Imperial College London

M4R Project

Deep unsupervised learning methods for the identification and characterization of TCR specificity to Sars-Cov-2

Author:
Yanis Miraoui
CID: 01731821

Supervisor:
Dr. Barbara Bravi
Lecturer in Biomathematics

This is my own work except where otherwise stated.

Department of Mathematics
Imperial College London
London, United Kingdom
June 13, 2023

Abstract

The T-cell receptor (TCR) is one of the key players in the immune response to the Sars-Cov-2 virus. In this study, we used deep unsupervised learning methods to identify and characterize TCR specificity. Our research focused on developing and applying state-of-the-art modelling techniques, including AutoEncoders, Variational AutoEncoders and transfer learning with Transformers, to analyze TCR data. Through our experiments and analyses, we have achieved promising results in identifying TCR patterns and understanding TCR specificity for Sars-Cov-2. The insights gained from our research provide valuable tools and knowledge for interpreting the immunological response to the virus, ultimately contributing to the development of effective vaccines and treatments against the viral infection.

Contents

1 Introduction

1.1 The immune response

The immune response is a complex biological process that is essential for protecting the body from harmful pathogens such as viruses, bacteria, and parasites. This process involves various cells and molecules that work together to identify and eliminate foreign invaders.

One of the key players in the immune response is the **T** **c**ell **r**eceptor (**TCR**). T cells are a type of white blood cell responsible for recognizing and targeting specific antigens, which are short proteins on the surface of pathogens. When a pathogen enters the body, T cells that recognize the antigen will become activated and multiply into a large population of identical cells. Some of these cells will differentiate into effector T cells, directly attacking and eliminating infected cells. Other T-cells will differentiate into memory T-cells, which can provide long-term immunity against future infections by the same pathogen.

As one can be observe in Figure 1 (Sun et al., 2021), T cell receptors (TCRs) are mainly composed of:

- an α protein chain
- a β protein chain

Each of these chains contains both constant and variable regions. The variable region of the alpha and beta chains is created by the combination of V (Variable), D (Diversity), and J (Joining) gene segments, which undergo a process of recombination during T cell development to create a unique TCR that can recognize a specific antigen (Figure 2) (Ralph et al., 2015). Both chains are responsible for the binding of the TCR with the antigen peptide via **c**omplementarity-**d**etermining **r**egions (CDRs). In particular, the third **c**omplementary **d**etermining **r**egion (CDR3) in the beta chain has the highest variability and is the most significant region for antigen binding and recognition. As of today, we estimate that more than 4×10^{11} different TCRs exist in an adult body (Jenkins et al., 2010). This large number of possible combinations allows the body to develop T cells that can recognize a wide range of antigens and therefore a wide range of pathogens.

On the other hand, the role of T-cells in the immune response is multifaceted. They are essential for coordinating and regulating the overall immune response, ensuring that it is both effective and targeted. T-cell receptors also play a critical role in the process of immunological memory, which is the ability of the immune system to remember and mount a rapid response to a previously encountered pathogen (Pennock et al., 2013).

Overall, the TCRs play an essential role in maintaining health and fighting off infections. The immune response depends heavily on the recognition and binding of T-cell receptors to the antigens of harmful pathogens that invade our bodies.

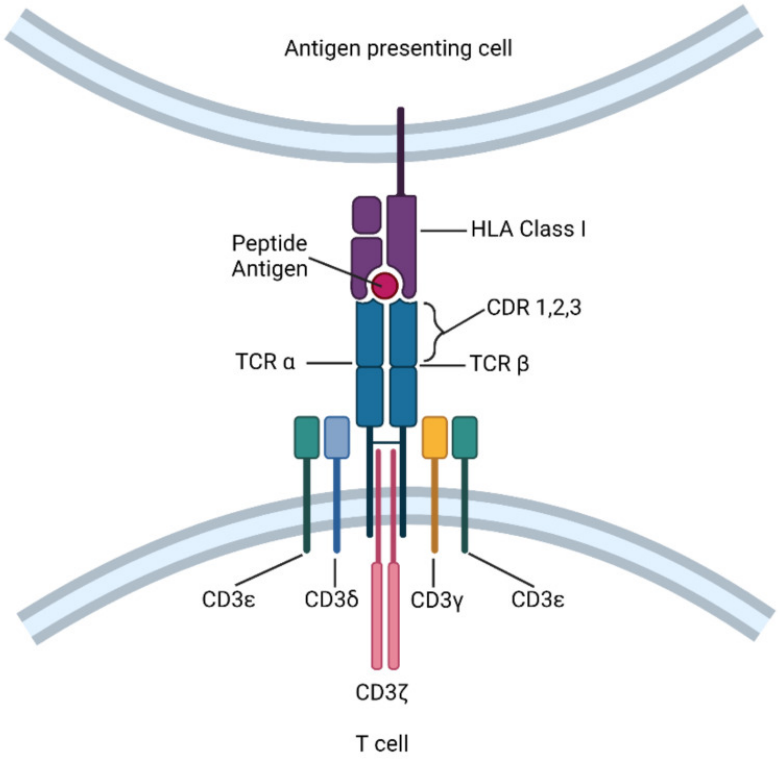


Figure 1: Structure of a **T** **c**ell **r**eceptor (TCR) and its components (Sun et al., 2021)

1.2 Background of our study

The ability to identify T-cell receptors specific to the antigens of a given virus is key to the design of vaccines and therapies. Designing effective vaccines and therapies against a particular virus requires knowledge of which TCRs are specific to the antigens of that virus (Gallagher, 2023). This information can be used to develop vaccines that stimulate the immune system to produce T-cells with those specific TCRs or to develop therapies that use T-cells with those TCRs to target and destroy infected cells. It could, for example, help adoptive T-cell therapies, in which T-cells with specific receptors are infused into cancer patients to target and destroy cancer cells using the specific TCRs. It can also aid in understanding the immune response to the virus and how it evolves over time.

However, identifying the specific TCRs involved in the immune response to a virus can be a complex and daunting task. The world has been experiencing this challenge in particular in the recent past with the emergence of Sars-Cov-2 in 2019:

- Firstly, Sars-Cov-2 is a novel virus, meaning that researchers did not have any pre-existing knowledge about the virus's antigens or the TCRs involved in the immune response to it. This made it necessary to start almost from scratch to identify the relevant TCRs (Mateus et al., 2020).

- Secondly, TCRs are highly diverse and can vary greatly between individuals. This means that different people can have different TCRs specific to the same virus. Therefore, it is essential to analyze TCRs from a large number of individuals to identify the TCRs that are most commonly associated with the immune response to COVID-19.

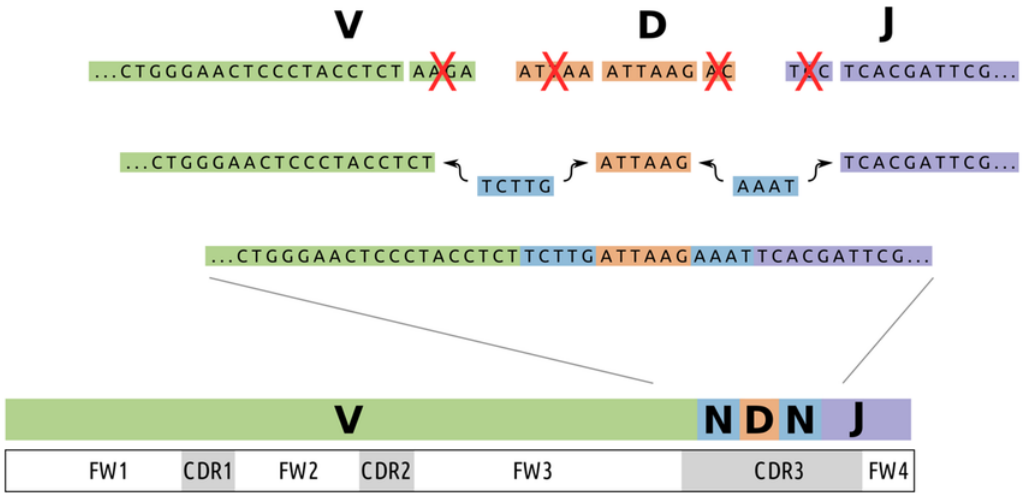


Figure 2: The VDJ recombination process that defines the TCR sequence (Ralph et al., 2015)

- Finally, TCR sequencing is a complex and time-consuming process, which can limit the speed and scale of TCR analysis. However, advances in sequencing technologies and computational methods have now made it possible to analyze TCRs from large numbers of individuals more efficiently (Bravi et al., 2021). Nonetheless, we could take this even further, as modelling the mapping from TCR sequences to binding specificities would allow us to narrow down the number of receptors to test. In other words, we could establish links between TCR sequences and functional properties to help predict binding specificity. This integration would allow promising TCR candidates to be prioritized for experimental validation. This would significantly optimize time and resources.

Despite these challenges, researchers have made some progress in identifying the specific TCRs involved in the immune response to COVID-19. For example, multiple studies have identified TCRs specific to different antigens of the SARS-CoV-2 virus, which can help inform the design of vaccines and treatments. However, this task remains very complex and time-consuming. The exploration of an *automatic* method [1] to identify binding TCRs and antigens appears as the leading solution to overcome these barriers.

In the meantime, in the past few years, we have been observing a rapid proliferation of machine learning methods in all domains and particularly in the field of biology. As the *2022 AI Index Report* (Zhang et al., 2022) and *2023 AI Index Report* (Zhang et al., 2023) emphasize, machine learning tools have been democratized exponentially over time, enabling groundbreaking discoveries and accelerating research.

The goal of our study is therefore to find an *automatic* method able to make use of state-of-the-art machine learning tools in order to characterize and identify the antigen specificity of TCRs.

[1] In this context, the term "*automatic*" refers to a method or approach that uses computational tools and algorithms to identify the specific TCRs involved in the immune response to COVID-19.

1.3 Literature review and related work

Multiple research studies have already led to very insightful results in identifying specific immune responses to Sars-Cov-2. In fact, some previous works have been able to relatively well cluster similar TCR sequences based on their antigen specificity using various statistical and machine learning techniques:

- By employing an entirely statistical approach, a custom similarity score based on the frequencies and rates of TCRs can be defined (Altan-Bonnet et al., 2020). For example, some similarity measures are defined using the amino acid sequence of the CDR3 part. This approach can be complemented with foundational clustering methods like k-means or DBSCAN clustering (Cheung, 2022).
- Building a new distance-based TCR repertoire (*tcrdist3*) in order to group TCR based on their metaclonotype (Mayer-Blackwell et al., 2021).
- Using innovative and state-of-the-art deep learning methods in a supervised procedure (Weber et al., 2021) but also an unsupervised approach using AutoEncoders models for example (Sidhom et al., 2021).

Nevertheless, this task is still new and while there is still room to improve the predictive performance of these methods, it also seems essential to analyze the interpretability of these models as it can provide valuable biological insights. The comprehensiveness of these models is crucial because it allows us to understand and relate their predictions to the actual molecular properties and interactions that occur in real life. Furthermore, interpreting the weights, behaviors and internal processes of machine learning methods has become essential to building robust models (Olah et al., 2018).

In our study, we aim to fulfill these precise shortfalls. We intend to compare different unsupervised machine learning methods able to cluster T cell receptors based on their antigen specificity. Our goal is not only to over-perform current existing methods but also to provide a complete interpretation of the weights and behaviours of these models. We will first analyse the performance of simple models such as the AutoEncoder before building more advanced architectures using a Variational AutoEncoder, Transformers and Transfer Learning.

1.4 The Sars-Cov-2 datasets

In our study, we will mainly investigate the antigens linked to the Sars-Cov-2 virus. For this purpose, we will make use of a dataset made freely available here (Nolan et al., 2020). It consists of multiple tabular data files that contain the CDR3, the v-gene and the j-gene sequences of different TCRs (separated by + signs) along with the list of antigens with which they have bound to. This dataset is of a very large size ($+300,000$ rows) but for computational as well as vizualisation reasons, we will mostly use some extracts of it. We present a small extract of this dataset in Table 1 to better understand the general structure of the data.

	TCR Bioidentity	Amino Acids
0	CASSIAAADNSPLHF+TCRBV19-01+TCRBJ01-06	AFLLFLVLI,FYLCFLAFL
1	CASSSGLAPYEQYF+TCRBV19-01+TCRBJ02-07	FYLCFLAFL,FYLCFLAFLL
2	CASQLSGSLSGANVLTF+TCRBV25-01+TCRBJ02-06	FLAFLLFLV,FYLCFLAFL
3	CASSEQADTQYF+TCRBV06-01+TCRBJ02-03	FYLCFLAFL,FYLCFLAFLL
4	CASTLTPPRLLRRNF+TCRBV06-X+TCRBJ02-03	AFLLFLVLI,FLAFLLFLV

Table 1: Extract of 5 rows from the dataset; it contains TCR sequences along with the binding list of Sars-Cov-2 antigens.[3]

Moreover, we will also employ a dataset containing different TCR sequences labeled with their metaclonotype group [4] derived from the *tcrdist3* tool (Mayer-Blackwell et al., 2021). Each *.tsv* file in this dataset represents a different metaclonotype group of TCR sequences. We therefore have pre-processed and aggregated it into one file by tagging and numerating each group. This dataset will in particular, help us to evaluate our methods and models and provide valuable tools for the interpretation of our models' outputs. This dataset is available and can be downloaded from here through the immuneRACE project.

	TCR Bioidentity	Metaclonotype_Group
0	CASSIAAADNSPLHF+TCRBV19-01+TCRBJ01-06	1
1	CASSSGLAPYEQYF+TCRBV19-01+TCRBJ02-07	4
2	CASQLSGSLSGANVLTF+TCRBV25-01+TCRBJ02-06	1
3	CASSEQADTQYF+TCRBV06-01+TCRBJ02-03	6
4	CASTLTPPRLLRRNF+TCRBV06-X+TCRBJ02-03	3

Table 2: Extract of 5 rows from the dataset; it contains TCR sequences along with their metaclonotype *tag*.

2 Modelling techniques

2.1 One pipeline

Throughout our study, we will make use of **one** similar pipeline. This analogous pipeline will consist of the following:

1. A data pre-processing
2. An encoding of the TCR sequences
3. A modelling and fitting process
4. The leverage of a clustering method

Within this pipeline, all the steps will remain similar except for the *modelling component* which we will modify and adapt to take advantage of different approaches. This will allow us to accurately compare the different modelling techniques used and determine which are better suited to this task.

[3]Please note that we have selected only the columns meaningful for our analysis for this visualisation. The original datasets contains 52 columns

[4]Groups of TCRs that are biochemically similar

2.1.1 Data pre-processing

The first step that we have to carry out is the pre-processing of our dataset. This consists of the cleaning and the transformation of our data after having carefully inspected its structure. We mainly focus on the pre-processing of the CDR3 part (the first part of the TCR Bioidentity in our dataset in Tables 1 and 2). In fact, the two other parts consist of the *v-gene* and *j-gene* and these will later be encoded into categories.

The first step that we have to conduct is to make sure that there are no errors in the CDR3 sequences of our dataset. In other words, we have to check that each CDR3 sequence that we will be using for our study can be accurately and reliably identified without any ambiguity or inconsistencies. This quality control process is crucial to ensure the integrity of our dataset and the validity of our study's findings. For this purpose, we employ the computational tools of the *soNNia* package that are designed explicitly for CDR3 sequence analysis (Isacchini et al., 2021). These tools are capable of identifying potential errors, such as inexistent or unrecognized genes, missing values, or other anomalies that could compromise the accuracy of the data.

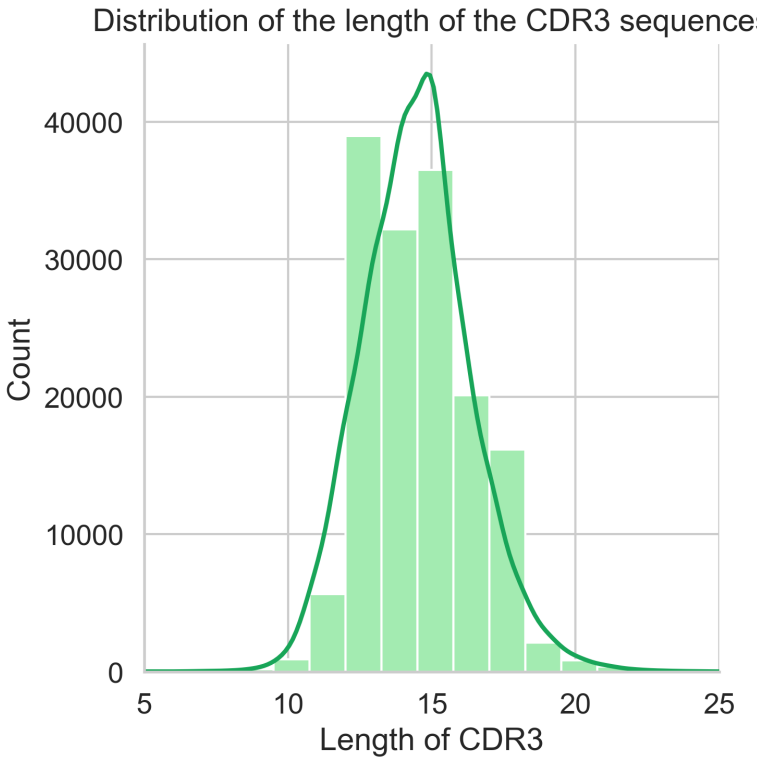


Figure 3: Distribution of the length of the CDR3 sequences.

One of the main obstacles to the exploitation of machine learning methods for the analysis of TCR sequences is the variability in the length of CDR3 sequences. In fact, CDR3 length can vary from a few up to 30 amino acids. We therefore aim to unify this aspect by padding and aligning the sequences according to the procedure presented in a previous study (Bravi et al., 2021). By calculating the length of the CDR3 sequences and the number of amino acids to which the TCR has bound, we are able to have a closer look at the specificities of our dataset. In Figure 3, we can

see that the lengths of the CDR3 sequences are symmetrically distributed around 15. By building the boxplot of the lengths of the sequences in terms of the number of antigens the TCR has bound to in Figure 4, we also notice that the lengths of the CDR3 sequences are similar across the different groups of antigens. Only a few *outlier* sequences have a very small (less than 5) or very large (greater than 20) size. Furthermore, we can compute that only **less than 0.4%** have a length greater than 20. Therefore, similarly to the previous study (Bravi et al., 2021), we can choose to align the sequences to all have a fixed length of 20 and drop the small subset of sequences that have larger lengths. The methodology used to align the sequences utilizes Hidden Markov Models (HMMs) and the code is available here. It has also been shown in this previous study that the use of this sequence alignment procedure, with gaps located in the central variable region of the CDR3, does not add any extra bias to our analysis. By using an RBM-LR model that does not require alignment, it has been shown that this alignment routine maintains the known biological structure of CDR3 sequences (Bravi et al., 2021). Therefore, using this alignment process, we obtain CDR3 sequences such as the ones in Table 3.

