Heidelberg University
Institute of Computer Science

Project report for the lecture
Advanced Machine Learning

Prof. Dr. Köthe
Summer semester 2021

Prediction of the next SARS-CoV-2 mutations by a Transformer based GAN Framework

https://github.com/nilskre/AML-covid-project

Team Member: Felix Hausberger, 3661293,
Applied Computer Science
eb260@stud.uni-heidelberg.de

Team Member: Nils Krehl, 3664130,
Applied Computer Science
pu268@stud.uni-heidelberg.de

Plagiarism statement

We certify that this report is our own work, based on our personal study and/or research and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication.

We also certify that this report has not previously been submitted for assessment in any other unit, except where specific permission has been granted from all unit coordinators involved, or at any other time in this unit, and that we have not copied in part or whole or otherwise plagiarized the work of other students and/or persons.

Member contributions

Nils Krehl

Came up with the project idea and was the domain expert for the project's biomedical domain. Initial research for the topic and feasibility analysis. Extensive research in the areas of the project's overall composition, data sources, dataset creation and dataset insights. Responsible for the data selection and download, the dataset creation, the generation of dataset insights and the pretraining and evaluation of the Transformer model.

Felix Hausberger

Extensive research in the area of ML models, especially the connection between NLP methods and our project, previous work on mutation prediction and different ML model architectures such as Sequence to Sequence, Transformer and GANs. Responsible for the data preprocessing, the model architecture and implementation, the GAN training and evaluation.

Contents

List of Abbreviations

BLEU Bilingual Evaluation Understudy

DNA Deoxyribonucleic Acid

GAN Generative Adversarial Network

GISAID Global Initiative on Sharing All Influenza Data

GPU Graphics Processing Unit

LSTM Long Short-Term Memory

ML Machine Learning

NLP Natural Language Processing

NMT Neural Machine Translation

ORF Open Reading Frames

ReLU Rectified Linear Unit

RKI Robert Koch Institut

RNA Ribonucleic Acid

RNN Recurrent Neural Network

SARS-CoV-2 Severe Acute Respiratory Syndrome Corona Virus 2

UTR Untranslated Region

0 Project Setup

For a detailed description of how to set up the project, please have a look at `https://github.com/nilskre/AML-covid-project/blob/main/README.md#setup`.

1 Introduction

During the genome replication random mutations can appear. As a consequence the encoded protein sequence could be changed, which can lead to different behavior. If this change increases the fitness, it is probably passed on to the next generation. [4]

A currently well known example for a mutating virus is SARS-CoV-2. Due to the developed vaccines the hope for an end of the pandemic arises. Nevertheless this is only true, if the vaccines, which are developed against the wild type of SARS-CoV-2, also remain effective against new mutations. To enable fast responses to new arising mutations it would be helpful to know the possible next mutations in advance. This can influence the treatment and prevention of diseases, by enabling the development of countermeasures and preventive measures in advance. [4]

Machine Learning (ML), especially Deep Learning enabled improvements in lots of different domains. This work applies Deep Learning to the area of virus genome mutation prediction. Due to the fact, that genome sequences could be treated as text data, methods from the Natural Language Processing (NLP) area can be applied. The success of Deep Learning for NLP tasks has already been shown in various areas such as text generation, text summarization or translation. [4]

Our research question is whether a ML model can be trained to predict the next possible SARS-CoV-2 mutations. In this paper, we propose three novelties:

- **Model architecture**: A new Generative Adversarial Network (GAN) based architecture influenced by [4]. Our novelty is the usage of transformers instead of Long Short-Term Memory (LSTM) in the seq2seq model.
- **Dataset**: Generation of a dataset for SARS-CoV-2, consisting of 9199 parent-child data instances.
- **Application domain**: The training of the network for SARS-CoV-2.

2 Fundamentals and Related Work

2.1 Basics from molecular biology

This chapter gives a brief introduction into the underlying biological domain.

2.1.1 Genome sequences

The human genome is encoded as DNA. The DNA carries genetic instructions how DNA based organisms develop and behave. It is structured as a double helix and consists of two nucleotide pair combinations: adenine (A), thymine (T) and cytosine (C), guanine (G). [11, p. 8]

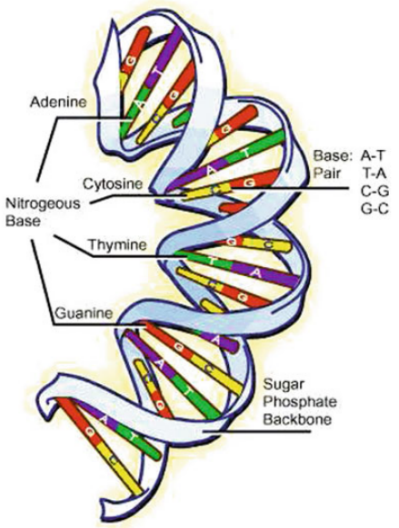


Figure 1: Deoxyribonucleic Acid (DNA) double helix [11, p. 8]

2.1.2 Protein biosynthesis

The protein biosynthesis (also known as the central dogma of molecular biology) describes how proteins are produced based on DNA sequences. The proteins then produce characteristics (e.g. human hair color). So the DNA directly influences which human characteristics are developed. [**schererStatisticalGeneticsGenetic2021**]

The process of the protein biosynthesis can be seen in figure 2. The single steps are described in the following:

1. Transcription: The DNA is transcribed by the RNA-Polymerase into pre-messenger RNA. As part of this process the nucleotide thymine is replaced by uracil (U). [**schererStatisticalGeneticsGenetic2021**]

2. mRNA Processing: The pre-messenger RNA is transformed into the messenger RNA by removing the non-coding regions (introns). So the main difference between DNA and Ribonucleic Acid (RNA) is that the DNA is structured as a double helix, whereas the RNA consists of a single strain. [**schererStatisticalGeneticsGenetic2021**]

3. Translation: Ribosomes translate the mRNA into amino acids. Three nucleotides (one codon) decode one amino acid. The matching which codons decode for which amino acid can be seen in figure 3. [**schererStatisticalGeneticsG**

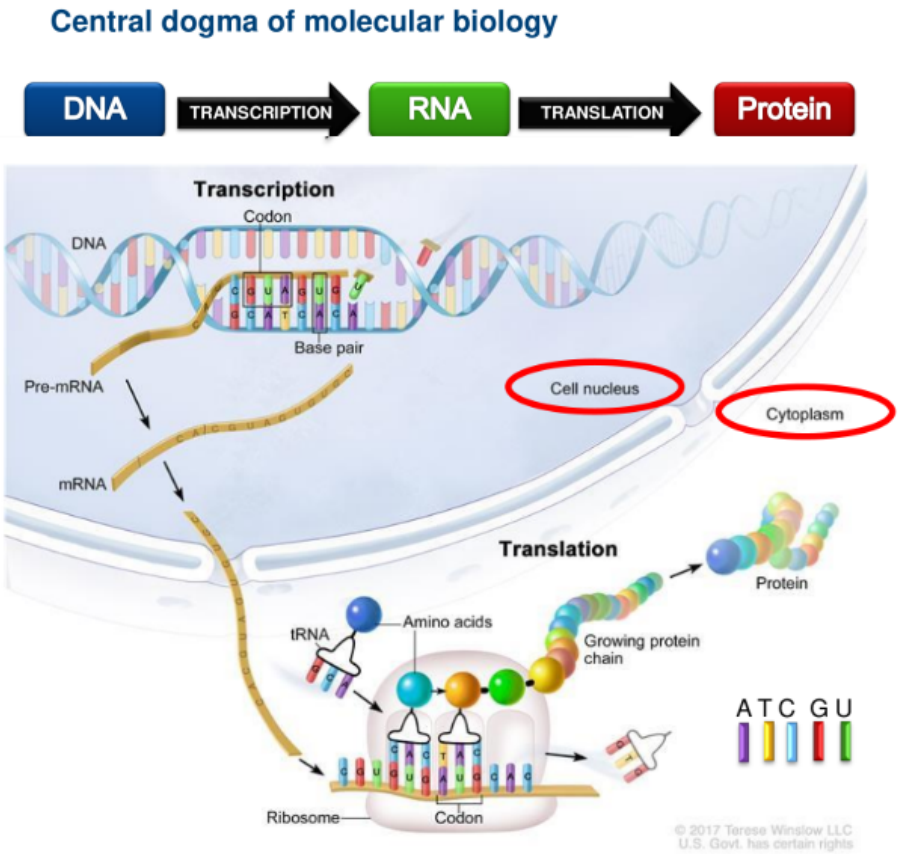


Figure 2: Protein biosynthesis [**schererStatisticalGeneticsGenetic2021**]

2.1.3 Structure of SARS-CoV-2

RNA based viruses (like SARS-CoV-2) use RNA to encode their genetic information. That means that the genetic information is present as a single strained RNA sequence. This sequence contains for example the information which proteins form the surface structure of the SARS-CoV-2 virus and how this surface looks like. [14]

Based on Naqvi et al. [14] the four proteins forming SARS-CoV-2 are:

- spike (S)
- envelope (E)

