736	0.725	7.329
	PWM2Vec (Ali et al., 2022)	0.740	0.700	0.739	0.688	0.694	0.676	6.615
	Minimizer	0.710	0.710	0.710	0.681	0.581	0.771	6.426
	Spike2CGR	0.633	0.577	0.633	0.559	0.376	0.663	6.193
% improv. of Spike2CGR from SOTA Chaos (Löchel et al., 2020)		−6.7	−9.2	−6.4	−9.3	−18.8	1.8	3.14
3-Layer CNN	Chaos (Löchel et al., 2020)	0.740	0.722	0.739	0.717	0.696	0.809	5.658
	Spike2Vec (Ali and Patterson, 2021)	0.750	0.723	0.750	0.715	0.725	0.838	6.919
	PWM2Vec (Ali et al., 2022)	0.751	0.715	0.751	0.716	0.732	0.846	7.458
	Minimizer	0.750	0.729	0.750	0.721	0.719	**0.851**	6.332
	Spike2CGR	0.770	0.724	0.767	0.734	0.712	0.845	4.758
% improv. of Spike2CGR from SOTA Chaos (Löchel et al., 2020)		3	0.2	2.8	1.7	1.6	3.6	31.23
4-Layer CNN	Chaos (Löchel et al., 2020)	0.740	0.686	0.737	0.706	0.678	0.728	7.986
	Spike2Vec (Ali and Patterson, 2021)	0.750	0.686	0.749	0.712	0.720	0.842	7.447
	PWM2Vec (Ali et al., 2022)	0.750	**0.733**	0.745	0.736	**0.747**	0.847	7.720
	Minimizer	0.750	0.726	0.750	0.706	0.709	0.846	7.068
	Spike2CGR	**0.7708**	0.731	**0.768**	0.738	0.714	0.843	10.658
% improv. of Spike2CGR from SOTA Chaos (Löchel et al., 2020)		3	4.5	3.1	3.2	3.6	11.5	−33.45

Table 3 (continued)

DL Model	Method	Acc. ↑	Prec. ↑	Recall ↑	F1 (Weig.) ↑	F1 (Macro) ↑	ROC AUC ↑	Train Time (hrs.) ↓
RESNET50 Pre-Trained Model	Chaos (Löchel et al., 2020)	0.680	0.644	0.676	0.641	0.547	0.743	10.654
	Spike2Vec (Ali and Patterson, 2021)	0.711	0.657	0.710	0.666	0.644	0.759	10.746
	PWM2Vec (Ali et al., 2022)	0.680	0.589	0.675	0.606	0.507	0.757	10.264
	Minimizer	0.723	0.665	0.723	0.673	0.647	0.802	11.732
	Spike2CGR	0.740	0.661	0.736	0.683	0.626	0.780	14.299
% improv. of Spike2CGR from SOTA Chaos (Löchel et al., 2020)		6	−1.7	6	4.2	7.9	3.7	−34.21
VGG-19 Pre-Trained Model	Chaos (Löchel et al., 2020)	0.480	0.233	0.483	0.315	0.050	0.500	27.398
	Spike2Vec (Ali and Patterson, 2021)	0.470	0.221	0.470	0.301	0.049	0.500	26.599
	PWM2Vec (Ali et al., 2022)	0.464	0.215	0.464	0.294	0.048	0.500	23.781
	Minimizer	0.480	0.227	0.477	0.308	0.496	0.500	24.459
	Spike2CGR	0.495	0.245	0.495	0.327	0.050	0.500	24.355
% improv. of Spike2CGR from SOTA Chaos (Löchel et al., 2020)		1.5	1.2	1.2	1.2	0	0	8.4

Table 4 Classification results for different models and algorithms for host classification using **coronavirus host dataset**. The ↑ and ↓ mean higher and lower values are better, respectively