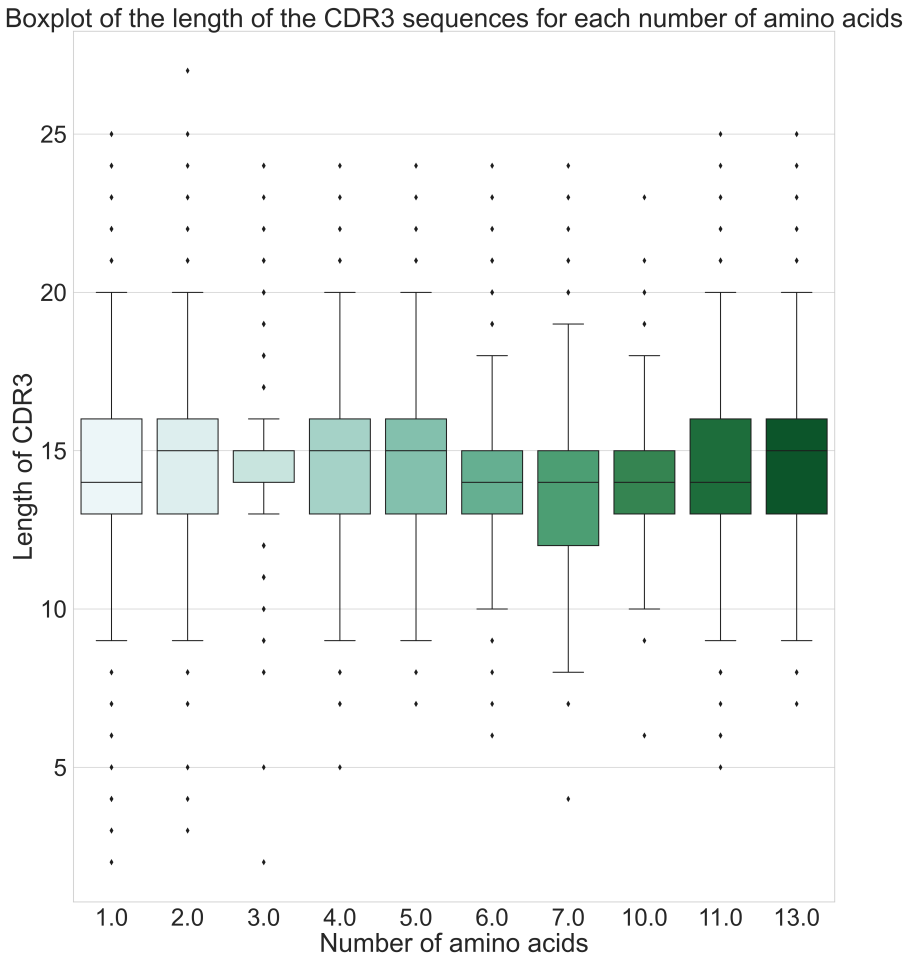


Figure 4: Boxplot of the length of the CDR3 sequences in terms of the number of amino acids the TCR has bound to.

Furthermore, in order to focus solely on the spatial representation of our TCRs and their antigen specificity clustering, we remove all duplicate rows from the dataset, as the frequency of occurrence is not of interest. Another important aspect that requires pre-processing is the presence of a multi-labeled target for the majority of

TCRs. This target is provided as a list of antigens to which the TCRs have bound. However, for our analysis, we only consider the first element of this list as the designated label. By doing this, we aim to evaluate the spatial representations of the TCRs and their clustering based on their antigen specificity.

	Original CDR3 sequence	Aligned CDR3 sequence
0	CASSAQGTGDRGYTF	CASSA- - - - -QGTGDRGYTF
1	CASSLVATGNTGELFF	CASSL - - - -VATGNTGELFF
2	CALKVGADTQYF	CALKV- - - - - - - -GADTQYF
3	CASSLWASGRGGTGELFF	CASSLW- -ASGRGGTGELFF
4	CASSLLGWEQLDEQFF	CASSL- - - -LGWEQLDEQFF

Table 3: Examples of the result of the CDR3 sequence alignment process.

2.1.2 Encoding

The next essential step in our pipeline involves the encoding of our CDR3 sequences as well as the *v-genes* and *j-genes* by converting their characters into numerical representations. By converting the sequences and genes into numerical values, we enable the application of various mathematical and statistical techniques for further analysis and modeling of the data.

To encode our CDR3 sequences, we construct an alphabet of all the possible characters that can be present in that string. This alphabet of 21 characters is employed to convert the sequences into one-hot encoded matrices of size 20×21. The alphabet is of size 21 as it is composed of 20 amino acids and the separator **"-"** used during the alignment procedure.

On the other hand, our goal is to make the most of the data available. Given that we have access to both the *v-gene* and *j-gene* sequences, we choose to exploit this additional information as we expect it to improve the performance of our models significantly. To achieve this, we use a one-hot categorical encoder to transform these genes from their character representation into a one-hot categorical format. By using this encoding technique, we convert the *v-genes* and *v-genes* into binary vectors where each gene is represented by a unique category. This transformation allows our models to effectively capture and utilize the distinct features of these genes, potentially improving the overall performance and reliability of our clustering.

Finally, we combine our encoded CDR3 sequences with our two categorically encoded vectors representing the *v-gene* and *j-gene*. As we can see in Figure 5, our input data for our models is therefore a list of 3 matrices. Note that in the following sections, we will refer *input of size 20* to a vector of a sequence that has been encoded into a matrix of size 20×21.

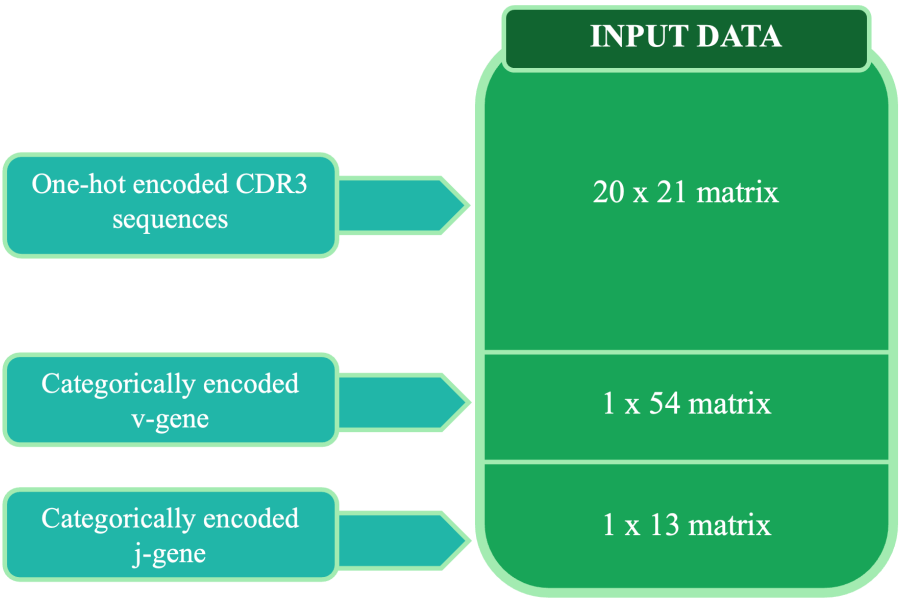


Figure 5: Diagram of the different components of the input data.

2.1.3 Modelling

The modelling component plays a crucial role in our pipeline as it provides the flexibility to modify and adapt the models used to obtain representations of TCRs. By using different deep learning techniques, we will explore a range of approaches and algorithms to capture the complex patterns and features inherent in TCR data.

Indeed, deep learning techniques provide powerful tools for learning high-level representations from TCR sequences. These techniques include AutoEncoders (AE), Variational AutoEncoder (VAE) and transformer-based models (TF). Each approach has its own strengths and limitations, so it is essential that we compare their performance in effectively clustering TCRs.

Through rigorous experimentation and analysis, we assess the effectiveness of each deep learning technique in capturing the diverse characteristics of TCR sequences. By comparing and contrasting the performance of different models, we can make informed decisions about which approach best captures the underlying structure of TCR data. We will be specifically focusing our analysis on 4 different deep learning approaches:

1. A simple AutoEncoder
2. An optimized deep AutoEncoder
3. A Variational AutoEncoder
4. Transfer Learning and Transformers

2.1.4 Dimensionality reduction

The modelling techniques previously presented represent TCR sequences as vectors in a latent dimensional space called embeddings. In most cases, the latent space

has dimensions of considerable size (up to 50 dimensions), making it difficult to visualize the results. To overcome this, we will first apply dimensionality reduction to visualize the latent space in two dimensions, thereby facilitating visualization.

For this purpose, we will employ a relatively recent and effective technique called **UMAP** (**U**niform **M**anifold **A**pproximation and **P**rojection) (McInnes et al., 2020), which has been utilized in recent studies in the field of TCR analysis (Wu et al., 2021). This technique is commonly used for visualizing high-dimensional data. It is particularly effective in preserving the local and global structure of the data while reducing its dimensionality. More precisely, UMAP is based on the concept of constructing a topological representation of the data, where similar data points are connected by shorter paths in the low-dimensional space. We will use UMAP to transform our large vectors as it offers several advantages over other dimensionality reduction techniques such as PCA or t-SNE:

- Proximity preservation: If two points are close in the high-dimensional space, they are likely to be close in the low-dimensional embedding space. This characteristic is particularly crucial for our analysis.
- Handling non-linear and complex structures: UMAP can effectively capture both global and local patterns in the data, making it suitable for complex data structures.
- Computational efficiency: UMAP is computationally efficient, making it well-suited for handling large datasets, especially when compared to t-SNE.

As described in Figure 6, the UMAP algorithm starts by constructing a high-dimensional graph representation of the data, where each data point is connected to its nearest neighbors. Then, it optimizes the low-dimensional embedding by minimizing the cross-entropy between the pairwise similarities of the data points in the high-dimensional space and the low-dimensional space. The optimization process involves minimizing the cross-entropy cost function using gradient descent:

$$C_{UMAP} = \sum_{i \neq j} p_{ij} log\left(\frac{p_{ij}}{q_{ij}}\right) + (1 - p_{ij}) log\left(\frac{1 - p_{ij}}{1 - q_{ij}}\right)$$

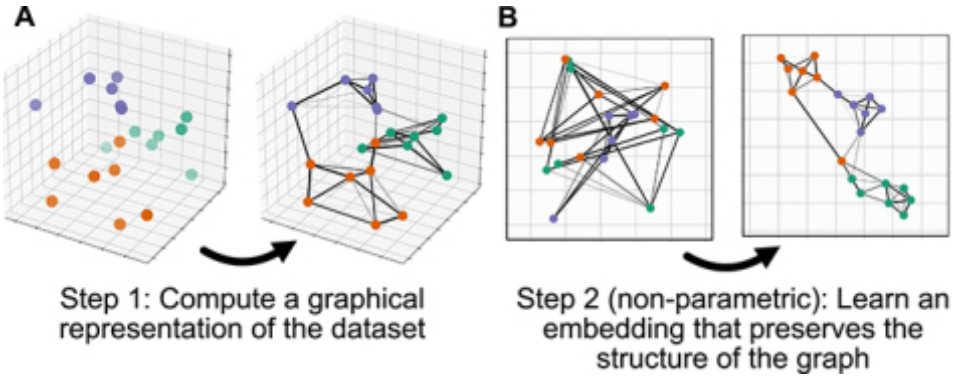


Figure 6: Overview of UMAP optimization (Sainburg et al., 2021).

The UMAP technique enables us to obtain a visualization of the data that effectively capture the underlying structure of the high-dimensional space. Although UMAP is a relatively new technique and not yet implemented in popular Python libraries such

as *scikit-learn* or *TensorFlow*, it can be easily implemented using the *umap-learn* package in Python.

2.1.5 Clustering methods

We can visualize this as the task of finding a mapping (or 'assignment') between the N samples and k clusters, where we also have to determine the number of clusters k. In fact, clustering techniques allow the grouping of similar TCR sequences, facilitating the identification of patterns and similarities in the data. Various clustering methods such as K-means, DBSCAN, Gaussian Mixture Model (GMM), OPTICS, etc., can be used to group closely related TCRs accurately. In our study, we will apply K-means to our transformed embeddings as it is computationally efficient and scalable for large datasets. Moreover, after the modelling and the dimensionality reduction steps, we should expect that our embeddings could be clustered straightforwardly.

The goal of the algorithm is to minimize the within-cluster sum of squares which quantifies the compactness of the clusters $C^1, ..., C^k$:

$$\min_{C^1,...,C^k} \sum_{i=1}^{m} \min_{h=1,...,k} \left(\frac{1}{2} \|x^i - C^h\|_2^2 \right)$$

Moreover, we can describe the k-means algorithm step-by-step as in Figure 7:

1. Initialization: We choose the number of clusters, k, and randomly initialize k cluster centroids. These centroids serve as the initial centers of the clusters.

2. Assignment: We assign each data point to the nearest centroid based on the Euclidean distance or any other distance metric. This step creates k clusters.
$$l_i^{t+1} = \arg\min_l \|x^i - m_l\|^2$$

3. Update: We recalculate the centroids of the clusters by computing the mean of all the data points assigned to each cluster. The centroids represent the new centers of the clusters.
$$m_l = \frac{1}{|c_l|} \sum_{i \in c_l} x^i$$

4. Iteration: We repeat steps 2 and 3 until convergence or until a predefined number of iterations is reached. Convergence occurs when the centroids no longer change significantly or when we reach a minimum.

5. Result: The algorithm produces k clusters, each associated with a centroid representing the center of the cluster. The data points are assigned to the clusters based on their proximity to the respective centroids.

One of the obstacles to using K-means is regarding the choice of the number k. To determine the number of clusters (k), we use 5-fold or 10-fold **C**ross-**V**alidation (CV) using the silhouette score as the performance metric. The silhouette score is a metric used to evaluate the quality of clustering results. It measures the compactness and separation of clusters based on the distances between data points within and between clusters.

The silhouette score for a single data point i is calculated as follows:

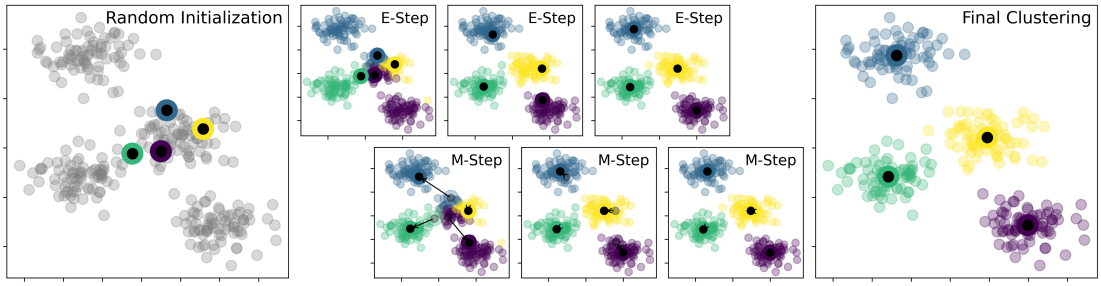


Figure 7: Visual explanation of the step-by-step K-means algorithm (VanderPlas, 2016).

1. Compute the average distance between the data point and all other data points within the same cluster A: $a(i) = \frac{1}{N} \sum_{x^k \in A} \|x^i - x^k\|^2$. This is known as the "intra-cluster distance" $a(i)$.

2. Compute the average distance between the data point and all data points in the nearest neighboring cluster B: $b(i) = \frac{1}{N} \sum_{x^k \in B} \|x^i - x^k\|^2$. This is known as the "inter-cluster distance" $b(i)$.

3. Calculate the silhouette coefficient for the data point using the formula:

$$S(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

.

The silhouette score S for a clustering result is the average of the silhouette coefficients $S(i)$ across all data points i. It ranges between -1 and 1, where:

- A score close to 1 indicates that the data points are well-clustered, with a clear separation between clusters and compactness within each cluster.
- A score close to 0 indicates overlapping clusters or that the data points are on or very close to the decision boundary between clusters.
- A score close to -1 indicates that the data points may have been assigned to the wrong clusters, with significant overlap and poor separation.

The silhouette score provides a quantitative measure to compare the different K-means clusterings using different choices of k. It helps in determining the optimal number of clusters. Moreover, we can also use this metric to evaluate the different modelling approaches that we build. It is important to note that the silhouette score should be interpreted in conjunction with a visual inspection of the clustering results, as it has limitations and may not always provide a complete assessment of the clustering quality.

2.1.6 Performance metrics

Clustering performance is key for evaluating the quality of TCR representations. To have a complete and full view of the performance of our models and methods, we will make use of 3 different metrics:

- Silhouette score
- Calinski-Harabasz index
- Davies-Bouldin index

As we have seen earlier, the Silhouette score measures the average distance of a sample with all other points in the next nearest cluster against all other points in its cluster. On the other hand, the Calinski-Harabasz index measures the sum of between-cluster dispersion $BGSS$ against the sum of within-cluster dispersion $WGSS$, where dispersion is the sum of distance squared:

$$CH = \frac{BGSS}{WGSS} \times \frac{N-K}{K-1}$$

where:

$$BGSS = \sum_{k=1}^{K} n_k \times ||C_k - C||^2$$

$$WGSS = \sum_{k=1}^{K} \sum_{i=1}^{n_k} ||X_{ik} - C_k||^2$$

with C_k the centroid of cluster k, C the centroid of the dataset and n_k the number of observations in cluster k.

Finally, the Davies-Bouldin index measures the between-cluster dispersion $BGSS$ against within-cluster dispersion $WGSS$:

$$DB = \frac{1}{K} \sum_{i=1}^{K} \max_{i \neq j} \frac{S_i + S_j}{d_{ij}}$$

where:

$$S_i = \left(\frac{1}{n_i} \sum_{j=1}^{n_k} ||X_j - C_i||^q \right)^{1/q}$$

$$d_{ij} = ||C_i - C_j||$$

using the same notation as for the Calinski-Harabasz index.