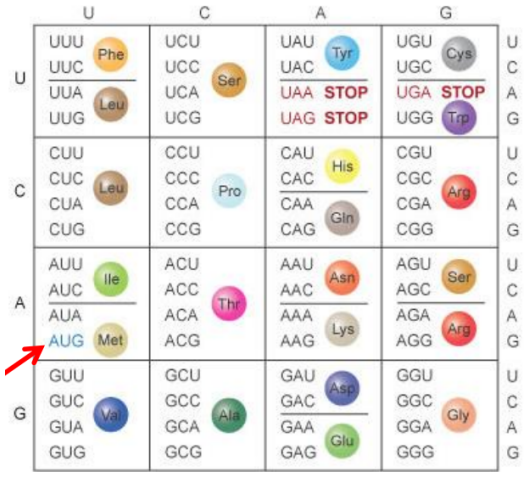


Figure 3: Genetic code [**schererStatisticalGeneticsGenetic2021**]

- membrane (M)
- nucleocapsid (N)

A structural representation of SARS-CoV-2 and a host cell is visualized in figure 4. [14]

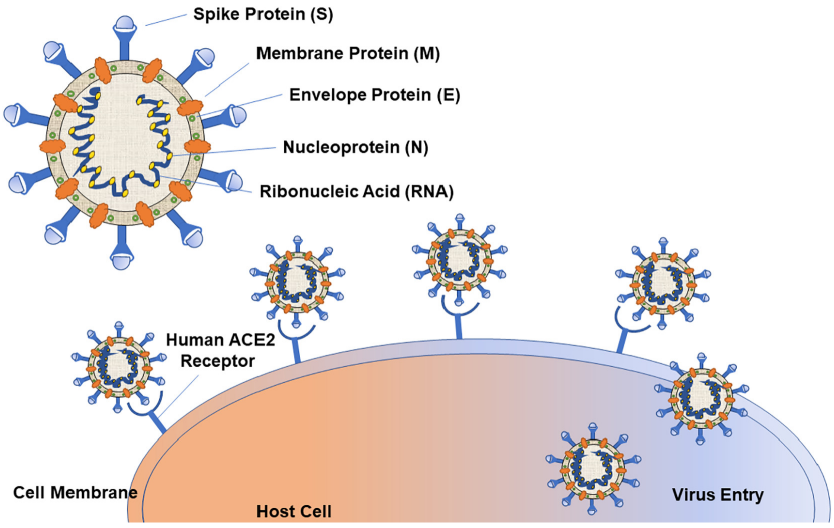


Figure 4: SARS-CoV-2 structure [14, p. 2]

Furthermore figure 4 shows the RNA in the center. The SARS-CoV-2 RNA consists of about 30,000 nucleotides, which belong to different subgroups, as shown in figure 5. Padded by a 5' Untranslated Region (UTR) at

the beginning and a 3' UTR in the end (UTR are markers for the beginning and the end of protein coding sequences), the middle part contains 12 functional Open Reading Frames (ORF), which contain the four coding regions for the proteins S, E, M and N.

Figure 5: SARS-CoV-2 genome structure [14, p. 3]

2.1.4 Relationships between biological objects

One main question in biology is to determine relationships between objects, such as species or genome sequences. The observed objects are called taxa. Most commonly used for determining these relationships is the phylogenetic analysis which generates a phylogenetic tree. In this phylogenetic tree each leaf node corresponds to exactly one taxa. The inner nodes of the tree are the inferred hypothetical ancestors, which are no taxas. The relatedness between different taxas can be evaluated by their distance. [5]

Figure 6 shows an example phylogenetic tree calculated based on the species genomes. On the right side each leaf node (taxa) corresponds to one current species. From left to right one can see the infered historical development from the ancestors to the current species. One can see, that Gorillas and Orangutans have earlier developed apart then the other species. Furthermore one can see, that the Chimpanzees and Bonobos share a most recent common ancestor (inner node next to both). That is why they are more related to each other. [**mallawaarachchiMolecularPhylogeneticsUsing2018**]

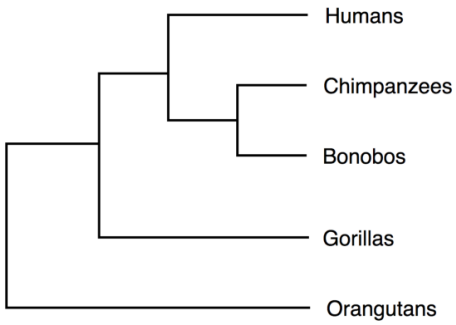


Figure 6: Example phylogenetic tree [**mallawaarachchiMolecularPhylogeneticsUsing2018**]

2.2 From Probabilistic Language Models to modeling Evolution Theory

A probabilistic language model tries to approximate the probability distribution

$$P(w_1, ..., w_n) = \Pi_{t=1}^{n} P(w_t | w_1, ..., w_{t-1}) \tag{1}$$

with w_t being a word at position (time step) t in a sentence of length n. To build language models Recurrent Neural Networks (RNNs) were used to model such probability distributions.

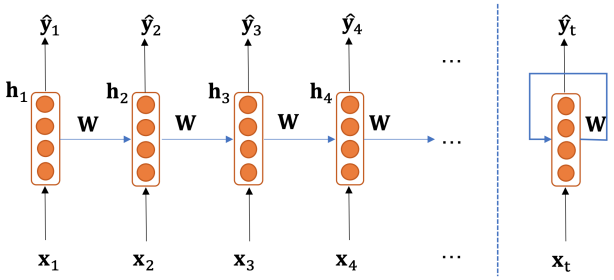


Figure 7: Architecture of a conventional RNN [7, p. 46]

At each time step t it outputs a probability distribution $P(w_t | w_1, ..., w_{t-1})$ given the words read so far in the current instance (see Figure 7). Words are read as a vectorized numerical representation, often given by pretrained so-called word embeddings x_t which are lower dimensional and more semantically-enriched compared to simple one-hot encodings. One then calculates the

hidden state h_t by

$$h_t = f(W^{(h)}h_{t-1} + W^{(x)}x_t + b_1) \tag{2}$$

and the corresponding output probability distribution by

$$\hat{y}_t = softmax(U^{(h)}h_t + b_2). \tag{3}$$

The applied weight matrix is always the same for each time step t giving the RNN its name. One can therefore simplify the unrolled RNN architecture on the left side of Figure 7 to the one on the right, where the hidden state is continuously passed as an input to the next time step. To achieve a better convergence behavior during training, one can also provide the expected hidden state of time step $t-1$ instead of using the predicted hidden state, which is called teacher forcing. RNNs are able to process input of arbitrary length and are by their recurrent character capable to use information from previous time steps. Unfortunately, they are vulnerable to vanishing and exploding gradient problems. LSTM is a special RNN architecture that solves such vulnerabilities by owning a separate long-term cell state besides a short-term hidden state (introduced in subsection 2.6). It is able to preserve information over many time steps. [7]

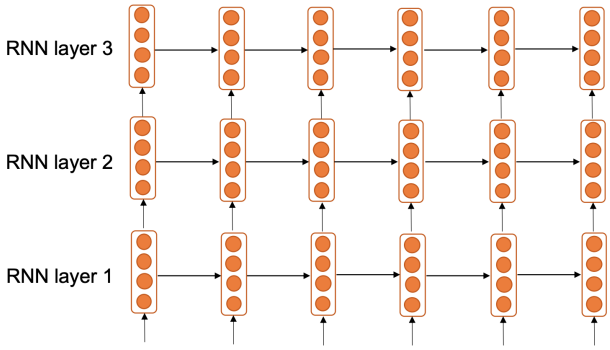


Figure 8: Architecture of a multi-layer RNN [7, p. 87]

One can also use two RNNs, one traversing a sentence from left to right and another one vice versa, with two different weight matrices to model the probability distribution bidirectionally. One therefore simply concatenates the hidden states of each RNN before applying the weight matrix U and the $softmax()$ function. Also multi-layer RNN can be utilized to generate higher-order features (hidden states) for the prediction task (see Figure 8). [7]

The RNNs or even better LSTMs architectures used for probabilistic language modeling can be reused in a more complex domain called sequence to

sequence modeling for Neural Machine Translation (NMT) from one language to another. Here one first tries to learn a fixed-dimensional input representation from an input sequence using an encoder architecture based on an LSTM. The so-called context vector is then decoded by a second LSTM into a new sequence of words preserving the grammar but having a different meaning. [18]

Sequence to sequence models are introduced together with the LSTM architecture in subsection 2.6. Here the connection to evolution theory can be drawn. RNA sequences made of a concatenation of nucleotides [1] can be represented textually using the FASTA format [**cockBiopythonFreelyAvailable2009**]. A sequence to sequence model can then transferably be applied in the domain of RNA sequences to model how RNA-based viruses change their structure to avoid the detection by the human immune system but still to preserve their infectivity and evolutionary fitness [9].

