Deep Evolutionary Generative Molecular Modeling for RNA Aptamer Drug Design

Cameron Andress

Submitted in partial fulfillment
of the requirements for the degree of

Master of Science

Department of Computer Science
Brock University
St. Catharines, Ontario

Abstract

Deep Aptamer Evolutionary Model (DAPTEV Model).

Typical drug development processes are costly, time consuming and often manual with regard to research. Aptamers are short, single-stranded oligonucleotides (RNA/DNA) that bind to, and inhibit, target proteins and other types of molecules similar to antibodies. Compared with small-molecule drugs, these aptamers can bind to their targets with high affinity (binding strength) and specificity (designed to uniquely interact with the target only). The typical development process for aptamers utilizes a manual process known as Systematic Evolution of Ligands by Exponential Enrichment (SELEX), which is costly, slow, and often produces mild results.

The focus of this research is to create a deep learning approach for the generating and evolving of aptamer sequences to support aptamer-based drug development. These sequences must be unique, contain at least some level of structural complexity, and have a high level of affinity and specificity for the intended target. Moreover, after training, the deep learning system, known as a Variational Autoencoder, must possess the ability to be queried for new sequences without the need for further training. Currently, this research is applied to the SARS-CoV-2 (Covid-19) spike protein's receptor-binding domain (RBD). However, careful consideration has been placed in the intentional design of a general solution for future viral applications.

Each individual run took five and a half days to complete. Over the course of two months, three runs were performed for three different models. After some sequence, score, and statistical comparisons, it was observed that the deep learning model was able to produce structurally complex aptamers with strong binding affinities and specificities to the target Covid-19 RBD. Furthermore, due to the nature of VAEs, this model is indeed able to be queried for new aptamers of similar quality based on previous training. Results suggest that VAE-based deep learning methods are capable of optimizing aptamer-target binding affinities and specificities (multiobjective learning), and are a strong tool to aid in aptamer-based drug development.

keywords:

Aptamer, RNA, DNA, nucleic acid, Rosetta Commons, Variational Autoencoder, VAE, deep generative model, neural network, genetic algorithm, GA, deep learning, machine learning, artificial intelligence, drug development, Covid-19, SARS-CoV-2

Acknowledgements

First of all, I would like to express my sincerest thanks and gratitude to my MSc supervisor and mentor Dr. Yifeng Li for this wonderful opportunity and all the support he provided, even before the start of my master's degree and throughout. Dr. Li was patient with me and my disability, understanding when I needed a break, and provided a great deal of knowledge and guidance. This is especially noteworthy as we started working together at the onset of the SARS-CoV-2 pandemic, which is still active and has involved multiple lockdowns, mental health afflictions, personal infection, and many lives lost, some of which were friends and family. Dr. Li's influence played a significant role in the expansion of my awareness and understanding of artificial intelligence, machine learning, deep learning, probability theory, and even concepts outside of computer science such as aptamers, drug development, microbiology, and chemistry. Dr. Li encouraged me to seek autonomy and to network, while holding me to a relatively high standard. In doing so, he helped me to develop, in my own way, into a researcher that I can be proud of. I could not imagine having a better supervisor than Dr. Li and I am eternally grateful for his influence in my life.

Additionally, I would like to thank the thesis committee members Dr. Cheryl Mccormick, Dr. Sheridan Houghten, Dr. Ali Emami, and Dr. Tony Yan for taking the time to review my work and offering their insights. Any feedback on this thesis only serves to further my knowledge, skills, and output quality. As such, they are essentially sacrificing their own time during a very busy period of the year for my benefit. Again, thank you very much Dr. Cheryl Mccormick, Dr. Sheridan Houghten, Dr. Ali Emami, and Dr. Tony Yan for doing so. It is greatly appreciated and any feedback is graciously received.

Besides Dr. Li and the thesis committee, I would be remiss not to thank everyone else who has aided me throughout this endeavour: Dr. Kappel's research and implementation with Dr. Das on the RNA docking capabilities of Rosetta is what my research is built upon. Furthermore, Dr. Kappel's willingness to help and her provided support, guidance, suggestions, and clarification when I reached out was incredibly impactful. To Dr. Rhiju Das, Dr. Ramya Rangan, and Dr. Andy Watkins, for all for the guidance, support with the Rosetta systems, and for the introductions. To Dr. Tony Yan and his student/my peer Hayley Bennett for any and all support provided when I reached out with questions relating to aptamers, microbiology, and chemistry. For also being willing to look over some of my work and confirm or correct my logic. To Yage Zhang, Muhetaer Mukaidaisi, and Karl Grantham, my fellow peers under Dr. Li, for coordinating with me when computer maintenance or CPU usage occurred. To Dr. Robson De Grande for teaching me about computer systems and parallelization concepts in class during my undergraduate program and for refreshing me on these concepts during my thesis research. To Cale Fairchild for the timely updates and investigations into recent power outages at Brock University. These updates allowed me to promptly restart my interrupted experiments and not lose too much time being unaware that an issue occurred. To David Wright for providing some technical support when I experienced unknown operating system-related errors. To Dr. Brian Ross, Dr. Ke Qui, Dr. Beatrice Ombuki-Berman, Riley Aitken, everyone at SAS, Lauryn Carrick, Joanna Amodeo, and Elena Genkin for all the administrative support and guidance provided throughout the graduate program. To Dr. Poling Bork, Dr. Renata Dividino, and Jonathan Radue for your friendship, working with my schedule, and for providing inspiration. To Leila Meskine, for all the tutoring provided. To Monica Drenth and everyone at A-Z learning services for their support and being a major contributor to the university experience. To the National Research Council Canada for sponsoring and collaborating on this project. To the Ontario Graduate Scholarship administration for providing funding these last two years. Thank you all!

Last but certainly not least, thank you to my family and friends. Jessica Andress, my loving, supportive, and incredibly patient wife, did everything she could to accommodate both my undergraduate and my graduate endeavours. We put our lives on hold for me to accomplish this milestone and I am forever grateful for her support, companionship, and partnership! Lori and Rick Andress, my parents, who have supported me however they could whenever I needed it. I would not be here today if not for them. Francis Lundrigan, my uncle, who helped refresh my math knowledge and skills at the beginning of the undergraduate program. I have lost of count how many times I had to rely on these skills. Michael Schnell, Agnes and Vince Richard, Ellen Richard, Joel Dube, Craig Styles, Andrew Kornelsen, Tennyson Demchuk, Brett Goddard, Rowan Fraser, Jessica Graham, for all the support, encouragement, and patience with me over the years during my academic career. Jesse-Daniel Belleau-Karaskewich, for introducing me to artificial intelligence. Vanessa Van Decker, Christopher (Cici) Hall, Lucy Ma, Zeenia Desai, Simran Bhaskar, and Anusha Ravikumar, my fellow SEDS exec team members and friends, for the continued patience, encouragement, and for accommodating my timeline. Finally, to all those remaining who chose anonymity, for providing much needed relief and distractions in times of great stress, you know who you are, thank you!

Contents

List of Tables

List of Figures

Acronyms

Notation	**Description**
$_m$RNA	messenger ribonucleic acid.
$_r$RNA	ribosomal ribonucleic acid.
$_t$RNA	transfer ribonucleic acid.
AI	artificial intelligence.
ANN	artificial neural network.
BCE	binary cross entropy.
CNN	convolutional neural network.
CPU	central processing unit.
CSV	comma-separated value.
DAPTEV Model	deep aptamer evolutionary model.
DEL	deep evolutionary learning.
DGM	deep generative model.
DL	deep learning.
DNA	deoxyribonucleic acid.
EC	evolutionary computation.
GA	genetic algorithm.
GC	guanine-cytosine.
GRU	gated recurrent unit.
KL	Kullback-Leibler divergence.
LSTM	long short-term memory.
ML	machine learning.
MLP	multilayer perceptron.
MSE	mean-squared error.
NLP	natural language processing.

Notation	**Description**
nt	nucleotide.
PDB	Protein Data Bank.
RBD	receptor-binding domain.
RMSD	root-mean-squared deviation.
RMSE	root-mean-squared error.
RNA	ribonucleic acid.
RNN	recurrent neural network.
SELEX	systematic evolution of ligands by exponential enrichment.
Seq2Seq	sequence-to-sequence.
VAE	variational autoencoder.

Definitions

Notation	**Description**
Å	Angstroms. A metric unit of length that equates to 10^{-10}m. This measurement is used when discussing the distance between atoms.
Affinity	In the context of aptamers, affinity means the aptamer is well suited to bind to its target (binds strongly to the target with low chance of separation).
Aptamer	A short, single-stranded, oligonucleotide (DNA or RNA), that can bind to specific targets with high affinity and specificity.
Binding	A process in which a molecule's atoms are attracted to each other, resulting in close, stable, physical proximity. This can be thought of as molecules attaching to each other.
Docking	A virtual simulation of the binding process in which one molecule attempts to bind to a given target.
Epoch	A term in AI/ML/DL to indicate that the model has seen all training data one time. This term is usually accompanied by some context of iteration.
Fitness Function	A function which produces a numerical value (the fitness) representing how well a data point performs in a given task. These values are used when comparing data points against each other.
Heuristic	A method used to more quickly solve, or approximate a solution to, a problem based on known features of the problem or data, but does not learn from data.

Notation	Description
In Silico	Pertaining to tests that take place in a computer-simulated environment.
In Vitro	Pertaining to tests that take place in a controlled environment, typically test tubes and petri dishes.
In Vivo	Pertaining to tests that take place in a living organism, typically within animals, plants, and the human body.
Monte Carlo	In computing, a Monte Carlo algorithm utilizes randomness to produce result with a low probability of being incorrect. These algorithms are often used when performing a full search/computation would take too long.
Multiprocessing	The utilization of multiple CPUs to run separate tasks/programs/instructions at the same time (in parallel). Not to be confused with multithreading, hyper-threading, or concurrency.
Overfitting	A concept in AI whereby the model learns to represent the training data too well and testing performance suffers as a result.
Receptor-Binding Domain	The location of a virus protein that binds to cell receptors and injects infectious DNA.
Specificity	In the context of aptamers, specificity means the aptamer binds **only** to its target and nothing else.
String	The computing terminology for "text". Programming languages store text as "strings".

Chapter 1

Introduction

There are two main types of nucleic acids. The first type is known as deoxyribonucleic acid (DNA) and the other is known as ribonucleic acid (RNA). DNA is typically double-stranded, houses instructions for protein creation, and resides in the nucleus of the cells within one's body. DNA contains four *bases*, each known as a type of nucleotide (nt). These bases are Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). RNA is similar to DNA except that it is single-stranded and has bases of Adenine (A), Cytosine (C), Guanine (G), and **Uracil (U)**. While there are many types of RNA, for the purposes of this introduction, RNA can be further divided into three types - messenger ribonucleic acid ($_m$RNA), ribosomal ribonucleic acid ($_r$RNA) also known as the *ribosome*, and transfer ribonucleic acid ($_t$RNA).

The synthesis of $_m$RNA also occurs in the cell nucleus. The $_m$RNA will read the sequence information from the DNA and produce an RNA-relative *copy* (complementary bases) in a process known as *transcription*. Surrounding the cell nucleus is an area of cytoplasm. This area of cytoplasm between the cell walls and the cell nucleus contains other types of molecules such as amino acids, $_t$RNA, $_r$RNA, and others outside the scope of this research.

These three types of RNA are crucial in the protein creation process. The $_m$RNA is responsible for carrying instructions from the DNA to the ribosome. The ribosome then interprets the instructions from the $_m$RNA and tasks various $_t$RNA to collect surrounding amino acids from the cytoplasm area. This process is known as *translation*. When the $_t$RNA return with the requested amino acid, the ribosome will connect these amino acids in a chain according to the $_m$RNA instruction. This chain of amino acids is known as a polypeptide chain. This is the first form of a protein. Then the polypeptide chain conforms to its secondary, then tertiary structure (a literal three-dimensional structure). At this point, many three-dimensionally conformed polypeptide chains are considered proteins. However, there exist some proteins which interact with other polypeptide chains that form a quaternary structure and this too would be considered a protein [21, 23, 40, 85].

1.1 Problem Statement

Viruses have outer protein shells and contain infectious DNA or RNA. A virus is incapable of self-reproduction and must commandeer the cell's protein creation capabilities to reproduce. After the virus protein has successfully reproduced, it then goes on to infect other cells in a process known as proliferation. The attaching and injecting of infectious DNA is achieved through a cell's receptor. Thus, these viral proteins have what is known as a receptor-binding domain (RBD), an area on the viral protein that has adapted to the conditions of the host cell receptor. In the case of the SARS-CoV-2 spike protein, which is the subject of this research topic, the RBD has adapted to the lung cell ACE2 receptors [43, 48, 70, 72, 76, 81, 86].

Typical drug development focuses on either triggering a host's immune system or interrupting the life cycle of the virus. The former is typically thought of as a preventative measure. This usually takes the form of vaccines preparing the host's immune system. The latter is known as a therapeutic, attempting to halt the infection process in a currently-infected host [43, 84, 87].

For antibody-related preventative drugs (vaccines), if all goes well, the biological response is to develop antibodies that last for a period of time. In the case of SARS-CoV-2, the antibody immunity period lasts approximately five to seven months [68]. Should the host become infected with the virus targeted by the vaccine during the immunity period, the antibodies will bind to the virus and inhibit it from interacting with cell receptors.

These vaccines can be cultivated in several ways. Usually, this involves introducing the host to a weakened virus, injecting the DNA to begin minor viral protein production in host cells or, in the case of the SARS-CoV-2 $_m$RNA vaccine, skipping the DNA transcription step and proceeding right to translation via $_m$RNA [16].

There are some issues inherent with the vaccine and vaccine development process, however. The first issue is the amount of time required to cultivate the virus, DNA, or RNA into a vaccine. Another issue is that there is often a large development cost associated with researching and producing vaccines. Lastly, these vaccines tend to focus on infection prevention. If the host is already infected, the immune system is weakened and attempting to produce antibodies. Injecting a weakened version of the virus or the infectious nucleic acid will not help [66, 87].

This is where aptamers can help. Aptamers are short, single-stranded, oligonucleotides (DNA or RNA), that have folded into their tertiary structures and can bind to specific targets with high affinity and specificity. Notice that aptamers binding mimics antibody binding. Here, affinity refers to how well the aptamer will bind to its target and specificity refers to the ability to **only** bind to its target. However, unlike vaccines which invoke the immune system to produce antibodies prior to infection, aptamers can halt a virus during its life cycle by binding to the virus protein RBD, inhibiting the protein from binding to the host cell receptor. This classifies aptamer-based drugs as therapeutics. Also, aptamers can be created and modified easily and can bind to many molecules, unlike antibodies. Unfortunately, one of the main ways to develop aptamers is through a process known as systematic evolution of ligands by exponential enrichment (SELEX) which can take anywhere from 4 weeks to eight or more months (depending on required additional steps) just to produce aptamers with mild binding qualities on average [1, 71]. Also, the SELEX process is very labour-intensive and is essentially a random search similar to the "hill climber" algorithm as described in "Experiments" [9, 15, 42, 43, 44, 71, 85, 87, 88].

With all of the issues regarding the SELEX process, this creates a need to digitally simulate the binding process. This is the main focus of this research, applied only to RNA data against the SARS-CoV-2 spike protein. RNA data was chosen specifically because DNA data has been explored extensively, but RNA data seems to be under-researched. Note that when binding is simulated in silico, it is called docking. Rosetta will be used to simulate and score the docking procedures [41].

1.2 Contributions

The goal of this research is to determine if a deep generative model (DGM) can accelerate the RNA aptamer drug development process. Thus, the DGM must display three capabilities. First, the DGM must produce a list of aptamers that yield strong docking scores to the SARS-CoV-2 spike protein's receptor-binding domain (RBD), suggesting a strong target binding affinity. Secondly, the DGM must produce structurally complex sequences, suggesting a strong target binding specificity. Thirdly, the model should provide the ability to produce new sequences immediately rather than having to retrain every time. Furthermore, it would be ideal if the training for this model could be achieved in a reasonable amount of time.

In addition to these requirements, this research also contributes the following: (1)

a method to generate an entire dataset rather than having to wait for experimental data to be gathered; (2) entire secondary structure prediction capabilities over substructure motif selection; (3) tertiary structure prediction for better control and understanding of aptamer information; (4) aptamer-target RBD docking predictions, allowing the user to target a specific virus; and (5) a pipeline that connects the input and output of well-performing prediction programs to a deep learning (DL) system, serving as an end-to-end model. While this research is applied to the SARS-CoV-2 spike protein, careful consideration has been placed into the universal design of nearly any protein target that can be obtained from the Protein Data Bank (PDB) website or elsewhere. Henceforth, “the model” will be referred to as the deep aptamer evolutionary model (DAPTEV Model).

1.3 Related Work

At the beginning of this research, during the literature review (completed around June, 2020), finding sources of related work was difficult. There were many papers on aptamers, nucleic acid design, and drug development, but it was only recently that the patents expired for the SELEX process [85]. This limited a lot of the potential exploration and innovation regarding aptamers. Moreover, machine learning (ML) applied to aptamers was even more scarce. However, shortly after performing the literature review, several related papers were released. Some of which implemented a model similar to the one proposed in this research.

1.3.1 Similar Research

Grantham et al. have developed a DL model that they refer to as deep evolutionary learning (DEL) for molecular design. They found their method was “beneficial for improving sample populations” and that their model “exhibits improvements on property distributions, and dominates samples generated by other baseline molecular optimization algorithms” [26]. Their model serves as inspiration for this DAPTEV Model. Published April 12, 2022.

Im et al. used a generative model to build DNA and RNA sequences that bind to a “target protein”. At the time of publication, it was specified that their research was ongoing, but they were able to train a long short-term memory (LSTM) recurrent neural network (RNN) on a “huge dataset of sequences from high-throughput experimental technologies”. It was found that the produced sequences possessed structural motifs similar to known motifs, and that the produced sequences had a strong binding affinity and specificity to their intended target [36]. This research was continued by Park et al. and a similar conclusion was drawn for RNA aptamers specifically [62]. Published August 15, 2018 and January 07, 2020 respectively.

Iwano et al. have a similar approach to Im et al. and Park et al., also utilizing DeepBind. Except, Iwano et al. implemented a convolutional neural network (CNN)-LSTM as their encoder and a profile hidden Markov model as their decoder for their variational autoencoder (VAE). Additionally, while Iwano et al. started with static sequence lengths, they utilized some post-processing techniques to extend the sequence lengths to a set of other fixed sizes, technically achieving sequence generation with variable lengths [38]. Published February 17, 2021. Their code and dataset is available at `https://github.com/hmdlab/raptgen`.

Lee et al. produced research that is the most similar to this DAPTEV Model. Lee et al. created a generative model which produces RNA of various lengths (30, 50, 70, and 90 nt) **and** their method allows for the specifying of a target sequence through the use of ZDOCK. It should be noted that Lee et al. attempted to accept DNA in their model similar to how this DAPTEV Model accepts DNA data, by converting thymine to uracil. However, Lee et al.'s "generative model" is a decision tree with a random forest algorithm that results in the "generating" of sequences. There is no mention of Gaussian distributions or a latent space, nor a decoder. In other words, their method does not use a DGM/VAE nor probabilistic learning. Lee et al. explain that they also use a "Monte Carlo tree search approach" to mimic an end-to-end generative model. Thus, Lee et al.'s model is not a DL algorithm. Lee et al.'s score function is the Matthew's correlation coefficient. This research was not applied to Covid-19. Although, it does seem like this would be possible. Results indicated equal or better performance of generated aptamers to target proteins than previous benchmarks [50]. Published June 25, 2021.

Song et al. produced similar results to Lee et al. except their implementation involved a Markov Cluster-based ML approach. Song et al. attempted to address the current issue of building secondary substructures and using these structures with SELEX data, stating this negatively affects the accuracy. Song et al.'s implementation focuses on classification rather than generation [71]. Published December 26, 2019.

Wornow et al. propose a conditional VAE, utilizing LSTMs as well, to generate novel high-binding aptamers. This research is applied to daunomycin. This too is similar to the DAPTEV Model's work and findings. However, Wornow et al.'s work utilized 8 rounds of SELEX data as a proof of concept. This creates a limitation of requiring experimental data to be collected before their model's efficacy can be illustrated or be used in practice [85]. Published April, 2020.