Metric	Range	Interpretation
Silhouette score	$[-1, 1]$	A higher score signifies better-defined clusters.
Calinski-Harabasz Index	$[0, \infty)$	A higher score signifies better-defined clusters.
Davies-Bouldin Index	$[0, \infty)$	A lower score signifies better-defined clusters.

Table 4: Description of the metrics using to evaluate the performance of the different methods.

Since each of these metrics has its own strengths and limitations, by comparing the various methods using all three metrics, we can effectively assess performance and gain a comprehensive understanding of the situation.

2.2 A simple AutoEncoder

The first model that we build, fit and evaluate is a simple AutoEncoder model. This seems like a natural choice as the AutoEncoder model is a type of artificial neural network that is commonly used for unsupervised learning tasks. It belongs to the family of feed-forward neural networks and is designed to encode and decode input data with the objective of reconstructing the original input as accurately as possible. Therefore, we will first apply this model to our input data previously defined in Figure 5.

2.2.1 Model architecture

The architecture of an AutoEncoder consists of two main components:

- an encoder g_ϕ
- a decoder f_θ

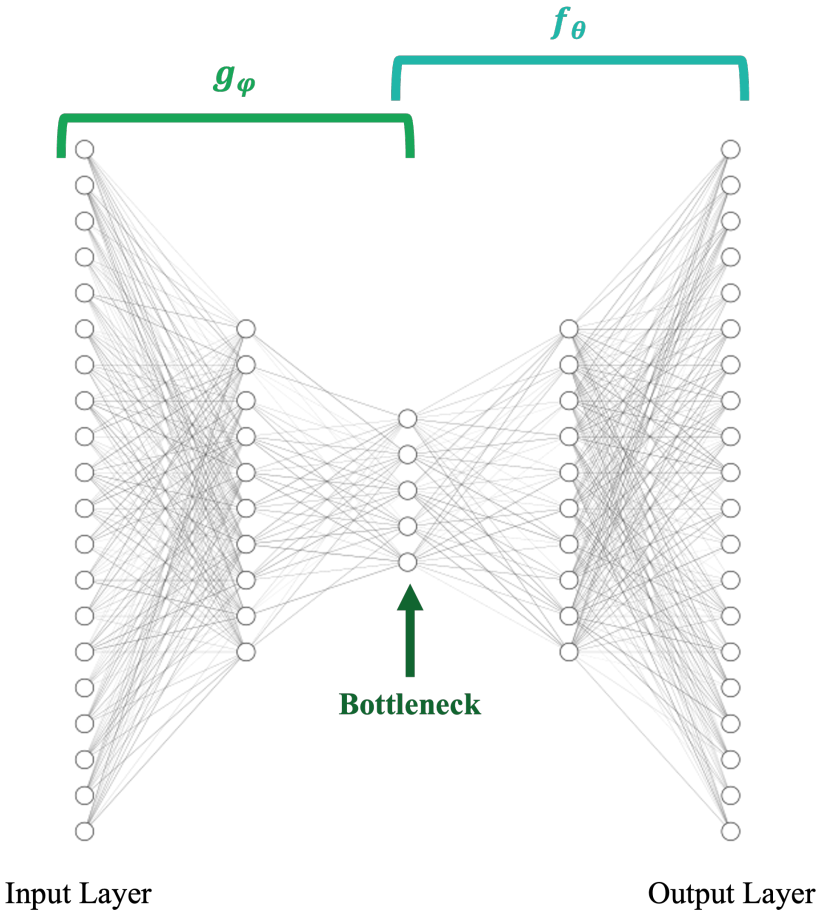


Figure 8: Diagram of a typical AutoEncoder architecture.

The encoder network takes the input data and maps it to a lower-dimensional representation, also known as a latent space. This mapping is achieved through a series of hidden layers that reduce the dimensionality of the input. The final layer of the encoder typically produces the compressed representation or code. For the simple AutoEncoder model, we will choose to have one input layer of size 20, one hidden layer of size 40 and a latent space of size 2 as it could be schematize in Figure 9.

The decoder network takes the encoded representation produced by the encoder and attempts to reconstruct the original input from it. Each layer has an increasing size until the output matches the dimensions of the input. The reconstruction is performed by applying an activation function to the output layer, which generates the output values.

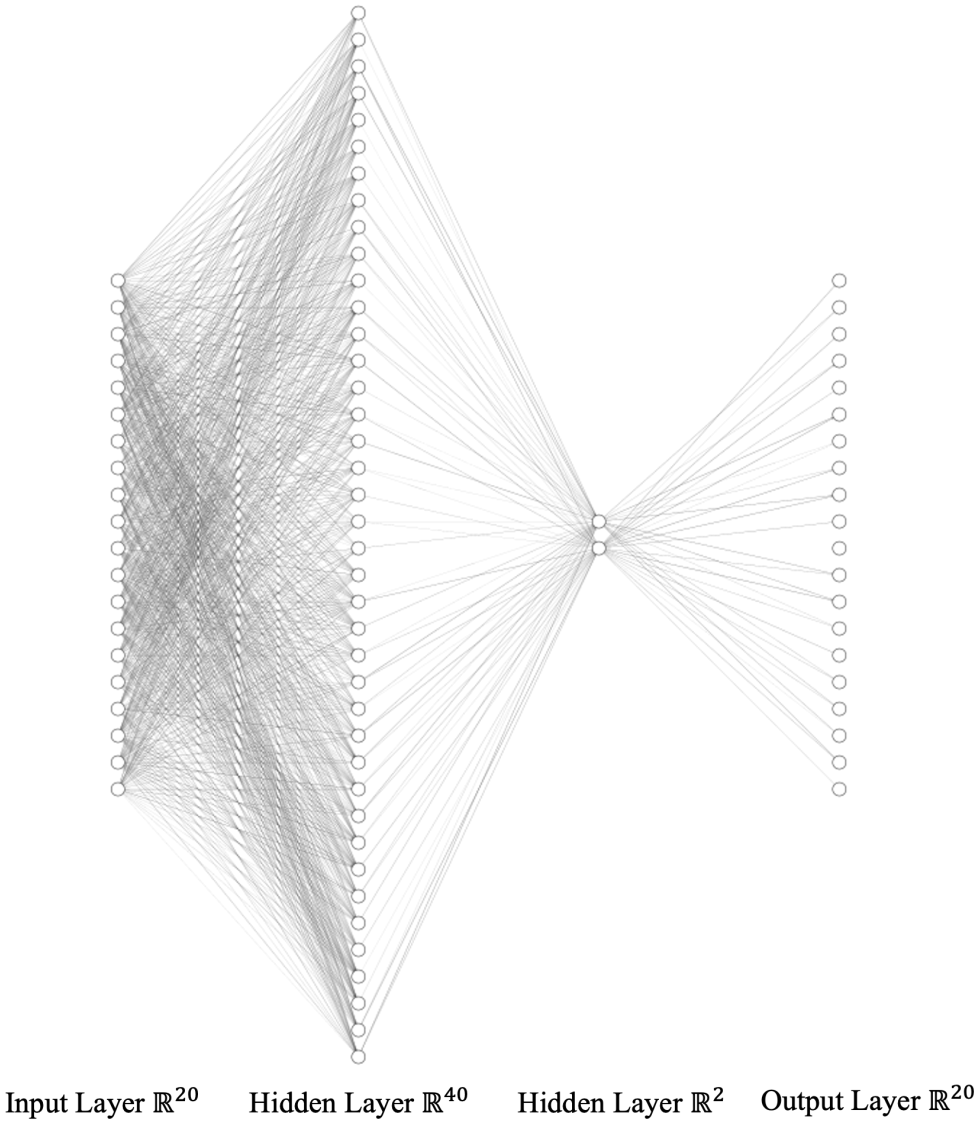


Figure 9: Model architecture of the simple AutoEncoder.

Each layer of our simple AutoEncoder uses **R**ectified **L**inear **U**nit (ReLU) activation functions (Figure 10):

$$ReLU(x) = \max(0, x)$$

The ReLU activation function is a popular choice for neural networks because of its simplicity and ability to handle non-linearities effectively. It allows the model to capture complex patterns and relationships in the data. In addition, ReLU activations help create sparse representations because they set negative values to zero, effectively suppressing irrelevant or noisy information.

One of the key features of the AutoEncoder is its ability to learn valuable representations or features from the data without explicit supervision. By compressing the

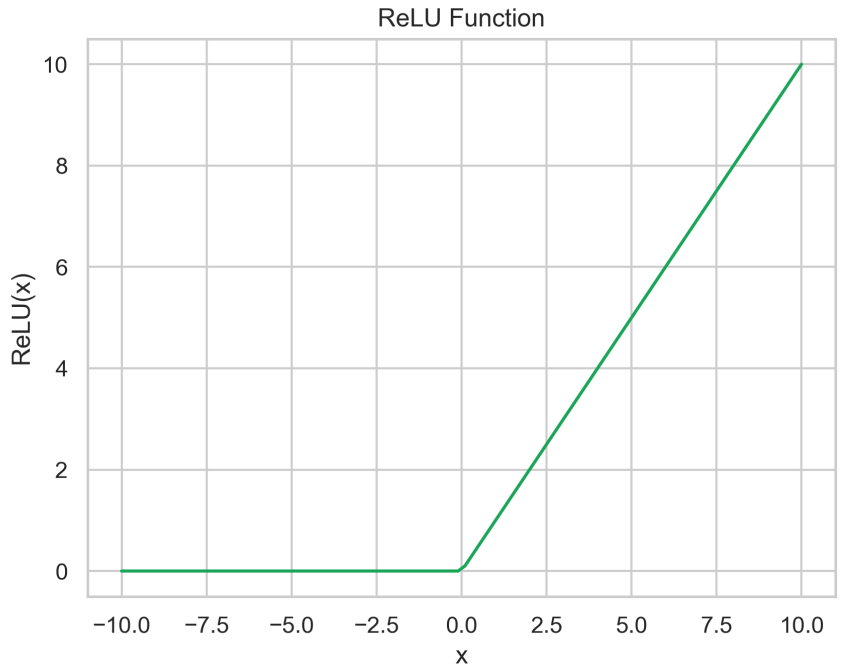


Figure 10: Plot of the ReLU function.

input into a lower dimensional space, the AutoEncoder automatically captures the important patterns and structures in the data.

Furthermore, AutoEncoders can also be used for generative purposes, where they are trained to generate new data similar to the input data. By sampling points from the latent space and passing them through the decoder, the AutoEncoder can produce new samples that have similar characteristics to the training data.

2.2.2 Training

During training, an AutoEncoder aims to minimize the reconstruction error, which is the difference between the original input and the reconstructed output. This is typically measured using a loss function. For this first model, we will use a simple and common loss function: the **M**ean **S**quared **E**rror (MSE). The training process involves adjusting the weights and biases of the network using gradient descent (or a similar optimizer), with the goal of reducing the reconstruction error:

$$\mathcal{L}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} (x^i - f_\theta(g_\phi(x^i)))^2$$

where g_ϕ is the encoder, f_θ is the decoder and ϕ and θ are a set of parameters or weights.

In fact, the gradient descent algorithm aims to minimize this loss function by iteratively updating the parameters of the model. The basic idea behind gradient descent is to move in the direction of steepest descent of the cost function to reach the minimum. By following the opposite direction of the gradient, we can iteratively

update the parameters to reach a minimum. We can describe the gradient descent algorithm step-by-step in the following way:

1. Initialize the parameters: Start by initializing the parameters of the model or function to some initial values: θ_0 and ϕ_0.

2. Calculate the cost function: Evaluate the loss function for the current parameter values. This function measures how well the model is performing, and we want to minimize it.

3. Calculate the gradients: Compute the gradients of the loss function $\nabla\mathcal{L}(\theta_t, \phi_t)$ with respect to each parameter. This step involves taking the partial derivatives of the cost function with respect to each parameter.

4. Update the parameters: Update the parameter values by taking a small step in the opposite direction of the gradients:

$$\theta_{t+1} = \theta_t - \eta_t \nabla\mathcal{L}(\theta_t, \phi_t)$$

$$\phi_{t+1} = \phi_t - \eta_t \nabla\mathcal{L}(\theta_t, \phi_t)$$

   This step is defined by the learning rate η_t, which determines the size of the steps taken during each iteration t.

5. Repeat steps 2-4: Iterate the process by calculating the loss function, gradients, and updating the parameters until convergence is achieved. Convergence occurs when the cost function reaches a minimum or when a predefined stopping criterion is met (e.g., a maximum number of iterations).

In particular, in our case, we will use a famous and more advanced optimizer that uses a similar logic as the simple gradient descent: the Adam optimizer (Adaptive Moment Estimation). It is an optimization algorithm that combines ideas from both the gradient descent and adaptive learning rate methods. The Adam algorithm maintains adaptive learning rates for each parameter by considering both the first-order (∇) and second-order (∇^2) moments:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

$$\phi_{t+1} = \phi_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

where $\hat{v}_t$ and $\hat{m}_t$ are the estimates of the first and second moments.

Adam is a popular optimization algorithm for training neural networks, as its adaptive learning rate mechanism automatically adjusts the step size for each parameter individually based on the magnitude of the gradients and the history of the moment estimates (Figure 11). This helps in efficiently navigating complex loss landscapes and speeding up convergence. This essential feature will allow us to ensure that our model accurately and quickly converges to a minima. We train our model in this way for 10 epochs.

Figure 11: Schematization of the Gradient Descent and the Adam Optimizer algorithms.

2.3 An optimized deep AutoEncoder

The previous AutoEncoder model discussed earlier is characterized by its simplicity and smaller architecture size. These uncomplicated models often have significant advantages, such as a reduced risk of over-fitting to the data. However, there is a trade-off as our model may be too simple and may not capture all the information contained in the data due to its limited structure. To address this concern, we decided to construct a larger AutoEncoder model with an optimized architecture and hyperparameters. Although this deeper AutoEncoder model is expected to show some degree of over-fitting to the training data, we may find that its performance exceeds that of the previous model.

2.3.1 Model architecture

As mentioned above, for this iteration, we develop an extended deep AutoEncoder model with a greater number of layers and larger layer sizes. Another notable difference is the inclusion of additional inputs, namely the v-gene and j-gene of a TCR, in addition to its CDR3 sequence. We expect this inclusion of additional information to have a significant impact on both the complexity and performance of our model, as the v and j genes contain crucial information regarding the specificity of a TCR.

In this iteration, our model is constructed with a dense layer of 87 units, followed by two substantial layers of 75 units each. In addition, the bottleneck of our autoencoder model has been significantly expanded and now consists of 20 units. Figure 12 illustrates the architecture, where the decoder section consists of a layer of 40 units followed by an output layer of size 20.

2.3.2 Training

During the training of our deep Autoencoder, we maintain consistency by using the same loss function, Mean Squared Error (MSE), and optimizer, Adam optimizer. However, a notable change this time is the inclusion of hyperparameters tuning for our model. This involves optimizing parameters such as the learning rate λ and the batch size, allowing us to further improve the performance and effectiveness of our model. We optimize these hyperparameters depending on the MSE loss value that we obtain.

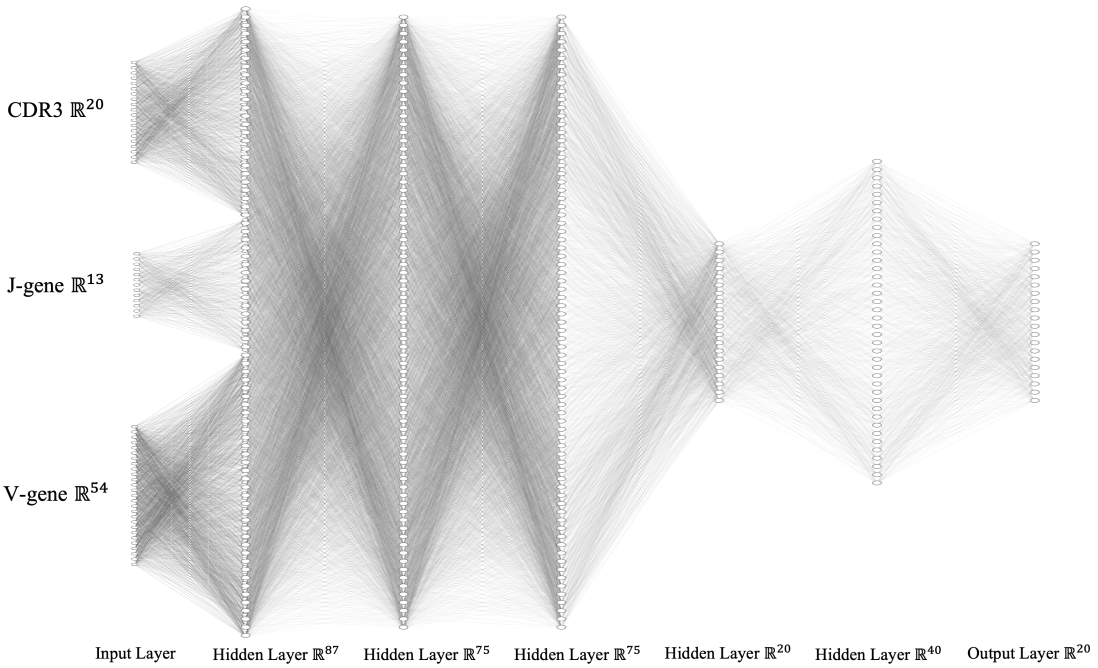


Figure 12: Model architecture of the optimized deep AutoEncoder.