2.3 GISAID EpiFlu Data Platform

The data needed for this project consists of the SARS-CoV-2 genome data and the corresponding metadata. The most complete database is hosted by the GISAID initiative [1]. It is a public-private-partnership between the initiative itself and the governments of Germany (official host of the platform), Singapore and the United States of America. [**gisaideditorGISAID2021**]

The main aim of GISAID is to enable the fast sharing of epidemic and pandemic virus data. In comparison to other databases such as GenBank the genome sequences are shared fast through GISAID. For enabling this fast sharing the GISAID database is only usable after registration and the redistribution of the data is forbidden, whereas the GenBank database is publicly available. [**shuGISAIDGlobalInitiative2017**]

Thus, until september 2021 over three million SARS-CoV-2 genome sequences are available through GISAID, whereas only about one million SARS-CoV-2 genome sequences are available through GenBank. Figure 9 shows the selection view of the GISAID database. [**gisaideditorGISAID2021**, **nationallibraryofmedicinenc**

[1] We restrict the representation of nucleotides solely to their nucleobases parts consisting of the distinct nucleobases guanine, adenine, cytosine and thymine. We therefore do not include the phosphate group and the five-carbon sugar components.

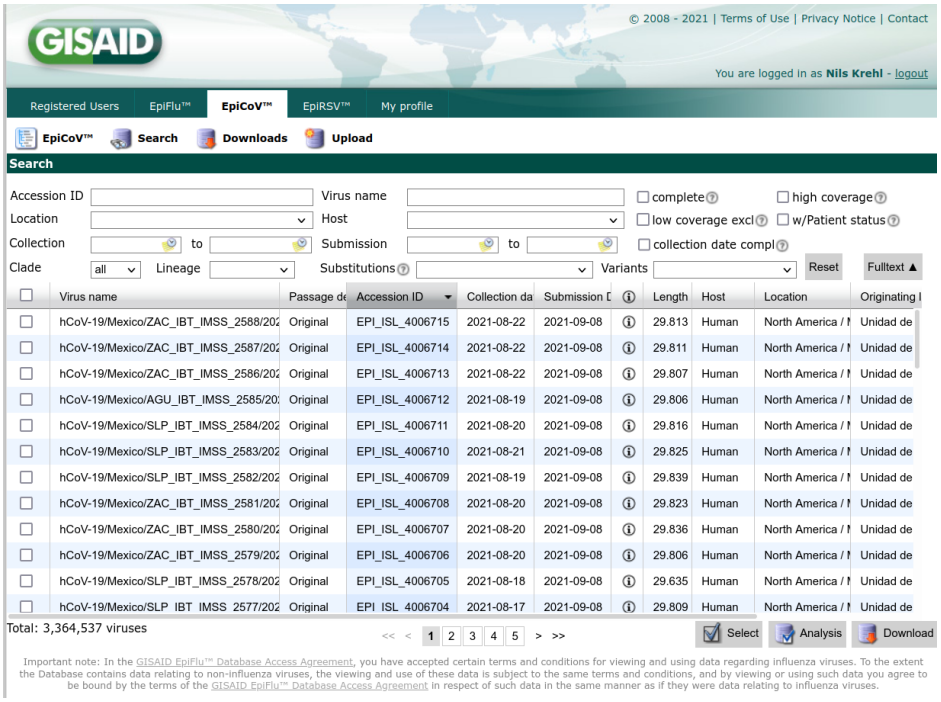


Figure 9: GISAID database [**own screenshot**]

2.4 Domain-Specific Methodologies to create Evolutionary Datasets for Mutation Prediction

From databases such as GISAID or GenBank an unordered list of genome sequences and metadata can be downloaded. For training a ML model to detect changes and predict future mutations the dataset must contain parent-child pairs.

Berman et al. [4] have done this based on two steps. First they created a phylogenetic tree from the genome sequences. Therefore first all genome sequences are aligned. The initial phylogenetic tree is calculated by Fasttree using the approximate maximum likelihood method. In the next step the tree is refined by a generalized time-reversible (GTR) model and a gamma model. The final phylogenetic tree is achieved by optimizing the branch length and the used models for tree generation. Like in standard phylogenetic trees each leaf node corresponds to one data instance, whereas the inner nodes are inferred from the other data instances. Secondly based on the rooted phylogenetic tree, parent-child pairs are calculated. Therefore first a marginal ancestral sequence reconstruction is executed based on the phylogenetic tree from the previous step. Now also the inner nodes correspond to data instances. In the next step BioPython's [6] Bio.Phylo package

is used to determine parent-child pairs. From each edge one parent-child pair is generated. [4]

Mohamed et al. [12] used existing datasets from Ogali et al. [**ogaliMolecularCharacterizati** in their work about mutation prediction. Ogali et al. [**ogaliMolecularCharacterizationNewca** performed a phylogenetic analysis. After selecting only genome sequences with high quality, they removed duplicate sequences and added reference sequences to the dataset. Now the alignment is executed. In the next step MEGA (Molecular Evolutionary Genetics Analysis) is used to create the phylogenetic tree. This is done by using the maximum likelihood method, the best-fit general time-reversible (GTR) model and gamma-distributed rate variation.

Hadfield et al. [8] have created the nextstrain project. It aims to examine pathogen's spread and evolution by providing a viral genome database, a bioinformatics pipeline for phylodynamics analysis and a visualization platform. Steps executed by the bioinformatics pipeline for phylodynamic analysis are: "subsampling, alignment, phylogenetic inference, temporal dating of ancestral nodes and discrete trait geographic reconstruction, including inference of the most likely transmission events" [8]. For calculating the phylogenetic tree, TreeTime's maximum likelihood method is used. [8]

2.5 Previous Work on Mutation Prediction

Even before the rise of Covid-19 there had been studies trying to predict mutations of RNA viruses. In the collection of [20, 21, 22] the authors predict the mutation positions in hemagglutinins from influenza A virus using logistic regression and plain neural networks and then use the resulting amino acid mutating probabilities to derive possible mutated amino acids. The same approach is further used for H5N1 neuraminidase proteins.

Salama et al. [16] proved that nucleotides in an RNA sequence can change based on their local neighborhood. Neural networks are used to predict new strains of the Newcastle virus and subsequently a rough set theory based algorithm is introduced to extract the according point mutation patterns.

Mohamed et al. [12] used a more modern sequence to sequence approach based on LSTMs to learn nucleotide mutations between time-series species of H1N1 Influenza virus and the Newcastle virus as mutations can also be influenced by long-distance relations of amino acids. Therefore one hot-encoded RNA sequences of a parent generation preprocessed to words is given as an input and the output is the predicted offspring generation evaluated by accuracy to the compared true offspring generation. The achieved accuracy in this paper is questionably high with 98.9% on the H1N1 Influenza virus and 96.9% on the Newcastle virus, possibly because of overfitting to the few

4.609 samples for H1N1 Influenza virus and only 83 for the Newcastle virus. Our approach therefore tries to increase the number of samples available for training when building the dataset.

Our approach will neither use any of the just mentioned architectures, but uses a transformer based architecture coupled with a GAN-style training architecture. A short introduction into sequence to sequence models and the underlying long short-term memory components shall be given to better point out our architectural decisions.

2.6 Sequence to Sequence Models based on Long Short-Term Memory

The original LSTM unit was introduced by Hochreiter and Schmidhuber in [10] and can be used for language modeling instead of using plain RNNs to prevent running into vanishing or exploding gradient problems [17]. The architecture of an LSTM is shown in figure 10.

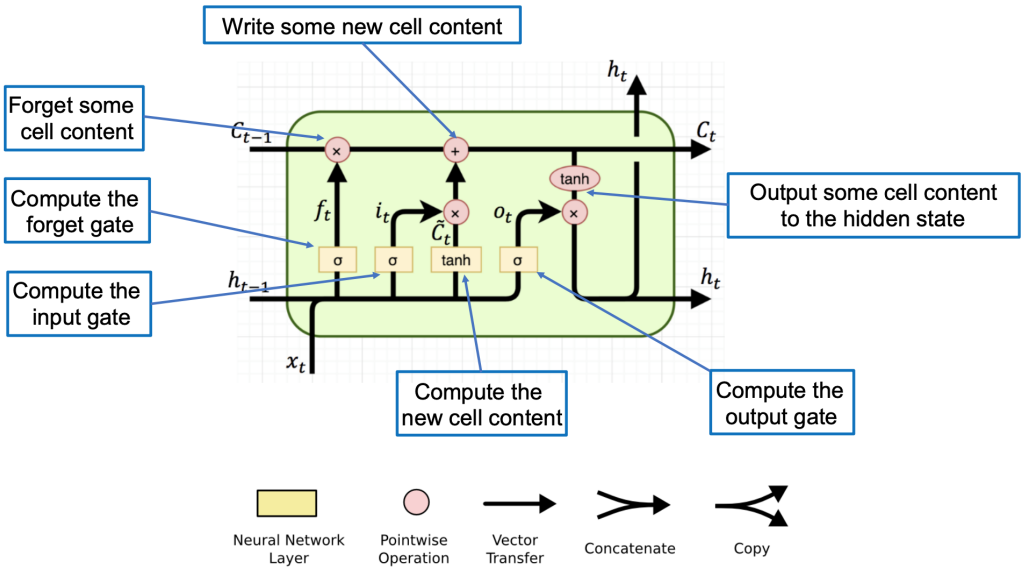


Figure 10: Architecture of an LSTM [7]