DL Model	Method	Acc. ↑	Prec. ↑	Recall ↑	F1 (Weig.) ↑	F1 (Macro) ↑	ROC AUC ↑	Train Time (hrs.) ↓
3-Layer Tab CNN	OHE (Kuzmin, 2020)	0.625	0.626	0.625	0.566	0.335	0.663	0.032
	WDGRL (Shen et al., 2018)	0.304	0.137	0.304	0.182	0.041	0.499	**0.029**
4-Layer Tab CNN	OHE (Kuzmin, 2020)	0.613	0.478	0.613	0.534	0.323	0.662	0.067
	WDGRL (Shen et al., 2018)	0.312	0.130	0.312	0.167	0.035	0.498	0.054
1-Layer CNN	Chaos (Löchel et al., 2020)	0.680	0.707	0.680	0.670	0.517	0.761	0.984
	Spike2Vec (Ali and Patterson, 2021)	0.728	0.738	0.728	0.711	0.412	0.710	0.738
	PWM2Vec (Ali et al., 2022)	**0.753**	**0.745**	**0.753**	**0.745**	0.496	0.743	0.950
	Minimizer	0.737	0.735	0.737	0.727	0.514	0.766	1.028
	Spike2CGR	0.743	**0.745**	0.743	0.739	**0.569**	**0.797**	0.711
% improv. of Spike2CGR from SOTA Chaos (Löchel et al., 2020)		6.3	3.8	6.3	5.9	5.3	3.6	27.7
2-Layer CNN	Chaos (Löchel et al., 2020)	0.668	0.684	0.668	0.655	0.410	0.710	1.046
	Spike2Vec (Ali and Patterson, 2021)	0.735	0.728	0.735	0.713	0.354	0.674	0.821
	PWM2Vec (Ali et al., 2022)	0.742	0.742	0.742	0.729	0.508	0.7641	0.970
	Minimizer	0.685	0.718	0.685	0.671	0.426	0.706	1.098
	Spike2CGR	0.740	0.734	0.740	0.726	0.428	0.716	0.688
% improv. of Spike2CGR from SOTA Chaos (Löchel et al., 2020)		7.2	5	7.2	7.1	1.8	0.6	34.2
3-Layer CNN	Chaos (Löchel et al., 2020)	0.681	0.677	0.681	0.672	0.470	0.74	0.681
	Spike2Vec (Ali and Patterson, 2021)	0.718	0.690	0.718	0.695	0.283	0.632	0.717
	PWM2Vec (Ali et al., 2022)	0.697	0.724	0.697	0.682	0.395	0.689	0.795
	Minimizer	0.731	0.734	0.731	0.719	0.424	0.716	0.960
	Spike2CGR	0.729	0.729	0.729	0.715	0.354	0.677	0.831
% improv. of Spike2CGR from SOTA Chaos (Löchel et al., 2020)		4.8	5.2	4.8	4.3	−11.6	−6.3	−22.02
4-Layer CNN	Chaos (Löchel et al., 2020)	0.624	0.617	0.624	0.606	0.262	0.623	0.991
	Spike2Vec (Ali and Patterson, 2021)	0.720	0.708	0.720	0.695	0.282	0.630	0.686
	PWM2Vec (Ali et al., 2022)	0.732	0.720	0.732	0.712	0.294	0.635	0.981
	Minimizer	0.718	0.716	0.718	0.695	0.290	0.636	0.995
	Spike2CGR	0.686	0.668	0.686	0.672	0.283	0.632	0.684
% improv. of Spike2CGR from SOTA Chaos (Löchel et al., 2020)		6.2	5.1	6.2	6.6	2.1	0.9	30.97

Table 4 (continued)

DL Model	Method	Acc. ↑	Prec. ↑	Recall ↑	F1 (Weig.) ↑	F1 (Macro) ↑	ROC AUC ↑	Train Time (hrs.) ↓
RESNET50 Pre-Trained Model	Chaos (Löchel et al., 2020)	0.662	0.665	0.662	0.639	0.267	0.621	0.840
	Spike2Vec (Ali and Patterson, 2021)	0.706	0.672	0.706	0.685	0.277	0.621	0.786
	PWM2Vec (Ali et al., 2022)	0.663	0.673	0.663	0.663	0.627	0.614	1.020
	Minimizer	0.694	0.671	0.694	0.665	0.266	0.621	1.730
	Spike2CGR	0.691	0.683	0.691	0.663	0.270	0.624	0.786
% improv. of Spike2CGR from SOTA Chaos (Löchel et al., 2020)		2.9	1.8	2.9	2.4	0.3	0.3	6.42
VGG-19 Pre-Trained Model	Chaos (Löchel et al., 2020)	0.519	0.475	0.519	0.442	0.158	0.572	3.738
	Spike2Vec (Ali and Patterson, 2021)	0.506	0.405	0.506	0.407	0.149	0.566	3.390
	PWM2Vec (Ali et al., 2022)	0.491	0.401	0.491	0.386	0.166	0.579	3.535
	Minimizer	0.509	0.427	0.509	0.427	0.186	0.576	3.969
	Spike2CGR	0.458	0.409	0.458	0.363	0.129	0.559	3.409
% improv. of Spike2CGR from SOTA Chaos (Löchel et al., 2020)		−6.1	−6.6	−6.1	−7.9	−2.9	−1.3	8.80