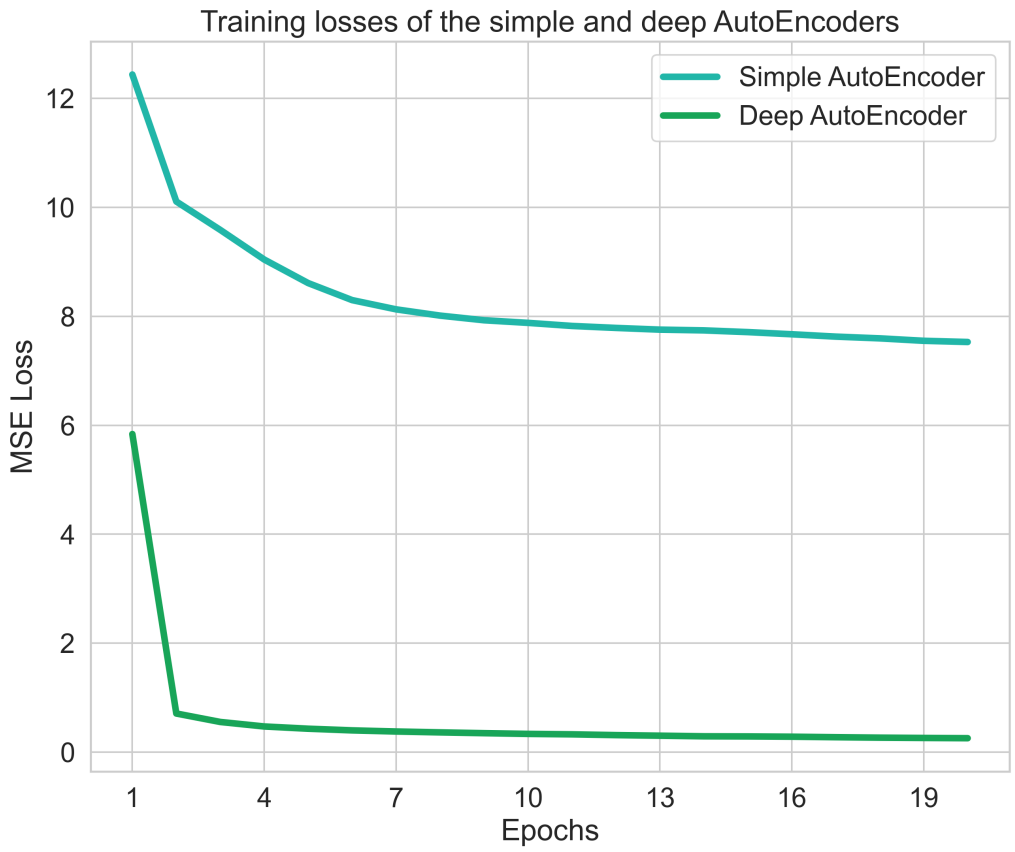


Figure 13: Training losses of the simple and deep AutoEncoders.

As depicted in Figure 13, it is evident that our deep AutoEncoder achieves a significantly lower loss value at a much faster pace compared to the simple AutoEncoder.

This outcome was expected since the deep AutoEncoder, with its increased complexity, possesses a greater capacity to capture and represent the patterns in the training data effectively. Moreover, the deep AutoEncoder also makes use of some additional information with the v-gene and j-gene tags. By observing Figure 13, we can identify a potential risk: our model is prone to substantial over-fitting to the training data, as indicated by the remarkably low MSE loss achieved as early as the 4^{th} epoch. However, when training the model for much more epochs ($\approx$ 200 epochs), we notice in Figure 14 that the deep AutoEncoder does not really overfit as its validation MSE decreases and remains relatively stable through the epochs.

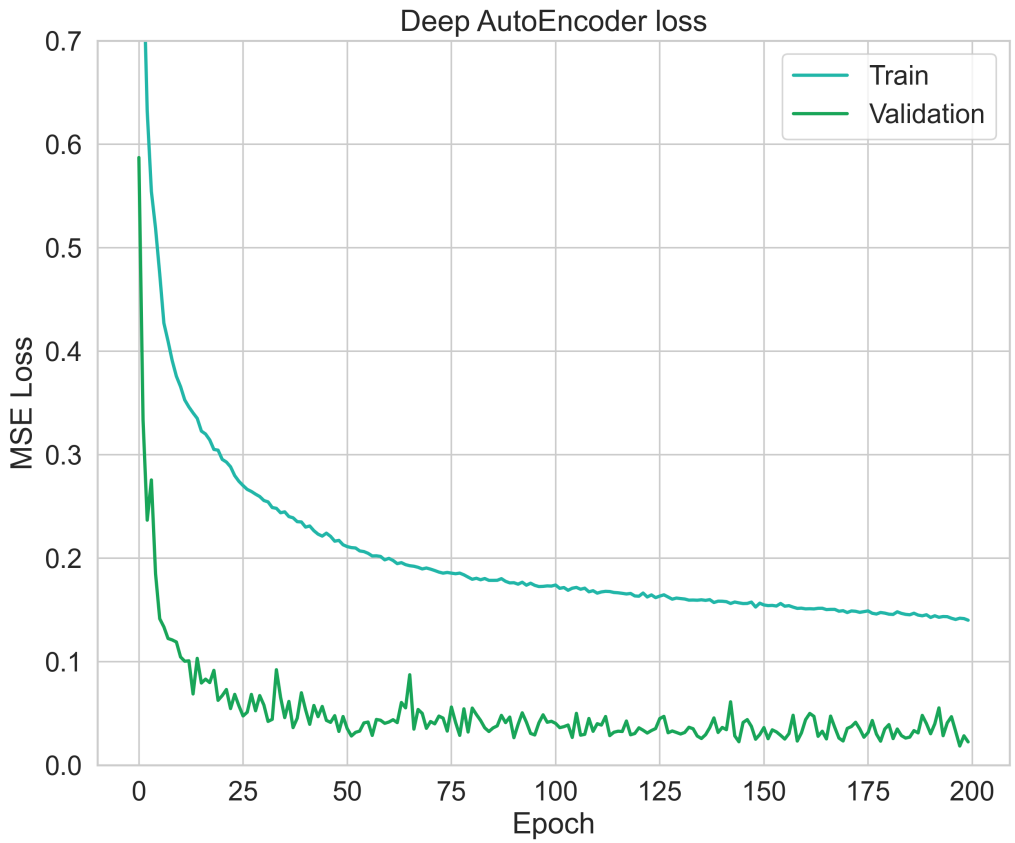


Figure 14: Training and validation losses through a large number of epochs of training for the deep AutoEncoder model.

2.4 A Variational AutoEncoder

The **V**ariational **A**uto**E**ncoder (VAE) is a type of generative model that combines concepts from traditional AutoEncoders and variational inference. It is also specifically designed to generate new samples that resemble the input data distribution.

Similarly as in the classic AutoEncoder, in a VAE, the encoder network takes an input data point and maps it to a latent space representation. This time, however, this lower dimensional bottleneck is characterized by a mean and a variance. In essence, the encoder learns to capture the salient features of the input data and represents them in the form of a probability distribution in the latent space (Kingma and Welling, 2022).

One distinctive feature of the Variational Autoencoder is its emphasis on imposing a normal distribution structure on the latent space. This is achieved by incorporating the **K**ullback-**L**eibler (KL) divergence term into the VAE's objective function. By including this term, the VAE encourages the learned latent space to follow a desired prior distribution, typically a standard Gaussian. This constraint ensures that the latent representations have a smooth and continuous nature, allowing for meaningful interpolations and a more interpretable latent space. The VAE's focus on enforcing normal distributions in the latent space enables it to capture the underlying statistical properties of the data, facilitating the generation of diverse and realistic samples. This also facilitates the effective clustering of the data points by promoting normal distributions in the latent space. This clustering property enables the VAE to capture underlying patterns and structure in the data, further enhancing the model's ability to generate coherent and meaningful samples based on different modes of the data distribution.

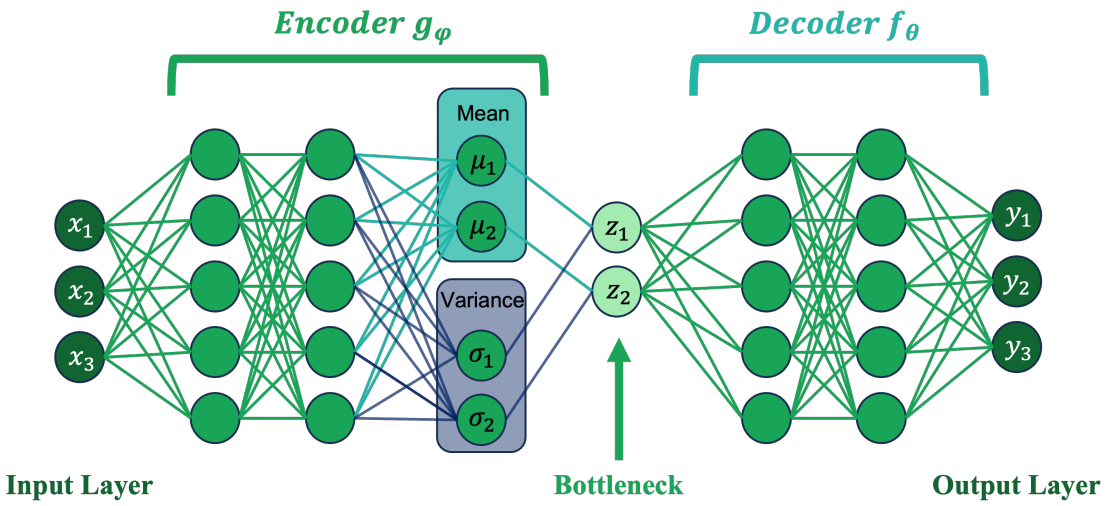


Figure 15: General structure of a typical Variational AutoEncoder model (VAE).

In general, the VAE model offers several advantages: it allows for unsupervised learning of complex distributions, facilitates meaningful latent space interpolations, and supports the generation of novel samples. This is particularly relevant for our study as this model architecture could greatly improve our embedding and clustering performance.

2.4.1 Model architecture

In general, a Variational AutoEncoder (VAE) has a similar structure to the one of a traditional AutoEncoder. The architecture of a VAE consists of two main components: an encoder network and a decoder network. As depicted in Figure 15, the encoder network takes the input data and maps it to the latent space. However, for the VAE, the final layers of the encoder produce the mean and variance parameters that describe the probability distribution of the latent space. On the other hand, the decoder network takes the sampled latent vectors and tries to reconstruct the original input data. The decoder progressively up-samples the latent vectors

to match the size and complexity of the input data, ultimately producing a reconstructed output.

More specifically, in our study, we will reproduce and use the architecture developed in a previous study (Davidsen et al., 2019). This architecture has the advantage of having already been optimized for a similar task. In particular, it has been shown that this specific architecture is able to accurately learn some important features of the data. This architecture is illustrated in Figure 16 and consists of:

1. 3 Input layers for the CDR3 sequence, the v-gene and the j-gene
2. An embedding of each input element
3. A concatenation of the 3 elements' embeddings into one matrix
4. Dense layers that deduce important information from the data
5. The construction of the mean and log variance of the latent space probability distribution
6. Sampling of the latent space vector
7. Dense layers of the decoder network that try to retrieve the previous inputs
8. Dissociation and decoding of each element
9. 3 Output layers for the CDR3 sequence, the v-gene and the j-gene

2.4.2 Training

The training of the VAE involves optimizing the parameters of both the encoder and decoder networks. We optimize those parameters in terms of a weighted combination of 3 objective functions for the 3 inputs and outputs of our VAE model:

- The main objective function for the CDR3 sequence $\mathcal{L}(\theta, \phi)$
- The cross-entropy loss function for the v-gene $\mathcal{L}_{CL}^{V}$
- The cross-entropy loss function for the j-gene $\mathcal{L}_{CL}^{J}$

The main objective function for the CDR3 sequence is a combination of two terms: the reconstruction loss, which measures the discrepancy between the input data and the generated output, and the KL divergence, which regularizes the latent space to follow a desired prior standard Gaussian distribution. The KL-divergence between two probability distributions simply measures how much they diverge from each other. We define the objective function mathematically as follows:

$$\mathcal{L}(\theta, \phi) = \mathcal{R}(x^i, f_\theta(g_\phi(x^i))) + \beta D_{KL}(q_\phi(g_\phi(x^i)|x^i)||p_\theta(g_\phi(x^i)))$$

where β is chosen constant, $\mathcal{R}(x^i, f_\theta(g_\phi(x^i)))$ is the reconstruction loss and D_{KL} is the KL-divergence:

$$D_{KL}(q_\phi(z^k)|x^k)||p_\theta(z^k)) = \sum_{z_i \in z^k} q_\phi(z_i) \log\left(\frac{q_\phi(z_i)}{p_\theta(z_i)}\right) = -\sum_{z_i \in z^k} q_\phi(z_i) \log\left(\frac{p_\theta(z_i)}{q_\phi(z_i)}\right)$$

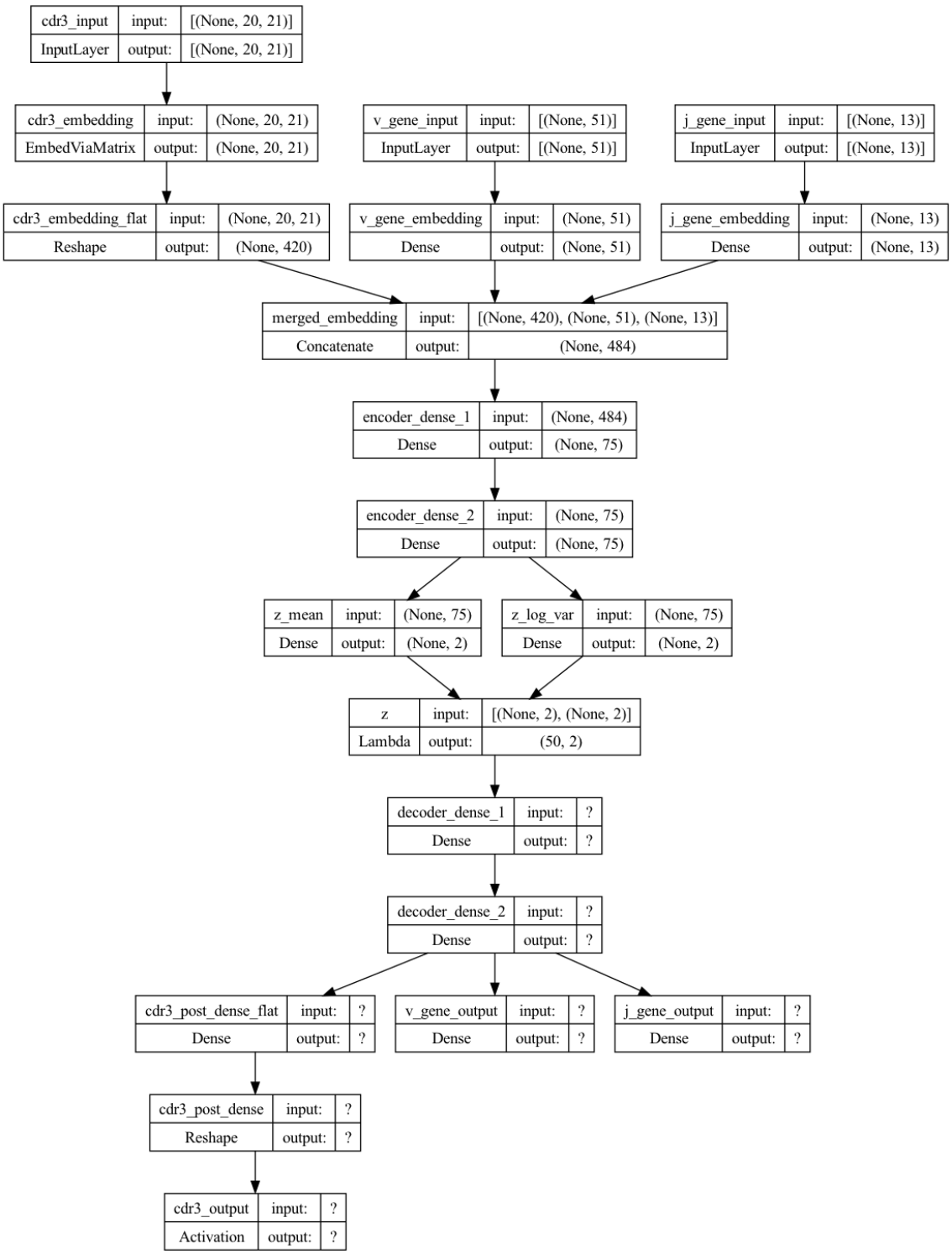


Figure 16: Architecture of the VAE (Davidsen et al., 2019).

which can be simplified for the Variational AutoEncoder as it uses the Gaussian distribution (Kingma and Welling, 2022):

$$D_{KL}(q_\phi(z^k)|x^k)||p_\theta(z^k)) = \frac{1}{2}\sum_{i=1}^{N}(1 + log(\sigma_i^2) - (\mu_i^2) - (\sigma_i^2))$$

On the other hand, for this VAE model, we choose the reconstruction loss to be the cross-entropy similarly as for the v-gene and j-gene:

$$\mathcal{L}_{CL} = -\sum_{i=1}^{n}\sum_{c=1}^{C} \mathbb{1}_{f_\theta(g_\phi(x_i))\in C_c} \log(p(f_\theta(g_\phi(x_i)) \in C_c))$$

$$= -\sum_{i=1}^{n}\sum_{c=1}^{C} \mathbb{1}_{f_\theta(z_i)\in C_c} \log(p(f_\theta(z_i) \in C_c))$$

$$= -\sum_{i=1}^{n}\sum_{c=1}^{C} \mathbb{1}_{y_i\in C_c} \log(p(y_i \in C_c))$$

These loss functions are combined to construct a unique weighted objective function:

$$\mathrm{L}(\theta, \phi) = w_1\mathcal{L}(\theta, \phi) + w_2\mathcal{L}_{CL}^{V} + w_3\mathcal{L}_{CL}^{J}$$

The weights used are chosen to be the optimized ones found in a previous study (Davidsen et al., 2019):

$$w_1 = 1, w_2 = 0.8138, w_3 = 0.1305$$

We jointly optimize our parameters θ and ϕ to minimize this objective function using as before the Adam optimizer for 100 epochs. As we can observe in Figure 17, the three losses seem to decrease likewise through the epochs. We also note that through the 100 epochs, the combined objective function for the validation set decreases significantly, thus justifying the choice of such a high number of epochs for the training of the model.