It consists of a hidden state h_t and an additional cell state c_t. The cell state stores long-term information and is used to derive a new hidden state. Information flows through three different gates inside the LSTM. The forget gate is used to control which parts of the cell state are potentially carried on to the next time step, the input gate is responsible to decide which parts

of the cell state should be updated and the output gate determines what is being passed on as the new hidden state. All three gates depend on the previous hidden state and the current input. They provide factors limited to the interval [0,1] by the sigmoid function and are multiplied with the cell state, the changes to be added to the cell state and the new hidden state derived from the cell state. Through the cell state an LSTM therefore makes it possible to capture long-distance dependencies. [7]

Sutskever et al. [18] introduced sequence to sequence learning following a multi-layer encoder-decoder style model architecture. One layer consists of one LSTM that is used as an encoder to learn a large fixed-dimensional vector representation of a size-unrestricted input sequence called the context vector. This vector consists of the last cell and hidden state of the encoder and incorporates the structure of the input sequence helping the following decoder LSTM to provide qualitative predictions for the output sequence. The second LSTM therefore serves as a beam search[2] decoder to map the context vector to a corresponding output sequence whose length does not need to match with the length of the input sequence. The output probability distribution is therefore given by the equation

$$p(y_1, ..., y_{T'}|x_1, ..., x_T) = \Pi_{t=1}^{T'} p(y_t|v, y_1, ..., y_{t-1}) \tag{4}$$

with v being the context vector. Using an LSTM is preferred over a normal RNN as it is used to capture the long range temporal dependencies of the input data. The encoder-decoder architecture uses four layers in total partitioned onto four Graphics Processing Units (GPUs). A corpus of 160k words for the input sequence and another one of 80k words for the target sequence was used to create the word embeddings of dimension 1000. Unknown words were replaced by an *UNK* token. The sequence to sequence model approach was evaluated for NMT and reached a 34.81 BLEU score. One finding during training was that reversing the input sequence introduces many short term dependencies as the minimal time lag of the problem is reduced making optimization easier. [18]

2.7 Applying Generative Adversarial Networks

Using a plain sequence to sequence model for mutation prediction does not necessarily guarantee that the generated sequences are evolutionary offsprings of a parent generation as not being included as is in the ground

[2]Do not choose the most probable word but the B most likely word hypothesis and pass them to the next time step in the LSTM. Whichever hypothesis results in the lowest loss is kept. To avoid combinatorial explosion limit the beam depth size.

truth data. The generated sequences might occur realistic and biologically relevant, but a plain sequence to sequence architecture does not inherently check for natural parental descent and therefore does not make sure whether the predicted mutations lead to improved fitness. Berman et al. [4] developed a novel sequence to sequence framework based on the GAN idea to predict genetic mutations and future biological populations of the influenza virus (see Figure 11). MutaGAN describes a sequence to sequence generator within an adversarial framework that predicts protein sequences augmented with possible mutations. By using a sequence to sequence generator and a discriminator specialized on separating fake evolutionary mutations from real ones, one can then guarantee to a certain degree that the evolutionary parent-childhood coherence is given. In MutaGAN a mutation is considered correct if the change in amino acid and location within the RNA sequence is equal to the parent's true offspring. [4]

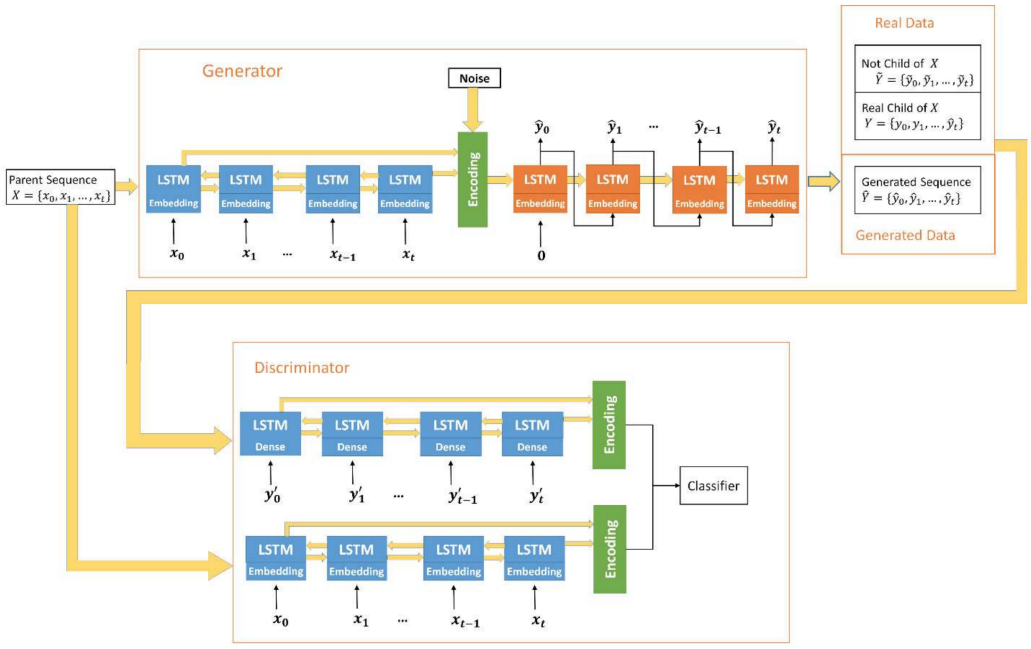


Figure 11: Architecture of MutaGAN [4]

MutaGAN's generator consists of a sequence to sequence model. The encoder is built from of a bidirectional LSTM and the resulting context vector of dimensionality 512 is combined with noise from a standard normal distribution. The decoder LSTM predicts the resulting sequence of amino acids greedily using the $argmax()$ of the $softmax()$ output for every time step. The discriminator is trained on three different parent-child pair config-

urations to optimally compete against the generator, real parent-child pairs, real parent and generated child pairs and pairs of real sequences that are not parent-child pairs. Two bidirectional LSTM encoders sharing the same weight matrix as the encoder of the generator produce a fixed-dimensional encoding of the provided parent-child pair used for classification of evolutionary descent by a plain neural network. The child encoder in the discriminator uses a dense layer having the same weight matrix as the embedding layer of the encoder of the generator. The dense layer is required to directly input the predicted child sequence as its probability distribution rather than the final output after applying the $argmax()$ as it would not enable backpropagation to train the generator. When a true child sequence is given a simple one-hot encoding is used. [4]

Interestingly Berman et al. [4] state planned improvements by utilizing bigger datasets acquired from the GISAID database EpiFlu and by using more length-robust attention-based models to directly work on nucleotide sequences instead of sequences of amino acids. Therefore transformers and the attention mechanism are introduced in the following section.

2.8 Transformer and Attention Mechanism

Using a sequence to sequence model as introduced based on an encoder-decoder architecture of LSTM cells has some drawbacks. First the input needs to be processed sequentially in time steps which makes parallelization difficult and increases training time, especially for longer sequences. Furthermore the hidden and cell state vector passed through every timestep tries to encode information of *all* previous time steps without knowing which information is especially important for the current time step. This not only makes long distance dependencies hard to capture, but also might not get the prioritization of the previous time step inputs right. [3, 19]

To tackle these problems the so-called transformer architecture was introduced in [19]. It feeds an entire sequence into the encoder to be processed in parallel denying any concept of recurrence or convolution. Only using a so-called self-attention mechanism the transformer makes sure that during processing every input position, receives the information that is most important to it. This way modeling long dependencies becomes much more easy compared to LSTMs as the view on the input sequence is more global. In this architecture, the encoder also passes all computed hidden states of every position to the decoder, which therefore [19] can generate the target sequences based on more semantically enriched features and also in parallel for every position. [19]

The transformer architecture achieved state-of-the-art quality results with

a BLEU score of 41.8 on the WMT 2014 English-to-French translation task (cf. BLEU score of [18] was 34.8), while still being much faster to train due to its parallelization capabilities. State-of-the-art results are already achieved after just twelve hours of training on eight P100 GPUs. As this is still far beyond the scope and resources given for this project, this project provides a proof-of-concept transformer model trained for far shorter and fewer sequences as one would need to predict entirely new RNA sequences. [19]

First the transformer architecture should be introduced in the following:

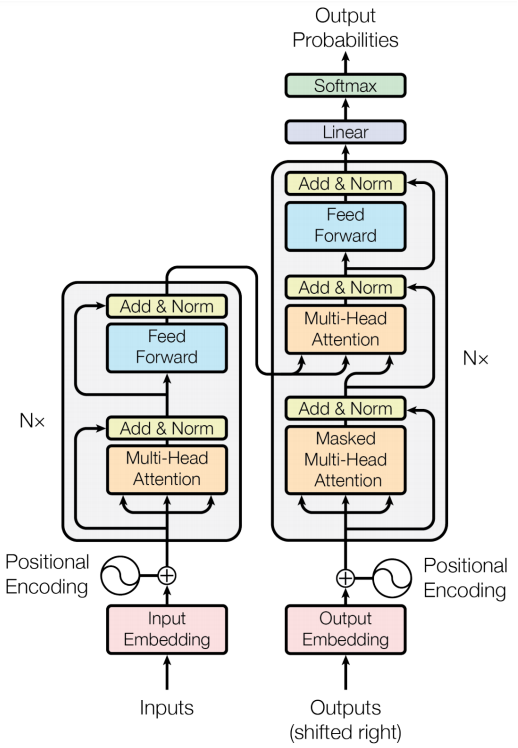


Figure 12: Architecture of the transformer [19]