1.3.2 Tangentially Related Research

Li et al.'s work attempted to classify aptamer-target pairs. In other words, they attempted to discern if an aptamer would or would not bind to its target. This is a classification model and not a generative model [51]. Published January 22, 2014.

Hamada discusses in silico approaches for RNA aptamers. However, very little was discussed regarding artificial intelligence (AI)/ML/DL and instead a focus was given to discrete applications [28]. This was a useful source of RNA design knowledge. Published February 01, 2018.

Chen et al. performed a review discussing the advancements of different discrete and ML/DL approaches to the prediction of aptamer binding ability. It was found that structure-based methods are the most widely used in silico method for drug design. It was also mentioned that Chen et al.'s hope is to facilitate ML-aided applications of high-throughput aptamer selection [15]. Published March 30, 2021

Bashir et al. explore the usage of ML models as a means to predict DNA aptamer binding affinity. For example, one of their models received sequences as one-hot encoding vectors and the output was the predicted affinity score. It was found that ML algorithms are capable of learning structural motifs and other aptamer features, and can be a useful tool in expediting the search for therapeutic agents [8]. Published April 22, 2021.

Heredia et al. developed a novel ML approach which extracted sequence information via a natural language processing (NLP) model. This extracted information was used in the training of a logistic regression model, a decision tree, a Gaussian Naïve Bayes model, and a support vector machine. Heredia et al.'s goal was "to help discriminate binding between DNA aptamers from those unspecified targets of DNA-binding sequences". In other words, Heredia et al. attempted to discern if a sequence was indeed an aptamer or not [31]. Published August 3, 2021.

A worthy mention is the work performed by Fare et al. They attempted to combine a genetic algorithm (GA) with a neural network, acting as a fitness function, to optimize the SELEX process for RNA aptamers. However, there does not seem to be any published work, and most online sources are no longer operational [35].

1.3.3 Deviations from Past Research

When this research first began, it seemed that similar publications mostly focused on predicting the binding affinity of DNA aptamers, or classifying if an aptamer would bind to its target. Since then, more research was published attempting both DNA and RNA aptamer sequence generation utilizing a plethora of implementations. Im et al. and Song et al. where among the few to previously publish sequence generating models. However, the work published in this space seems to be limited by a number of factors.

Many models utilize a previously-developed and trained ML system known as "DeepBind" to predict "the interactions between proteins and nucleic acids" [62]. DeepBind was trained on hundreds of distinct proteins. However, DeepBind only yields binding affinity scores without the context of the target's RBD and does not provide the ability to specify target proteins, nor the target's RBD. Furthermore, the usage of DeepBind restricts the user to the proteins on which DeepBind trained. This means any new viruses or mutations are not updated within the previously trained model. Even if this **was** included, the scores provided by the system would be generalizations and not specific to the target unless an individual model for the target was trained specifically. Conversely, this DAPTEV Model utilizes Rosetta to "model RNA-protein complexes by simultaneously folding and docking RNA to a protein surface" via a highly accurate statistical RNA-protein scoring function [41]. This allows the user of the DAPTEV Model to specify a target protein and the target's RBD. The target in question for this research is the SARS-CoV-2 spike protein's RBD, but any protein target can be used. The main drawback for the DAPTEV Model is the computational demand and time required to perform docking simulations. However, this limitation can be overcome via the use of multiprocessing and cluster-distributed computing.

Most models seem to require a starting dataset derived by high-throughput experimentation. Even the dataset obtained through DeepBind, which is currently unavailable, is developed using this method. This creates the limitation of requiring previous high-throughput data before the model can be used. This DAPTEV Model allows users to generate random sequences, and a starting dataset, with control over the percentage of guanine-cytosine (GC) in each sequence, and the ability to only allow folded structures into the model. This also allows the user to start with experimental data if they so choose, but does not require it. Additionally, further augmentation to the random sequence generation can serve to improve initial data generation and can provide greater control over the initially produced sequences.

Most of the aforementioned generative models utilize either an LSTM RNN, or some form of tree structure. The DAPTEV Model implements a gated recurrent unit (GRU) RNN for the encoder and decoder models. Additionally, the DAPTEV Model also implements a multilayer perceptron (MLP) model into the VAE for the purpose of binding classification and serves as an additional regularization method.

Lastly, there is an observed commonality that previous implementations either restrict the length of the sequences or allow for accepted, predetermined, sequence lengths. The DAPTEV Model uses and produces variable length sequences with a range of 20 to 40 nt chosen in the parameters. However, this range can increase or decrease according to the user's preference. Also, there does not seem to be any secondary or tertiary structure prediction capabilities in the majority of these model. While this does add additional complexity and computational demand, the tradeoff is finer control and knowledge over the produced sequences. It is for this reason that the DAPTEV Model implements these features.

To the best of our knowledge, this is the first attempt to combine: (1) an RNA-aptamer sequence probabilistic DGM model with evolutionary computation (EC); (2) the ability to specify the target and the target's RBD; (3) the inclusion of non-canonical random aptamer sequence generation of true variable length; (4) complete secondary structure prediction and tertiary structure prediction capabilities; (5) docking predictions; and (6) the ability to also include DNA if the user converts to RNA via thymine to uracil base change.

Chapter 2

System Design

The approach for the development of RNA aptamers designed for a specific virus is as follows:

1. Download target virus data from the PDB website
2. Modify the raw PDB and FASTA files
3. Compile starting dataset
4. Set Rosetta parameters
5. Set dataset preparation script parameters
6. Generate dataset
7. Set starting DAPTEV Model parameters
8. Load PDB virus and compiled dataset into DAPTEV Model
9. Deep learning and evolutionary computation
10. Random RNA sequence generation
11. RNA secondary structure prediction
12. Multiprocessing: tertiary structure prediction & docking simulation
13. Merge data, sort by score, take top M sequences
14. Repeat from step 9 until complete

The user must first download the target virus data from the Protein Data Bank (PDB) website [10] and modify the contents to be compatible with Rosetta [41]. For a more in-depth description of this process, see "Data Preparation". Following this step, the user will compile all (if any) known RNA aptamer sequences and secondary structures into one or more comma-separated value (CSV) files. It is expected that the secondary structures for these sequences are known ahead of time. If secondary structures are unknown, these can be quickly computed using Arnie [83] (see "RNA Secondary Structure Prediction").

The next step is for the user to create the dataset as it relates to the target virus. This can be accomplished by initially setting the Rosetta and dataset preparation script parameters (see "Parameters"), and running the dataset creation Python code (called "create_dataset.py"). The execution of this script is required as it will perform several necessary actions. First, this script will check each RNA sequence against the user-specified requirements established during parameter entry. If any starting data does not conform to the user's requirements, the script will filter out and replace these sequences with random RNA sequences and Arnie-predicted secondary structures until the desired number of data points has been reached. Then, the script will run each data point through Rosetta's RNA tertiary structure prediction and docking simulation to obtain target virus-related docking scores. Multiprocessing is utilized during this step to accelerate the prediction computation time. See "Tertiary Structure & Docking Prediction", "Score Functions", and "Multiprocessing" for more on information. Lastly, the script will save the resulting dataset in a DAPTEV Model-compatible format. Note that the user can generate an entire dataset of random RNA structures with this script but this is not recommended. More information on this can be found in "The Data", "Random Sequence Generation", and "RNA Secondary Structure Prediction".

With the last step complete, the user has a fully prepared dataset. Now, the user must enter their desired starting parameters for the ML system (the DAPTEV Model) and direct the DAPTEV Model to the recently-constructed dataset. At this point, the user is ready to run the DAPTEV Model program via the "run_all.py" script. What follows is a description of the steps performed by the DAPTEV Model. A flowchart visualizing this process can be seen in Figure 2.1. Note that the VAE and EC approach is inspired by similar research conducted by Grantham et al. [26].

Upon execution of the DAPTEV Model, the dataset is loaded in. Then, the sequences and their docking scores are sent into a deep probabilistic neural generative model, or DGM for short, known as a VAE. This VAE is responsible for learning how to effectively simplify (encode) key features of the sequences, storing these features in a latent space, then rebuilding (decoding) samples from the latent space. These produced samples will be similar, possibly even identical (depending on how well the VAE learns), to the input sequences based on the learned features. The nature of the learnable latent features is respective to the application. For this application, some examples of "key features" are nucleic acid base pair interactions, structural motifs like stem loops and bulges, and other observable characteristics of RNAs. See "Deep Learning and Evolutionary Computation" for more information on this subject and how features are stored.

After the VAE has encoded key features to the latent space, Darwinian evolutionary operations such as selection, crossover, and mutation are performed on the latent space to explore the docking score landscape for the specified target. Each generation requires a new round of structure predictions and docking simulations to obtain new scores and confirm the next generation's performance. This process is repeated multiple times until a certain number of generations have occurred. The optimal number of generations depends on one's parameter choices and is determined empirically. Further discussion on this can be found in "Deep Learning and Evolutionary Computation" and Figure 2.1.

Additionally, as a comparison method, another AI model, known as a genetic algorithm (GA), will perform the same operations as the ones performed on the latent space of the VAE. Except, this GA will operate only on the initial input sequences and not the latent space. This way, the sequences obtained from the VAE can be compared against the separate GA-produced sequences, serving as a benchmark. Figure 2.2 illustrates the GA process. It is expected that the encoding nature of the VAE will be able to identify the most important features of the sequences, making the search space more robust.

Once the evolutionary operations are complete, if the predicted data contains duplicates, these duplicates will be removed and replaced with random sequences that contain at least one connection/base pairing in its secondary structure (similar to the dataset script). "Duplicates", in this case, refers to repeat sequences in the current generation's prediction, predicted sequences that already exist in the training set, or predicted sequences that have been predicted in a previous generation.

Following the duplicate replacement process, the newly updated dataset of sequences will have their secondary structures predicted. Then, this data is sent to Rosetta for another round of tertiary structure and docking predictions, computed using multiprocessing. The associated scores are added to the dataset.

At this point, the dataset is two times its original size. The dataset will then be sorted according to the score values from smallest to largest. The top M sequences will be collected and the remainder will be discarded. "M" in this instance refers to the user's chosen population size for the DAPTEV Model (see "Machine Learning Parameters"). If this is not the final generation, this data will be used to fine-tune (continue training) the VAE, and the process will repeat. If this is the final generation, this data will be returned to the user as the output from the DAPTEV Model.

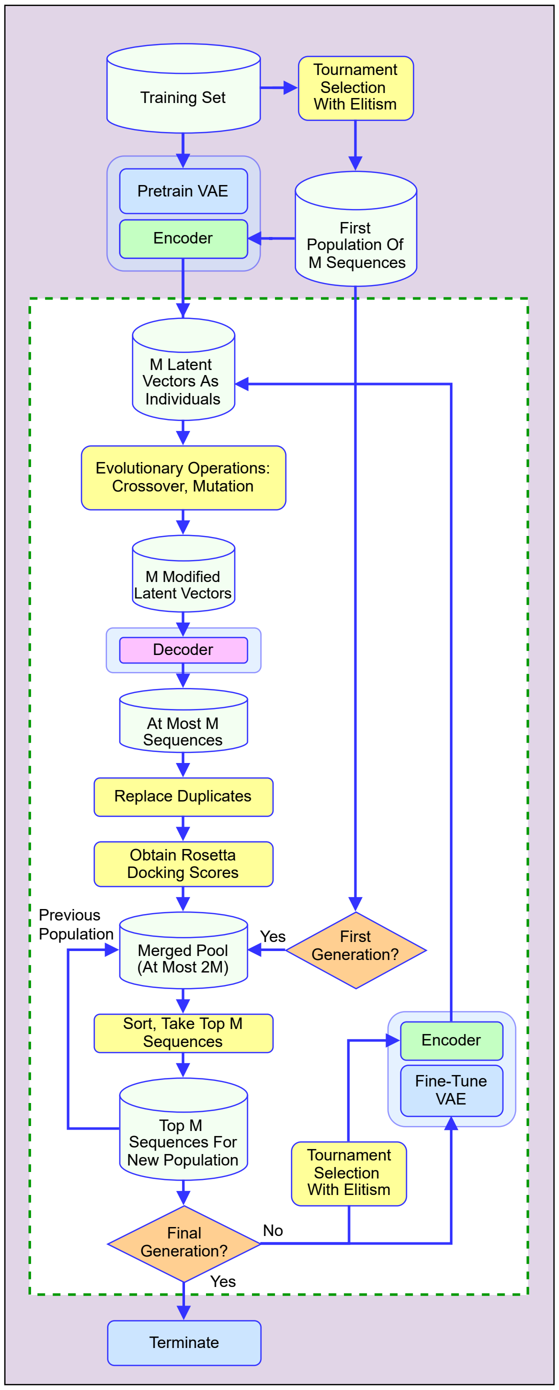

Training Set
Tournament Selection With Elitism
Pretrain VAE
Encoder
First Population Of M Sequences
M Latent Vectors As Individuals
Evolutionary Operations: Crossover, Mutation
M Modified Latent Vectors
Decoder
At Most M Sequences
Replace Duplicates
Obtain Rosetta Docking Scores
Previous Population
Merged Pool (At Most 2M)
Yes
First Generation?
Sort, Take Top M Sequences
Encoder
Fine-Tune VAE
Top M Sequences For New Population
Tournament Selection With Elitism
Final Generation?
No
Yes
Terminate

Figure 2.1: A flowchart depicting the deep aptamer evolutionary model (DAPTEV Model) process for aptamer design.

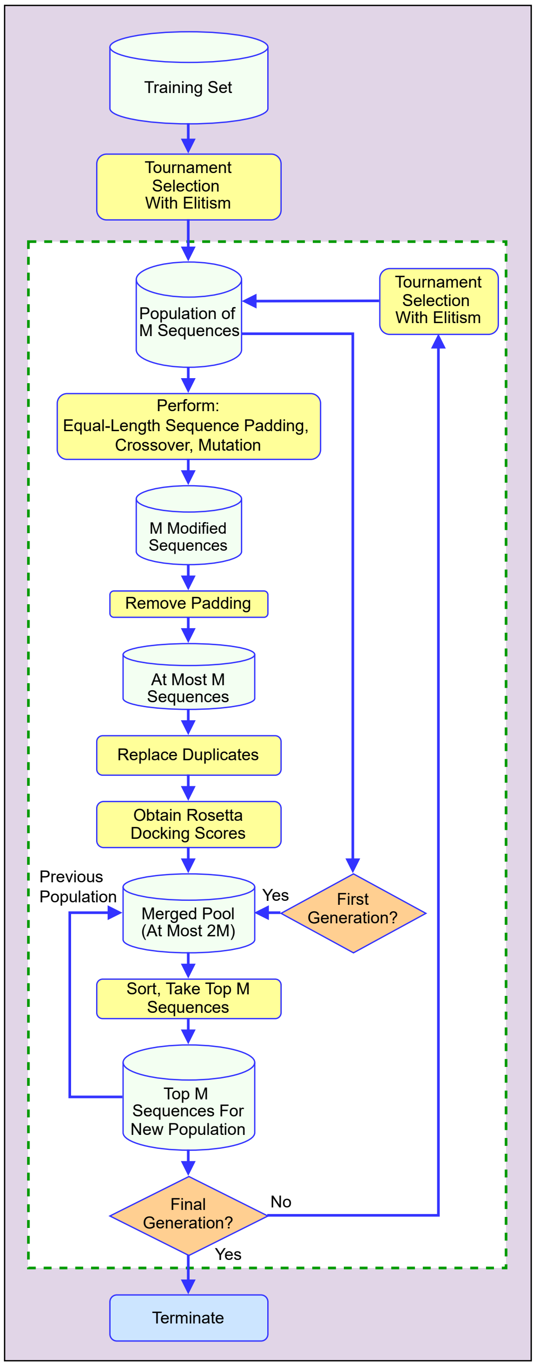

Training Set
Tournament Selection With Elitism
Population of M Sequences
Tournament Selection With Elitism
Perform: Equal-Length Sequence Padding, Crossover, Mutation
M Modified Sequences
Remove Padding
At Most M Sequences
Replace Duplicates
Obtain Rosetta Docking Scores
Previous Population
Merged Pool (At Most 2M)
Yes
First Generation?
Sort, Take Top M Sequences
Top M Sequences For New Population
Final Generation?
No
Yes
Terminate

Figure 2.2: A flowchart depicting the genetic algorithm (GA) process for aptamer design.

2.1 Deep Learning and Evolutionary Computation

In the following sections, the example of performing an experiment with parameters of 800 for the population size, 10 for the number of generations, and 3 for the number of runs in the context of reducing computation time is discussed. It should be noted these values can be considered conservative for DL applications. Additionally, DL systems typically utilize thousands of data points in the initial dataset for training.

Traditional ML methods usually require some data preprocessing (filtering out noisy data, outliers, incorrect information, etc.) before data can be used for training. This is because the quantity of data points is often relatively low and we do not want the ML system to learn inaccurate associations or base its conclusions on errors. DL, however, is a subset of ML that typically uses significantly more data and, as a result, often does not require as much data preprocessing. Note that, while DL is a subset of ML, not all ML models can be considered DL models. The distinction here is between ML models and DL models that do not overlap.

DL aims to model human learning as closely as possible. Humans are often subjected to noisy and incorrect data, but human brains are proficient at filtering these inaccuracies out and updating logic during the learning process. As such, DL models are often considered to be more effective than traditional ML models at exploring a fitness landscape.

2.1.1 Fitness Landscape

The term "fitness/problem/score landscape" has been mentioned multiple times now, but what exactly does it mean? Consider a problem's solution search space as a landscape containing hills and valleys as depicted in Figure 2.3. A problem may have many potential solutions, but some solutions will be inherently better than others. For example, when searching for a needle in a haystack, one could sift through the hay one at a time or one could use a metal detector. This landscape, therefore, constitutes all possible solutions to any problem containing numeric values representing the quality of the solutions. These numerical values are known as a solution's "fitness". Thus, changes in elevation on this plane are indicative of how well suited, how fit, a potential solution is to the task at hand. Neighbouring solutions also tend to have similar fitness values. As such, this landscape can have hills and valleys of different sizes and in different locations. For optimization tasks, where one attempts to find the single best answer to a problem, the goal is to locate the solution at the *global maximum* (tallest hill) or *global minimum* (lowest valley). Here, "global" means "for the entire

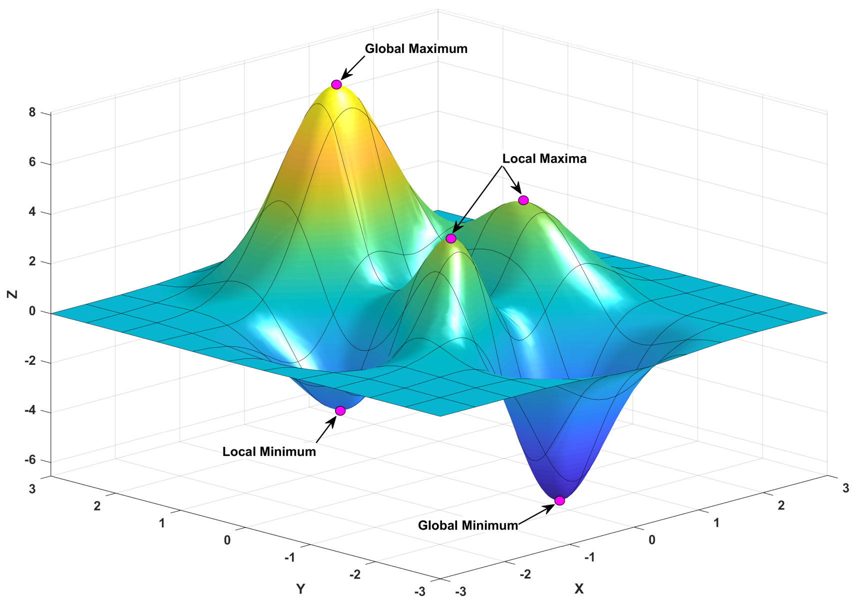


Figure 2.3: A three-dimensional plot of a "fitness landscape". Source: the "peaks" function in MATLAB (a translation and scaling of Gaussian distributions) [56]
Formula: $z = 3(1-x)^2e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}$

problem's search space". Conversely, a *local* minima or maxima means the potential solution is only the best of its neighbours and translates to a smaller hill or more shallow valley. Note that the best solution or goal can be either a minimum or a maximum. It all depends on whether the goal is to minimize or maximize the solution's numerical values. In the case of aptamer discovery for drug development, the goal is to minimize the returned docking scores, thus prioritizing the minima.