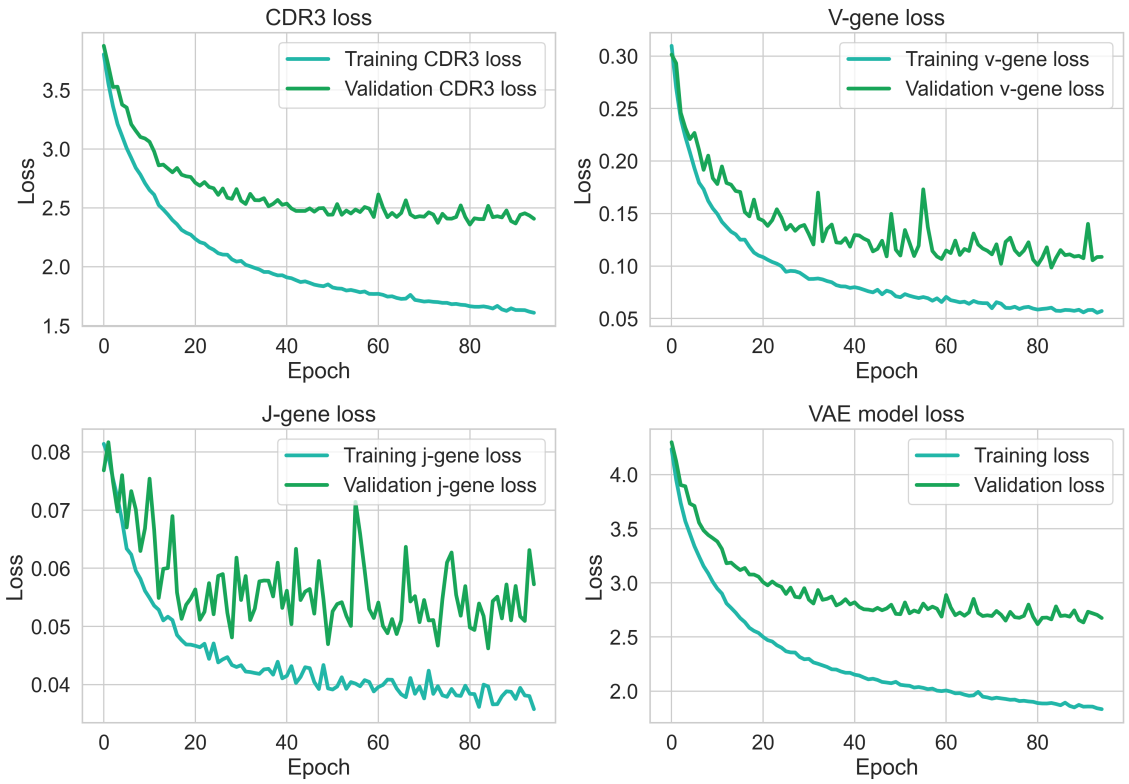


Figure 17: Training and validation losses of the Variational AutoEncoder model.

An critical aspect during the model training process is the computation of the relationship between each parameter in the network and the final output loss. In general, for neural networks, this calculation is achieved by a technique called backpropagation. However, applying backpropagation to a random sampling process is not feasible. Fortunately, there is a clever approach called the "reparameterization trick" that provides a solution. This trick, described in Figure 18 involves randomly sampling from a unit Gaussian distribution and then adjusting the sampled values by the mean and variance of the latent distribution. By using this reparameterisation, we can optimize the parameters of the distribution while retaining the ability to generate random samples from it.

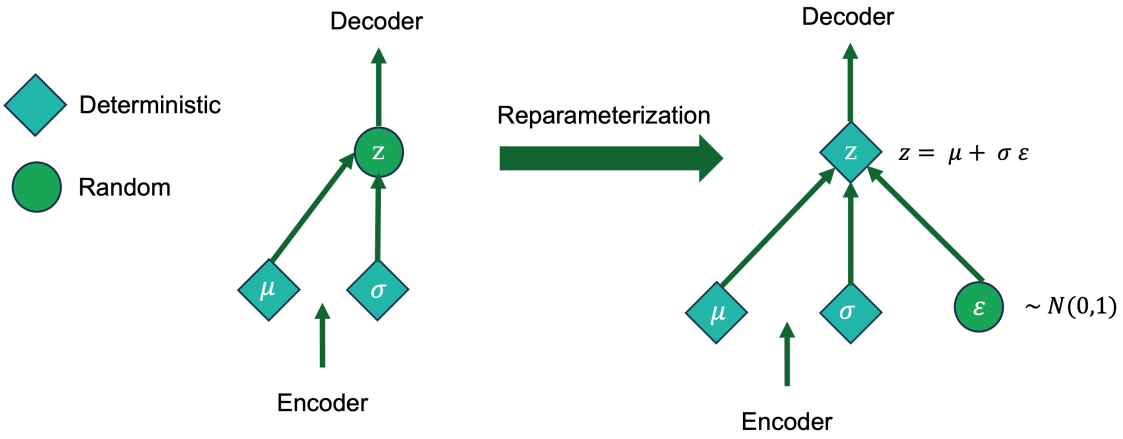


Figure 18: Schematization of the reparameterization trick for the VAE model.

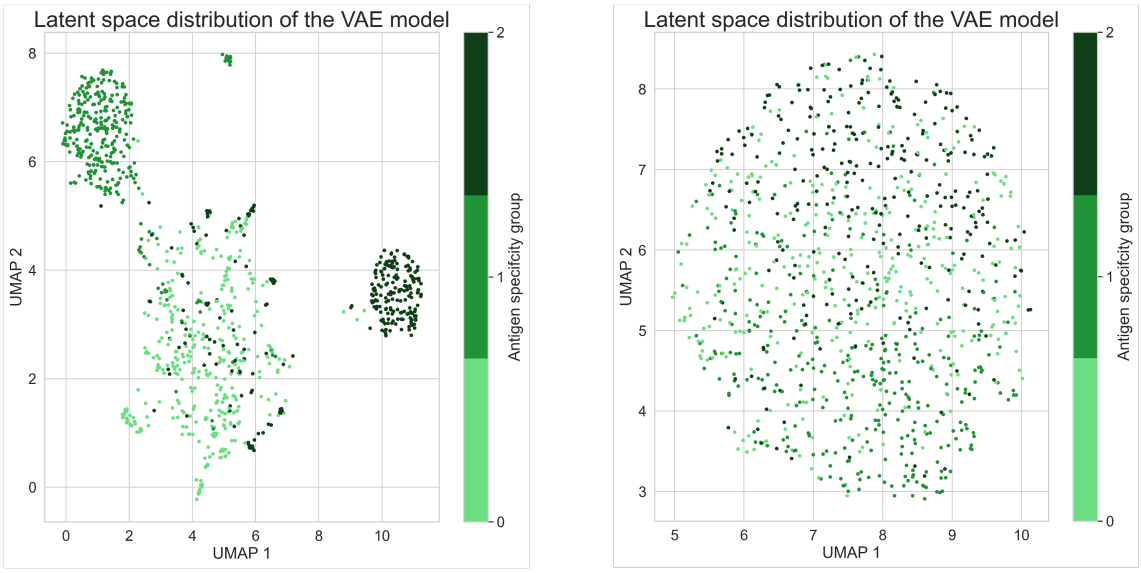
Another technique used in our model addresses the challenge of the network learning negative values for the standard deviation value. To overcome this problem, we train the network to learn the logarithm of the standard deviation ($\log(\sigma)$) instead.

2.4.3 Choosing the entanglement parameter

In the previous section, we have defined the objective function for the VAE model using a constant β. This constant β is essential for our VAE model and is often called the *entanglement*. By varying its value between 0 and 1, we attach more or less importance to the cross-entropy loss or the KL-divergence. We have for example as in Figure 19:

- If β is close to 0, our VAE model is mainly optimized to minimize the cross-entropy loss. Therefore, the embeddings on a 2D latent space reveal the formation of distinct clusters. However, we can observe that the latent space is highly discontinuous greatly limiting the relevancy and performance of our model.

- If β is close to 1, our VAE model optimizes the KL-divergence loss. Intuitively, this loss encourages the encoder to distribute all embeddings evenly around the center of the latent space. If it tries to "cheat" by clustering them apart into specific regions in the space (e.g., away from the origin) it will be penalized. This results in a model that has a poor clustering performance even though its probability distribution space seems to be centered and continuous.



$\beta = 0.1$

Mainly cross-entropy loss

$\beta = 0.9$

Mainly KL-divergence loss

Figure 19: Latent space distributions for a VAE model purely optimized for reconstruction loss or for KL-divergence loss.

On the other hand, optimizing the two losses together with a balanced objective function results in the generation of a latent space that maintains the notion of similarity or dissimilarity of nearby embeddings on the local scale via clustering. Moreover, in this setting, the embeddings are still very densely packed near the latent space origin as it is depicted in Figure 20. Therefore, choosing the correct value of β is essential to maintain these important performance properties. For our dataset, this corresponds to choosing a value of β around 0.2 ($\beta \approx 0.2$).

2.5 Transfer Learning and Transformers

2.5.1 Advantages of employing NLP techniques for our task

TCR sequence analysis can greatly benefit from the use of NLP-like methods and techniques. In fact, there are a number of similarities between NLP tasks and TCR sequence analysis that make NLP models well-suited and that could significantly improve our predictive performance:

- First, both NLP tasks and TCR sequence analysis involve the examination of sequential data with contextual dependencies. NLP models, particularly transformers, have been shown to be highly effective at capturing patterns and dependencies within sequences of symbols, be they words in sentences or amino acids in TCR chains. NLP models also excel at understanding the surrounding context of words in a sentence, while in TCR sequences the arrangement of amino acids influences the function of the receptor. The ability of NLP models to capture long-range dependencies and contextual information can be used to interpret the complex relationships within TCR sequences.

- Transfer learning is a technique widely used in NLP and can be applied to TCR

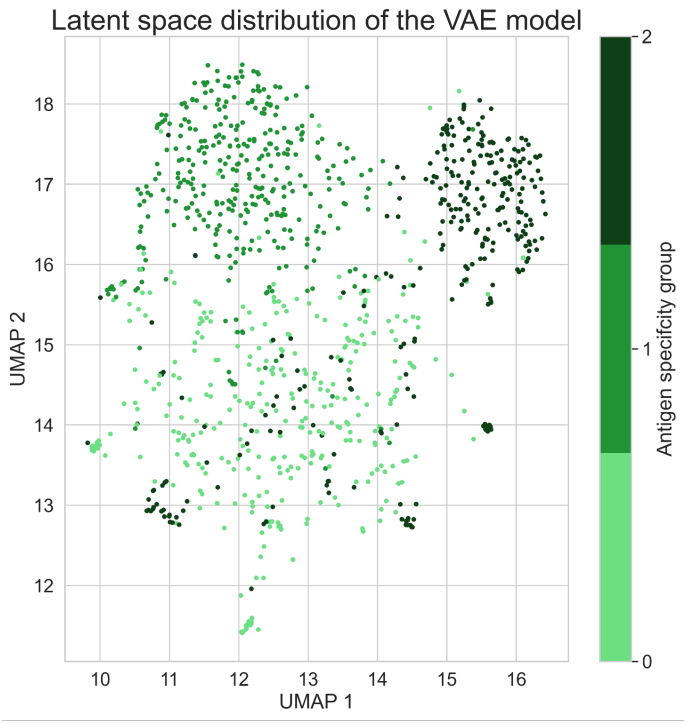


Figure 20: Latent space distribution for a VAE model evenly optimized for reconstruction loss and for KL-divergence loss.

sequence analysis. NLP models are often pre-trained on a large dataset and then fine-tuned for specific tasks. On the other hand, TCR sequence analysis faces challenges in obtaining labeled data due to the experimental limitations mentioned earlier in the introductory section 1. By adapting transfer learning techniques from NLP, pre-trained models can capture general patterns in TCR sequences and be fine-tuned for specific downstream tasks, even with limited labeled data.

- NLP models are known for their ability to learn meaningful representations of text. Similarly, TCR sequences carry important information about the structure, function and antigen recognition of T cell receptors. By adapting NLP models, we could learn rich representations of TCR sequences, allowing the extraction of important features and patterns that aid in tasks such as TCR clustering, classification or prediction.

Therefore, the use of NLP-like models for TCR sequence analysis is fully justified and relevant to our study. The context dependencies, transfer learning capabilities, representation learning and availability of unlabeled data in both NLP and TCR sequence analysis provide a solid foundation for adapting and applying NLP models to improve the classification and prediction of TCR sequences.

2.5.2 Impact and benefits of Transformers and Transfer Learning

Transformer models are a type of deep learning model that has gained significant attention and popularity in Natural Language Processing (NLP) tasks. They were

introduced in a 2017 paper (Vaswani et al., 2017) and have since become the backbone of many state-of-the-art NLP systems. Their ability to capture contextual relationships and dependencies within text has led to remarkable advances in tasks such as machine translation, sentiment analysis, text generation and more.

Transformers are a type of NLP models that has revolutionized the field of NLP by overcoming some of the limitations of traditional sequence-based models such as Recurrent Neural Networks (RNNs). They use a self-attention mechanism that allows them to capture dependencies between different words in a sentence or sequence. This mechanism allows the model to focus on different parts of the input sequence when making predictions, without the need for explicit sequential processing.

Like AutoEncoders or Variational AutoEncoders, transformers consist of an encoder and a decoder. The encoder takes an input sequence and encodes it into a set of hidden representations, while the decoder generates an output sequence based on these representations.

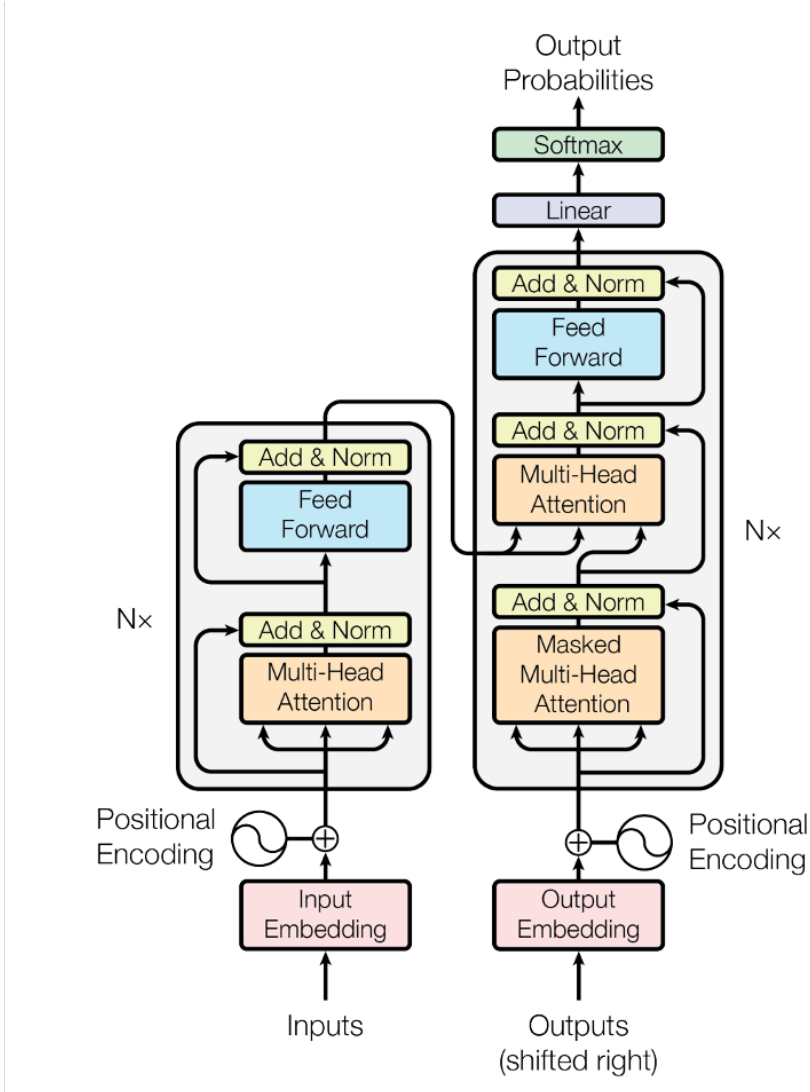


Figure 21: General architecture of a Transformers model (Vaswani et al., 2017).

Transformers have become essential for a number of reasons:

- Parallelization: Transformers can process inputs in parallel, making them

faster and more efficient, particularly for long sequences.

- Long-term Dependencies: Transformers can capture long-term dependencies. This makes them more effective in tasks that require understanding context over a span of characters.

- Attention Mechanism: The self-attention mechanism in transformers allows the model to attend to relevant parts of the input, resulting in better contextual representation and more accurate predictions.

- Transfer Learning: Transformers have played a pivotal role in transfer learning, where models pre-trained on large datasets can be fine-tuned on specific downstream tasks with smaller datasets. This approach has significantly reduced the need for massive amounts of labeled data and improved the performance of unsupervised models.

In particular, the attention mechanism is an essential feature of the Transformers models. We can think of the attention mechanism, or self-attention, as a mechanism that enhances the information content of an input embedding by incorporating information about the input's environment. As shown in Figure 21, this mechanism is used multiple times through *"Multi-Head Attention"* steps. Self-Attention compares all input sequence members with each other and modifies the corresponding output sequence positions. In other words, the self-attention layer differentiatively key-value searches the input sequence for each input, and adds the results to the output sequence. Mathematically, self-attention is calculated using dot products between query, key and value vectors, which are linear projections of the input sequence:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where:

1. Q, K, V are matrices packing together sets of queries, keys, and values, respectively.

2. k which takes the value of each of the elements of the input, corresponds to a key.

3. d_k is the dimension for each key k.

The weighted values are then combined to form the output representation. Self-attention allows transformers to capture dependencies between different positions in the sequence, providing a flexible and powerful way to model relationships.

The multi-head attention mechanism in transformers and described in Figure 22 involves multiple linear projections of queries, keys and values. The resulting projections are processed in parallel by a single attention mechanism. These intermediate results are concatenated and projected again to produce a final result. This multi-head attention approach enhances the model's ability to capture diverse relationships and dependencies within the input sequence, leading to more comprehensive and expressive representations.

On the other hand, the use of transfer learning in our task can significantly improve accuracy and efficiency by leveraging knowledge from pre-trained models, reducing

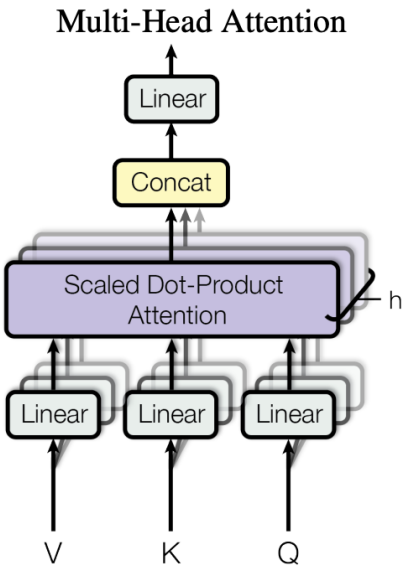


Figure 22: Multi-Head attention mechanism (Vaswani et al., 2017).

data requirements, and capturing complex relationships through domain-specific knowledge transfer. This approach improves the generalization, scalability and biological relevance of TCR clustering, leading to a deeper understanding of TCR repertoires and facilitating the development of personalized immunotherapies.