The transformer consists of an encoder and a decoder part just as normal sequence to sequence models. One layer of the encoder contains a self-attention layer before passing the hidden state representations of an input sequence to a feed forward neural network consisting of two linear transformations separated by Rectified Linear Unit (ReLU). Thus it makes sure that semantic and contextual dependencies between different positions in the input sequence are also modeled. Also skip connections are added for both components with subsequent layer normalization. [2, 19]

To calculate the self-attention output of a sequence, a query Q, key K

and value V matrix is calculated through multiplication of the sequence representation[3] with learned transformation matrices. Each row of the three received matrices corresponds to a specific input position. The output of the self-attention layer is then calculated through

$$(ScaledDot - Product)Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V. \quad (5)$$

The dot product between Q and K calculates scores that are used as weighting factors for V. This describes for every input position how relevant all other input positions are for each of the hidden state encodings. The scores are divided by the square root of the dimensionality of the query/key values of the hidden state representations, which is chosen to be 64 (square root eight), to receive more stable non-vanishing gradients. The softmax guarantees that the positional scores lie between zero and one and add up to one. Note that the score of the current input position itself will most likely have the highest score. [2, 19]

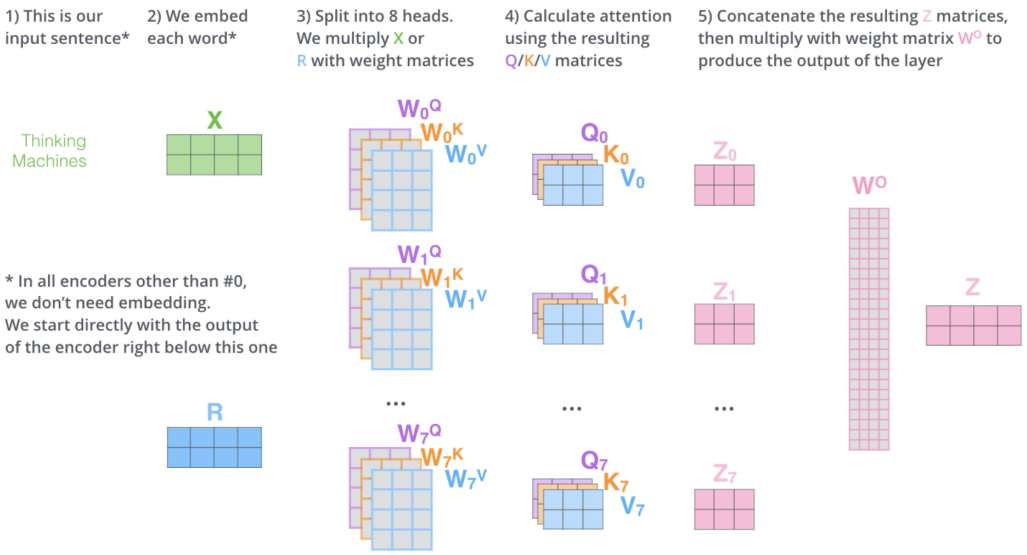


Figure 13: Multi-head attention architecture [2]

One can expand self-attention towards a multi-head attention. In this configuration each head produces its own query, key and value matrices and therefore its own self-attention output in its own representation subspace. The resulting hidden state is calculated by concatenating each hidden state

[3] A matrix containing the hidden state representations of every input position

output of the different heads and multiplying it with a contraction matrix that outputs a hidden state vector in its single-head dimensionality. Multi-head attention is needed to be able to better focus on multiple parts of the input sequence when encoding an input position, otherwise it might happen that most of the focus is set on the input position itself. Vaswani et al. [19] used 8 heads. [2, 19]

Overall six layers of encoder and decoder components having individual weight matrices are stacked on top of each other to capture low-level as well as high-level features. The dimensionality of the hidden state is 512. Note that each position of the input sequence is encoded on an individual path through the transformer, which makes parallelization possible compared to LSTM based architectures. [2, 19]

The decoder uses almost the same architecture as the encoder, but contains an additional multi-head attention component that integrates all the hidden state encodings from the encoder. Therefore the final hidden state encodings resulting from the top most encoder are transformed into key and value attention vectors and given to the integrating mutli-head attention component of every decoder. Once again this helps to better focus on the relevant input positions for the current output position based on hidden state information extracted from the encoder. Note that the decoder works in a sequential manner again meaning that the input to the decoder are the already decoded sequence tokens of previous time steps, future input positions are masked away. The first multi-head attention component therefore capture inter-dependencies in the generated output sequence, the second one, the so-called encoder-decoder attention, enriches the hidden state encoding with the information given from the encoder hidden states. Finally a linear layer mapping the hidden state vector to a logits vector with dimensionality of the given output dictionary and a $softmax()$ function produce the output sequence using the greedy or beam search approach. The EOS token symbolizes the end of the output to be generated. [2, 19]

Input to the transformer are word embeddings also of fixed dimensionality of 512. Onto each word embedding a positional encoding following a specific pattern is added. These patterns are either learned or fixed and make sure that the distances of the input positions are projected onto the queue, key and value representations to be utilized during scaled dot-product attention. This also enables self-attention to be scaled to unseen lengths of sequences. Transformers usually contain an upper bound of sequence length to be processed, which is different to recurrent architectures that can process inputs of arbitrary length. Input sequences are usually padded up until the specified sequence length, the used PAD tokens are masked to be excluded from the self-attention components. [2, 19]

3 Approach

Figure 14 visualizes the steps of our approach, which are described in detail in the following chapters.

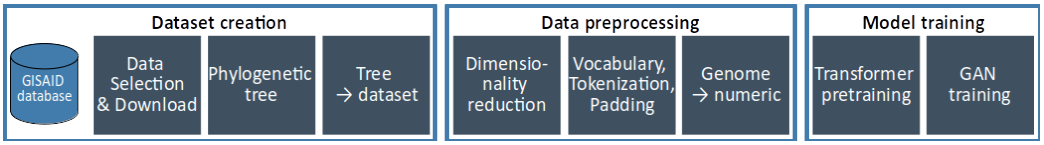


Figure 14: Approach pipeline [**own representation**]

3.1 Dataset Creation

As described in chapter 2.4 a dataset consisting of parent-child pairs is needed to learn a ML model for mutation prediction. The steps to create this dataset are described in the following chapters.

3.1.1 Raw data selection from GISAID

In the first step of the dataset creation the raw genomes and their metadata are downloaded from GISAID. Therefore a selection of a suitable subpart of the data is necessary. The GISAID platform only allows the download of 5000 genomes at once. That is why we decided to focus our analysis on Germany (for one country the selection of < 5000 genomes is rather simple compared to the selection of data from multiple countries). By looking at the latest "Report on virus variants of SARS-CoV-2 in Germany" from the Robert Koch Institut (RKI) we decided for a time period. Figure 15 shows the distribution of different variants over time. Starting from calendar week 18 the Delta variant gradually displaces the previously widespread Alpha variant.

That is why we selected the data from 04.05.2021 (calendar week 18) until 15.07.2021 (calendar week 28). This results in a raw dataset of 35.818 genomes. Each genome consists of a FASTA file containing the genomes sequence and a record in a tsv file with the metadata.

3.1.2 Generation of a phylogenetic tree

As described above we need parent-child genome sequence pairs to be able to train a ML model. Therefore we need to evaluate the ancestral relationships between the genome sequences. As described in chapter 2.1.4 this is done in

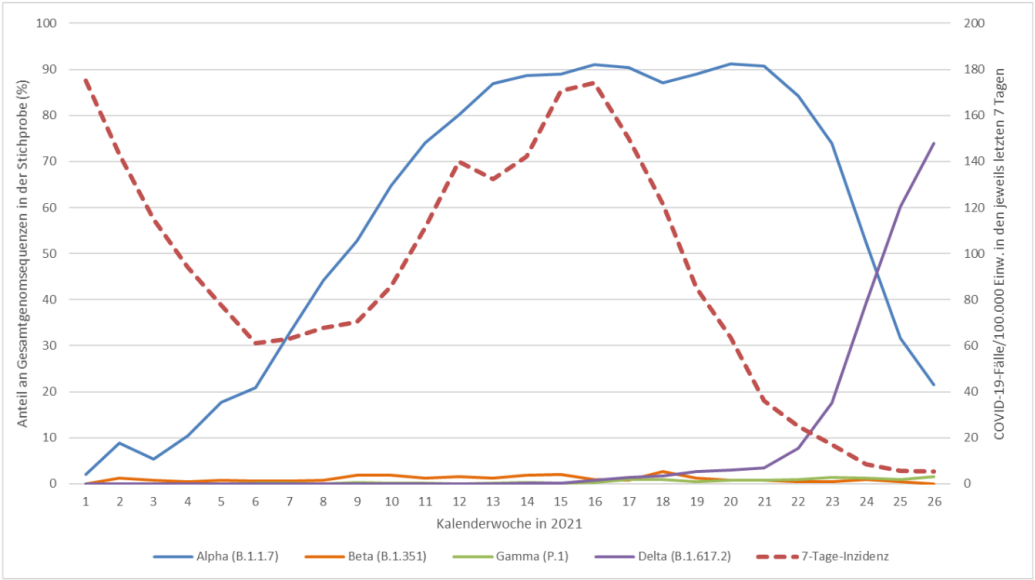


Figure 15: Variant distribution in Germany over time [**robertkochinstituteditorBerichtVirusvariantenSARSCoV22021**]

biology through phylogenetic trees. For the data preparation (e.g. alignment) and the calculation of the phylogenetic tree we use the nextstrain pipeline [8]. The pipeline was executed on a local computer with Intel Core i7-7500U (4*2.70GHz), NVIDIA GeForce 940MX and 16 GB RAM. Unfortunately after two days of calculation, the nextstrain pipeline exited with an out of memory exception. That is why the amount of data is decreased. This was done through iterative reduction of the dataset size. The largest locally computable dataset consists of 11.773 genome sequences.