Improvement >-5% could be considered as significant

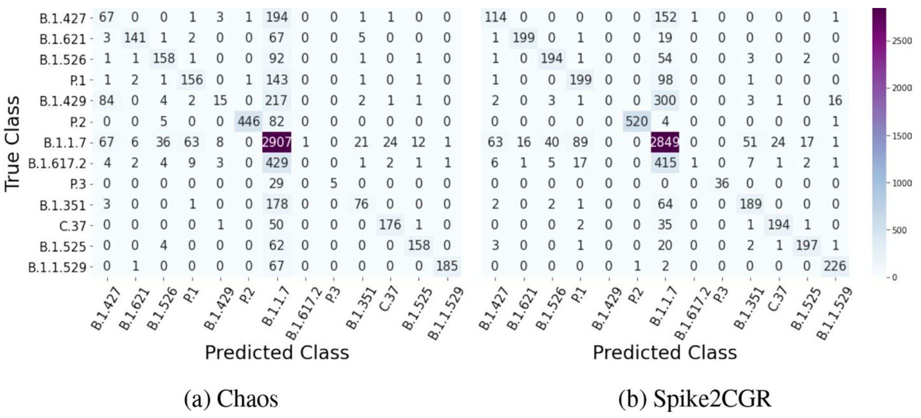


Fig. 8 Confusion matrices comparison of Chaos and Spike2CGR based encoding to perform variant classification using the best performing model (4-layer CNN)

5.5 Generalizability and scalability of spike2CGR

Since our method is able to avoid confusing two closely related lineages in the presence of many other lineages (for the SARS-CoV-2 dataset), we believe that adding more data (i.e., sequences) would improve the performance of our approach since it would be able to build a more sophisticated model of the data. If the phylogenetic tree for the lineages grows in depth (having more data) and in breadth (having more lineages), both of these are a function of the mutation rate of the virus and time — both for which sequence diversity increases, in any case. Since the mutation rate of SARS-CoV-2 is fairly slow compared to other RNA viruses like HIV or Hepatitis-C (HCV) (Markov et al., 2023), and it only has been around for a few years (unlike these other viruses), SARS-CoV-2 sequences are hence highly similar to each other in general. SARS-CoV-2 lineage classification is thus an ideal (worst case scenario) test case to assess the classification performance of our tool for viral sequences, i.e., it has the potential to perform even better in the case of HIV or HCV. In terms of scalability in the computation time

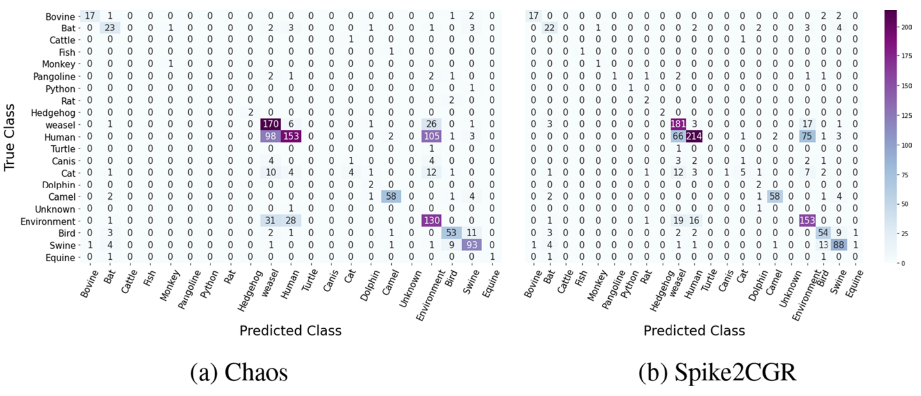


Fig. 9 Confusion matrices comparison of Chaos and Spike2CGR based encoding to perform host classification using the best performing model (1-layers CNN)

of Spike2CGR, creating images for the given SARS-CoV-2 dataset took around 10 h. This indicates that we could make our method scalable by enhancing the computation resources.

6 Conclusion

In this paper, we propose different embedding-based methods to convert the protein sequences into 2D visual representations using a chaos game representation-based approach. We show that by manipulating the sequences in a biologically meaningful way, we can improve the classification performance of DL models as compared to the SOTA. This could help biologists and doctors to understand the behavior of coronavirus and hence take efficient preventive measures in advance to reduce its impact on humans. In the future, we will extract more data and perform experiments using other DL methods for classification. Similarly, evaluating the scalability of our system by investigating the data consisting of SARS-CoV-2 lineages having more similarity with each other and using large-size data is also an interesting future direction. Moreover, we would also train a DL model using more biological data, which researchers can use as a pre-trained model for the analysis of different biological sequences in the future. Using other methods for sequence embeddings, such as spaced *k*-mers, is also an interesting area to explore. Another possible future extension is to explore the tradeoff between pairwise distances and some phenetic analysis (i.e., some neighbor-joining).