2.5.3 TCR-BERT

Therefore, in this section we will present and use a Transformer model that mimics the usual architecture of an NLP model, but has been adapted for TCR specificity analysis: TCR-BERT (Wu et al., 2021). In fact, similar to the AutoEncoder and the Variational AutoEncoder, TCR-BERT uses unlabelled TCR sequences to learn a versatile representation of TCR sequences in a latent space. BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) models excel in capturing the contextual relationships and semantic understanding of language. In the case of TCR-BERT, the pre-training process allows it to learn the grammatical structure and patterns within TCR sequences. While this does not directly encode TCR specificity information, it enables TCR-BERT to capture relevant sequence features and dependencies that may contribute to specificity. Using a similar method as before, we can then leverage UMAP to perform dimensionality reduction and K-means for clustering.

Specifically, the goal of the TCR-BERT model is to create a continuous embedding of T-cell receptor sequences that can be used for a variety of subsequent tasks. Prior to training, the TCR-BERT model learns to predict hidden amino acids based on the context around them, capturing the structural patterns in TCR sequences. Pre-training is performed on a large corpus of sequences, without regard to the understanding of antigen binding affinities. The model is then further trained to predict the antigen from a collection of 45 antigen labels to which a given amino acid sequence will bind. It is shown in this research that TCR-BERT can be used for various tasks such as antigen binding prediction and TCR clustering for in-depth TCR analysis.

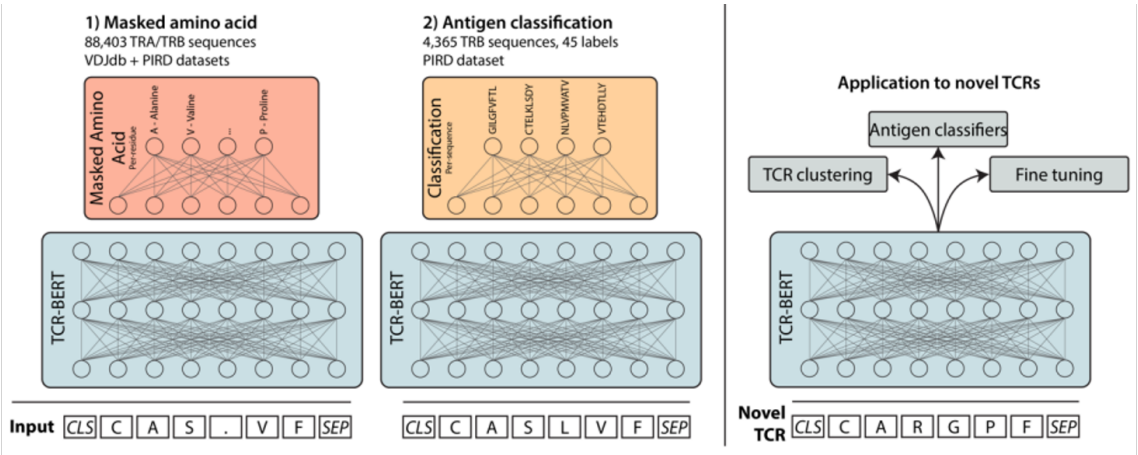


Figure 23: TCR-BERT self-supervised training (Wu et al., 2021).

We will make use of the pre-trained version of TCR-BERT freely available online on the *Hugging Face* platform. Through the Hugging Face library and platform, we can easily leverage pre-trained models for various tasks. Moreover, *Hugging Face*'s user-friendly interfaces, documentation, and model repositories provide a seamless experience, enabling us to harness the power of advanced models with ease.

2.5.4 Pre-trained vs Fine-tuned

We now want to compare the performance of two Transformers models in the context of TCR analysis, focusing specifically on the analysis of TCR sequences associated with Sars-Cov-2. The first model we can evaluate is the pre-trained TCR-BERT without fine-tuning to assess its performance on our Sars-Cov-2 dataset. This helps us to understand how well the pre-trained TCR-BERT captures the relevant features and patterns in the TCR sequences related to Sars-Cov-2. From Figure 24, this model performs relatively well for clustering our TCR sequences. We obtain quite clear and distinct clusters with a relatively high silhouette score. In addition, we can inspect the logo plots of each cluster as they reveal the patterns in the sequences of each cluster. We notice clear patterns: sequences that end with 'EAFF' tend to have small *UMAP 1* embeddings while sequences ending with 'EQYF' tend to have larger *UMAP 1* embeddings. However, as the model was trained on a different dataset and is not optimized to understand well the specific structures and similarities of TCR sequences related to Sars-Cov-2, we could improve its performance further.

The second model we examine is the fine-tuned version of TCR-BERT using $100,000$ TCR sequences from our dataset. Indeed, fine-tuning involves training further the pre-trained TCR-BERT model using our specific Sars-Cov-2 dataset. By fine-tuning the model, we aim to enhance its performance and adapt it to the specific characteristics and nuances of the TCR sequences associated with Sars-Cov-2 while still leveraging some essential prior knowledge. We produce the same method as previously to build Figure 25. By computing some clustering performance metrics, we can deduce that this fine-tuned model would perform slightly better for our task. Moreover, we notice similar behaviors and patterns as for the pre-trained only model: sequences that end with 'EAFF' tend to have small *UMAP 2* embeddings while

Figure 24: K-means clustering of the embeddings from TCR-BERT along with the corresponding logos for each cluster (without fine-tuning).

sequences ending with 'EQYF' tend to have larger *UMAP 2* embeddings.

By comparing the performance of the pre-trained TCR-BERT and the fine-tuned TCR-BERT model on our Sars-Cov-2 dataset, we have gained insights into the effectiveness of fine-tuning for TCR analysis tasks. Even though, the performance improvement is only small, this could have important impacts and consequences on future downstream tasks.

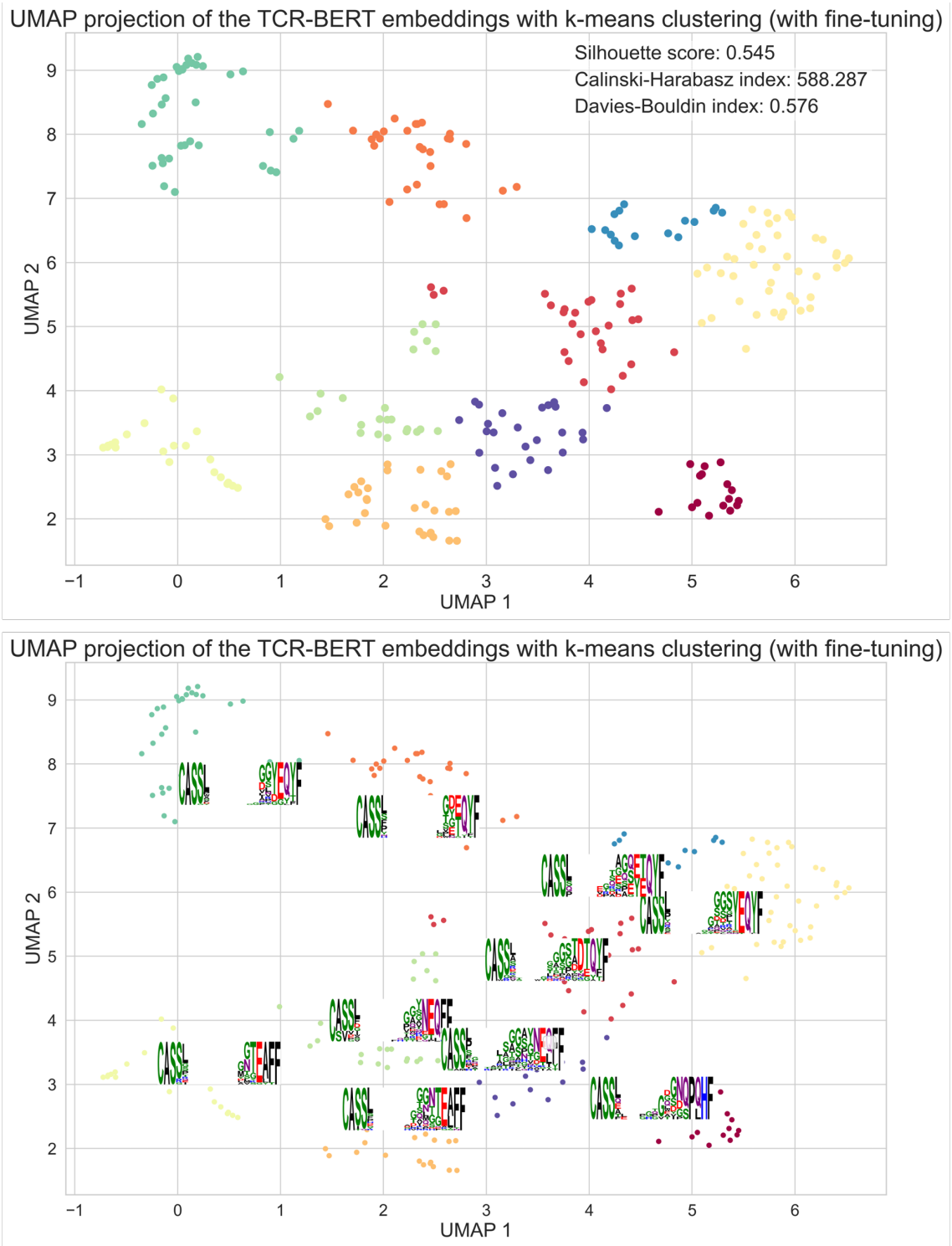


Figure 25: K-means clustering of the embeddings from the fine-tuned TCR-BERT along with the corresponding logos for each cluster.

3 Results

3.1 Comparison of results

In this section we compare the results obtained by applying the different methods and models described above in section 2 to our dataset. As mentioned in section 2.1.6, the performance of each method is evaluated using three metrics: the Silhouette score, the Calinski-Harabasz index and the Davies-Bouldin index. We compute these metrics as part of our pipeline, following the steps described in the 2.1 section. We then plot our results for the Silhouette score and the Davies-Bouldin index in Figure 26, and provide a comparison table for the Calinski-Harabasz index in Table 5.

Model	Calinski-Harabasz index
Simple AutoEncoder	1009.71
Deep AutoEncoder	3769.26
Variational AutoEncoder	1712.20
Fine-tuned Transformer	1506.37

Table 5: Calinski-Harabasz index value for each model investigated in our study.

Looking closely at the results, we can see that the Deep AutoEncoder and the Variational AutoEncoder consistently outperform the Simple AutoEncoder on all three metrics. Both methods achieve higher Silhouette scores, indicating better cluster separation, and lower Davies-Bouldin scores, indicating more compact and well-separated clusters. In addition, the Calinski-Harabasz score is significantly higher for the deep AutoEncoder and the VAE, indicating better cluster quality.

Among the methods, the VAE shows the best overall performance, with the highest Silhouette score and the lowest Davies-Bouldin score. These results suggest that the VAE's probabilistic encoding contributes to its superior performance in capturing TCR specificity patterns. Therefore, the VAE represents a promising solution for the effective clustering of TCR sequences.

Furthermore, the TCR-BERT method, which uses transfer learning and transformers, achieves competitive results compared to the other methods. It achieves moderate Silhouette and Davies-Bouldin scores and a relatively high Calinski-Harabasz score, indicating its effectiveness in capturing TCR specificity patterns and clustering TCR sequences. It is reasonable to assume that this fine-tuned model could be even more powerful if trained on an even larger dataset.

Overall, the deep AutoEncoder, the Variational AutoEncoder and the fine-tuned TCR-BERT methods show promise in capturing TCR specificity patterns and clustering TCR sequences effectively. However, the VAE stands out as the most successful method, demonstrating its strong potential to accurately and confidently cluster TCR sequences.

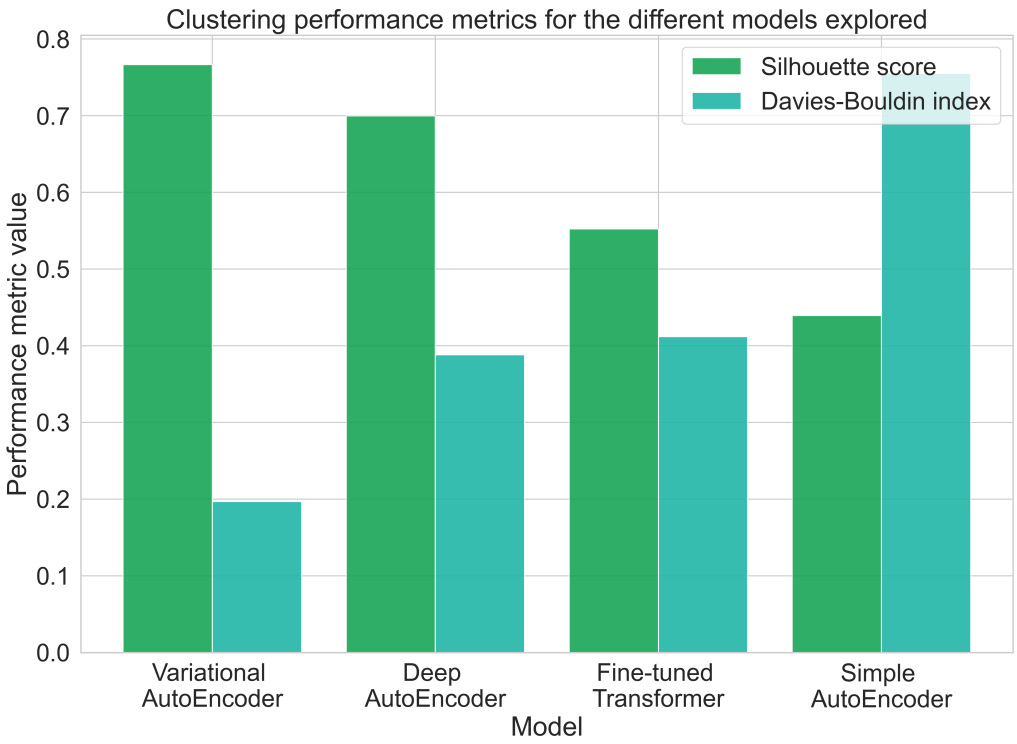


Figure 26: Clustering performance metrics of the models and methods explored in our study.

3.2 Interpretability of our models

Interpretability of deep unsupervised learning models is crucial in the context of identifying and characterizing T-cell receptors specificity to SARS-CoV-2 for several reasons. Here are a few reasons for the importance of model interpretation:

- Interpretable models provide a better understanding of the underlying mechanisms and decision-making processes of the model. This transparency helps to build trust and confidence in the model's predictions and results. It allows us to validate and verify some of the results of the model, ensuring that the conclusions drawn are reliable. This is even more important for neural networks and deep learning models, which are often seen as *"black-box"* methods.

- Interpretation of deep unsupervised learning models can provide insights into the features or patterns that contribute to TCR specificity. Understanding these patterns can lead to a better understanding of TCR-antigen interactions and inform the design of targeted therapeutics or vaccines.

- Model interpretation helps to understand the generalization and transferability of the learned representations. It allows us to assess whether the model has learned biologically meaningful features that are applicable beyond the training dataset. This knowledge is crucial for deploying the model in real-world scenarios and adapting it to other related problems.

Interpreting deep unsupervised learning models in the context of TCR specificity to SARS-CoV-2 is vital for understanding the underlying mechanisms, gaining insights into TCR-antigen interactions and identifying relevant features and potential

bio-markers. That is why, we will carefully study the inner mechanisms of each of our model structures in the following sections.

3.2.1 Interpretability for our AutoEncoder model

In order to evaluate the interpretability of our AutoEncoder model and to gain a deeper insight into its inner workings, we use a separate dataset that we presented earlier in section 1. This dataset contains different TCR sequences grouped into different metaclonotype groups. Metaclonotype groups consist of TCRs that are known to share biochemical similarities. By testing the ability of our model to cluster these biochemically similar TCRs correctly, we aim to assess its ability to capture and represent important features related to TCR structure.

For the purpose of testing, we select three metaclonotype groups that are well balanced. This ensures that we have a representative sample from each group to evaluate the performance of the model. We then use our AutoEncoder model to extract latent representations of the TCR sequences from these selected metaclonotype groups. These embeddings encapsulate the compressed information about the TCRs, capturing their essential features.

To assess the quality of the clustering, we plot the extracted embeddings in a two-dimensional space, as shown in Figure 27. The resulting plot provides a visual representation of the distribution and proximity of the TCRs. If the TCRs belonging to the same metaclonotype group are clustered closely together in this space, it indicates that our AutoEncoder model has successfully learned the relevant structural features solely from the provided inputs (the CDR3 sequences, j-gene and v-gene).

As we can see in Figure 27, our model is able to cluster biochemically similar TCRs based on these inputs accurately. This means that it has captured meaningful patterns and relationships within the data. This is an important indication of its interpretability, suggesting that it can recognize and represent important structural features of TCRs that are relevant to their specificity for SARS-CoV-2.

3.2.2 Interpretability for our Variational AutoEncoder model

To explore the interpretability of our Variational AutoEncoder model, we focus our analysis on the weight matrices and the variance of the means in latent space. By examining these components, we can gain insights into the patterns captured by the model to differentiate and associate TCRs.