A good example of encountering a local minimum in this application would be the production of an unfolded RNA in silico as discussed in "Additional Considerations" and "Native Rosetta Score Function". Unfolded RNAs are less likely to experience penalties during the structure prediction and docking process due to the RNA's lack of self-interaction and its malleability to the target RBD conditions. This, in turn, can yield very low structure prediction and docking scores, resulting in the algorithm prioritizing unfolded RNAs. However, aptamers are designed not only for high target binding affinity but also for target specificity [9, 15, 42, 43, 44, 85, 87, 88]. Thus, this scenario is unlikely to occur in vitro or in vivo and is, therefore, not the best solution for aptamer drug development. For an example of a docked unfolded RNA, see the best and median complexes in Figure 3.12b.

When one begins learning AI concepts, they are often taught that models have a single, static, learning rate to teach students the basic concept. This learning rate tells the model how large of a step to take on the fitness landscape to increase proximity to the global optimum. The issue with implementing a static learning rate is the possibility that the model may have this value set too high and could pass by the optimum. In this scenario, the model would not consider this a success and would likely continue stepping back and forth, never settling. Conversely, having too small of a learning rate means the model may take a very long time to learn or may never learn what it needs to due to encountering a local minimum. To address this issue, the DAPTEV Model model implements learning rate *cosine annealing* [54, 63]. This way, the learning rate can start large, causing the model to make bigger leaps at the beginning, and then the model can reduce the learning rate to discern more nuanced details thereafter [11, 25]. Note that the transition from the starting learning rate to the end learning rate occurs in the first generation for a given run. All subsequent generations in that run will have their beginning learning rate set to the same value as the ending learning rate.

2.1.2 Variational Auto-Encoder

A VAE is a type of probabilistic DGM that receives data, learns how to represent (encode) the important features of the data in a latent space, then attempts to recreate (decode) the latent data [45]. For typical VAEs, the goal is to perfectly recreate the input. This way, one can provide new random data to the VAE and be confident that the output will be something that seems plausible within the problem space. For example, one could train a VAE on images of handwritten digits between 0 and 10. Then provide a random, normally distributed, sample to the VAE as input and receive a predicted digit output that looks as if it was hand-written, but that exact image likely would not exist in the training set. It is for this generative capability that a VAE was used in this research. Figure 2.4 illustrates this process.

If both the encoding and decoding models of a VAE are a type of RNN, then sequence information can be provided to the VAE in a manner similar to NLP systems. These are known as sequence-to-sequence (Seq2Seq) models. Considering RNAs are in fact sequences, this makes VAEs an ideal tool for learning and producing sequences. The RNNs implemented for this DAPTEV Model's encoder and decoder are known as a GRU and are an evolution of an LSTM model.

The RNN encoding and decoding models of a VAE cannot learn possible sequence information for anything without a starting vocabulary. If the VAE was being applied

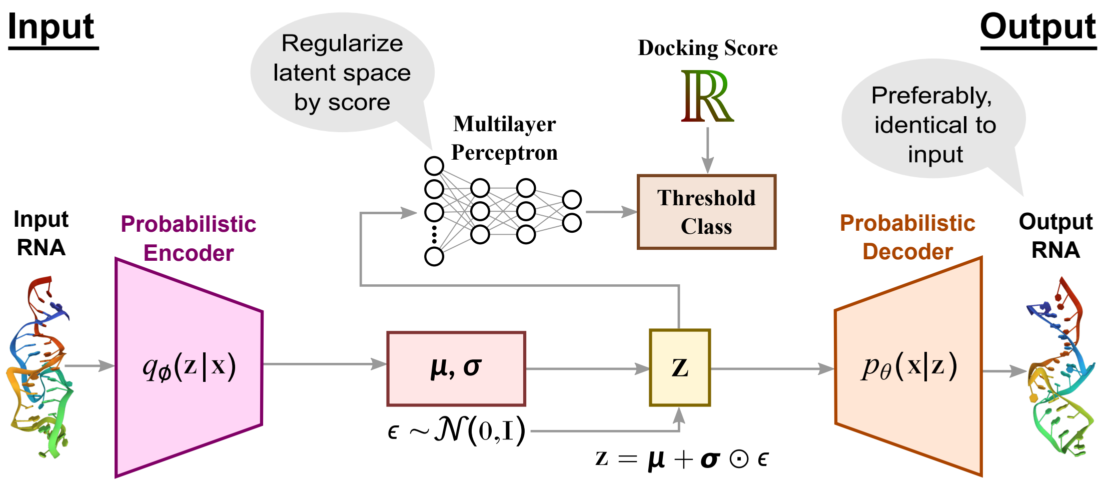


Figure 2.4: Mathematical and graphical representation of the implemented VAE.

to an NLP application, the known vocabulary might be the English alphabet or, for more advanced learning, it could be all the words contained in the entire dictionary. In this application, the vocabulary would represent the characters required to build RNA sequences ("a" for adenine, "c" for cytosine, "g" for guanine, and "u" for uracil). However, this alone does not provide the VAE with enough context for how an RNA structure will interact with itself and the possible structural motifs that can occur. Thus, it is a good idea to include all the possible combinations of these four base characters with respect to the maximum sequence length. For example *A, C, G, U, AA, AC, AG, AU, CA, CC, ...*. Conversely, adding too many combinations could drastically increase the computation time as each increase in combination length (1 character, 2 characters, 3 characters, ...) essentially equates to 4^x possible combinations. Here, the 4 represents the four base characters and the x represents the maximum character combination length.

In the context of this DAPTEV Model, the data starts as strings of RNA base characters (see "The Data" for more on this). Then, the VAE will detect all unique individual characters from the provided data. For example, "a", "c", "g", "u". From these base characters, the VAE will build all possible combinations up to the user-specified vocabulary length limit. Note that the combinations implemented in this model excludes duplicates produced from palindromes. Once the vocabulary has been built, three more values are inserted. These values are "<*BOS*>" for *beginning of sequence*, "<*EOS*>" for *end of sequence*, and "<*PAD*>" for *padding*. When the VAE encounters a <*BOS*> value, it knows that the sequence has started. When the

VAE produces a $<BOS>$ value, the DAPTEV Model will know to consider what follows as a new sequence and will filter out the $<BOS>$ before proceeding to the scoring procedure. $<EOS>$ is treated similarly, except this value marks the end of the sequence. While the $<PAD>$ value will be filtered in the same manner as the $<BOS>$ and $<EOS>$ values, it is used slightly differently. When working with sequences, one inevitably must choose how to deal with sequences of variable length. One method is to restrict all sequences to the same length. However, doing so limits the versatility of the model and its learning capabilities. Instead, this DAPTEV Model implements a $<PAD>$ value to be inserted into each sequence as they are presented to the VAE. As this DAPTEV Model uses a “maximum sequence length” parameter, the amount of padding added to any sequence will simply be equal to this *max length parameter minus the current sequence length.* If this parameter did not exist, the amount of padding added to a sequence would depend on the largest sequence provided to the VAE. When the VAE encounters a $<PAD>$ value, it knows that this is not important and will essentially ignore it.

Once the vocabulary is built, all produced combinations will be saved as “embeddings”. Embeddings are essentially just ways of numerically representing combinations and their associations with other combinations. Each combination in the vocabulary list is assigned an index. Then, a square, two-dimensional look-up table is made with the indices along both the top row and the side column. The values stored at each intersection represents how strongly related the pair of combinations are according to the learning of the VAE [57, 65]. Thus, these embedding values are constantly updating as the VAE learns.

Next, the training sequence data, now in the form of embeddings, is passed through the GRU encoding model which is responsible for learning important features of the provided data. Learned features are unique to the given problem or task. For the DAPTEV Model, “features” can represent nucleic acid base pair interactions, structural motifs like stem loops and bulges, and other observable characteristics of RNAs. All output (important features) from the encoder is then compressed and saved in a latent space.

The latent space is a collection of latent vectors (z) which represent a probabilistic distribution of the learned features. These vectors are considered samples from the distribution and are characterized by the mean (μ) and standard deviation (σ) for each z, as seen in Figure 2.4. The VAE also uses log variance when performing calculations, but this can be determined from the σ value [45, 64]. To obtain the μ and σ vectors, the output (hidden state) from the GRU is passed through two separate linear layers (see Pytorch documentation for more details [63]). This process of *passing data through linear layers* is the aforementioned "compression" process, meaning the encoder determines important features, then represents these features as the statistical parameters μ and σ. It is important to mention that this DAPTEV Model assumes the latent space is a Gaussian distribution with a mean of 0 and a variance of 1.

The μ and log variance vectors are then used to calculate the Kullback-Leibler divergence (KL) loss. This loss function measures the difference between the current distribution of the latent space and the standard normal distribution. The goal is to shift the current distribution to be as close to the standard normal distribution as possible. In doing so, the KL loss function acts as a regularizer, encouraging the posterior closer to the prior [13, 25, 45, 47].

Following the latent space contributions from the encoder, the decoder will then sample from this space and attempt to recreate the input data. First, the sample is passed through a linear layer to scale the latent data. Then the scaled data is sent to the decoder GRU which attempts to recreate the sequence. This recreation is then sent through another linear layer which outputs the probabilities for each vocabulary value.

In an ideal world, the input data would be perfectly reconstructed. To measure this, a reconstruction loss is computed [45]. While the reconstruction loss was developed in relation to the KL divergence, representing the negative log-likelihood, in implementation it can be represented by a cross entropy function. This is because the cross entropy function also calculates the difference between two distributions. As such, the cross entropy function is implemented in the DAPTEV Model.

If left to its own devices, the KL loss function can tend to vanish during training. This can negatively affect the total loss function and, thus, also affect the reconstruction learning process. Additionally, the raw KL value can greatly influence the decoder's learning at the beginning of training. To address these issues, the DAPTEV Model implements a technique known as KL annealing [22]. The annealing process applies a small weight, between 0 and 1, to the KL loss value and linearly increases this weight during training.

For typical VAEs, the total loss function would simply be the addition of the KL and the reconstruction loss values. This is because VAEs are usually concerned with recreating data as its sole purpose. Thus, an encoder and decoder model is generally sufficient.

Up to this point, the VAE is not necessarily learning what makes an RNA sequence good or bad with respect to the Rosetta docking score. Technically, as long as the VAE is provided with only well-performing sequences, it would likely recreate similar-performing sequences. This, however, is not “learning”. This is preferential omission and qualifies more as a heuristic, which is not the aim of this research. The goal is to supply all data, good and bad, to the DL system and to have the model determine the qualities of “good RNA” for itself. The VAE should then be able to produce sequences with strong docking capabilities to a specified target. It is for this reason that an MLP model was included in the architecture of the VAE. This MLP exists between the VAE’s encoder and decoder, training at the same time. However, the MLP is not learning how to reproduce sequences. Instead, it is learning the associations between sequences and their docking classifications as determined by the score threshold. The associated MLP loss value is also added to the total loss function.

After the encoder has seen the sequence data and produced a latent vector, this vector and its associated score will be given to an MLP. The MLP is responsible for performing score-based classification and to further regularize the model with respect to the sequence scores. It was found during model testing that performing mean-squared error (MSE) and root-mean-squared error (RMSE)-based regression on the raw scores performed poorly, which negatively affected the overall performance of the encoder and decoder. The same was true when the scores were normalized based on the length of the sequence (number of nts). There was even an attempt to institute a custom function which penalized sequences for having too many or too few connections. The formula for this penalty function was a simple parabolic function: $y = n(0.8(x-2.5)^2 + 0.001)$ where y is the new “score” that has been normalized and penalized, n is the original score normalized by sequence length, and x is the number of nt pairings in the secondary structure. These normalized and penalty-normalized scores also yielded poor performance by the regression model. Ultimately, via the suggestion of Dr. Li, the method arrived at was to classify the original score as “a good docking score” ($\leq$ score threshold parameter) or “a bad docking score” ($>$ score threshold parameter). This converted the scores to 1 (good) or 0 (bad) and allowed to the MLP to perform classification with a binary cross entropy (BCE) loss function rather than regression. This is the third and final value added to the total loss function for the VAE.

It should be noted that one cannot simply set the score threshold parameter to any low value desired. DL systems require a certain amount of data from which to learn. If the starting dataset has too few instances of sequence scores equal to or below the chosen score threshold, the DL system will not be able to learn properly and will overgeneralize all sequence scores to be “bad”. To determine the optimal score threshold, it is recommended that the user perform some preprocessing dataset analytics. In doing so, the user can determine the best score threshold that contains at least 25% of the data in the dataset. For example, upon analyzing the starting dataset in this research, it was found that 26.28% of the data fell within the range of 3,500 and below. Thus, a score threshold of 3,500 was chosen.

See Figure 2.5 for a depiction of the implemented VAE’s process. Note that the evolutionary operations performed after the MLP occurs later as shown in Figure 2.1. These operations are shown here simply to illustrate that the evolutionary operations are applied to the latent vector, not the sequences.

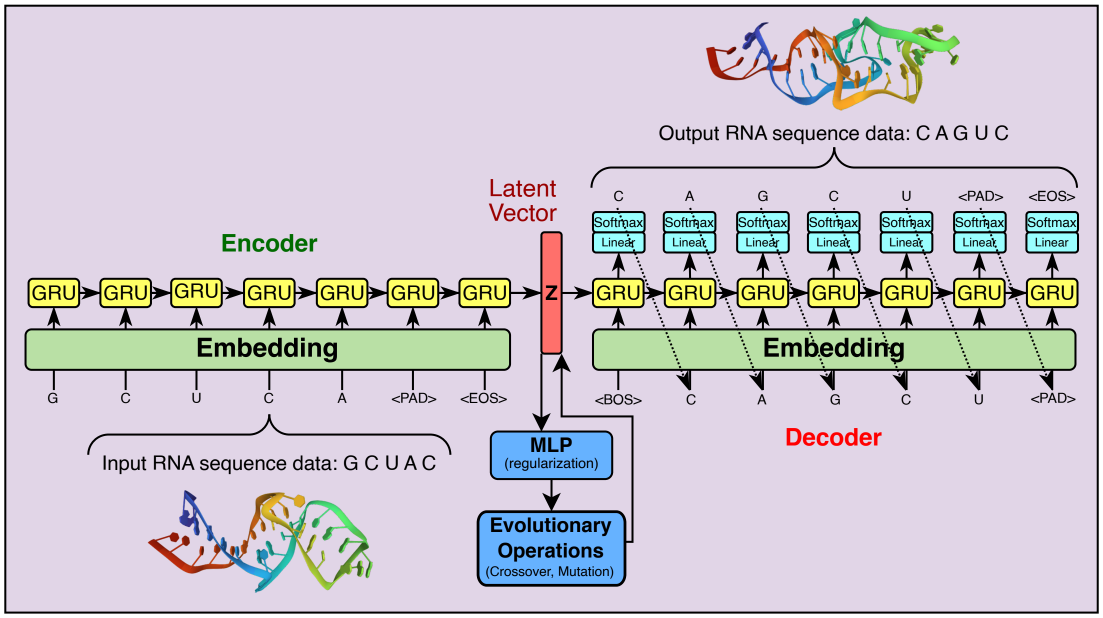

Output RNA sequence data: C A G U C
Latent Vector
Encoder
GRU
Embedding
G
C
U
C
A
<PAD>
<EOS>
Input RNA sequence data: G C U A C
z
MLP (regularization)
Evolutionary Operations (Crossover, Mutation)
Softmax
Linear
<BOS>
Decoder

Figure 2.5: Depiction of the implemented VAE’s process.

See tables 2.1, and 2.2 for more information on the model structure. See subsections “Machine Learning Parameters” and “DAPTEV Model Parameters” for chosen model parameter. For a summary of the VAE algorithms, see the pseudocode 1 below.

Table 2.1: The chosen parameters for the VAE structure. Note: for vocabulary and embedding size, as well as other VAE parameters, see Table 3.6.

	Input Size	**Hidden Size**	**Num. Layers**	**Dropout**	**Z Size**	**Bidirectional**
Encoder	embedding size	256	1	0	10	True
Decoder	embedding size + Z size	512	3	0	10	False

Algorithm 1 DAPTEV Model's VAE algorithm

```
1: Build vocab
2: for each Run do
3:     for each Generation do
4:         for each Epoch do
5:             for each Batch do
6:                 for Encoder do
7:                     convert sequences to embedding values
8:                     convert scores to 0 or 1 based on score threshold
9:                     pass embedding values to encoder GRU
10:                    get hidden state $h_e$
11:                    pass $h_e$ to linear layers
12:                    get $\mu$ and $\sigma$
13:                    use $\mu$ and $\sigma$ to calculate $logvar$ and $z$
14:                    use $\mu$ and logvar to get KL Loss
15:                for MLP do
16:                    pass $z$ into MLP to get output
17:                    pass output to sigmoid function to classify as 0 or 1
18:                    get BCE Loss via known score comparison
19:                for Decoder do
20:                    pad embedded sequences to equal lengths
21:                    pass $z$ into decoder linear layer to scale up features
22:                    pass scaled $z$ and padded sequences to decoder
23:                    pad output
24:                    send output to fully connected layer
25:                    get prediction for each element in vocabulary
26:                    compute cross entropy loss to get reconstruction loss
27:                compute total loss function
28:                update model via backprop
```

Table 2.2: The VAE's implemented model structures. Note (1): LL stands for "Linear Layer". Note(2): the output for the GRU says "N/A" because it does not have an "output" parameter. See Pytorch documentation for more details [63].

	Input	**Hidden**	**Layers**	**Dropout**	**Bidir.**	**Output**
Encoder Structure						
GRU	Encoder Input	Encoder Hidden	Encoder Layers	Dropout	Encoder Bidir.	N/A
Mu LL	2x Encoder Hidden Size	-	-	-	-	Z Size
Sigma LL	2x Encoder Hidden Size	-	-	-	-	Z Size
MLP Structure						
LL 1	Z Size	-	-	-	-	64
LL 2	64	-	-	-	-	32
LL 3	32	-	-	-	-	1
Decoder Structure						
Decoder GRU	Decoder Input	Decoder Hidden	Decoder Layers	Dropout	Decoder Bidir.	N/A
Latent LL	Z Size	-	-	-	-	2x Z Size
Predict LL	2x Z Size	-	-	-	-	Vocab Size

Lastly, a point must be made about the efficacy of this DAPTEV Model. The results from the DAPTEV Model are obtained via the last generation's training output. In fact, every data point seen by the VAE is solely training data (predicted sequences eventually become training data). This means the VAE sees this data multiple times depending on the *number of epochs* parameter. This can affect the VAE output such that the VAE may produce duplicates due to learning too well how to reconstruct the input data. These duplicates are then replaced as explained in "The Data". Additionally, as the output from the DAPTEV Model is obtained from the last set of sequences in the final generation, the scores will be slightly higher than they could be. This is because the VAE trains on the initial dataset for many more epochs than the future predicted sequences. As this DAPTEV Model performs optimization, this means that the dataset likely contains many poor-performing sequences at the beginning of training which could skew the final output. Note that the GA does not suffer from this issue as much due to the nature of simply carrying forward strong sequence attributes and forgetting poor attributes. See "Evolutionary Operations" and "Genetic Algorithm" for more on genetic operations and GAs. The initial dataset can still heavily influence the GA, but the VAE is likely more affected by this due to the patterns and features the VAE learned earlier at the beginning of training. The choice to do this, however, was one made out of necessity. The VAE requires input to return a prediction. However, to select strong candidates via tournament selection, one must know the sequence's score. This would be unknown if a random normally distributed sample was provided. Thus there would be no way to know if the prediction yielded from the input was based on a well-performing RNA or not.