Author contributions T. M. and S. A. worked on the idea of converting sequences into visual representations. T. M. did the literature review. M. P. and I. K. worked on brainstorming the main idea and supervised the project. M. P., I. K., and S. A. worked on data collection. T. M. and S. A. worked on computing results. S. A. and T. M. worked on implementing baselines. All authors worked on the writeup.

Data availability The dataset is available at https://github.com/sarwanpasha/Spike2CGR/tree/main

References

Ahmed, I., & Jeon, G. (2021). Enabling artificial intelligence for genome sequence analysis of covid-19 and alike viruses. *Interdisciplinary Sciences: Computational Life Sciences, 14*(2), 504–519.

Ali, S., Bello, B., Chourasia, P., Punathil, R. T., Zhou, Y., & Patterson, M. (2022). Pwm2vec: An efficient embedding approach for viral host specification from coronavirus spike sequences. *MDPI Biology Journal, 11*(3), 418.

Ali, S., Sahoo, B., Ullah, N., Zelikovskiy, A., Patterson, M., Khan, I. (2021). A k-mer based approach for sars-cov-2 variant identification. In: International symposium on bioinformatics research and applications. pp. 153–164

Ali, S., Ali, T.E., Khan, M.A., Khan, I., Patterson, M. (2021). Effective and scalable clustering of sars-cov-2 sequences. In: Proceedings of the 5th international conference on big data research, pp. 42–49

Ali, S., Patterson, M. (2021). Spike2vec: An efficient and scalable embedding approach for covid-19 spike sequences. In: International conference on big data (Big Data), pp. 1533–1540. IEEE

Barnsley, M. F. (2012). *Fractals Everywhere* (New). New York: Dover Publications Mineola.

Bishop, C. M., & Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*. Berlin: Springer.

Cherkasov, A., et al. (2014). Qsar modeling: Where have you been? where are you going to? *Journal of medicinal chemistry, 57*(12), 4977–5010.

Chowdhury, B., & Garai, G. (2017). A review on multiple sequence alignment from the perspective of genetic algorithm. *Genomics, 109*(5–6), 419–431.

Cui, J., Liu, Q., Puett, D., & Xu, Y. (2008). Computational prediction of human proteins that can be secreted into the bloodstream. *Bioinformatics, 24*(20), 2370–2375.

Deber, C. M., Wang, C., Liu, L.-P., Prior, A. S., Agrawal, S., Muskat, B. L., & Cuticchia, A. J. (2001). Tm finder: a prediction program for transmembrane protein segments using a combination of hydrophobicity and nonpolar phase helicity scales. *Protein Science, 10*(1), 212–219.

Devijver, P., Kittler, J.: Pattern recognition: A statistical approach. In: GB: Prentice-Hall, London pp. 1–448 (1982)

GISAID. (2021). Website https://www.gisaid.org/. [Accessed 29-December-2021]

Galloway, S., et al. (2021). Emergence of sars-cov-2 b. 1.1. 7 lineage. *Morbidity and Mortality Weekly Report, 70*(3), 95.

Hadfield, J., Megill, C., Bell, S. M., Huddleston, J., Potter, B., Callender, C., Sagulenko, P., Bedford, T., & Neher, R. A. (2018). Nextstrain: Real-time tracking of pathogen evolution. *Bioinformatics, 34*, 4121–4123.

He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep residual learning for image recognition. In: IEEE conference on computer vision and pattern recognition, pp. 770–778

Hirst, J. D., & Sternberg, M. J. (1992). Prediction of structural and functional features of protein and nucleic acid sequences by artificial neural networks. *Biochemistry, 31*(32), 7211–7218.

Hoang, T., Yin, C., & Yau, S.S.-T. (2016). Numerical encoding of DNA sequences by chaos game representation with application in similarity comparison. *Genomics, 108*(3–4), 134–142.

Jeffrey, H. J. (1990). Chaos game representation of gene structure. *Nucleic acids research, 18*(8), 2163–2170.

Kuzmin, K., et al. (2020). Machine learning methods accurately predict host specificity of coronaviruses based on spike sequences alone. *Biochemical and Biophysical Research Communications, 533*(3), 553–558.