Our first approach is to examine the weights of the encoder component in our VAE model. This examination allows us to understand the relative importance given to each element of the input. As shown in Figure 28, we observe a consistent pattern across specific weight matrices. In particular, we notice that higher weight values are assigned to input positions between 12 and 16. This suggests that the amino acids present in the CDR3 sequence at positions 12 to 16 play an essential role in the differentiation and grouping of TCRs. This finding is consistent with previous

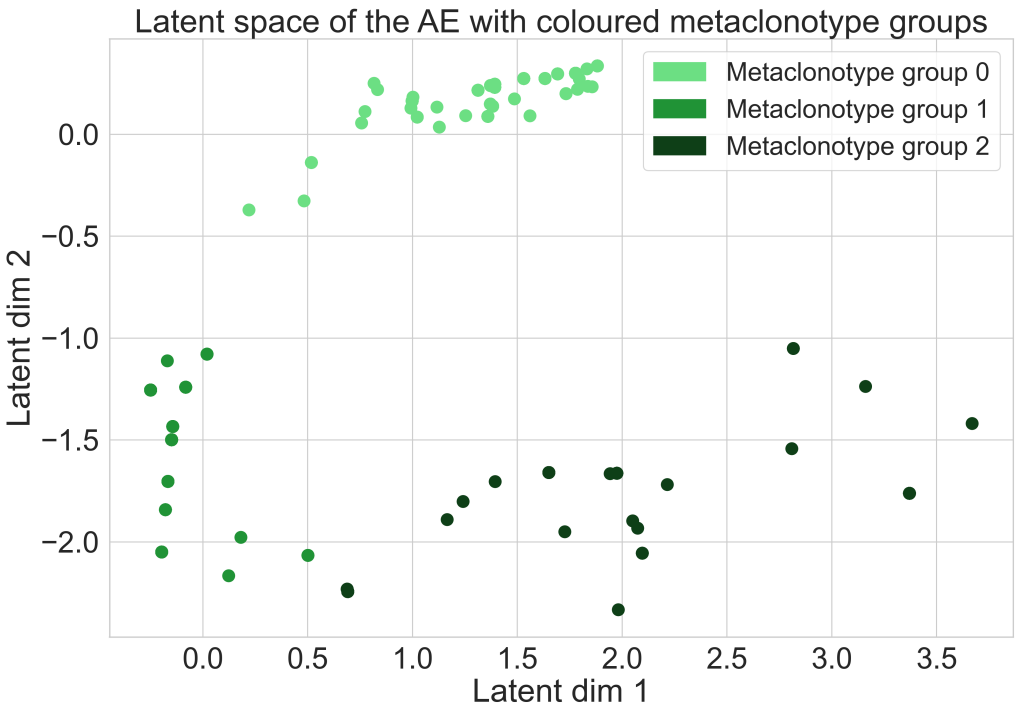


Figure 27: Latent space of the AutoEncoder model with data points coloured with their metaclonotype group.

scientific research which has shown that the middle part of the CDR3 sequence contains the most variable region and is the main contributor to TCR specificity.

By analyzing the weight matrices, we gain valuable insights into the learned representations of the model. The emphasis on specific positions within the CDR3 sequence provides evidence that the VAE has successfully recognized the relevant patterns associated with TCR differentiation and grouping. This understanding improves our interpretation of the model's behavior and strengthens the link between its internal mechanisms and the underlying biological processes.

Alternatively, if we examine the variance of the means of the distributions in latent space, we observe a similar behavior. To investigate this, we introduce variations in each amino acid at each position and calculate the variance of the means of the resulting embeddings. This approach provides a more intuitive understanding, while giving results that are very close to those obtained by analyzing the weight matrices.

If we plot the variance of the means per position, as shown in Figure 29, we observe a consistent pattern. Once again, we find that the means have a higher variance in the middle of the CDR3 sequence, specifically between positions 10 and 16. This result confirms that our VAE model effectively captures meaningful patterns and relationships present in the data. It serves as an important indication of the interpretability of the model, suggesting that it is capable of recognizing and representing important structural features of TCRs that are relevant to their specificity for SARS-CoV-2.

By thoroughly analyzing both the weight matrices and the variance of means, we gain a comprehensive understanding of how the VAE model interprets and represents TCR data. This insight strengthens our ability to interpret and utilise the

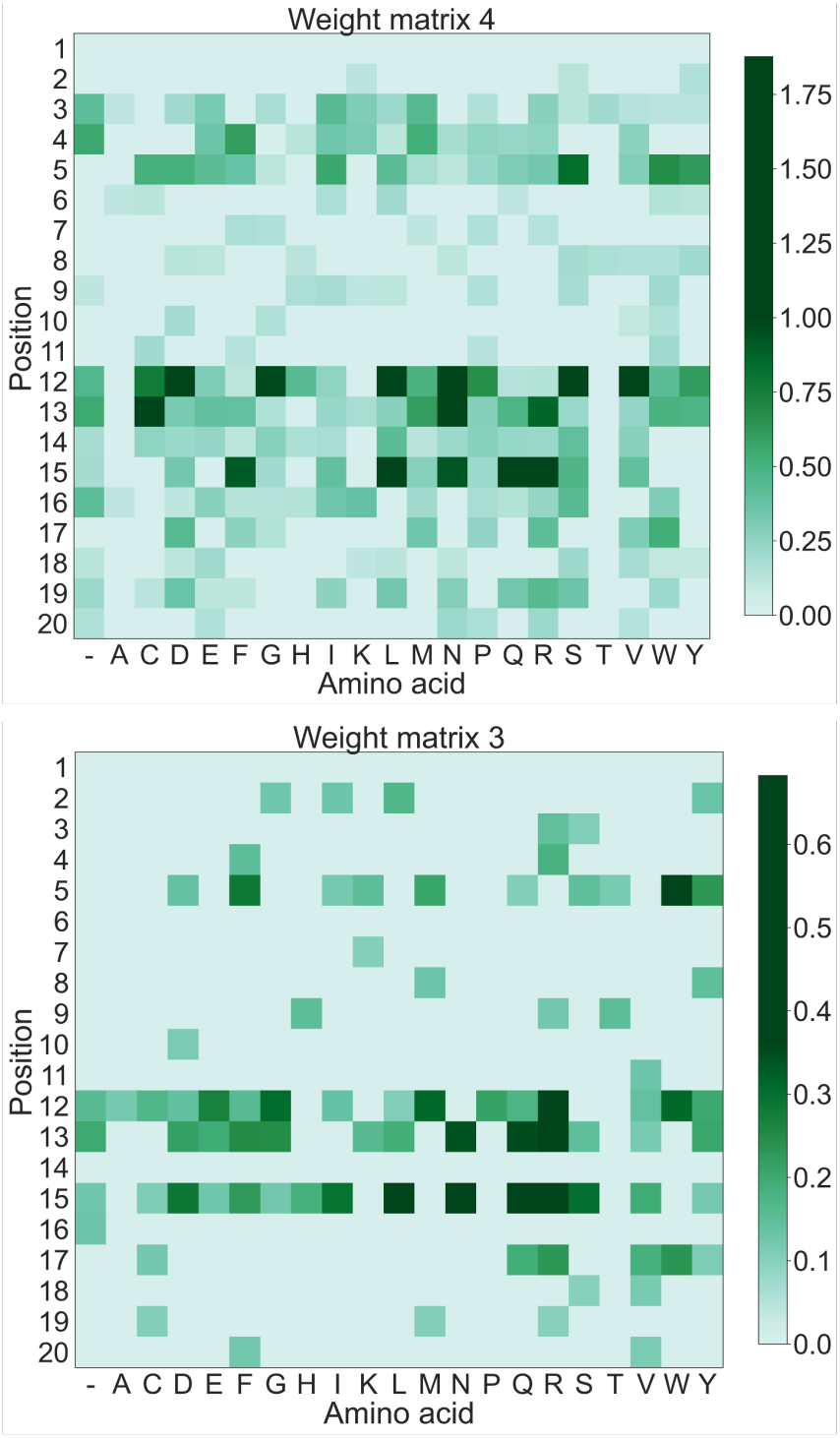


Figure 28: Matrices of the weights 3 and 4 of the VAE model.

results of the model to further our understanding of TCR specificity in the context of SARS-CoV-2.

3.2.3 Interpretability for our Transformer model

In this section, we explore the interpretability of our fine-tuned transformer model, TCR-BERT, by examining its attention heads. By extracting the attention heads from our trained model, we can analyze where the model focuses its *"attention"*.

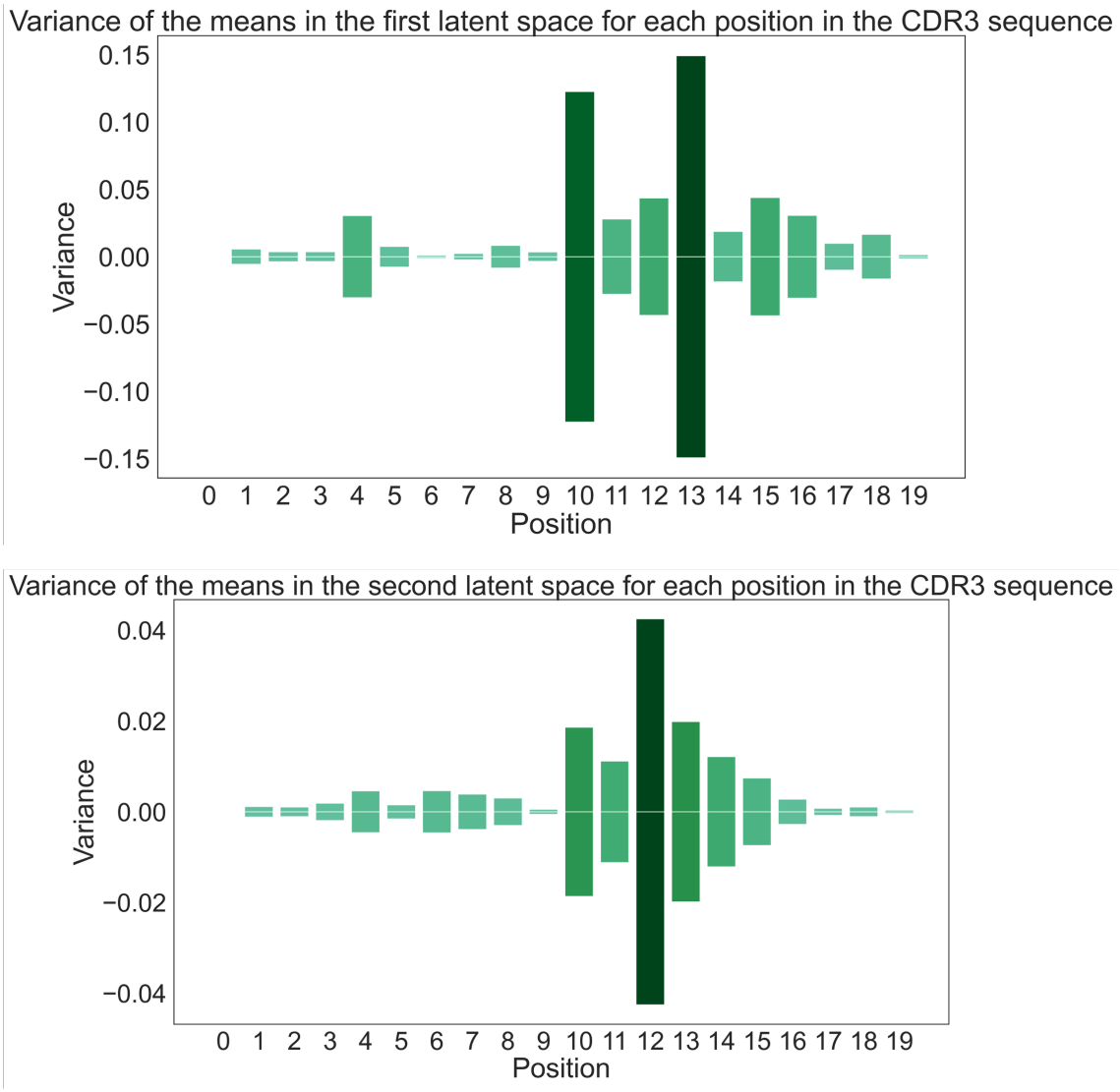


Figure 29: Variance of the means in the latent space at the different positions in the CDR3 sequence.

Plotting the attention values for different positions and amino acids, as shown in Figure 30, provides valuable insights and interpretations.

Similar to our observations with the VAE model, we find that the attention heads in the transformer model primarily focus on amino acids between positions 8 and 15. This alignment with the VAE model findings is consistent with previous scientific research which, as mentioned above, highlights the middle portion of the CDR3 sequence as the most variable region contributing significantly to TCR specificity.

This result reinforces the notion that both the VAE and Transformer models emphasize the same parts of the input to represent TCRs in a latent dimensional space accurately. The focus on the middle part of the CDR3 sequence, where significant structural variations occur, suggests that both models capture and prioritize By using the attention heads of the transformer model, we gain a deeper understanding of how it processes and attends to different parts of the input. This interpretability allows us to elucidate the mechanisms by which the model distinguishes and rep-

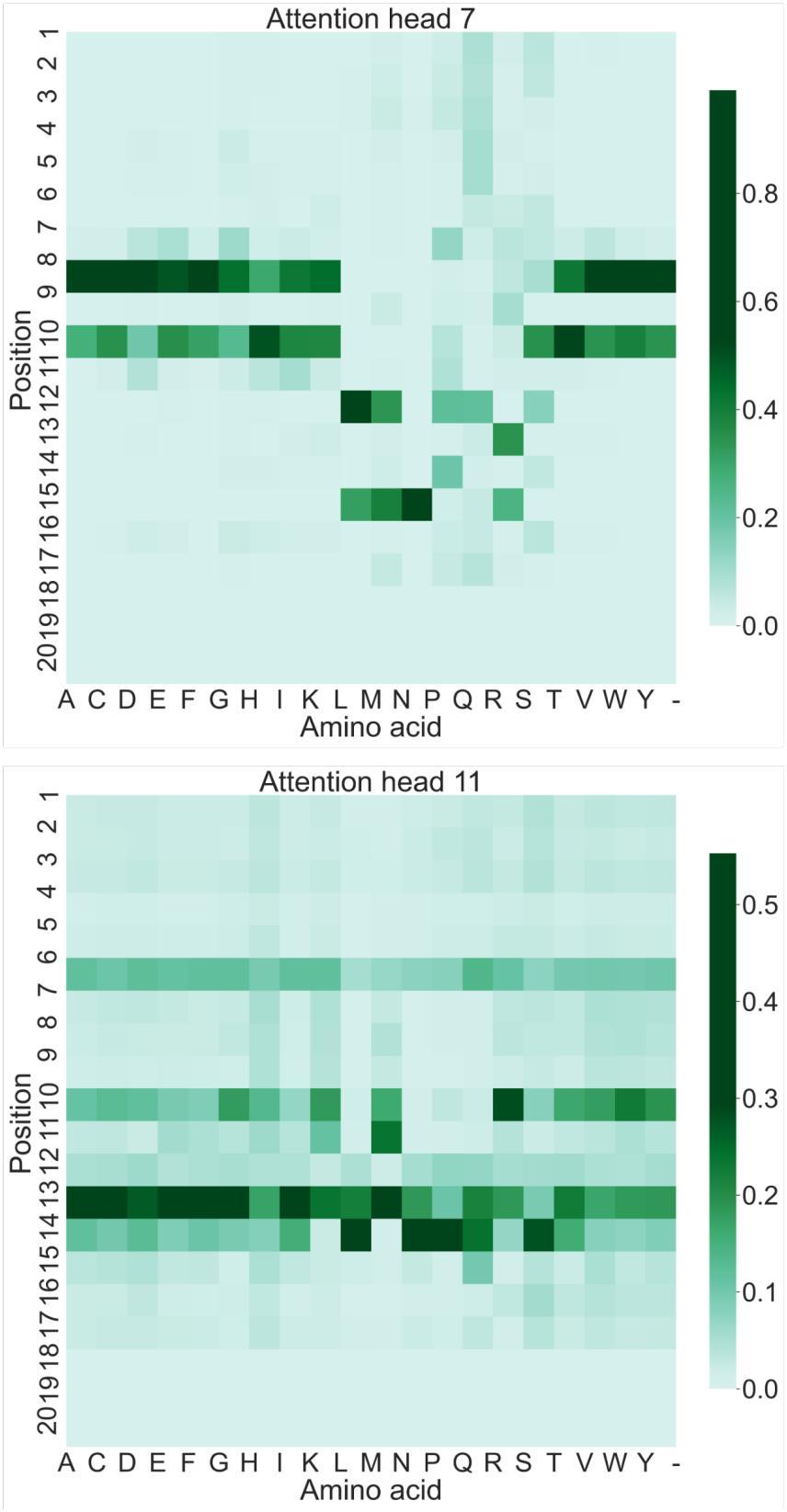


Figure 30: Matrices of the weights for the 7^{th} and 11^{th} attention heads of the fine-tuned TCR-BERT model.

resents TCRs. The consistent emphasis on the same regions of the input between the VAE and the transformer model provides further validation of the importance of the middle portion of the CDR3 sequence in characterizing TCRs.

3.3 Generating new TCR sequences

In this section we will explore how we can use our previously constructed models, such as the VAE or the fine-tuned Transformer model, to generate new TCR sequences with a desired specificity. This will allow us to explore the vast sequence space and potentially discover novel TCRs with solid affinity and specificity for, for example, SARS-CoV-2 antigens. This can help in the development of targeted therapeutics and vaccines. In addition, the generation of new sequences can help to understand the underlying mechanisms of TCR recognition and specificity by studying the patterns and motifs present in the generated sequences.

First, to generate new TCRs using our fitted VAE model, we can make the assumption that our latent space distribution is continuous as explained earlier in section 2.4.3. We can therefore sample vectors from our latent space distribution and then use our decoder to generate meaningful amino acid sequences. Similarly, for the fine-tuned TCR-BERT, we can use the generative capabilities of the Transformer architecture to generate new TCR sequences. By using the learned representations in latent space, we can sample latent vectors and pass them through the decoder component of the model. This process allows us to reconstruct the corresponding TCR sequences, incorporating the desired specificity and structural features.

Several criteria can be used to evaluate the quality of sequences generated from our VAE and Transformer models. One important aspect is to assess the diversity of sequences generated. A high-quality model should be able to generate a diverse set of TCR sequences representing a wide range of potential specificities. This can be assessed by exploring our latent space and generating sequences based on random points from this space. Figure 31 visually demonstrates the rich diversity of CDR3 sequences present in the latent space as each color corresponds to a different generated CDR3 sequence. This analysis provides evidence of the model's capability to produce diverse and unique sequences, reinforcing its potential for generating novel TCRs with specific specificities.

Another critical factor is the structural and biochemical similarity of the generated sequences to known TCRs that show specificity for SARS-CoV-2. The generated sequences should show similar patterns in terms of conserved motifs, amino acid composition and other structural features known to contribute to TCR specificity. Evaluation methods such as motif analysis can be used to assess the similarity between the generated sequences and the desired specific TCRs. The TCRs generated from the Transformer model have previously been evaluated and shown to be similar engineered sequences to those that exist and bind to the same antigen as shown in Figure 32 (Wu et al., 2021).