2.1.3 Evolutionary Operations

Three important notes must be mentioned before proceeding. First, for the remainder of this and the "Genetic Algorithm" subsection, when discussing the topics of *genetics* and *evolution*, it is in reference to algorithmic/AI representations and not the actual scientific fields of study. The second note here is the concept of *genetic diversity.* In "Machine Learning Parameters" it is stated that too small of a population size will result in a low amount of genetic diversity, quickly converging to a local minimum. This should be avoided by increasing the population size parameter. In the context of this research, this means that there will not be enough sequences in a given generation to produce sufficiently dissimilar children from the parent sequences and other produced children. Eventually, all produced sequences could start to resemble each other and/or the ML algorithm will be unable to find better sequences for the given task due to the lack of *genetic diversity* among each sequence. *Genetic diversity*, with regard to EC algorithms like GAs, refers to the difference between the data (sequences). It is this inability to discover better options (sequences) for the given task that is referred to as *convergence.* Ideally, the goal is to prolong the search before convergence to more fully explore the problem/solution space.

Each generation ensures the DAPTEV Model's VAE has finished a round of training. Once a round has finished, the DAPTEV Model will begin the sequence optimization step. This is an important step as this is when the DAPTEV Model will search for better sequences and update its running data accordingly. This step occurs every generation and involves three Darwinian evolutionary operations.

The first operation is known as *tournament selection with elitism.* Elitism, as implemented in this DAPTEV Model, is the action of selecting the top x best individuals (sequences) in a population to be carried forward into the next generation. Here, x is a percentage of the population as specified by the *elitism* parameter, with at least one individual being the lowest possible value. This is done to ensure the best individuals remain in the running data and are not filtered during tournament selection. Tournament selection is the process of randomly sampling sequences from the dataset and having these candidates compete in a tournament. The winner of each tournament will be selected as a *parent* to produce the next generation of *children.* This process is repeated multiple times until the parent pool equals the chosen *population size* parameter. Note that the same individual can be selected more than once.

The tournament selection process has a couple parameters. One such parameter is the k value (see "Machine Learning Parameters") which indicates how many contenders with which to build a tournament round. The higher the k value, the more likely a stronger performing individual will be selected to win the tournament and proceed as a parent for the next generation. This, in turn, means there is a higher likelihood that poor-performing individuals will not be selected to proceed as parents. This is known as *selection pressure* and may be detrimental as sometimes things must worsen before they can become better, as discussed in "Fitness Landscape". As a visual, imagine a model, starting at the top of a mountain, that wants to find the lowest valley. On its way down, it encounters a small hill and does not want to climb up because that would be increasing in elevation, not decreasing. However, just past that hill is the lowest possible valley. The "poor performing individual" can be thought of as that hill and should be allowed in the next generation. Figure 2.3 illustrates this well, starting at the global maximum (mountain) and moving to the global minimum (valley), but a local maximum (hill) is in the way.

Another tournament selection parameter is the *selection rate.* This parameter creates a possibility for the best-performing individual in a tournament pool to opt out of the tournament, allowing some poor-performing individuals to be selected instead. This exists to further reduce selection pressure.

The second evolutionary operation is called *crossover.* This is the process of producing children from two parents. The crossover operator attempts to simulate a child obtaining features from their parents. After the selection operation is performed, the sequences are passed through the VAE's encoder model to obtain their latent representations (z). Then, the crossover operation is applied to these parent z vectors whereby parent z_pa and parent z_pb will each copy random parts of their latent vector to new children vectors z_ca and z_cb. This procedure is repeated until the number of children equals the population size parameter. This operation is performed to carry forward the best-performing aspects of the previous generation, while still exploring the fitness landscape. Similar to tournament selection, this operation also has a probability of not occurring (called crossover rate). For this DAPTEV Model, if a parent fails to perform crossover, it will simply be cloned into the next generation. Thus, it is best if the value for this rate is kept small.

The last evolutionary operation is *mutation*. Mutation, in terms of AI, is an attempt to mimic random mutations in evolution. In practice, this is used as a way to shift individuals to a random spot on the fitness landscape. If this operator was not present in the algorithm, the only exploration taking place would be along the same trajectory as the starting data. In other words, the starting dataset can be considered a starting position on the fitness landscape. If one only performed crossover, the search would essentially be moving in one direction along that landscape. This is still exploring, but it is possible that a better solution exists elsewhere entirely. Thus, some random influence should be allowed such that an individual may find themselves performing significantly better than the rest. For the DAPTEV Model, mutation is implemented by selecting a random individual, then selecting a random index of that individual's z vector and replacing it with a random, normally distributed, value. The parameter controlling this operator is the mutation *rate*. This controls the frequency/chance of mutation occurring. It is best if the value for this rate is very small. Too high of a mutation rate, and the algorithm will essentially be performing just a random search and could be corrupting any learning.

Once the new latent vectors have completed a round of evolutionary operations, they are sent through the VAE's decoder. This will reconstruct the sequences based on the VAE's previously trained decoding capabilities. The output will be a list of new sequences. Any duplicates in this list, as compared to itself, the starting dataset, and any previously predicted sequences will be removed and replaced with new, folded, random sequences. However, docking scores for these new sequences are unknown. Thus, the docking simulation must be repeated before the VAE can continue training.

2.1.4 Genetic Algorithm

At its core, a GA is an optimization algorithm that attempts to model evolution. In comparison to the DAPTEV Model, a typical GA is not able to learn key features. Instead, it starts with the sequences only and searches more randomly via crossover and mutation while identifying and prioritizing stronger generations based on docking score alone. Thus, while GAs are often utilized for global optimization tasks, in this scenario, a GA will tend to find and struggle with local minimums associated to RNA folds more easily. It is possible that further consideration of sequence folding and its relation to the docking score could be included in the GA's fitness function. However, (a) this would create another issue of determining the optimal method of numerically assessing how sequence folding affects the docking score, which is an entirely different field of research unto its own; and (b) this would be changing too many variables between the DAPTEV Model and a GA which would affect the comparability between these models. The goal was to simply remove the VAE from the DAPTEV Model, resulting in a GA operating on the problem space rather than the latent space, to see how their performance differs. To see the process for the GA comparison model and how it is similar to the DAPTEV Model process, see 2.2.

The GA comparison model behaves fairly similar to the DAPTEV Model. The main difference is that there is no VAE and the evolutionary operations are performed on the problem space (the sequences themselves) rather than the latent space. As such, the crossover and mutation steps have slightly different implementations. For the GA crossover function, parent sequences a and b be first be padded. Then, instead of selecting random indices of parent z_pa and parent z_pb (latent vectors), parents a and b will have random indices of the sequence space selected for crossover. The resulting children will be new sequences with padding values. This, however, introduces the issue of potentially selecting the $<BOS>$, $<PAD>$, and $<EOS>$ characters from the parent sequences and shuffling their locations in the children sequences. Should it occur that the $<BOS>$ and $<EOS>$ characters are not respectively at the beginning and end of the child sequence, the GA will simply filter these out. This is possible as the length of sequence is already known by the size of the data structure holding the sequence. Removing $<BOS>$, $<PAD>$, and $<EOS>$ will leave only the remaining sequence information. Thus, having shuffled $<BOS>$, $<PAD>$, and $<EOS>$ characters does not significantly affect the GA.

For the mutation function, rather than selecting a random index, generating a random normally distributed number, and replacing the value at that index with the random number, as implemented by the VAE. The GA, instead, selects a random index and a random value from the vocabulary list that is not $<BOS>$, $<PAD>$, or $<EOS>$. The GA then replaces the value at the random location with the random vocabulary selection. This introduces another issue. Now, there is a chance that the $<BOS>$ or $<EOS>$ characters will be overwritten. This too is accounted for in the GA as these characters are eventually removed anyway.

It is worth noting that the VAE and the GA are not restricted to producing only folded structures. As such, these models do have a possibility of generating unfolded sequences. This was allowed to more deeply explore the solution landscape and potentially encounter stronger sequences through the ML prediction process.

2.2 The Data

To understand how the parameters affect the DAPTEV Model, the data preparation, format, amalgamation, and considerations must be discussed. The first step is to obtain a target from the PDB website [10]. This DAPTEV Model uses the SARS-CoV-2 spike glycoprotein (closed state) [81] as the docking target with an RBD between residues 333 and 524 [48, 67, 70, 76, 81, 86]. See Figures 2.6 and 2.7 for an illustration.

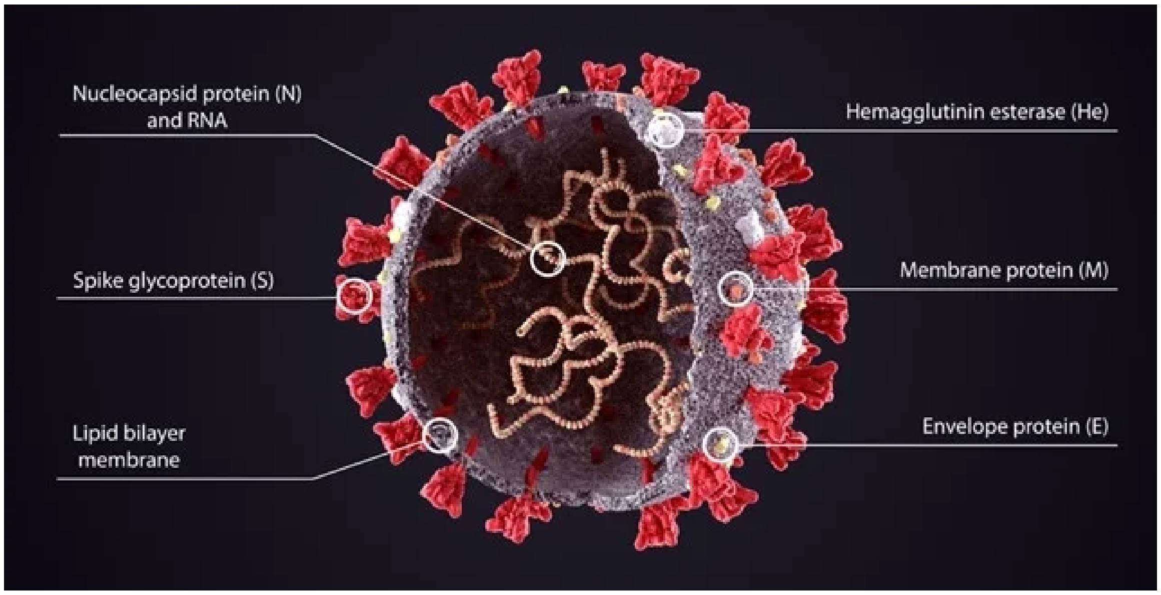


Figure 2.6: Realistic three-dimensional illustration of SARS-CoV-2 virus structure diagram [46].

2.2.1 Data Preparation

Once the PDB file and the respective FASTA file (sequence information) is downloaded, the user may experience some issues utilizing the raw data as input into Rosetta. Many FASTA files on the PDB website contain sequences which can be considered "inconsistent" to their PDB file counterparts. This is because the sequence downloaded from the PDB website likely includes residues which do not appear in the PDB file's atom coordinate area, leaving gaps in the PDB file's residue numbering system. This typically occurs because these residues could not be found at the time of protein sequencing. These residues in the FASTA file are essentially place-holders

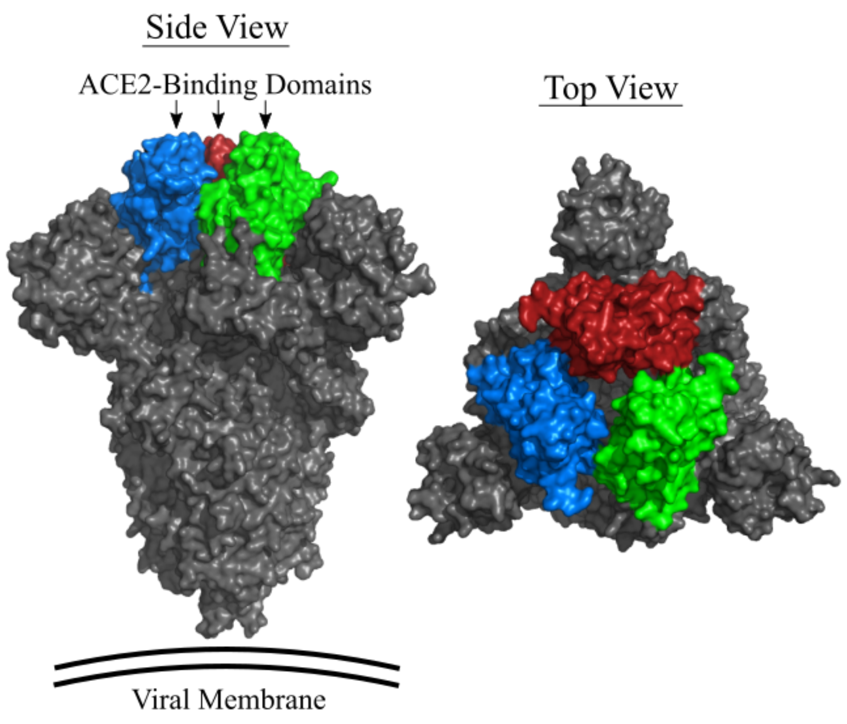


Figure 2.7: Visualization for SARS-CoV-2 Spike protein (closed) RBD.

used during the protein analysis phase to represent that some residue is present but which one specifically is unknown. The presence of these residues in the FASTA sequence can cause Rosetta to return errors during the tertiary structure prediction and docking process. Additionally, the downloaded FASTA file may only contain the sequence for one chain of the protein, as a protein can be made up of multiple repeating chains. This is an issue for Rosetta as the software expects to receive each chain's sequence that makes up the entire protein. Furthermore, in the case of the SARS-CoV-2 spike protein, as Figure 2.7 shows, the RBD is a relatively small area on top of the protein. If the entire spike protein is used for the docking procedure, this will drastically increase the docking simulation computation time as the protein is rather large. For example, performing five runs of the docking process with an RNA nt count of 25 on the full SARS-CoV-2 spike protein took over 98 minutes (5,891 seconds) to finish computing. This is the case even if a constraint file is present to restrict the docking region. See "Constraint Score Function" and "Implementation and Considerations" for more information on constraints and their implementation in this model. As the DAPTEV Model defaults to five runs of the docking procedure per RNA, and repeats the docking procedure according to the user-specified population size, number of generation, and number of runs (see "Parameters"), it is entirely possible for the model to take upwards of 4 years to complete if each docking process is performed sequentially.

To address these issues, some modifications to the raw data likely needs to take place prior to running the DAPTEV Model. First, the user will have to remove all unnecessary residues from the target PDB file and save the new PDB structure and the new FASTA sequence. Unnecessary residues are target residues that an RNA could not possibly interact with during the RBD-docking process due to constraint specification. Removing these residues will reduce the size of the target and the PDB file which will significantly accelerate Rosetta's computation time. For the implementation of this model, all but the spike protein's RBD and some neighbouring residues were removed, reducing the file to 30% of its original size. The new PDB and FASTA files were then saved for use. This was done via a molecular visualization software called PyMOL [69]. Running the docking simulation again after this change, with the same RNA of 25 nts, only took a little more than 6 minutes (385 seconds) to complete. A significant improvement over the previous 98 minutes. However, removing residues will likely create larger gaps in the PDB file residue numbering system and could change a considerable amount of the sequence. Additionally, with these edits, the PDB file may now have chains of differing sizes rather than equal-length chains of repeated residues.

This leads to the second modification. The new PDB file must be "cleaned" via a secondary Rosetta script called "clean_pdb.py". This script will renumber the existing residues sequentially, remove any extraneous information in the PDB file, and will write out a new PDB and FASTA file. Doing so will also change the protein's RBD residue range. This means, for the SARS-CoV-2 spike protein, the RBD of 333 to 524 established in recent scientific literature [48, 67, 70, 76, 81, 86] will not be the same anymore. Also, while this new PDB file is necessary for the remaining process, the user can discard the FASTA file written through the "clean_pdb.py" script as this FASTA file will likely contain an incorrect sequence. Instead, the user should use the sequence that was previously saved via PyMol. Now, the user can open the PDB file in a text editor and note the new ranges for each chain. These new ranges should be specified in the FASTA file header. Then, the user should open the new PDB file in their molecule visualization software and note the new RBD residue numbers. This will be important later when specifying constraint parameters (see "Score Functions" for more details). Performing these actions should prevent Rosetta from returning errors or taking an inordinate amount of time to complete. A summary of these steps can be seen below in the subsection "Modification Summary".

2.2.2 Data Format and Amalgamation

Following the modification of the raw PDB and FASTA files, the user should then provide starting RNA aptamer input data. Input data begins in one or more CSV files which contain RNA sequences and associated secondary structure information. These should be known and confirmed aptamers. The more known aptamer data provided, the better it is for the system to learn and explore these options.

RNA sequences are stored as strings of RNA base characters known as nt where “a” equates to adenine, “c” is cytosine, “g” is guanine, and “u” is uracil. An example of an RNA sequence is *5’ - cggcaacgaguuaaccucg - 3’*. The secondary structure information is represented as a string in dot-bracket notation. The secondary structure for the provided RNA sequence example is *......((((.....))))*.

2.2.3 Additional Considerations

While an entire dataset ***can*** be created using the random sequence generator, as described in “Random Sequence Generation”, it is best to augment the dataset with known aptamers. Providing existing aptamer data will add diversity into the dataset, will expose the model to physical aptamer characteristics found in nature (or post-development), and will help the model learn these traits. This will also allow the DAPTEV Model to rule out or confirm existing aptamers as potential solutions to the given problem as it is possible that a known aptamer is already a good fit for the specified target. As a result, supplying existing aptamer data could provide the ML system with a strong starting position before it begins exploring other options.

Any data provided by the user will be subjected to the minimum and maximum sequence lengths that the user specified during parameter entry (see “Dataset Preparation Parameters”). Thus, the user could provide, for example, 1000 aptamer data points, but not actually have the full 1000 data points captured in the output from the dataset preparation script. The result will be a subset of the loaded data. The GC percentage parameter will not affect entered data as this parameter is only used when generating new random sequences.

It is important to mention that this DAPTEV Model does not allow unfolded RNA secondary structures to be created when performing random sequence generation. Associated secondary structures will be restricted to having at least one set of brackets (base pair connections). At any point during the dataset creation process or the DAPTEV Model’s duplicate replacement process, if the generated secondary structure does not have at least one connection (base pairing), this sequence will be discarded and a new sequence and secondary structure will be generated. Sequences will also be discarded each time the VAE produces new sequence predictions, or during the sequence duplication removal process, if the sequences already exist within the starting dataset or in the previously predicted data. This discarding and generating process will be repeated until unique sequences are discovered.

In the case of unfolded RNA structures (structures represented entirely as dots in dot-bracket notation), this was decided during the literature review stage as very few aptamers with unfolded secondary structures were found, indicating that these structures are not optimal, are rarely developed, or are scarcely found in nature. Additionally, it has been observed during the experimental phase that unfolded secondary structures produce RNA tertiary structures that are more malleable when docking to the target's RBD. Thus, unfolded sequences can better form to the target's RBD conditions, producing strong binding scores (binding affinity). However, aptamers are designed not only for high binding affinity but also for target specificity [9, 15, 42, 43, 44, 85, 87, 88]. An unfolded structure would not be considered "specific" to a target because it could potentially bind just as well to another target. This is likely why unfolded aptamers were not found. Furthermore, the Rosetta scoring function for the docking process includes the quality of the RNA tertiary structure prediction. This means that the returned score for any sequence in this model may be deceptively "improved" due to encountering few stability-related penalties during the structure prediction process. As a result, the scores produced by unfolded RNA structures are artificially, and significantly, better than their more structurally-complex peers. For more details on this behaviour, see "Native Rosetta Score Function". If unfolded structures are introduced into the model from the starting dataset or during the duplication replacement process, it increases the chance of the algorithm becoming stuck in a local minimum (see "Fitness Landscape" for more explanation on fitness landscapes and local minima). This can result in the algorithm producing many unfolded RNA aptamers, which would not be ideal for aptamer drug development as the goal is to optimize the aptamer development process for a specific target only (specificity). For this reason, any data the user enters should be scrubbed of sequences with unfolded secondary structures to avoid introducing these instances into the algorithm.

With regard to the discarding of repeat sequence data, the intent was to only present unique data to the VAE to prevent overfitting. The DAPTEV Model fine-tunes itself (performs additional training steps) on the newly produced sequences each generation (see Figure 2.1 for illustration). Should the VAE train further on data that it has already seen, this could lead to the VAE associating greater importance to the features of these sequences over all other sequences. This would essentially equate to an unfair advantage for the repeat data and a biased output from the DAPTEV Model.