The generated phylogenetic tree can be seen in figure 16.

3.1.3 Phylogenetic tree to dataset

Based on the calculated phylogenetic tree the final dataset is generated. Therefore the patristic distances between all leaf nodes are calculated. From the resulting patristic distance matrix for each leaf node the most related next leaf node can be evaluated. The two leaf nodes with the smallest distance are the closest relatives to each other and therefore represent a parent-child pair. After the evaluation which leaf nodes are one parent-child pair, the question which node is the parent and which node is the child is clarified. Therefore the two nodes are sorted by their date, which is stored in the metadata file. The older genome sequence is the parent and the younger genome sequence is the child.

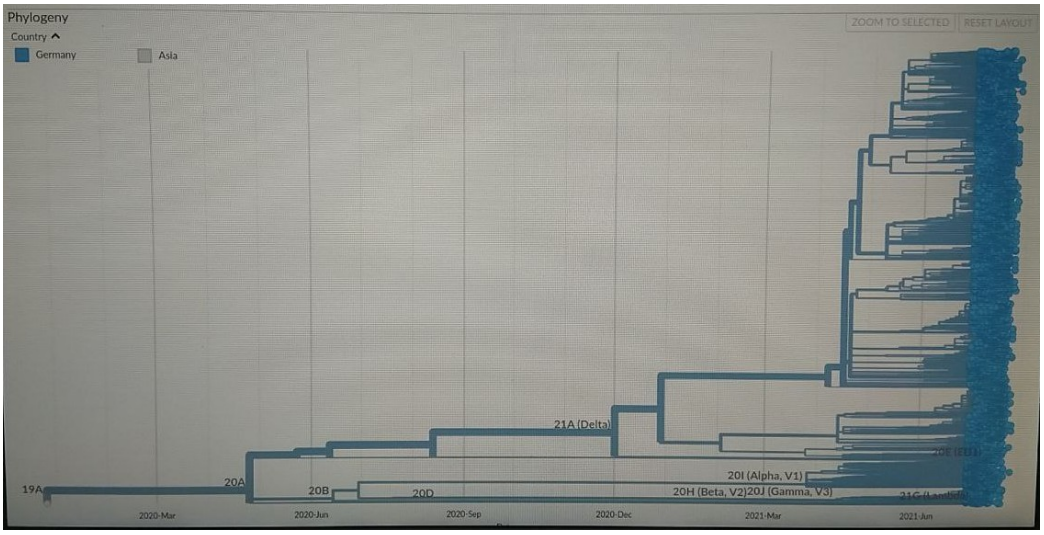


Figure 16: Phylogenetic tree of our dataset [**own representation**]

For calculating the patristic distance matrix we first used Dendropy (version: 4.5.2) [**DendroPyPhylogeneticComputing**]. Again this leads to problems due to limited RAM on our local computer. As a consequence we switched to PhyloDM (version: 1.3.1) [13], which is a library for calculating patristic distances in a memory and time efficient manner.

Finally we received a dataset containing parent-child genome sequences.

3.2 Data Preprocessing

3.2.1 Dimensionality reduction by selecting subpart of the genome

The full SARS-CoV-2 genome consists of 29.904 nucleotides. Due to the limited computing capacity on our local computers we need to reduce the dimensionality of the dataset. This is done by selecting a subpart based on the gained data insights from chapter 4.1. In figure 18 parts of the genome with lots of mutations are visible. Based upon this we selected 99 nucleotides (33 codons) in the range between position 21.800 and 21.899.

3.2.2 Transform genome sequence to numeric model input

Input to the model are numericalized codon sequences with each codon being composed of three nucleotides. Therefore one needs to build a vocabulary with all codons existing in the dataset that assigns each codon a unique number. Each genome string is therefore tokenized to codons and so far

unseen codons are added to the vocabulary. The vocabulary also contains special tokens like:

- $<SOS>$: start of sentence
- $<EOS>$: end of sentence
- $<UNK>$: unknown
- $<PAD>$: padding

A codon is encoded as unknown in case at least one nulceotide is unknown or at least one padding character is contained. Only in the special case of three padding characters the padding token is returned.

To integrate the dataset into the training routine, PyTorchs *DataLoader* is used (see subsection 3.4). Therefore a *CustomGISAIDDataset* class is derived from PyTorchs Dataset class that can be utilized by the *DataLoader*. It can be configured to return the training or test dataset instances. Each instance is already numericalized by vocabulary index lookups for each codon, padded with the numericalized $<SOS>$ and $<EOS>$ token and transformed to a PyTorch tensor.

3.3 Model Architecture

The connection of modeling evolution theory of virus mutations to probabilistic language modeling has already been explained in subsection 2.2. To create such a probabilistic language model an approach closely related to the one introduced by the MutaGAN paper [4] as explained in subsection 2.7 is chosen with the novelty that attention-based transformers are used instead of LSTMs[4]. To implement the GAN framework, PyTorch version 1.9.0 is used as it provides full-featured classes for transformers, embedding layers and other architectural components needed compared to other frameworks like Keras. This way one can specialize on the training routine instead of implementing a custom transformer architecture, which might be too error-prone[5].

The architecture of the generator was shown in Figure 12. The numericalized codon sequences of the source parent and the target child sequence first need to be transformed to input embeddings of size 32 for each codon. This size was chosen experimentally and seemed to work best in practice as

[4]This improvement was also named in the conclusion of the MutaGAN paper [4].

[5]In case one is interested in how to implement a custom transformer architecture, see `http://nlp.seas.harvard.edu/2018/04/03/attention.html`.

it only needs to model a vocabulary of size 40 for the parent vocabulary and 40 for the child vocabulary. To those input embeddings positional encodings are added that are formed by embedding each index in a sequence of positional indices for each instance. To both input sequences dropout layers are applied with a dropout rate of 10%. The transformer then predicts the child sequence as a hidden representation of dimensionality 64 for each codon. The hidden representation is the output of the feed forward layer in a transformer block, its dimensionality was also chosen according to best experimental performance. In total three encoder and decoder layers are used each having eight heads. The transformer also uses a dropout rate of 10%. The final feed forward network reduces the dimensionality from 64 back to the size of the vocabulary for each codon and applies the softmax function to receive the final output probabilities. Using greedy decoding the codon with the highest probability is returned in the evaluation phase as well as during training for simplicity reasons (instead of using a beam search approach). Note that padding tokens or so far unpredicted child tokens are masked away to not be considered when calculating the loss during the training routine.

The architecture of the discriminator is kept similar to Figure 11 using PyTorch's *TransformerEncoders* instead of LSTMs. Again one transforms the numericalized parent and either true or generated child sequence to embeddings of size 32 of each codon and adds the positional embeddings before applying dropout. Instead of using an embedding layer for the true or generated child sequence one uses a dense layer instead in order to directly feed in the probability distribution of generated child sequences. The true child sequence therefore needs to be one-hot encoded before being provided to the discriminator in order to match the shape. The transformer encoder then produces an encoding of dimensionality 64 for each codon. During the training phase dropout is once again used with a rate of 10%. Note that until this step the same weights are used as in the generator. In the following step the two hidden representations of the parent and true or generated child sequence are concatenated along the embedding dimension. The subsequent feed-forward network uses three blocks of dropout, batch normalization and a linear layer with ReLU activation. The ReLU layer in the last block is replaced with a sigmoid layer that makes sure each codon receives a score between zero and one that states how likely its a true or generated codon. Once again padding tokens are masked away during the training phase.