In addition to evaluating the quality of the generated sequences, it is also essential to study the probability of generating these TCRs. The probability distribution associated with the generation of TCR sequences provides insights into the likelihood of obtaining specific sequences. In order to produce these *probabilities of generation*, we leverage a computational tool called **OLGA**[5] (Sethna et al., 2019). OLGA is

[5]GitHub repository: https://github.com/statbiophys/OLGA

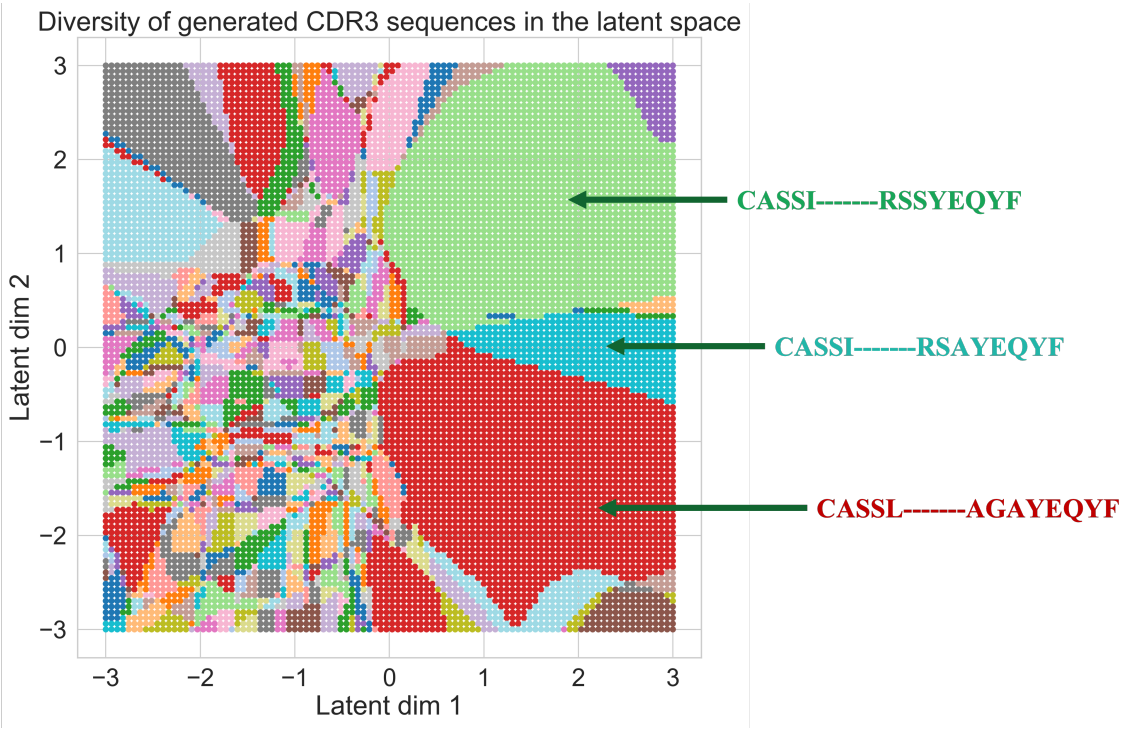


Figure 31: Plot showing how diverse is our latent space in terms of generated CDR3 sequences. Note that each color represents a different CDR3 sequence.

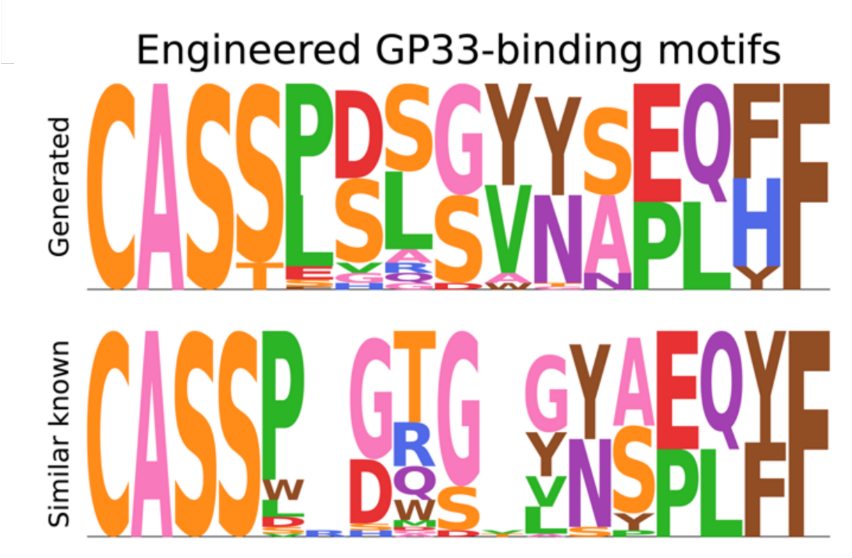


Figure 32: Example of sequences generated using the transformers TCR-BERT along with the motif of similar known sequences that bind to the same antigen GP33 (Wu et al., 2021).

developed to compute the generation probability of amino acid and in-frame nucleotide CDR3 sequences from a generative model. Using this tool, our evaluation method is then relatively straightforward:

1. We evaluate the *generation probabilities* of the *"real"* sequences present in our dataset in order to obtain the true distribution of the sequences from our dataset.

2. We then evaluate the *generation probabilities* of the **generated** sequences from our model in order to obtain the probability distribution for the *in-silico* TCR sequences.

3. Finally, we compare the two distributions as shown in Figure 33. If the two distributions are close, it could mean that our generated sequences could have also been part of our initial dataset. This is the case in our study: both distributions seem to be very similar and very close.

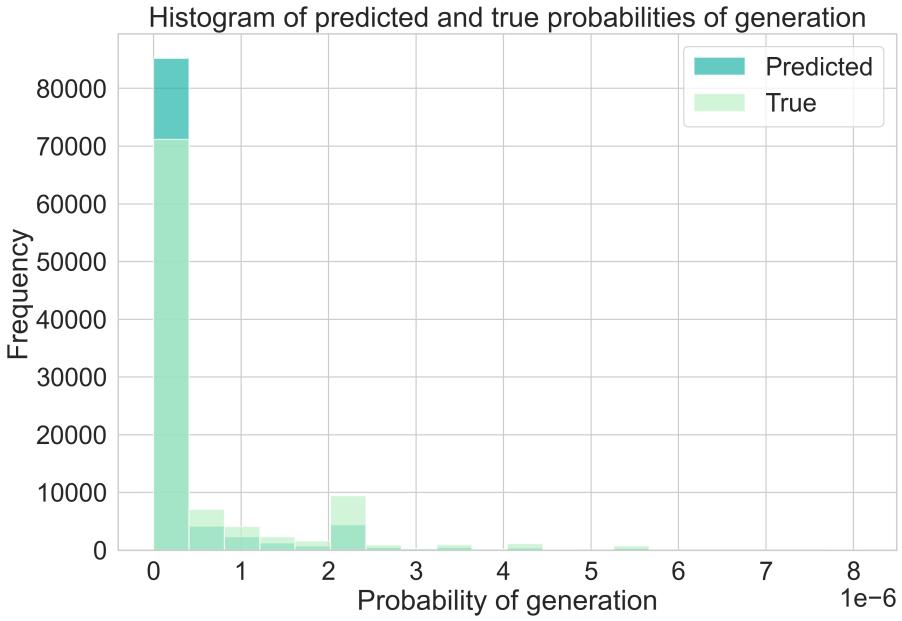


Figure 33: Histogram of the probability distribution for generating the TCR sequences using the fitted VAE and OLGA.

It is also important to consider the predicted binding affinity of the generated sequences to SARS-CoV-2 antigens. Practical experience can be used to estimate the binding affinity and assess whether the generated sequences are likely to have the desired specificity. As this evaluation would require *"hands-on"* biological experiments, we are not able to perform these tests in our study.

4 Discussion

4.1 Conclusions

In this study, we used deep unsupervised learning methods to identify and characterise T-cell receptor specificity for the Sars-CoV-2 virus. Our research focused on the development and application of state-of-the-art modelling techniques, including AutoEncoders, Variational AutoEncoders and transfer learning with Transformers, to analyse TCR data.

Through our experiments and analyses, we have achieved promising results in identifying TCR patterns and understanding TCR specificity for SARS-CoV-2. Our models have demonstrated their effectiveness in capturing meaningful representations of TCR sequences and clustering them based on their similarities. In addition,

the interpretability of our models has provided valuable insights into the features and mechanisms underlying TCR recognition and response to the virus.

Furthermore, our study has highlighted the potential of deep learning methods in TCR analysis for viral infections, such as Sars-Cov-2. The application of transfer learning and Transformers, in particular TCR-BERT, has demonstrated the benefits of applying Natural Language Processing techniques to TCR data, opening up new avenues for research and improving our understanding of immune responses.

The outcomes of our research could have significant implications for vaccine and treatment design against Sars-CoV-2. By identifying TCRs that are specifically involved in the immunological response to the virus, we can gain insights into the key targets for vaccine development and the design of therapeutic interventions. Our research has explored the exciting possibility of generating new TCR sequences using the insights gained from our deep unsupervised learning models. By understanding the patterns and characteristics of TCRs involved in the immunological response to Sars-Cov-2, we can utilize generative modeling techniques to create synthetic TCR sequences that mimic the desired specificity and functionality.

In conclusion, our study has demonstrated the effectiveness of deep unsupervised learning methods in characterizing and identifying TCR specificity to Sars-CoV-2. The insights gained from our research provide valuable tools and knowledge for interpreting the immune response to Sars-Cov-2, ultimately contributing to the development of effective vaccines and treatments against the virus.

4.2 Next steps and future research

While our findings are promising, there are still areas for further exploration. The following are some potential next steps to consider:

- As the availability of TCR data continues to increase, the inclusion of more extensive and diverse datasets can improve the generalisability and robustness of our models. Including data from different populations and diseases (e.g. influenza) could provide an even more comprehensive understanding of TCR specificity for Sars-Cov-2 and its variants.

- Further optimization of our deep learning models is essential to improve their performance and interpretability. Exploring different other architectures, hyperparameters and optimization techniques could further improve the accuracy and efficiency of TCR specificity prediction. We could for example try to fit a neural network with a different architecture (RNN, GNN, etc...). Moreover, the development of more advanced explicable AI approaches could provide additional insights into the learned representations and decision-making processes of our models.

- While our deep learning models have shown promise in identifying TCR specificity, experimental validation is critical. Conducting in vitro and in vivo studies to validate the predicted TCR-antigen interactions could provide concrete evidence of their functional relevance. In addition, collaborations with immunologists and clinicians could facilitate the translation of our findings into clinical practice.

Methods and models	PROS	CONS
Simple AutoEncoder	+ Not prone to over-fitting + Simple architecture, easy to reproduce and train	- Prone to under-fitting - May be too simplistic to capture complex relationships in the data - Struggle to represent the wide range of TCR specificities
Deep AutoEncoder	+ Convoluted architecture that can capture complex relationships in the data	- Prone to over-fitting - Complicated architecture - Training may be time-consuming
Variational AutoEncoder	+ Probabilistic encoding, enabling the generation of diverse TCR sequences + Continuous latent space for better interpretability	- Choice of the entanglement β - Complex training (sampling, losses, etc…)
Fine-tuned Transformer	+ Better performance and generalization + Better interpretability using attention heads and continuous latent space + Excellent performance for generating new sequences	- Training is time-consuming - Training requires GPU - Not completely unsupervised (pre-training)

Figure 34: Summary of the pros and cons of the different modelling methods and techniques explored from a data science viewpoint.

5 Acknowledgements

I would like to express my deep gratitude and appreciation to my supervisor, *Dr Barbara Bravi*, for her invaluable guidance, support, and mentorship throughout the course of this research. Her expertise, dedication, and insightful feedback have been instrumental in shaping this report and my overall growth as a researcher. I am truly grateful for her unwavering belief in my abilities and the opportunities she has given me.

Furthermore, I would like to extend my sincere thanks to the *AI for Health Lab* at Stanford University for their assistance to this study. The resources, expertise, and collaborative environment provided by *Kevin Wu* and *Kyle Swanson* have been instrumental in shaping some important parts of this research. The cutting-edge technologies and interdisciplinary discussions with the lab have enriched my understanding of the existing methods.

A Appendix

You can find freely access here the code that has been done during the elaboration of this study.
Most of the code has been cleaned and annotated with comments explaining the methods and techniques used. In fact, reproducible code is essential for conducting rigorous and reliable research. The data analyses and scientific claims can be verified and replicated by others using the same data and software.
Please read the README.md file for more information on how to load the datasets and run the code successfully. A requirements.txt file has also been added to the GitHub repository in order to simplify the installation of all the necessary packages and avoid any conflicts.

B Appendix

For an easier and faster training and testing of the Transformers model, here is a link to a shared Google Colab document. It already contains the necessary code to run the training and fine-tuning of the model using a custom dataset. This implementation allows anyone to make use of GPU computational resources for an accelerated training. Please note that you have to select *GPU runtime* in the *Runtime* parameters of the notebook.

C Appendix

Moreover, you can also find here a web application simulating some TCR-antigen sequences binding examples and their predictions based on the chosen model (AutoEncoder, Variational AutoEncoder, etc...): https://m4r-dash.yanismiraoui.repl.co/. You can also ask any questions that you may have about this research paper to a Chatbot. The code of this web application is available here and has been deployed and hosted on a standard virtual machine using Heroku, Replit and Amazon Web Services (AWS).
Please note that as this web application is hosted with limited resources, some results and figures may take some time to load.
On the other hand, the Figures 9 and 12 were created using this diagram creation website. It greatly facilitated the drawing of the full neural networks architectures along with the sizes of each layer.

References

Altan-Bonnet, G., Mora, T., and Walczak, A. M. (2020). Quantitative immunology for physicists. *Physics Reports*, 849:1–83.

Bravi, B., Balachandran, V., Greenbaum, B., Walczak, A., Mora, T., Monasson, R., and Cocco, S. (2021). Probing t-cell response by sequence-based probabilistic modeling. *PLOS Computational Biology*, 17(9).

Cheung, K. (2022). Methods for the characterization of specificity in immune response to sars-cov-2.

Davidsen, K., Olson, B. J., III, W. S. D., Feng, J., Harkins, E., Bradley, P., and IV, F. A. M. (2019). Deep generative models for t cell receptor protein sequences. *eLife*, 8.

Gallagher, J. (2023). New superbug-killing antibiotic discovered using ai. *BBC News*.

Isacchini, G., Walczak, A. M., Mora, T., and Nourmohammad, A. (2021). Deep generative selection models of t and b cell receptor repertoires with sonnia. *PNAS*. https://doi.org/10.1073/pnas.2023141118.

Jenkins, K., M., Chu, H., H., McLachlan, B., J., Moon, and J., J. (2010). On the composition of the preimmune repertoire of t cells specific for peptide-major histocompatibility complex ligands. *Annual review of immunology*, 28:275–294.

Kingma, D. P. and Welling, M. (2022). Auto-encoding variational bayes. *ICLR*.

Mateus, J., Grifoni, A., Tarke, A., Sidney, J., Ramirez, S. I., Dan, J. M., Burger, Z. C., Rawlings, S. A., Smith, D. M., Phillips, E., Mallal, S., Lammers, M., Rubiro, P., Quiambao, L., Sutherland, A., Yu, E. D., da Silva Antunes, R., Greenbaum, J., Frazier, A., Markmann, A. J., Premkumar, L., de Silva, A., Peters, B., Crotty, S., Sette, A., and Weiskopf, D. (2020). Selective and cross-reactive sars-cov-2 t cell epitopes in unexposed humans. *Science*, 370(6512):89–94.

Mayer-Blackwell, K., Cohen-Lavi, S. S., Crawford, J. C., Souquette, A., Gaevert, J. A., Hertz, T., Thoma, P. G., sPhilip Bradley, and Fiore-Gartland, A. (2021). Tcr meta-clonotypes for biomarker discovery with tcrdist3 enabled identification of public, hla-restricted clusters of sars-cov-2 tcrs. *eLife*, 10.

McInnes, L., Healy, J., and Melville, J. (2020). Umap: Uniform manifold approximation and projection for dimension reduction.

Nolan, S., Vignali, M., and et al., M. K. (2020). A large-scale database of t-cell receptor beta (tcr) sequences and binding associations from natural and synthetic exposure to sars-cov-2. *Research Square*.

Olah, Chris, Satyanarayan, Arvind, Johnson, Ian, Carter, Shan, Schubert, Ludwig, Ye, Katherine, Mordvintsev, and Alexander (2018). The building blocks of interpretability. *Distill*. https://distill.pub/2018/building-blocks.

Pennock, D., N., White, T., J., Cross, W., E., Cheney, E., E., Tamburini, A., B., Kedl, ., and M., R. (2013). T cell responses: naive to memory and everything in between. *Advances in physiology education*, 37(4):273–283.

Ralph, Duncan, IV, M., and Frederick (2015). Consistency of vdj rearrangement and substitution parameters enables accurate b cell receptor sequence annotation. *PLoS computational biology*, 12.

Sainburg, T., McInnes, L., and Gentner, T. Q. (2021). Parametric umap embeddings for representation and semisupervised learning. *Neural Comput*.

Sethna, Z., Elhanati, Y., Callan, C. G., Walczak, A. M., and Mora, T. (2019). OLGA: fast computation of generation probabilities of b- and t-cell receptor amino acid sequences and motifs. *Bioinformatics*, 35(17):2974–2981.

Sidhom, JW., Larman, H.B., Pardoll, and et al., D. (2021). Deeptcr is a deep learning framework for revealing sequence concepts within t-cell repertoires. *Nat Commun*, 12.

Sun, Y., Li, F., Sonnemann, H., Jackson, R., K., Talukder, H., A., Katailiha, S., A., Lizee, and G. (2021). Evolution of cd8+ t cell receptor (tcr) engineered therapies for the treatment of cancer. *Cells*, 10:2379.

VanderPlas, J. (2016). *Python Data Science Handbook*. O'Reilly Media, Inc.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *arXiv*.

Weber, A., Born, J., and Martínez, M. R. (2021). Titan: T-cell receptor specificity prediction with bimodal attention networks. *Bioinformatics*, 37:i237–i244.

Wu, K., Yost, K., Dániel, B., Belk, J., Xia, Y., Egawa, T., Satpathy, A., Chang, H., and Zou, J. (2021). Tcr-bert: learning the grammar of t-cell receptors for flexible antigen- binding analyses. *bioRxiv*.

Zhang, D., Maslej, N., Brynjolfsson, E., Etchemendy, J., Lyons, T., Manyika, J., Ngo, H., Niebles, J. C., Sellitto, M., Sakhaee, E., Shoham, Y., Clark, J., and Perrault, R. (2022). The ai index 2022 annual report.

Zhang, D., Maslej, N., Brynjolfsson, E., Etchemendy, J., Lyons, T., Manyika, J., Ngo, H., Niebles, J. C., Sellitto, M., Sakhaee, E., Shoham, Y., Clark, J., and Perrault, R. (2023). The ai index 2023 annual report.