2.2.4 Modification Summary

Steps to modify the raw PDB file and the sequence obtained from the PDB website:

1. Open the original PDB file in a molecular visualization software.
   - PyMOL [69] was used throughout the experiments.
2. Delete any unnecessary residues that are not pertinent to the RBD.
   - This can be accomplished by manual selection and deletion in PyMOL.
3. Save the new pdb file
   - In PyMOL, this can be achieved through the context menu.
4. Save the new sequence as a FASTA file.
   - In PyMOL, this can be achieved through the command "save name.fasta".
   - Note: this FASTA file will contain the entire sequence, but the sequence will be broken down into multiple lines. The user must remove the new-line characters, concatenating the entire sequence and placing it on one line.
5. Update the sequence in the fasta.txt file (the file Rosetta reads from) with the output FASTA file from the previous step.
6. Renumber the new pdb file with the Rosetta "clean_pdb.py" script by typing `python $ROSETTA_TOOLS/protein_tools/scripts/clean_pdb.py 6vxx.pdb ignorechain` into the terminal with the current working directory set to the same location as the pdb file.
   - Python 2 is required for this process as Rosetta was written in Python 2. If one has a different python version installed, the "python" call at the beginning of this command should be "python2"
   - 6vxx.pdb is the SARS-CoV-2 spike protein. Substitute with the target's PDB file name.
   - "Ignorechain" tells the script to ignore any specific chain and simply number each residue sequentially.
7. Update the chain ranges in the fasta.txt file with newly produced residue ranges in the PDB file.
8. Make sure the "flag" file (see "Parameters") points to the fasta.txt file and the Rosetta script-produced renumbered PDB file.

2.3 Random Sequence Generation

Following the data loading operation, the remainder of the dataset (dataset size parameter minus selected elements from the loaded data) will be built using a custom random RNA sequences generator and a thermodynamic-based secondary structure prediction system (see "RNA Secondary Structure Prediction"). Random RNA sequences will be generated according to the minimum and maximum sequence length and the GC percentage parameters. The GC percentage is required as it has been discovered that there is a strong correlation between the amount GC content and the stability of the RNA structure [23]. By default, the minimum length for RNA sequences is set to 20 characters (nt), the maximum length is set to 40 characters, and the GC percentage is set to 50%. If the user does not have any starting aptamer data to load, an entire dataset can be created according to the user's entered parameters. However, it is best not to do this as discussed in "The Data". This random sequence generator is also used after each round of predictions provided by the VAE during the sequence duplicate replacement process. This too is discussed in "The Data".

2.4 RNA Secondary Structure Prediction

During the primary sequence generation, the DAPTEV Model will perform secondary structure predictions and will associate these structures with the respective RNA sequences. This is achieved utilizing EternaFold, an accurate thermodynamic-based RNA secondary structure prediction software, interfaced by a python package called Arnie [83]. The outputs from the Arnie package are RNA secondary structures in dot-bracket notation.

The secondary structure information must be present before proceeding as Rosetta requires it to perform tertiary structure prediction (see section "Tertiary Structure & Docking Prediction"). For example, the following RNA sequence will yield the associated secondary structure when passed through Arnie. The two-dimensional structure is shown in Figure 2.8.

- 5' - cgcugucuguacuuguaucaguacacugacgagucccuaaaggacgaaacagcg - 3'
- ((((((.((((((......)))))).......((((.....))))...))))))

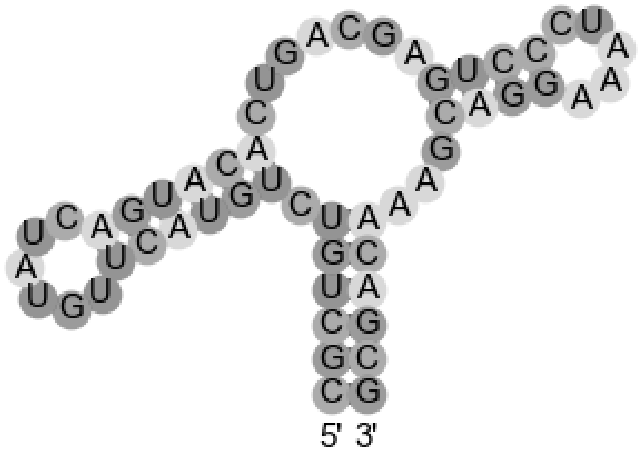


Figure 2.8: Visualization for the Arnie-returned secondary structure output for the provided RNA sequence.

2.5 Tertiary Structure & Docking Prediction

Once the initial dataset of primary sequences and secondary structures has been loaded or created, the DAPTEV Model sends this information to Rosetta [41] to begin the tertiary structure prediction and docking simulation. Both the tertiary structure prediction and docking simulation are performed per sequence before moving on to the next sequence. Rosetta does this to indicate how well the RNA conformed to its tertiary structure ***and*** how well that structure docked to the target. The quality of the structure prediction and the docking simulation is expressed through Rosetta's score function (see "Score Functions" for more on this). During this process, the produced score is associated with the corresponding RNA sequence and utilized as a fitness function (a measure of performance) in the ML process. This process is affected by the Rosetta-specific parameters as discussed in "Parameters". A visual of the Rosetta-predicted tertiary structure for the sequence provided in "RNA Secondary Structure Prediction" can be seen in Figure 2.9. Notice that the pentagonal structures (the nucleic bases) between the RNA backbone folds appear to connect with their complementary bases on the opposing side of the RNA strand or point outwards in the same pattern as the secondary structure in section "RNA Secondary Structure Prediction". For a depiction of the same RNA when docked to the PP7 coat protein dimer [14], see Figure 2.10.

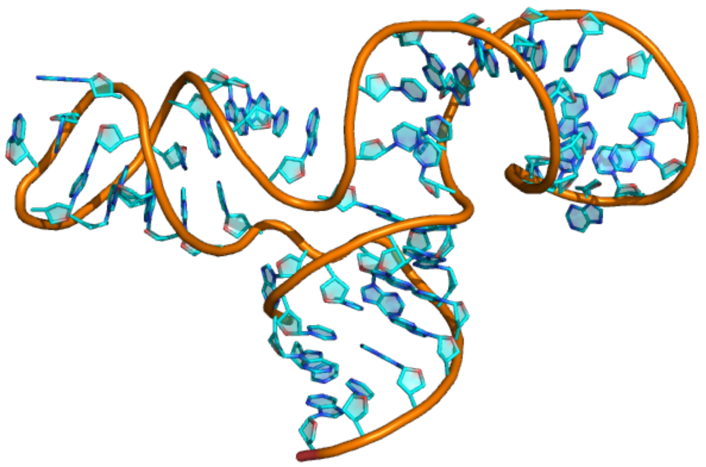


Figure 2.9: Visualization for the Rosetta-predicted tertiary structure of the example RNA sequence in section “RNA Secondary Structure Prediction”.

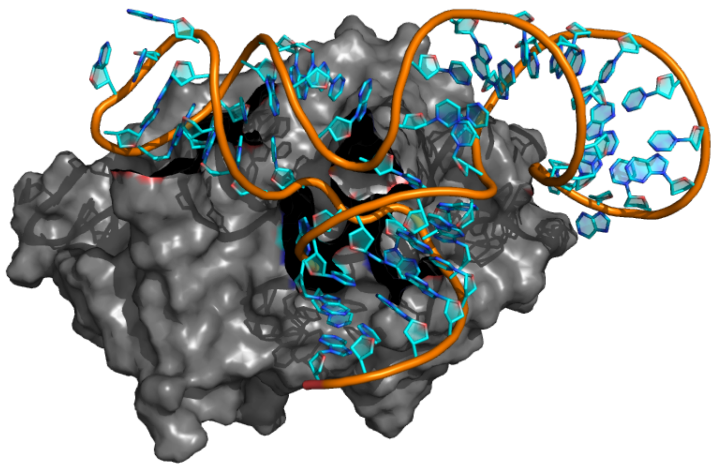


Figure 2.10: Visualization for docked complex of the example RNA sequence in section “RNA Secondary Structure Prediction” and the PP7 coat protein dimer as predicted by Rosetta.

It is worth noting that Rosetta does allow the user to provide their own full, or partial, RNA tertiary structures. This capability exists for those whom already know the RNA tertiary information and want to enforce this full, or partial, RNA tertiary structure during the docking process. Should the user wish to do so, they must also point Rosetta to the RNA tertiary structure PDB file prior to docking by specifying the file path and file name in the “flag” file. Providing a partial RNA PDB file will tell Rosetta to predict any unspecified conformations. Rosetta even allows the user to predict just the RNA tertiary structure as a separate process prior to performing docking. Doing so provides the user with the ability to visually inspect the resulting tertiary structure predictions and to choose one for the docking process. However, the implementation of this DAPTEV Model does not provide access to these functionalities, favouring the combined structure prediction and docking simulation process instead.

The decision to utilize the combined prediction and simulation process was made for a few reasons. Firstly, it was noticed that predicting just the RNA tertiary structure took almost as long as the structure prediction and docking process. Secondly, providing a pre-built RNA tertiary structure yielded no improvement in the docking simulation speed. In other words, providing an RNA PDB file to the docking simulation took the same amount of time to compute as the combined structure prediction and docking simulation process. This suggests that it is either a trivial matter to predict the tertiary structure during the combined structure prediction and docking process or that the combined process is optimized, in some capacity, for computation time. Regardless, performing tertiary structure prediction first, then performing the docking process, rather than just performing the combined process, essentially doubled the computation time. Moreover, performing the structure prediction process requires the writing and reading of multiple RNA PDB files, and an additional step of either manual (visual) inspection of the produced RNA structures or the utilizing of some computational method to determine "the best" RNA to proceed with for docking. Note that utilizing a "computational method" is irrelevant as the prediction and simulation process already yields a single, combined, structure and docking score. Lastly, the starting aptamers dataset did not contain any tertiary structures, requiring tertiary structure predictions for every data point in the dataset. Thus, there was no need to preemptively predict the aptamer tertiary structures.

2.6 Score Functions

There are two scoring functions within this DAPTEV Model's usage of Rosetta. The first is the native Rosetta docking score function. The second is a constraint scoring function. For the Rosetta scoring function, the lower the returned score is after the tertiary structure prediction and docking simulation, the more stable the predicted complex is considered to be for the given protein, RNA, and their combined conformation. The constraint score affects incurred penalties the RNA will experience during the docking process and is used to specify the target's RBD.

2.6.1 Native Rosetta Score Function

The native Rosetta scoring function is a low-resolution, coarse-grained, knowledge-based (statistical) RNA-protein potential. This serves as an energy function for scoring Monte Carlo steps within the tertiary structure prediction and docking simulation. In a recent study published by Dr. Kappel and Dr. Das, it was stated that they included all previously published score terms describing RNA structure and RNA-protein interactions in this scoring function while also providing rapid computation and maintaining coarse-granularity. It was also found that, over ten popular scoring systems, the best-performing scoring models achieved an average atomic root-mean-squared deviation (RMSD) of 11.6 Å for 3dRPC and 10.2 Å for DARS-RNP, whereas Rosetta's score function achieved an RMSD of 6.4 Å [41].

Rosetta's docking score function includes the prediction and scoring of the provided RNA's tertiary structure. This means two elements are being represented in one produced score. It is, therefore, possible to have a low score for the RNA tertiary structure prediction due to the RNA's lack of self-interaction, reducing penalties associated with stability, with an average or high score for the predicted combined complex, producing an artificially well-performing complex overall (when only considering the score). Furthermore, it is also possible that the predicted RNA structure could return unfolded, as a result of an unfolded secondary structure, before proceeding to the docking simulation. In this instance, an unfolded RNA would experience few penalties during the docking process (see following paragraphs from more on "penalties") as the RNA is less restricted in its tertiary structure and can better adapt to the target's RBD conditions. This too would produce an artificially low score and could outperform other predictions upon quick inspection or during algorithmic comparisons.

2.6.2 Constraint Score Function

The constraint scoring function applies to where on the protein the RNA docks. This constraint function allows the user to specify the target's RBD region without having to force a ***fixed*** binary interaction between specific atoms of the RNA and the target protein. Instead, the user indicates to Rosetta their chosen constraint type, to which atoms on the target and the RNA the constraint applies, and what built-in formula the constraint will use to calculate an energetic penalty. This penalty will be applied against the returned Rosetta score to passively discourage the RNA from docking elsewhere on the target. As a result, the RNA has a range of acceptable distances it can deviate from the target's RBD during the docking simulation. If the RNA atom leaves the specified region, Rosetta will apply the scaling penalty.

To illustrate this concept, let us use the example of choosing an atom pair constraint with the flat harmonic function. The Rosetta documentation describes an "atom pair" constraint as constraining the distance between a specific RNA atom and the target's atom. While the atoms are technically "constrained" to each other with this option, it is the choice of function and function parameters that dictates whether these two atoms will have a fixed or variable distance between them. The formula of the harmonic function is shown in equation 2.1, where x represents the distance between the two atoms (this varies as Rosetta attempts multiple conformations), $x0$ is the ideal distance between the atoms, sd is the acceptable standard deviation from the ideal distance, and $f(x)$ is the returned penalty. The flat harmonic function is an extension of the harmonic function which produces a penalty of 0 between the ranges of $x0 - tol$ to $x0 + tol$, where tol means tolerance.

$$f(x) = \left(\frac{x - x0}{sd}\right)^2 \tag{2.1}$$

An example input for the constraint file is as follows:

AtomPair CA 201A C5 838 FLAT_HARMONIC 0 1 1

Here, "AtomPair" specifies the type of constraint, indicating that a specific atom of the target and the RNA are bound to each other in some capacity. This tells Rosetta what geometric property to measure. "CA" is the chosen atom on the target to be bound to the RNA atom. "201A" is the atom residue number for the target in chain "A" (a location within the target's RBD). This number can be found in the pdb file next to the chain letter or it can be obtained from the FASTA file based on the index (column number) of the desired residue. "C5" is the chosen atom on the RNA to be bound to the target atom. 838 is the nucleotide number for the RNA according to its expected position (index) in the FASTA file. "Flat_Harmonic" tells Rosetta what function to use when calculating the penalty. "0" ($x0$) is the ideal distance between these atoms measured in Å. The first "1" (sd) affects the magnitude or severity of the returned penalty. The severity of the returned penalty increases the closer the sd approaches 0. The second "1" (tol) is the acceptable range (+/-) that the atom is allowed to move from $x0$ before incurring a penalty. For the flat harmonic formula, see 2.2. For a plotted visual of this example penalty function, see Figure 2.11. To see how this constraint function affects the docking process, an example applied to the SARS-CoV-2 RBD is provided in Figure 2.12.

$$f(x) = \begin{cases} 0 & \text{for} \quad x0 - tol \leq x \leq x0 + tol \\ \left(\dfrac{x - x0}{sd}\right)^2 & \text{Otherwise} \end{cases} \tag{2.2}$$

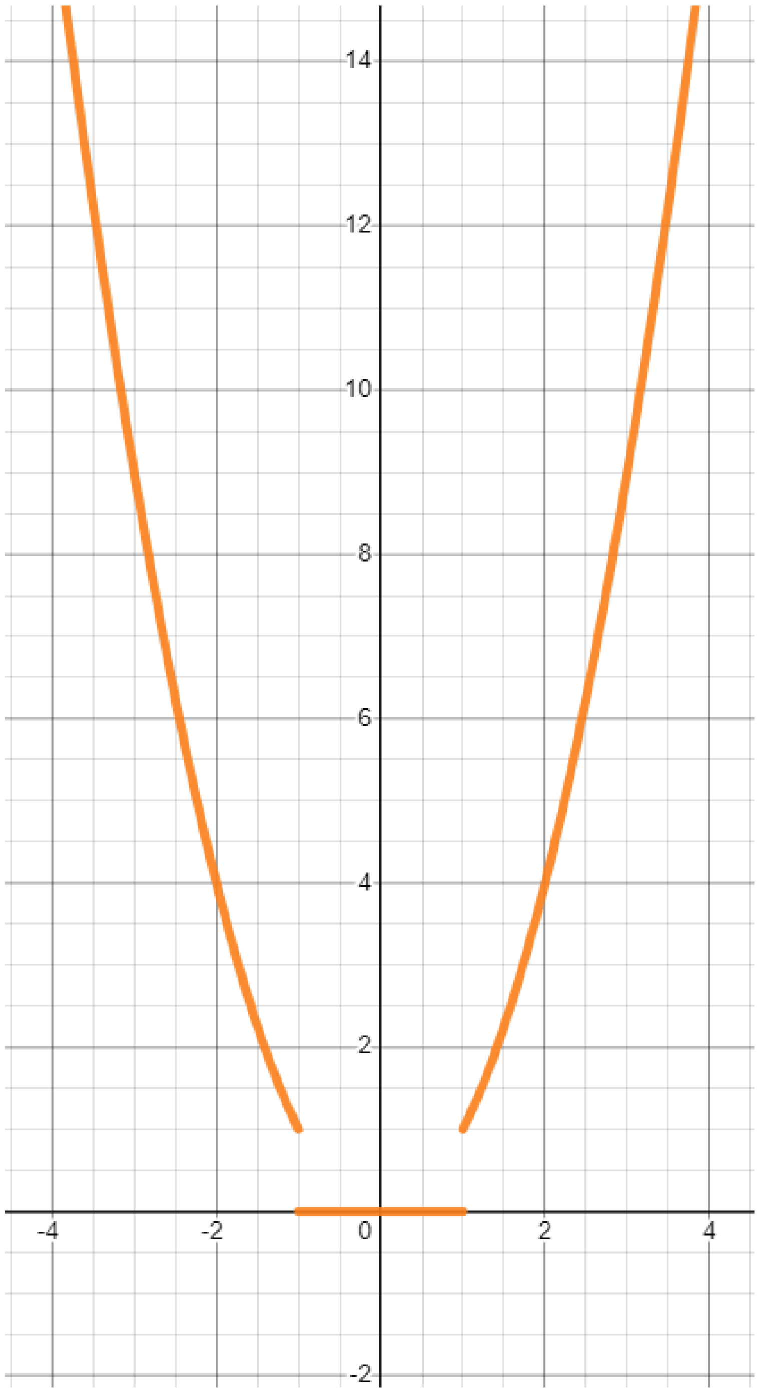


Figure 2.11: The Rosetta flat harmonic constraint penalty function for inputs $x0 = 0$, $sd = 1$, $tol = 1$.

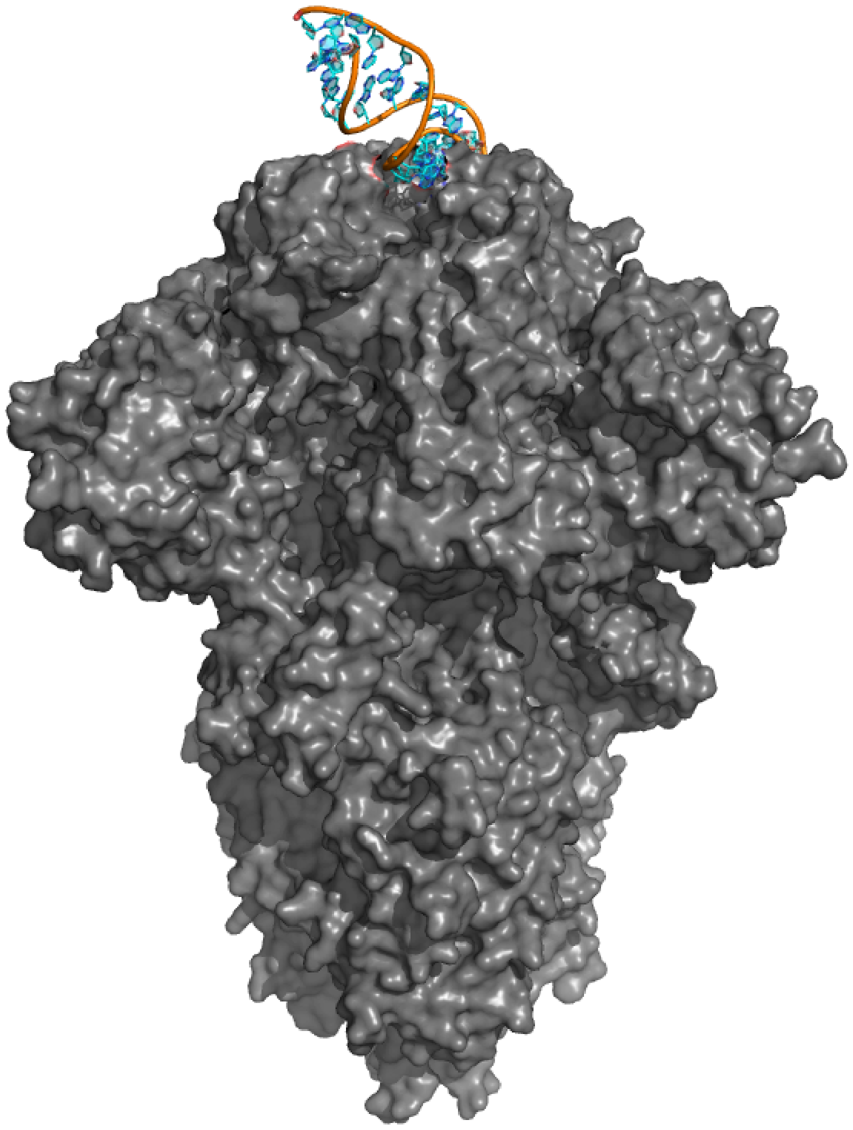


Figure 2.12: Depiction of the SARS-CoV-2 spike protein with an RNA molecule of sequence 5' - ggcacagaagauauggcuucgugcc - 3' and secondary structure (((((.((((......))))))))) docked at the RBD site.