Löchel, H. F., Eger, D., Sperlea, T., & Heider, D. (2020). Deep learning on chaos game representation for proteins. *Bioinformatics, 36*(1), 272–279.

Ma, Y., Yu, Z., Tang, R., Xie, X., Han, G., & Anh, V. V. (2020). Phylogenetic analysis of hiv-1 genomes based on the position-weighted k-mers method. *Entropy, 22*(2), 255.

Majumder, J., & Minko, T. (2021). Recent developments on therapeutic and diagnostic approaches for covid-19. *AAPS Journal, 23*(1), 1–22.

Markov, P. V., Ghafari, M., Beer, M., et al. (2023). The evolution of SARS-CoV-2. *Natural Reviews Microbiology, 21*, 361–379. https://doi.org/10.1038/s41579-023-00878-2

Matsuda, S., Vert, J.-P., Saigo, H., Ueda, N., Toh, H., & Akutsu, T. (2005). A novel representation of protein sequences for prediction of subcellular location using support vector machines. *Protein Science, 14*(11), 2804–2813.

Minh, B. Q., et al. (2020). IQ-tree 2: New models and efficient methods for phylogenetic inference in the genomic era. *Molecular Biology and Evolution, 37*(5), 1530–1534.

Naveca, F., et al. (2021). Phylogenetic relationship of sars-cov-2 sequences from amazonas with emerging brazilian variants harboring mutations e484k and n501y in the spike protein. Virological. org **1**

Phylogenetic assignment of named global outbreak lineages (Pangolin). (2022). https://cov-lineages.org/resources/pangolin.html. [accessed 27-March-2022]

Pickett, B. E., Sadat, E. L., Zhang, Y., Noronha, J. M., Squires, R. B., Hunt, V., Liu, M., Kumar, S., Zaremba, S., Gu, Z., et al. (2012). Vipr: an open bioinformatics database and analysis resource for virology research. *Nucleic Acids Research, 40*(D1), 593–598.

Rizzo, R., Fiannaca, A., La Rosa, M., Urso, A. (2016). Classification experiments of dna sequences by using a deep neural network and chaos game representation. In: Proceedings of the 17th international conference on computer systems and technologies 2016, pp. 222–228

Roberts, M., Haynes, W., Hunt, B. R., Mount, S. M., & Yorke, J. A. (2004). Reducing storage requirements for biological sequence comparison. *Bioinformatics, 20*, 3363–9.

Shen, J., Qu, Y., Zhang, W., Yu, Y. (2018). Wasserstein distance guided representation learning for domain adaptation. In: AAAI conference on artificial intelligence

Simonyan, K., Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In: International conference on learning representations

Spänig, S., & Heider, D. (2019). Encodings and models for antimicrobial peptide classification for multiresistant pathogens. *BioData Mining, 12*(1), 1–29.

Strichartz, R. S. (2000). Evaluating integrals using self-similarity. *The American Mathematical Monthly, 107*(4), 316–326.

Tzanov, V. (2015) Strictly self-similar fractals composed of star-polygons that are attractors of iterated function systems. arXiv preprint arXiv:1502.01384

Uyangodage, L., Ranasinghe, T., & Hettiarachchi, H. (2021). Transformers to fight the COVID-19 infodemic. In: NLP for Internet Freedom: Censorship, Disinformation, and Propaganda, pp. 130–135

WHO. (2021). Website https://www.who.int/en/activities/tracking-SARS-CoV-2-variants/

West Jr, A., et al. (2021). Detection and characterization of the sars-cov-2 lineage b. 1.526 in New York. bioRxiv

Yadav, P., et al. (2021). Neutralization potential of covishield vaccinated individuals sera against b. 1.617. 1. bioRxiv **1**

Yao, H., Zhu, D.-l., Jiang, B., Yu, P. (2019). Negative log likelihood ratio loss for deep neural network classification. In: Future technologies conference, pp. 276–282. Springer

Zhang, W., et al. (2021). Emergence of a novel sars-cov-2 variant in southern california. *Jama, 325*(13), 1324–1326.

Authors and Affiliations

Taslim Murad[1] · Sarwan Ali[1] · Imdadullah Khan[2] · Murray Patterson[1]

✉ Murray Patterson
mpatterson30@gsu.edu

Taslim Murad
tmurad2@student.gsu.edu

Sarwan Ali
sali85@student.gsu.edu

Imdadullah Khan
imdad.khan@lums.edu.pk

[1] Department Of Computer Science, Georgia State University, Atlanta 30302, Georgia, USA

[2] Department Of Computer Science, Lahore University of Management Sciences (LUMS), Lahore, Pakistan