3.4 Training Process

The training phase is two-fold and is performed on a single GeForce GTX 1080 GPU. First a pre-training phase for the generator needs to be performed

in order to produce first reasonable child sequences. Also the weights of the embedding layers and the encoder part of the generator will be reused for the discriminator in the second training phase. For the pre-training phase the cross-entropy loss function typical for language modeling is used, the learning rate is set to 0.005. Teacher forcing is used in the initial pre-training phase to accelerate the training process by leading the generator into a right direction towards reasonable child sequences. In the main training phase the generator and discriminator compete against each other. When training the discriminator, the generator first generates a child sequence given a parent sequence. The true and generated child sequence is then given to the discriminator. Here the binary cross entropy loss is used to model the two parts of the classical GAN loss function:

$$\underset{G}{argmin}\ \underset{D}{argmax}\ E_{x\sim p*(x)}(log(D(x))) + E_{z\sim p(z)}(log(1 - D(G(z)))) \tag{6}$$

PyTorch's *BCELoss* function is given as:

$$l = -w(y\, log(x)) + (1 - y)\, log(1 - x) \tag{7}$$

We therefore pass $y = 1$ when calculating the left part of the loss using the true child sequence and $y = 0$ when calculating the right part of the loss using the generated child sequence. We use the same mechanism to calculate the loss for the generator, but maximize $log(D(G(z)))$ instead of minimizing $log(1-D(G(z)))$. After experiencing mode collapse problems the MutaGAN paper [4] switched to the Wasserstein loss function instead of the binary cross entropy function and therefore also changed the last layer of the discriminator to be a linear activation function. As no mode collapse was discovered and PyTorch does not offer a Wasserstein loss function the binary cross entropy loss function was kept. The learning rate for the generator was set to 0.0001 whereas the learning rate for the discriminator was set far lower to 0.00003 to avoid mode collapse. Besides providing true child sequences and generated child sequences to the discriminator, the MutaGAN paper [4] also provides unrelated, but true child sequences to the discriminator. This way it is hoped to not only learn which mutations are relevant as inherently given by the training dataset, but also to strengthen the knowledge which mutations are especially related to a specific parent sequence and which not. For time reasons the training dataset was not enriched with such true but unrelated child sequences.

Both training phases use batch gradient descent as their optimization method with a batch size of 32. Gradient descent is performed alternatingly between the generator and discriminator on every epoch. Gradients

are clipped to a maximal norm of one to avoid exploding gradients. Furthermore for both training phases the Adam optimizer is used. A learning rate scheduler makes sure to reduce the learning rate by a factor of ten in case the loss has not improved for ten epochs. On every batch gradient descent a tensorboard is updated to draw the loss curve. Every ten epochs a model checkpoint is saved. Pretraining is done for only 50 epochs whereas the main training phase runs for 300 epochs.

4 Experimental results

4.1 Dataset insights

This chapter presents the dataset insights of the complete dataset with full genome sequences and the dataset insights of the preprocessed dataset.

4.1.1 Complete dataset

The dataset consists of 9199 parent-child pairs. The distribution how the parent and child sequences differ can be seen in figure 17. The majority of parent-child pairs differ in less than 200 characters. The number of completely equal parent-child pairs is 396.

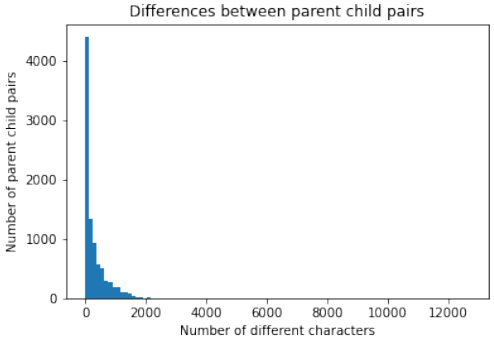


Figure 17: Distribution of the differences between parent and child sequences [**own representation**]

Figure 18 visualizes the distribution of mutations over the full SARS-CoV-2 genome.

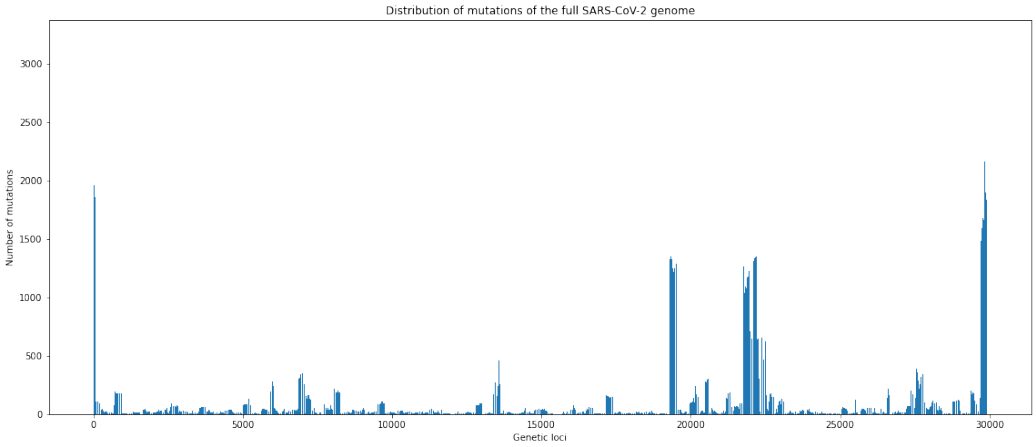


Figure 18: Mutated genetic loci [**own representation**]

4.1.2 Preprocessed dataset

The complete dataset is preprocessed as described in chapter 3.2. The distribution how the parent and child amino acids differ in the selected subpart can be seen in figure 19. The majority (6946) of parent-child pairs are equal. This is not optimal for training the ML model and is further discussed in chapter 4.2.

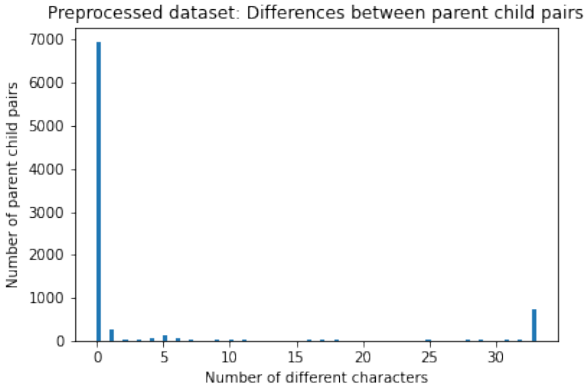


Figure 19: Preprocessed dataset: Distribution of the differences between parent and child sequences [**own representation**]

Figure 20 visualizes the differing amino acids and their positions of the selected subpart of the SARS-CoV-2 genome. They are distributed equally over the selected subpart.

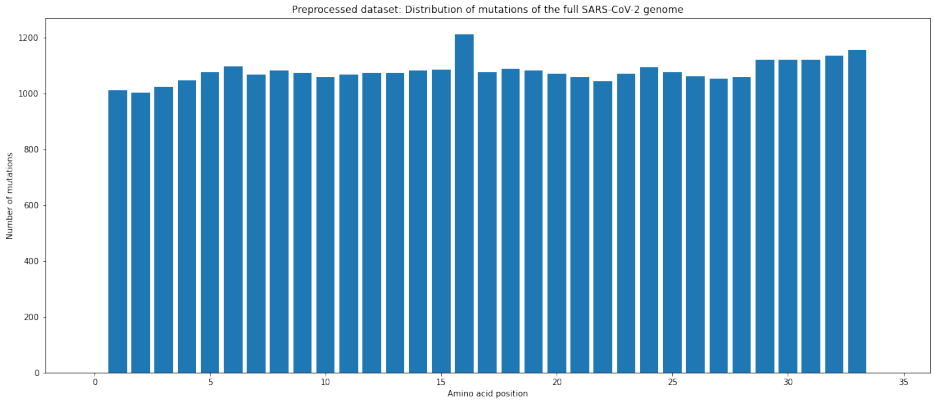


Figure 20: Preprocessed dataset: Differing amino acid positions [**own representation**]

4.2 Evaluation

4.2.1 Evaluation criteria

To measure the success of the transformer model different evaluation criteria need to be introduced that capture how well the mutations observed in the data can be reproduced by the generator. A mutation prediction is considered correct in case the prediction matches in codons and locations. Further to distinguish are partially correct mutation predictions in case mutations are observed at the right locations but in different codons. Mutation predictions can further be considered as wrong in case they were not observed in the true child sequences or missed in case they could not be observed in the generated sequences.

A common metric used in the domain of NMT is the so-called Bilingual Evaluation Understudy (BLEU) score [15]. Its so-called modified n-gram precision captures common n-grams between the generated sentence and the true reference translation sentence independently of their position. Also a sentence brevity penalty is incorporated into the score. The BLEU score ranges between zero and one with one being a perfect match to the reference translation. [15]

Using solely the BLEU score would mean less focus on the concrete mutation prediction at critical genome positions but rather a correct overall child sequence prediction. As the child sequence is mostly similar to the parent sequence, predicting a nearly identical child sequence is not too difficult and the BLEU score would resolves already to a high value at early stages of training.

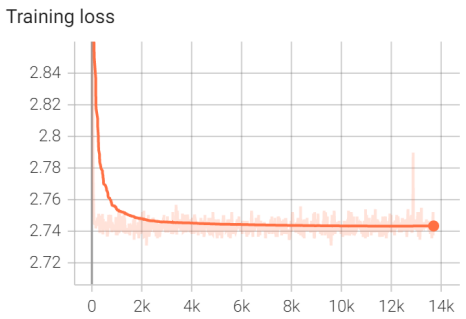
Therefore the MutaGAN paper [4] introduced several other metrics besides the BLEU score. For instance the distribution of specific codon mutations between the ground truth data and the generated data was calculated to evaluate the similarity of both mutation profiles. Furthermore a so-called sequence true positive rate is determined by capturing all observable predicted mutations for each parent sequence and comparing them to the list of actual existing mutations for each parent sequence in the ground truth data.

Besides the BLEU score, we also utilize the sequence true positive rate to evaluate the model using Google's *Diff Match and Patch* libraries to identify the mutations[6]. The Levenshtein distance to determine the degree of similarity between a parent and generated child sequence is not used as at most two mutations appear for a given parent genome during the evaluation phase making the Levenshtein distance neglectible.