For more information on the constraint file, constraint types, and penalty formula options, see `https://www.rosettacommons.org/docs/latest/rosetta_basics/file_types/constraint-file`. For a tutorial on constraint files implementations and interpretations, see `https://new.rosettacommons.org/demos/latest/tutorials/Constraints_Tutorial/Constraints`. The options for this score function are controlled in the constraint file (see "Parameters").

2.6.3 Implementation and Considerations

While the implementation of the native Rosetta docking score function was relatively straightforward, the implementation for the constraint function required some creativity and problem-solving regarding the RNA-specific parameters. The first issue was that the constraint function requires the specification of a certain nucleotide for theRNA (838 in the example above) but the DAPTEV Model generates RNAs of variable lengths during iteration. Even if tertiary structure prediction was performed ahead of time, there is no obvious way to know what would be the best nucleotide to choose for the constraint. Also, incorporating an additional tertiary structure prediction would drastically increase computation time as this does not speed up the docking simulation, but doubles the tertiary structure prediction process. The second issue is the atom choice for the chosen nucleotide (C5 in the example above). If the DAPTEV Model creates random RNA sequences of variable length, then it will also select a random base every time for the constraint parameter.

To address the first issue of selecting a specific nucleotide, the middle position (nucleotide) of every produced RNA sequence was utilized. An argument could be made that it would have been better to select the first or last quarter of the sequence as, once the RNA folds at the middle following tertiary structure prediction, a quarter of the sequence length would likely equate to somewhere in the middle of the new structure. However, an RNA folding perfectly in the middle of the sequence may not always occur as this largely depends on the location of the connections in the secondary structure. Additionally, it is possible to have an RNA with most of the connections towards one end, say near the 3' end, leaving a long unfolded tail. Selecting the nucleotide located a quarter of the length away from the 5' end may not be ideal at that point. With this in mind, choosing the middle of the sequence seemed acceptable. Especially as the constraint does not necessarily equal physical atom interaction and only forces general proximity. The other atoms and nucleotides are still free to interact and bind in the given range.

The second issue with regard to selecting a specific atom without knowing what nucleotide would be chosen by the DAPTEV Model. This is an issue because nucleic bases are comprised of many atoms, but not necessarily all the same ones. For example, the atom H1 does not appear in the base uracil, but it does appear in guanine. Initially, storing a unique atom for each base separately and selecting these atoms according to their respective base was considered. Then it was noticed that the atom C5 is ubiquitous among nucleic acid bases [55]. Thus, this research simply chooses to always use the C5 atom.

2.7 Multiprocessing

In “The Data”, it is mentioned that the docking procedure could take a long time if performed sequentially. Notice that the computation time went from 5,891 seconds per sequence to 385 seconds due to reducing the target PDB file. Assuming the same user-entered parameters for population size (800), number of generations (10), and number of runs (3) for one experiment, this reduces the total computation time from 4 years to roughly one-third of a year. However, one-third of a year is still far too long to wait for results from just one experiment. Thus, an additional strategy was implemented into the DAPTEV Model system. This strategy is known as multiprocessing.

Processes (separate programs or sets of instructions) in a computer utilize the computer’s central processing unit (CPU). Computers lately have multiple CPUs. The operating system is tasked with distributing and managing these processes among all available CPUs for that machine. Multiprocessing is a technique employed to run multiple processes, such as docking simulations, on the machine’s many CPUs at the same time (in parallel). For this DAPTEV Model, the percentage of available CPUs utilized to perform docking was 50%, or 18 CPUs. This means 18 docking commands were running at the same time.

This strategy increases the computation time per sequence from roughly 6.5 minutes to approximately 18 minutes due to the overhead involved with creating and managing multiple processes. However, the ability to run these commands in parallel effectively reduces the computation time down to roughly 1 minute per sequence because all 18 docking commands finish around the same time. Thus, 18 commands that finish in roughly 18 minutes total is approximately 1 minute per sequence (if viewed sequentially). Again, for the same parameter choices, this technique reduces the total computation time for the experiment from approximately one-third of a year down to roughly 16.5 days, or 5.5 days for one run of 10 generations at 800 docking simulations.

Ideally, it would be better to distribute these docking processes over a cluster of computers rather than just one computer. Then, one could perform multiprocessing on each computer, *drastically* accelerating computation time. For example, if one had 45 computers in a cluster, each with 18 available CPUs, one could finish one generation of the DAPTEV Model in just 18 minutes. This would make one experiment with starting parameters of 800 for the population size, 10 for the number of generations, and 3 for the number of runs, take only 9 hours to complete. This processing time could be further reduced if more of the machine's CPUs are utilized. However, this research could only be performed on one machine and it was a shared environment. Thus, only 50% of the available CPUs on the machine was utilized so others could use the remaining resources.

There are two important notes here for multiprocessing. The first is that there is no point in creating more processes than the available CPUs amount. While the operating system will manage these tasks and schedule them, one will not see a significant decrease in processing time because each process must either share the CPUs, or queued processes will have to wait until the previous ones finish. Secondly, it is best to leave at least one cpu available so the operating system has one to operate on or use for other purposes.

2.8 Development and Acknowledgments

This system was built by connecting the input and output from multiple tools. First, a random RNA sequence generator script was required to build an initial dataset and to replace any duplicate sequences produced during the DL training process. This was a custom tool created at the beginning of this research.

Next, a secondary structure prediction program was required. This was obtained from Das Lab's Arnie python package [83]. Luckily, this package could be imported into python directly. The output from the random sequence generator was fed into Arnie to obtain secondary structures.

Then, tertiary structure prediction and docking simulations were required. Both of these requirements were satisfied by the "RNP_denovo" capabilities of Rosetta [41]. The primary and secondary data was received and given to Rosetta to compute the tertiary structure and docking scores. However, while there is a python implementation of Rosetta, this implementation did not yet include the "RNP_denovo" script at the time of developing this DAPTEV Model. Thus, the subprocess module in python was required to call Rosetta from the terminal. This required some file-editing to be performed prior to running the "RNP_denovo" script as explained in "Rosetta Parameters". The reason for this is that Rosetta needs the FASTA, secondary structure, constraint, and flag files to contain the information on which Rosetta will operate prior to running the program. Unfortunately, this posed another issue. To accelerate computation time, these terminal calls had to be performed in parallel (multiprocessing), which required multiple instances of file editing. However, overwriting a file that Rosetta is using with new data will corrupt Rosetta's computation and could yield errors. Thus, multiple temporary files were created and Rosetta was redirected to those scripts. Then, python was instructed to repeatedly iterate over each running process and communicate with the terminals to collect the output without blocking and waiting. This was required to keep each terminal active and prevent the subprocesses from entering into sleep mode. During this communication step, python filters the output, via regular expression, searching for lines that start with "total score" as this was the output docking affinity score for a given sequence.

Once Rosetta returned the docking scores, all starting and obtained data was given to a VAE to begin training. This algorithm was implemented in Pytorch. Then, evolutionary operations were performed on the training data and the VAE-yielded results. This process was implemented in python itself. Both the VAE and the EC were coded directly into the DAPTEV Model and did not require separate terminal calls. The steps performed by the VAE and EC were inspired by Grantham et al. and Patel [26, 64].

2.9 Parameters

There are three groups of parameters the user must set. The first set of parameters pertains to Rosetta. The second group of parameters affects the dataset preparation script. The third parameter group influences the overall ML process (the DAPTEV Model itself). The user may notice some overlap between the dataset preparation script parameters and the DAPTEV Model parameters. The user required the ability to set these parameters separately as this script may need to be executed prior to performing some other crucial steps before running the DAPTEV Model.

2.9.1 Rosetta Parameters

The user starts by specifying parameters relating to the operation of Rosetta. This is done via file editing before running the DAPTEV Model. The following are the editable files and their purposes:

- Flag file
    - This is a parameter file Rosetta uses when performing docking simulations. In this file, the user specifies the following:
        - The read location and name of the FASTA file.
        - The read location and name of the secondary structure file.
        - The read location and name of the constraint file.

    * The read location and name of the protein PDB file.
    * The write location and name of the Rosetta output file.
    * The number of docking prediction and produced complexes per RNA sequence.
    * The number of random docking attempts per sequence.
    * Additional customization options for the docking simulation.

- Fasta file
  - A .txt file describing the protein and RNA data.
  - This file has a header which includes the protein PDB file name, the chains and chain ranges found in the protein PDB file plus the RNA chain and range at the end, and a second line containing the protein sequence with the RNA sequence appended at the end. The protein sequence must match the protein PDB file.
  - However, the DAPTEV Model automatically fills in the RNA information. Thus, RNA information is omitted from this file at run time.
- Secondary structure file
  - A .txt file describing the secondary structure of the protein (technically meant to contain RNA data, but the DAPTEV Model fills in this information as it runs as well).
  - This file contains two lines of data. The first is the dot-bracket notation for the protein sequence (this will be entirely represented as dots only) with the RNA notation concatenated at the end, and the second line is the protein and RNA sequence (same as the FASTA file sequence).
- Constraint file
  - A .cst text file describing to where, approximately, on the target protein the RNA should dock. This is useful for enforcing RNA docking to the target's receptor-binding domain (RBD).
  - This file requires a pairing of the target protein and RNA atoms, the scoring function choice, and the values for the scoring function to use.
  - Note: users likely will not be able to simply enter the established RBD atom numbers into this constraint file. A calculation is usually required first. See the note in the following "protein PDB file" point and section "Score Functions" for more information.
- Protein PDB file
  - A .pdb file describing the atom coordinates for a particular protein.
  - This is usually obtained from the PDB website [10].

  - Note: it is best to keep the size of this protein as small as possible. Only the affected RBD and anything else that might be affected during the RNA docking process should remain. Otherwise, the docking prediction will take a very long time per RNA for each generation of the ML algorithm. After the target is reduced in size, a new protein sequence and atom numbering must be calculated, usually via a visualization program such as PyMOL [69]. It is for this reason that users cannot simply enter the established RBD for their target as the RBD atom number has likely changed at this point. Users can use the same visualization software to find the new RBD atom number.

- Rosetta output file
  - This parameter specifies where (if at all) to write the output file produced by Rosetta.
  - This output file contains necessary data for generating Protein-RNA complexes in PDB format.
  - By default, this parameter is set to "/dev/null/" for the DAPTEV Model, meaning no complexes will be saved to disk, to save disk space. However, the sequences and the Rosetta docking seed is still written so the user can obtain the docked complex information at a later time.

2.9.2 Dataset Preparation Parameters

Next, the user set the parameters for the dataset creation, filtering, scoring and formatting script. The parameters for this script are edited in the python file itself. The following are the dataset creation script parameters and their descriptions:

- The minimum and maximum lengths of the RNA sequences.
  - Newly produced sequences are subjected to a minimum and maximum length of characters to keep Rosetta score calculation times minimized.
  - These parameters specify the minimum and maximum lengths (inclusive) for any new sequences.
- The approximate percentage of GC for new sequences.
  - Prior to generating new sequences, the system must also know the approximate amount of GC content to build into the new RNAs. For more information on why this is required, see "Random Sequence Generation".
  - This parameter specifies the approximate percentage of GC content in newly produced random RNA sequences.
- Whether to allow unconnected (unfolded) secondary structures during the prediction process.

  - Allowing unfolded secondary structures into the VAE can cause the algorithm to prioritize this feature, producing an output of entirely unfolded RNAs. See “The Data” for an explanation of why this is not desirable. See “Deep Learning and Evolutionary Computation” for an explanation of how this phenomenon occurs.
  - This parameter is a **True** or **False** setting indicating whether RNAs are allowed to be unfolded (True) or not (False).

- Data size.
  - This parameter allows the user to specify how many data points they want the dataset to have.
  - This is useful if the user does not have enough starting aptamer data, had too many starting aptamers filtered out during the execution of this script, or if the user wants to create an entire dataset.
  - Any starting aptamers filtered out will be replaced with random RNA sequences and associated Arnie-predicted secondary structures.

- Percentage of data points classified as training.
  - This parameter allows the user to choose what percentage of the total data will be classified as a training set. The remainder will be classified as a testing set.
  - For most ML models, the user usually specifies how many data points and which ones are to be used for training, validation, and testing. However, the implementation of this DAPTEV Models does not use training and validation sets. The classification capability was built into this script should the user desire to utilize the same data in another model, but this parameter is set to “1.0” (100% training data) for the DAPTEV Model.

- The percentage of available CPUs to use.
  - The dataset creation script utilizes multiprocessing (parallel computing) to run many instances of Rosetta at once when scoring sequences. Ideally, this task should be distributed over a cluster of computers to disperse the workload and speed up the scoring process. However, access to a cluster was not available. Instead, the Rosetta scoring system was distributed amongst the available CPUs on a single computer.
  - This parameter allows the user to choose what percentage of available CPUs to engage when the script sends sequences to Rosetta for scoring.

- Random seed value
  - Random sequence generation and the Rosetta docking simulation both use random number generators.
  - This parameter allows the user to recreate the same results from a previous run of the script.

2.9.3 Machine Learning Parameters

Next, the user will specify additional parameters in a .csv file for the entire ML experiment prior to running the DAPTEV Model. These parameters and their descriptions are as follows:

1. The desired number of runs to perform.
    - Each "run" performs a specified number of generations (see below) for a particular experiment, but with a unique random state.
    - The purpose behind the "run" parameter is to run the same parameter set (experiment) repeatedly for statistical purposes.
2. The desired number of generations to perform.
    - A "generation" in this context is referring to the number of times the ML algorithm will carry forward previous predictions and produce new data for the same parameter set.
    - The purpose behind the "generations" parameter is to simulate generations in evolution. See "Deep Learning and Evolutionary Computation" for more information on this.
3. The desired population size.
    - While the user can upload any amount of data, this parameter specifies how many sample sequences the ML algorithm will return per generation and upon completion.
    - This parameter also specifies the number of sequences on which each subsequent generation will be operating.
    - Setting the population size too small will reduce the total diversity of each generation, resulting in a quick convergence of the data to a local minimum. See "Deep Learning and Evolutionary Computation" for more information on this.
    - Conversely, setting the population size too large drastically increases computation time.
4. Parameters specific to the VAE:
    - The epochs.
        - An epoch is a unit of iteration in which the ML model has seen the entire training dataset once. Usually, ML algorithms train for many epochs. This means the ML algorithm will see the same dataset multiple times in an attempt to learn patterns within the data.

  - This DAPTEV Model trains the VAE multiple times each generation. Thus, a parameter was required for the number of epochs in the first generation, called "pre-epochs", and another was required for all subsequent generations, called "post-epochs", thereafter.
- The batch size.
  - This parameter represents how much of the total data the VAE will see at one time. Where an epoch depicts the VAE seeing the total data one time, the batch size represents the proportion of total data the VAE will see in any given epoch.
- The score threshold amount.
  - This DAPTEV Model utilizes a BCE function when performing regression in the additional regularization model, converting all scores above the chosen threshold to 0, and setting all scores equal to or less than the chosen threshold to 1. The regularization model views scores of 1 as positive cases (the target), and scores of 0 as negative cases (results to avoid).
  - Thus, this parameter represents the score value the user deems as acceptable for the given task.
  - However, the user cannot simply choose to set this value to some desired score value. While there is a range of acceptable score threshold values, the chosen value largely depends on the initial dataset. More information on this can be found in "Deep Learning and Evolutionary Computation".
- The starting learning rate value.
  - Most ML models implement a "learning rate" in the algorithm. This learning rate is a numerical value the model will use to make increments on previous predictions in an attempt to optimize the model's prediction accuracy. It can be thought of as the model making *logical leaps* when trying to understand new concepts. However, static learning rates suffer from the possibility of stepping over the goal/proceeding past it. This is why learning rate annealing has been implemented in this DAPTEV Model model (see "Fitness Landscape" for more on this).
  - This parameter specifies the starting value for the learning rate annealing process.
- The ending learning rate value.
  - Similar to "starting learning rate", this parameter tells the VAE at what learning rate value to end.
- The starting KL weight value.
  - The loss function for the encoder model in the VAE is known as the KL [45, 47]. See "Deep Learning and Evolutionary Computation" for more information on this.

  - This parameter specifies how much impact the KL loss has on the total loss function (the formula that represents the VAE's ability to learn).
  - In a similar manner to the learning rate, the KL loss weight changes per epoch in the first generation and is a static value thereafter. However, unlike the learning rate, the KL weight starts small and grows linearly to a maximum amount.
- The ending KL weight value.
  - This parameter is the upper limit value for the KL loss weight.
  - After the KL weight reaches this point at the end of the first generation, all future generations will have their starting KL weight set to this value.
- The vocabulary combination amount.
  - The RNN encoding and decoding models of a VAE cannot learn possible combinations of anything without a starting vocabulary.
  - "Vocabulary combination" represents all possible combinations of the four RNA base characters with respect to the maximum sequence length. For example *A, C, G, U, AA, AC, AG, AU, CA, CC, etc.*
  - Adding too many combinations could drastically increase the computation time as each increase in combination length (1 character, 2 characters, 3 characters, etc.) essentially equates to 4^x possible combinations. Here, the 4 represents the four base characters and the x represents the maximum character combination length.
  - The vocabulary combination amount parameter is the user's selection for x, choosing the maximum length of the character combinations.

5. Evolutionary operation parameters:

- The tournament selection K amount.
  - After the VAE is trained in each generation, the DAPTEV Model then performs a "tournament selection" process in which K sequences from the training set are randomly chosen, then the sequence with the lowest score value among those chosen sequences is selected to proceed as a "parent".
  - This parameter indicates how many contestants are randomly chosen to compete as a parent for the next round of predictions.
- The selection probability value.
  - Immediately after the process of selecting K sequences for the tournament, each sequence still has a chance of being chosen, even if they are not the best individual. This is implemented to further constrain selection pressure, the tendency to ***only*** select the best-performing sequences. This is because a high selection pressure prevents the exploration of temporarily "poor options" in the chance this exploration

  may lead to a better outcome. In practice, this would be akin to an individual deciding they do not want to be a parent, despite being a strong candidate.
  - The probability of an individual being selected as the victor for each individual in the K contestants is calculated as $P * (1 - P)^i$ where the variable P represents the user-entered *selection probability value* parameter and the variable i is the position of the individual assuming the contestants have been sorted from best to worst and that the position count starts at 0.

- The elitism value.
  - This parameter represents the percentage of best-performing sequences in a given generation's population pool, with a minimum of 1 chosen, that automatically proceed into the parent pool for the next generation.
  - Technically speaking, with GAs, elitism is supposed to represent the individuals who are guaranteed a place in the next generation's population (current generation's produced children), while these same individuals can still act as parents based on random selection. However, due to the sort and trim behaviour of the DAPTEV Model, this happens automatically. It is preferable, though, to guarantee that the elite among the population are used when obtaining samples from the VAE. Thus, the implementation of elitism in this DAPTEV Model is slightly different.
- The crossover rate value.
  - After all parent sequences have been selected from the tournament process, they must then go through the process of making children. To do this, similar to how children inherit the genes of both parents in biological terms, the children sequences inherit characteristics from the parent sequences. This is done by selecting two parents at a time and building two new child sequences from parts of the parents in a process known as "crossover". However, there is a chance that the crossover process will not occur. In this event, the children are simply clones of their parents.
  - The *crossover rate value* parameter represents the probability that the crossover operation will be performed.
- The Mutation rate value.
  - After child sequences have been created, they are then subjected to a mutation process. The implementation of this process depends on the problem space it is being applied to. An example of mutation might be the replacing of some or all of the child's data with some random value(s) that can realistically occur given the situation.
  - Similar to the crossover operation, the mutation operation also has

a chance of not being performed. However, this chance of failure is usually much greater than crossover.

  - The *mutation rate value* parameter represents the probability that the mutation operation will be performed.

- For a deeper explanation of these evolutionary operation concepts and how they affect the ML system, see "Deep Learning and Evolutionary Computation".

6. The minimum and maximum lengths of the RNA sequences.

   - It is possible that the VAE can produce duplicates in its predictions as a result of learning too well how to reconstruct the latent space (see "Deep Learning and Evolutionary Computation" for more on this). When this occurs, the governing DAPTEV Model will remove these duplicates and replace them with new random sequences (see "The Data").
   - These newly produced sequences are subjected to a minimum and maximum length of characters to keep Rosetta score calculation times minimized.
   - These parameters specify the minimum and maximum lengths (inclusive) for the newly created replacement sequences.

7. The approximate percentage of GC for new sequences.

   - Prior to the replacement sequences being generated, the system must also know the approximate amount of GC content to build into the new RNAs. For more information on why this is required, see "Random Sequence Generation".
   - This parameter specifies the percentage of GC content in newly produced random RNA sequences.
   - This parameter **does not** affect the predicted RNAs from the VAE.

8. The percentage of available CPUs to use.

   - The current implementation of this DAPTEV Model utilizes multiprocessing (parallel computing) to run many instances of Rosetta at once when scoring sequences.
   - This parameter allows the user to choose what percentage of available CPUs to engage when the DAPTEV Model must send sequences to Rosetta for scoring.

9. Random seed value

   - There are many influences of randomness in this DAPTEV Model. This parameter allows the user to recreate the same results from a previous run.

- Note: the seed for the Rosetta docking process is always set to the same value, regardless of the user's chosen random seed. This was done to keep the Rosetta prediction variables the same while only changing the sequence information, and to ensure the user knows what seed to use when attempting to recreate the same Rosetta docking result. However, the user still has control over this seed value if they set it from the top of the "run_all.py" script (which controls and runs the entire DAPTEV Model model). It just will not change during the DAPTEV Model's iteration.

Lastly, the user must specify the starting dataset by supplying the DAPTEV Model with the path to the file (including the file name) within the code. See "The Data" for more information.

Chapter 3

Experiments

The experiment of the DAPTEV Model is accompanied by two additional comparison models. This thesis has outlined the DAPTEV Model thoroughly. It was also mentioned that a GA, which follows the same process as the DAPTEV Model but does not employ a VAE, is included as a benchmark. The process for the GA can be seen in Figure 2.2. The third benchmark experiment included is a simple random sequence "hill climber" algorithm.

The hill climber algorithm generates random sequences using the dataset creation script with the same chosen parameters as the DAPTEV Model. As such, this process is repeated for the same number of generations and runs as the DAPTEV Model, and produces the same number of sequences as the DAPTEV Model's "population size" parameter per generation. The best performing sequences are carried forward into the next generation and all other sequences up to the chosen population size are discarded. This is known as a "hill climber" because this algorithm employs the simple heuristic of "always take the best" without any actual "learning". At first glance, this may seem like an ideal algorithm to use, but it has a serious issue.

If one considers the problem search space discussed in "Deep Learning and Evolutionary Computation" and shown in Figure 2.3, one hill was taller than the rest. While score reduction is the optimization task of this research, for the sake of this explanation, let us consider this hill as *the goal*. A "hill climber" is an algorithm that will search until it encounters a hill, which could be the tallest (the global maximum), but likely is not as the terrain can be chaotic. The algorithm will then climb that hill (improve its solution to the given problem) and will stop at the top because any other direction would be a reduction in elevation/fitness. Hence the name "hill climber". However, as the goal is to optimize the docking scores, the opposite scenario of stagnating in a local minimum is more likely. If the valley is not the global minimum (the goal for score optimization) but instead a *local* minima, the algorithm will essentially stagnate. As this algorithm solely performs random searching, it will have to rely on chance alone to discover a better solution. Thus, it is expected that this algorithm will compare poorly in relation to the DAPTEV Model and the GA, but is included as a benchmark to determine if the DAPTEV Model performs better than a random search.

3.1 Chosen Model Parameters

The following subsections describe the parameters chosen for this DAPTEV Model. The rationale behind the parameter choices will be explained in the same subsections. Note that, when the user enters parameters related to percentage or rates, the values must be represented as decimals. For example, the GC percentage parameter of 50% should be entered into the DAPTEV Model as 0.05. Additionally, to see additional Rosetta file parameters set for every model, as described in section "Parameters" subsection "Rosetta Parameters", see appendix "Rosetta File Settings".

3.1.1 Constraint Function Parameters

Table 3.1: The constraint parameters for both models.

Constraint Type	Protein Atom	Protein Residue	RNA Atom	RNA Residue	Penalty Function
AtomPair	CA	201A	C5	###	FLAT_HARMONIC

Table 3.2: The flat harmonic function parameters for both models.

Ideal Distance ($x0$)	Std Dev (sd)	Tolerance (tol)
0	0.125	1

As seen in Table 3.1, the chosen constraint type implemented in these experiments is the "***AtomPair***". This was chosen due to its ease of understanding, usage, and inclusion in the code. The following parameters pertaining to the *AtomPair* are easy to discern or calculate. The ***CA*** atom was found in the protein PDB file next to the associated protein residue chosen within the RBD. The chosen RBD residue, after removing unnecessary residues and renumbering the PDB file, was residue "***201A***".

Rosetta allows the user to specify the residue for the protein in multiple ways. For the implementation in these experiments, the residue number followed by the associated chain of the protein is used. "***201***" is the residue number, and "***A***" means the residue resides on chain A. Figure 3.1 shows a visual of the chosen residue.

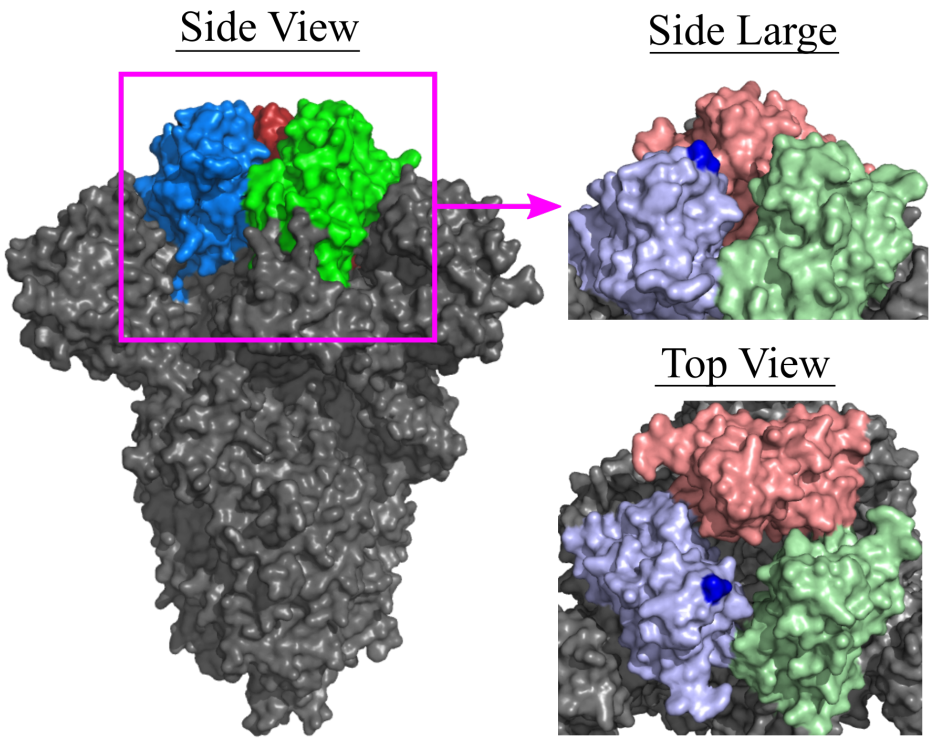


Figure 3.1: Location of the chosen residue (dark blue) on the renumbered SARS-CoV-2 Spike protein RBD as specified in the constraint function parameters. Note: the full spike protein shown (left) is for reference but the protein file utilized in the DAPTEV Model is cropped.

As previously mentioned in section "Score Functions" subsection "Implementation and Considerations", the atom "***C5***" can be found in every nucleic acid base [55]. This was the reason behind choosing "*C5*" as the RNA atom parameter. Doing so required no additional calculations to be performed and was simple to implement.

For each experiment, the parameters outlined in Tables 3.1 and 3.2 are static and do not change during iteration. The only parameter that changes during iteration is the RNA residue number, hence the "###" in Table 3.1 under "RNA Residue". This is due to the models generating new RNA sequences of variable lengths. As the middle of each RNA sequence is the chosen residue and this middle number must be added to the last residue number of the protein PDB file, this parameter will change frequently. Note that there can be fewer or more than 3 digits for this parameter. It all depends on the conditions of the protein sequence and the RNA residue chosen.

The "***Flat_Harmonic***" penalty function was chosen to accompany the *Atom-Pair* constraint type. An *AtomPair* constraint in combination with a *Flat_Harmonic* penalty function allows the RNA to attempt different conformations and orientations within an allowable range. Thus, the pair of atoms need not be physically interacting at all times.

Table 3.2 shows the parameters chosen for the "***Flat_Harmonic***" penalty function. A value of 0 Å was entered as the "ideal distance" between the target RBD residue and the calculated RNA residue. While this may seem as though the target residue and the RNA residue are touching, a standard deviation of 0.125 and a range of +/- 1 Å is allowed. This tight range was specified as it was observed during some empirical testing that many RNAs were docking either deep in-between the three RBD areas or underneath the edited protein.

Docking in-between the RBD areas seems implausible in real binding scenarios as an already-folded RNA would have to somehow fit its potentially larger structure through the small gap in the SARS-CoV-2 protein RBD conditions. Should the RNA dock in this area, the returned Rosetta score and penalty will be very good and could influence the ML model to prioritize this sort of unrealistic result. Thus, this should be discouraged during the docking procedure. Unfortunately, it is difficult to prevent this occurrence of unfolded structures, but it is also possible for an unfolded RNA to bind in this area during in vitro/in vitro applications. Therefore, this scenario was allowed in the model. See the side and top views of Figure 3.1 for a visual of the RBD "small gap", and the "worst" classified docking attempt in Figure 3.12c for an example of docking in-between the RBD area.

Docking underneath the edited protein is also an issue because the actual SARS-CoV-2 spike protein is larger than what was provided to the DAPTEV Model. The protein used in these experiments is just the SARS-CoV-2 RBD region some neighbouring residues. The remainder, including the lower portion of the spike protein, has been cropped out to accelerate computation time. Thus, the RNA would not be able to bind under this cropped area in real binding applications and should, therefore, not be allowed during the docking process.

The parameters in Table 3.2 were chosen for these reasons. To discourage the RNA from docking in-between the RBD area or docking under the cropped protein file. This tight range allows the RNA to still attempt different conformations and orientations while heavily penalizing folded RNAs if they attempt to dock too low on the protein or too deep in-between the protein RBD areas.

3.1.2 Dataset Preparation Script Parameters

Table 3.3: The parameters chosen to generate the initial dataset.

Min Seq Length	Max Seq Length	GC %	Allow Unfolded	Data Size	Train %	% of CPUs
20	40	50	False	12,000	100	50

Table 3.3 provides the parameters chosen during the dataset creation process. After obtaining 849 unique, known, aptamer sequences [10, 50, 51] (some of which had to be converted from DNA to RNA), only 390 met the condition of being between the minimum (20) and maximum (40) sequence lengths. Of those 390 aptamers, 344 contained at least one connection (base pairing) in their secondary structures. 44 sequences were too small, 417 sequences were too large and an additional 44 sequences were unfolded (based on the returned Arnie secondary structure predictions). As 12,000 data points were required, an additional 11,656 random sequences were generated and scored. These sequences were restricted to having lengths between 20 and 40 nt, an approximate GC percentage of 50%, and containing at least one secondary structure connection (affected by the "Allow Unfolded" parameter). This acted as the starting dataset. Every sequence in the dataset was classified as "training" data ("Train %" parameter) and the script used 50% of the computer's available CPUs to perform the docking procedures.

A dataset size of 12,000 was chosen to provide enough data for the DL system. For more information in this, see section "Deep Learning and Evolutionary Computation". A training classification amount of 100% was chosen to provide the ML model with as much unique training data as possible. Additionally, as the goal is to produce new sequences, there is no real way to compare an input test set against the VAE output. Thus, it did not make sense to have a testing set.

Sequence lengths between 20 to 40 nt were chosen for two reasons. The first reason is that aptamers are known as ***short*** single-stranded oligonucleotides [15, 42, 43, 44, 88], which typically means less than 100 nt in length. Secondly, as the docking molecule grows in length, the computation time increases non-linearly as well. However, too small of a sequence length may not produce large enough aptamers to successfully inhibit the SARS-CoV-2 spike protein from binding to the ACE2 cell receptors. The range of 20 to 40 nt seemed to be a comfortable middle ground.

The CPU percentage amount of 50 was chosen solely to allow for others using the computer at the same time. For the rationale behind the sequence GC percentage, see section “Random Sequence Generation”. For the rationale behind the disallowing of unfolded RNA secondary structures, see “Additional Considerations” and “Native Rosetta Score Function”.

3.1.3 DAPTEV Model Parameters

Table 3.4: The starting parameters for the DAPTEV Model.

Runs	Generations	Pop Size
3	10	800

Table 3.5: The parameters for the Rosetta-related aspects of the DAPTEV Model.

Min Seq Len	Max Seq Len	GC %	% of CPU	Allow Unfolded
20	40	50	50	False

Table 3.6: The parameters affecting only the VAE.

Pre Epochs	Post Epochs	Batch	Score Thresh	Start LR	End LR	Start KL Weight	End KL Weight	Vocab Size
45	10	32	3,500	0.003	0.0003	0.001	0.5	3

Table 3.7: The parameters affecting the DAPTEV Model’s evolutionary operations.

Tournament K	Selection Rate	Elitism %	Crossover Rate	Mutation Rate
2	95	1	90	1

Tables 3.4, 3.5, 3.6, and 3.7 show the parameter choices for these experiments as they relate to the DAPTEV Model’s starting parameters, Rosetta-related parameters, VAE-specific parameters, and the parameters affecting the evolutionary operations respectively. For the starting parameters, 3 runs, 10 generations, and a population size of 800 were chosen based on the remaining available time and to still produce a sufficiently comprehensive experiment. As section “Multiprocessing” explains, 1 run with these parameters takes the DAPTEV Model 5.6 days to compute. For 3 runs, this yields roughly 16.8 days. Then the comparison GA must also be run, taking another 16.8 days. This is over 1 month just to accumulate 3 runs of data per model.

The Rosetta-related parameters follow the same rationale as the dataset creation parameters for the most part. However, they are included here as well because the DAPTEV Model must also perform very similar actions as the dataset creation script does. This involves random sequence generation to remove duplicate sequences, secondary structure prediction of all newly produced sequences, and scoring new sequences per generation. As such, the same parameters as the dataset creation script were used.

All of the parameters chosen for the VAE were determined empirically. It was observed that training on the initial dataset for 30 epochs yielded an undertrained VAE which did not return adequate results. However, 60 epochs seemed to overtrain the system. 45 epochs yielded acceptable results. As the population size was only 800, having the VAE train on this data the same number of times could potentially also cause overfitting. Thus, it was determined that 10 "post-epochs" produced a satisfactory performance. 32 is a typical batch size for ML models, allowing models to train quickly. A score threshold of 3500 was obtained when upon the observation that too high of a threshold (5000) yielded under-performing, but many folded, RNAs and too low of a threshold (2000) produced well-scoring but many unfolded RNAs. The starting and ending learning rates and KL weights went through many iterations before finding a strong combination. The vocabulary size (maximum length of nt known to the VAE) was set to 3 to further reduce computation time. As vocabulary is essentially all combinations of the four nt bases, this essentially equates to 4^x where x is the size of the vocabulary. If this value was not limited, the number of calculations could quickly exceed the millions when considering the number of epochs, generations, and runs. For a discussion of the VAE performance and to view related plots, see section "Results".

With regard to the parameters related to evolutionary operations, a tournament K value of 2 was chosen to keep selection pressure low, allowing some poor and average-performing sequences to proceed as parents for the next generation. A selection rate of 0.95 was chosen to allow some of the better-performing sequences to opt out of the tournament, but not too many. An elitism amount of 1% guarantees the top performers of the entire generation will proceed into the next generation. However, more exploration over exploitation was preferred, so this value was kept relatively low. A crossover rate of 90% reflects the likelihood that a crossover will occur. This value was set relatively high (high exploration), but also allows for some reutilization of data (some exploitation). Lastly, the mutation rate was set to 1% to avoid random searching, but still have some element of an external random influence.

3.2 Results

The following subsections outline figures and tables that illustrate the performance for different aspects of the DAPTEV Model and comparison experiments. The first discussion point will explore the performance of the DAPTEV Model's VAE based on the parameters provided above. Following the performance of the VAE will be the overall performance of the experiments on the docking scores and the produced secondary structures.

3.2.1 VAE Performance

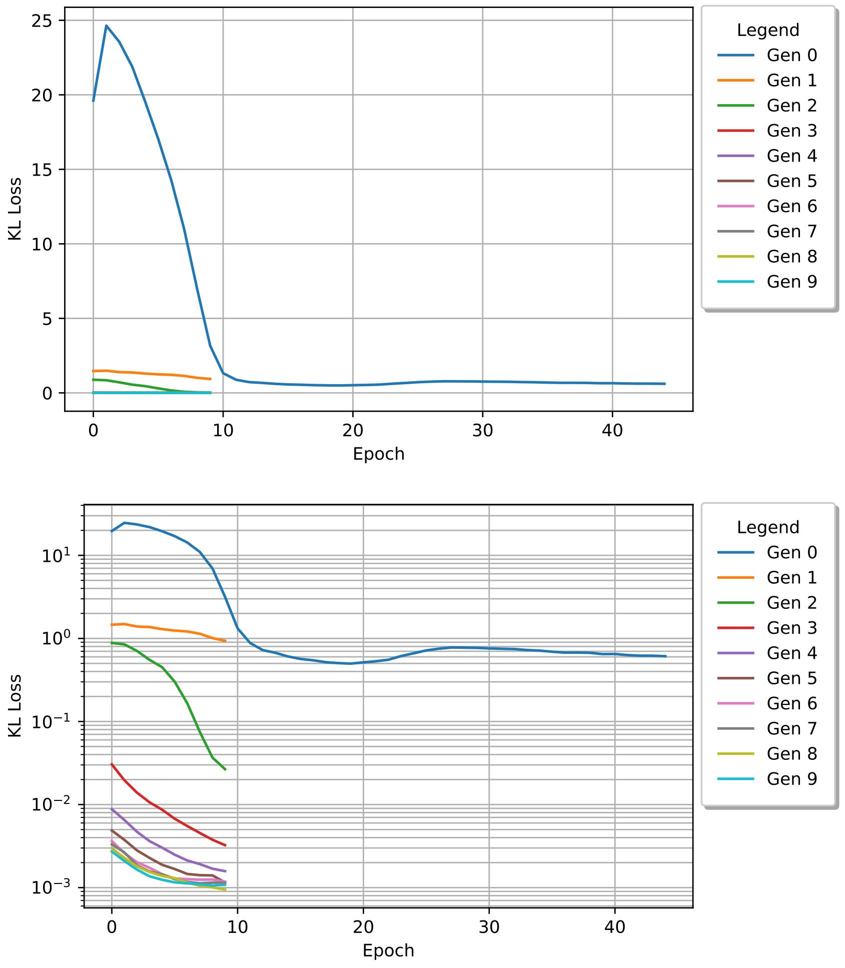


Figure 3.2: The KL loss values per epoch for each generation plotted over a linear scale (top) and a logarithmic scale (bottom).