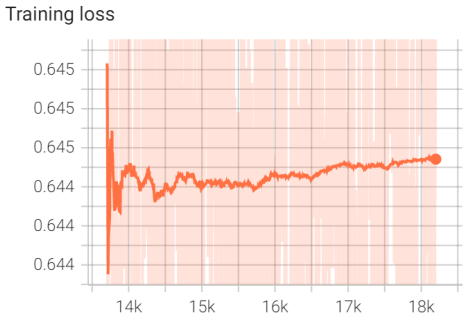
[6]For the Diff Match and Patch libraries, see `https://github.com/google/diff-match-patch`

4.2.2 Training phase and evaluation

Figure 21a shows the loss plot for the transformer pretraining. The model converges rather fast. Similar is the actual GAN training (see 21b).



(a) Transformer pretraining loss plot



(b) Transformer training loss plot

Whereas the trained model achieves a relatively high BLEU score with 87.42%, the sequence true positive rate only amounts to 31%. Thus one concludes that the model generates reasonable and close genomes from given parent sequences, but at first glance develops only partly the capability to predict the actual and correct mutations given in the ground truth data. When further analyzing the rather low sequence true positive rate, one recognizes that a lot of false predictions where made that can be traced back to cases where the parent sequence consists of a series of $<UNK>$ tokens that are replaced by actual codons. As these kinds of mutations are not present in the ground truth data, whose matching child pairs still contain the $<UNK>$ series, one can think of an additional capability of the trained transformer model as a gap filler of unknown sequence parts of parent genomes, that is even beneficial. Other mutations where captured really well, e.g. from codon TTG to CTG at strang location 45 or from codon G to A at strang location x. During evaluation at most two mutations on the same parent in one prediction appeared, matching to the actual ground truth. Some mutations in the ground truth data were also missed, especially in case they were not too frequent. During evaluation only 25 unique parent sequences were considered in the test dataset further reducing a diverse prediction capability of the model. But as data selection and preprocessing is very time and especially resource extensive[7], the scope of the project and its data used is intenionally kept minimal to provide a proof of concept. In this sense, on can conclude that the model first not only generates reasonable genome strang offsprings, but also embodies noticable codon mutations of the ground truth

[7] Phylogenetic tree reconstruction might take several days until completion.

data. When up-scaling the amount of genome sequences to use and doing more strict filtering of genome sequences to increase the data quality, predicting virus mutations using transformers is very well feasible. As a comparison MutaGAN achieves a BLEU score of 97.46% and an unweighted sequence true positive rate of just 21% [4].

5 Conclusion

Coming to a conclusion, our main research question was whether a ML model can be trained to predict the next possible SARS-CoV-2 mutations. The short answer to this is, that our trained ML model is definitely able to generate possible next sequences. The pretraining of the transformer achieved a BLEU score of 0.8559 and the training of the GAN achieved a BLEU score of TODO.

Based on related works introduced in chapter 2, we decided for a ML model architecture consisting of a transfomer based GAN Framework. This architecture was influenced by Berman et al. [4]. Our novelity is the usage of transformers instead of LSTM. This enables faster training, because transformers can be trained in parallel.

For answering our research question we created a new dataset based on raw data from GISAID. As part of this we developed a reusable pipeline for creating evolutionary datasets.

Another novelty is the application and training of such an ML architecture for SARS-CoV-2.

As an outlook, there are some improvements and further investigations we would like to evaluate in future work. As realized while generating data insights of the dataset, lots of our data instances are very similar to each other. To really model the worldwide development of SARS-CoV-2 mutation events, the raw dataset should be based on a subsampling of all available global data instances. Further improvements could be achieved by using a computing cluster for generating a larger dataset and for training the model on the full SARS-CoV-2 genome and not on a subset of the genome. Moreover in the model evaluation more domain specific metrics as proposed by Berman et al. [4] could be used in addition to the BLEU score.

References

[1] GISAID (editor). *GISAID - Mission*. GISAID - Mission. 2021. URL: https://www.gisaid.org/about-us/mission/ (visited on 07/05/2021).

[2] Jay Alammar. *The Illustrated Transformer*. 2018. URL: https://jalammar.github.io/illustrated-transformer/ (visited on 08/18/2021).

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *arXiv* (2016). URL: https://arxiv.org/abs/1409.0473 (visited on 08/19/2021).

[4] Daniel S. Berman et al. "MutaGAN: A Seq2seq GAN Framework to Predict Mutations of Evolving Protein Populations". In: *arXiv* (2020). URL: https://arxiv.org/abs/2008.11790 (visited on 07/05/2021).

[5] H.J. Böckenhauer and D. Bongartz. *Algorithmische Grundlagen der Bioinformatik: Modelle, Methoden und Komplexität*. XLeitfäden der informatik. Vieweg+Teubner Verlag, 2013. ISBN: 978-3-322-80043-5. URL: https://books.google.de/books?id=2HL3BQAAQBAJ.

[6] Peter J. A. Cock et al. "Biopython: freely available Python tools for computational molecular biology and bioinformatics". In: *Bioinformatics* 25.11 (Mar. 2009). tex.eprint: https://academic.oup.com/bioinformatics/article-pdf/25/11/1422/944180/btp163.pdf, pp. 1422–1423. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btp163. URL: https://doi.org/10.1093/bioinformatics/btp163.

[7] Michael Gertz. "Text Analytics - Text Classification, lecture script". University Heidelberg, 2020. (Visited on 08/15/2021).

[8] James Hadfield et al. "Nextstrain: real-time tracking of pathogen evolution". In: *Bioinformatics* 34.23 (May 2018). tex.eprint: https://academic.oup.com/bioinform pdf/34/23/4121/26676762/bty407.pdf, pp. 4121–4123. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty407. URL: https://doi.org/10.1093/bioinformatics/bty407.

[9] Brian Hie et al. "Learning the language of viral evolution and escape". In: *Science* (2021). URL: https://science.sciencemag.org/content/371/6526/284 (visited on 07/05/2021).

[10] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *ResearchGate* (1997). URL: https://www.researchgate.net/publication/13853244_Long_Short-term_Memory (visited on 08/13/2021).

[11] Nan M. Laird and Christoph Lange. *The fundamentals of modern statistical genetics*. 1st. Springer Publishing Company, Incorporated, 2010. ISBN: 978-1-4419-7337-5.

[12] Takwa Mohamed et al. "Long Short-Term Memory Neural Networks for RNA Viruses Mutations Prediction". In: *Hindawi* (2021). URL: https://www.hindawi.com/journals/mpe/2021/9980347/ (visited on 08/11/2021).

[13] Aaron Mussig. *aaronmussig/PhyloDM: 1.3.1*. Version 1.3.1. Oct. 2020. DOI: 10.5281/zenodo.4089111. URL: https://doi.org/10.5281/zenodo.4089111.

[14] Ahmad Abu Turab Naqvi et al. "Insights into SARS-CoV-2 genome, structure, evolution, pathogenesis and therapies: Structural genomics approach". In: *Biochimica et Biophysica Acta (BBA) - Molecular Basis of Disease* 1866.10 (2020), p. 165878. ISSN: 0925-4439. DOI: https://doi.org/10.1016/j.bbadis.2020.165878. URL: https://www.sciencedirect.com/science/article/pii/S092544392030226X.

[15] Kishore Papineni et al. "BLEU: a method for automatic evaluation of machine translation". In: *ACM* (2002). URL: https://dl.acm.org/doi/10.3115/1073083.1073135 (visited on 09/23/2021).

[16] Mostafa Salama, Aboul Ella Hassanien, and Ahmad Mostafa. "The prediction of virus mutation using neural networks and rough set techniques". In: *EURASIP Journal on Bioinformatics and Systems Biology* (2016). URL: https://bsb-eurasipjournals.springeropen.com/articles/10.1186/s13637-016-0042-0 (visited on 07/05/2021).

[17] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. "LSTM Neural Networks for Language Modeling". In: *ISCA Archive* (2012). URL: https://www.isca-speech.org/archive/archive_papers/interspeech_2012/i12_0194.pdf (visited on 08/12/2021).

[18] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *arXiv* (2014). URL: https://arxiv.org/abs/1409.3215 (visited on 05/15/2021).

[19] Ashish Vaswani et al. "Attention Is All You Need". In: *arXiv* (2017). URL: https://arxiv.org/abs/1706.03762 (visited on 08/17/2021).

[20] Guang Wu and Shaomin Yan. "Prediction of mutations engineered by randomness in H5N1 hemagglutinins of influenza A virus". In: *SpringerLink* (2007). URL: https://link.springer.com/article/10.1007/s00726-007-0602-4 (visited on 08/13/2021).

[21] Guang Wu and Shaomin Yan. "Prediction of mutations engineered by randomness in H5N1 neuraminidases from influenza A virus". In: *SpringerLink* (2007). URL: https://link.springer.com/article/10.1007/s00726-007-0579-z (visited on 08/13/2021).

[22] Guang Wu and Shaomin Yan. "Prediction of mutations in H1 neuraminidases from North America influenza A virus engineered by internal randomness". In: (2008). URL: https://link.springer.com/article/10.1007/s11030-008-9067-y (visited on 08/13/2021).