Figure 3.2 shows how the VAE's encoder model performed per epoch and over each generation (plotted separately on the same graph). Note that the epochs and generations start from zero rather than starting from one. Thus, 10 generations and 10 epochs are still being calculated and reported. The upper graph is plotted over a linear scale, while the lower graph uses a logarithmic scale with a base of 10. In the linear graph, one can see the first generation is significantly larger than the remaining generations. It is for this reason that a logarithmically scaled graph was provided to see the performance of the other generations. The first generation also continues for more than 10 epochs as shown in the parameters from Table 3.6. In the first generation (generation zero), the VAE is trained on the entire dataset of 12,000 sequences. It takes the encoder model between two to three epochs to learn the key features of the training data. This is the reason for the large jump. Then the encoder model begins rapidly learning these features. Each subsequent generation shows the encoder model continues to improve on this and approaches a zero loss value.

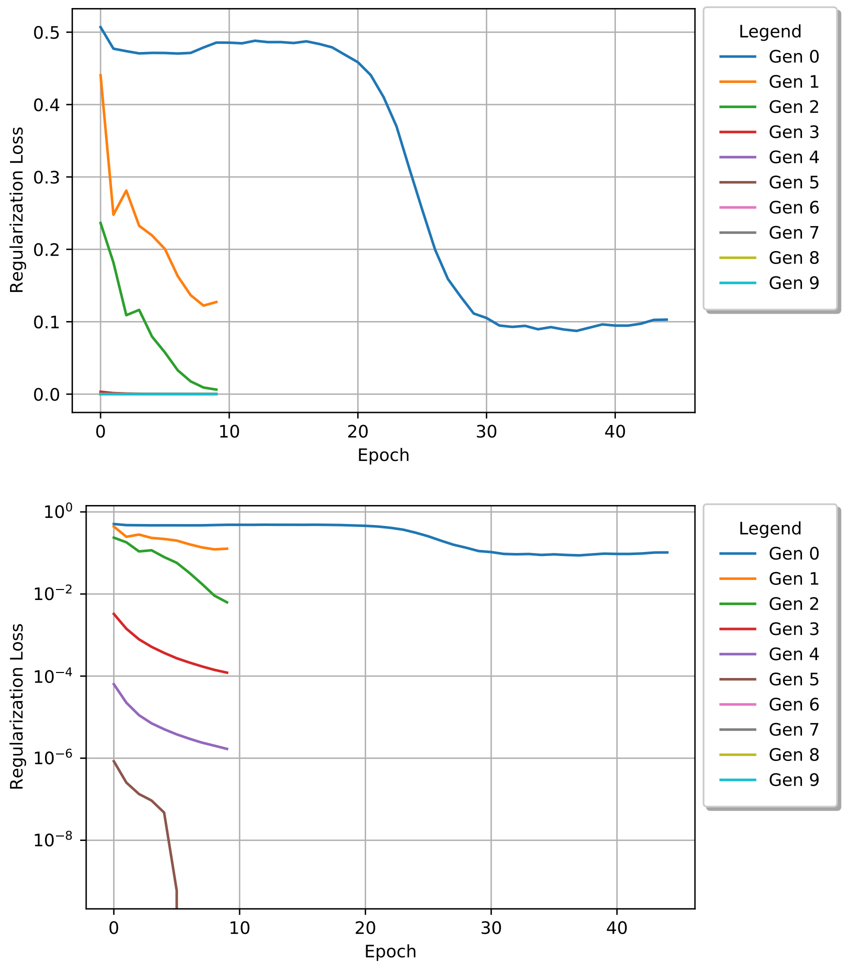


Figure 3.3: The regularization BCE loss values per epoch for each generation plotted over a linear scale (top) and a logarithmic scale (bottom).

The regularization model is subject to the same training scenario as the encoder model and, thus, performs similarly. However, as the DAPTEV Model implements a score threshold, the regularization model employs a BCE loss function. In Figure 3.3, it can be seen that the model is further refining its performance. In fact, as the logarithmic graph shows, the model learned to perfectly classify the data halfway through generation five. All generations after generation five are not shown on the graph for this reason.

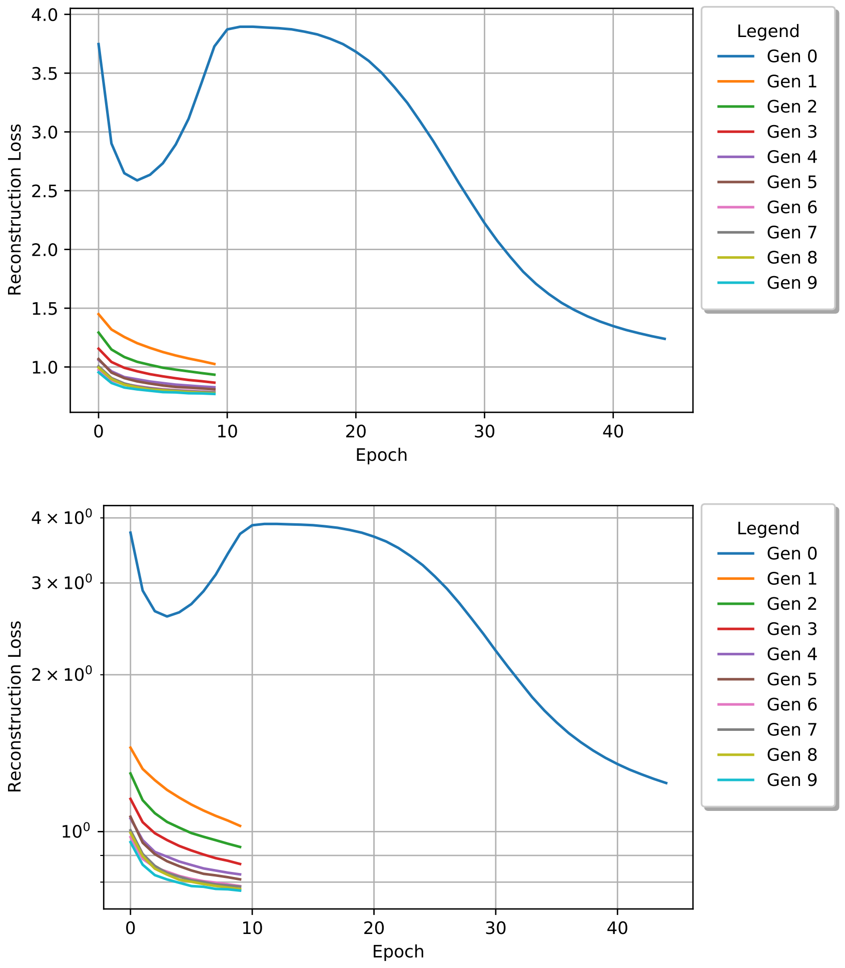


Figure 3.4: The reconstruction loss values per epoch for each generation plotted over a linear scale (top) and a logarithmic scale (bottom).

Figure 3.4 shows very little need for a logarithmic graph, but it still helps to see the slight differences in the decoder's performance per generation. Here, it is obvious that, after the first generation, the decoder is learning how to reproduce the encoded latent vectors very well. As both the encoder and the decoder are performing so well, this leads to duplicate sequences being produced in later generations.

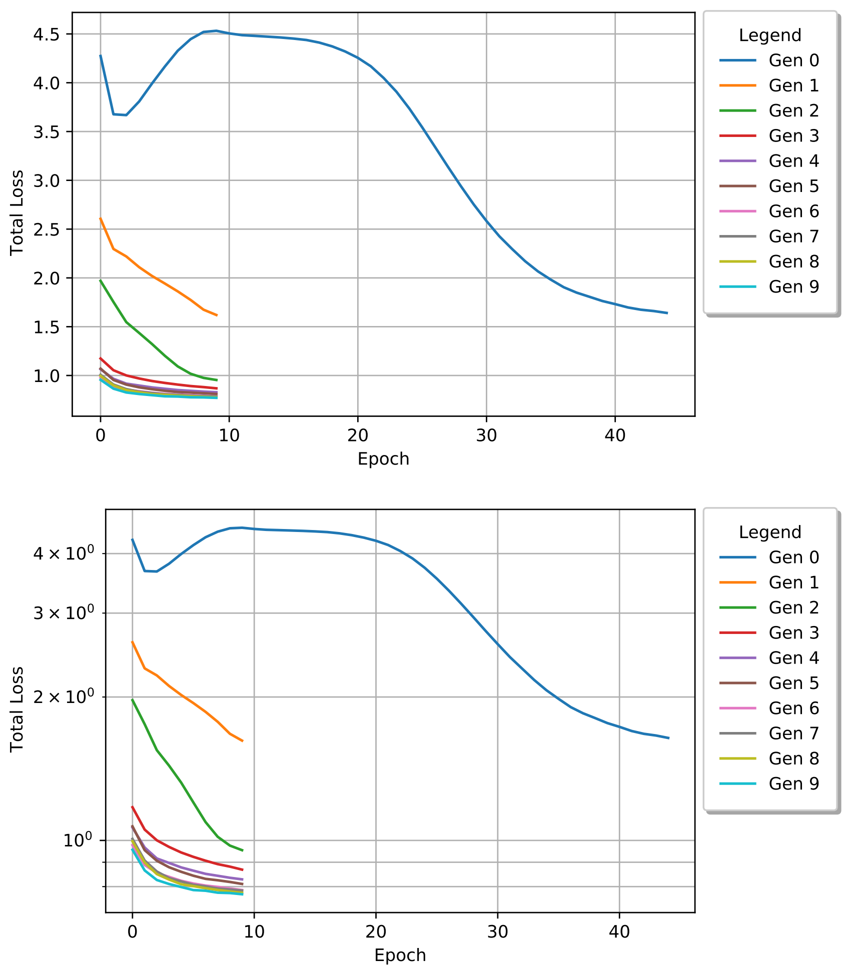


Figure 3.5: The total loss values per epoch for each generation plotted over a linear scale (top) and a logarithmic scale (bottom). Note: total loss values include the KL weight factor over time.

Figure 3.5 shows the total loss of the VAE per epoch per generation. As each model effectively reduced its loss values, it stands to reason that the total loss would reflect this performance. It is important to mention that the total loss is not simply an addition of all three loss values. The KL loss is also multiplied by the KL weight increments over time as specified by the parameters in Table 3.6.

For anyone reading this thesis who does not know how to read graphs in a logarithmic scale, each tick (horizontal lines corresponding to the numbers on the y-axis) is an increment in the order of magnitude and each minor tick (in-between the numbers on the y-axis) is one proportion of that magnitude change. For example going from $10^0 = 1$ to $10^1 = 10$ is one order of magnitude more and each minor tick in-between would equal +1 to the result of 10^0 (2, 3, 4, etc.).

3.2.2 Score Comparisons

Table 3.8: The score performance in the first and last generation for each experiment.

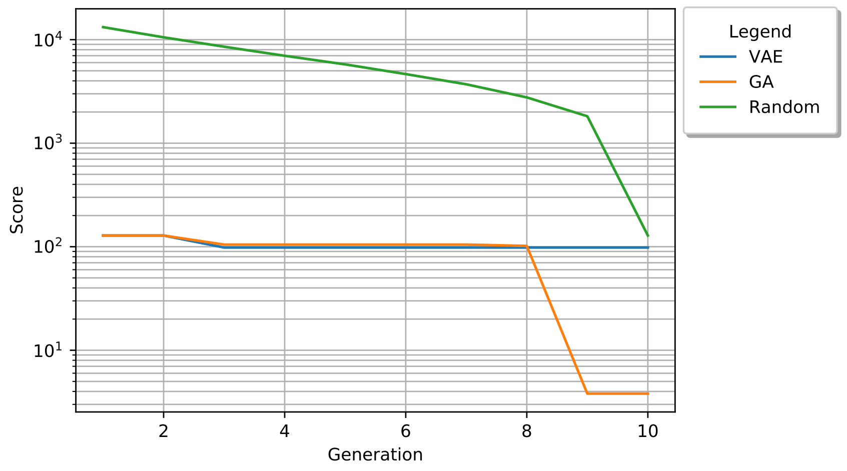
		DAPTEV	**GA**	**Hill Climber**
First Gen	*Best Score*	128.304	128.304	18,248.3
	Mean Score	1,046.470	1,016.521	23,628.530
	Median Score	1,071.148	1,040.083	21,956.050
	Worst Score	1,470.417	1,438.765	60,677.880
Last Gen	*Best Score*	98.205	3.811	128.304
	Mean Score	618.682	336.658	1,098.340
	Median Score	646.670	351.440	1,122.182
	Worst Score	857.166	459.842	1,530.179

Table 3.8 provides the before and after details of each model, illustrating overall docking scores optimization performance. Here, it can be seen that both the DAPTEV Model and the GA improved significantly from their worst scores to their best scores, with the GA outperforming the DAPTEV Model. However, in comparison to the hill climber model, both the GA and the DAPTEV Model have performed significantly better.

Notice here that the worst score in the original dataset (and the hill climber) was 60,677 and the best score was 128. Thus, the original dataset of 12,000 sequences already performed a lot of the random searching, leaving little room to grow. As such, observing just the best scores does not provide enough evidence of performance.

Figure 3.6 shows the best and the worst performing docking scores produced by each experiment per generation. The provided graphs are scaled logarithmically as the hill climber ("Random" in the legend) experiment performed significantly worse than the DAPTEV Model ("VAE" in the legend) and the GA. The linear graphs, therefore, do not properly illustrate the performance of the other two experiments. For a graph of the linear performance for just the DAPTEV Model and the GA, see Figure 3.7.

Upon observing Figure 3.7, it is clear that the GA produced better results than the DAPTEV Model. The lowest (best) score produced by the DAPTEV Model was 98.21 in generation three, whereas the GA obtained its lowest score of 3.81 in generation nine. Until Generation nine, it seems that the DAPTEV Model was exceeding the performance of the GA. It is possible that the GA simply encountered some luck in its



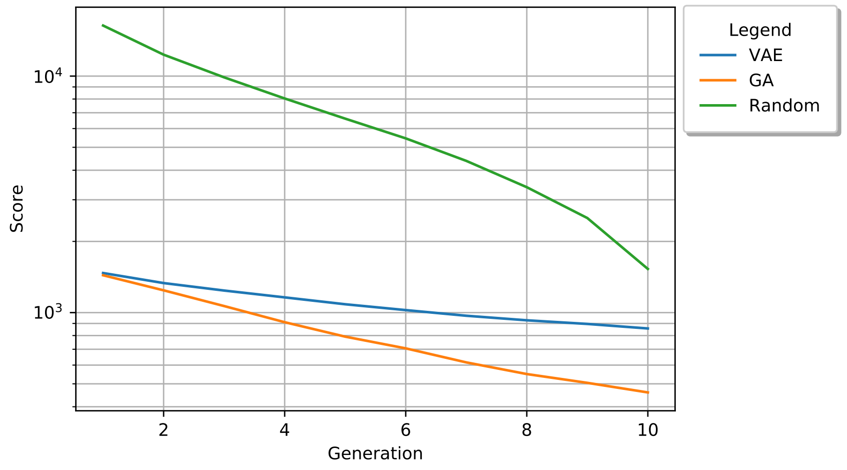
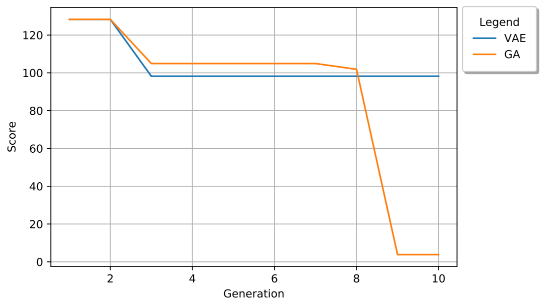


Figure 3.6: The three experiment's scores plotted per generation on the same logarithmically scaled graph. The top graph shows the best score per generation while the bottom graph shows the worst score.



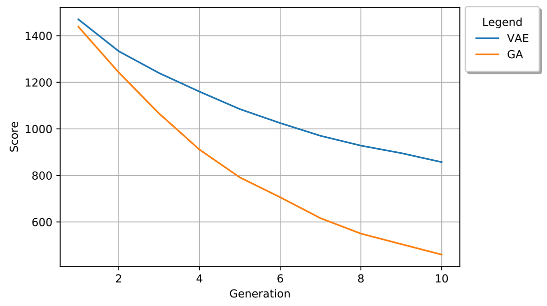


Figure 3.7: The DAPTEV Model and the GA experiment's scores plotted per generation on the same linearly scaled graph. The left graph shows the best score per generation while the right graph shows the worst score.

random search. However, the remaining graphs should be studied before concluding this.

When considering the worst score, again it can be seen that the GA outperformed the DAPTEV Model. The DAPTEV Model's worst scores fell from 1,470.417 to 857.166. The GA's worst score fell from 1,438.765 to 459.842. Based on these results, it seems that the GA may be more suited to score optimization than the DAPTEV Model. To confirm, additional metrics and figures are provided.

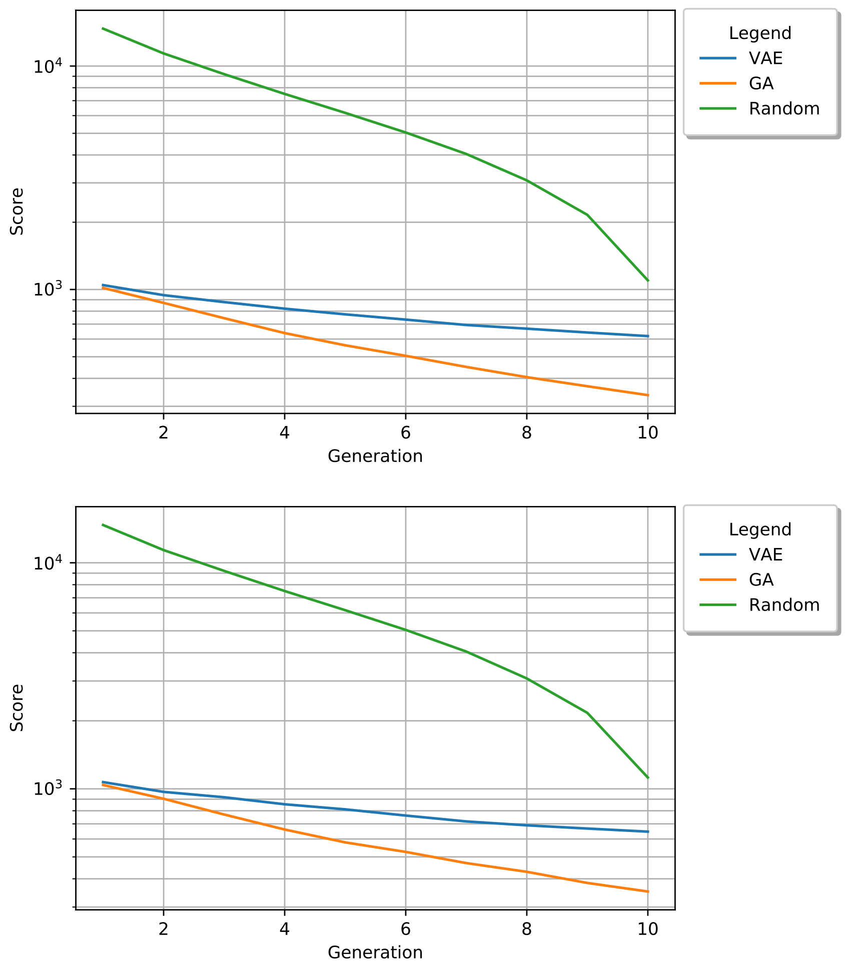


Figure 3.8: The three experiment's scores plotted per generation on the same logarithmically scaled graph. The top graph shows the mean score per generation while the bottom graph shows the median score.

In Figures 3.8 and 3.9, the means and medians of the scores plotted per generation can be seen. Both the means and medians seem to follow a fairly similar trend and are comparable in score output. Again, it seems that the GA is outperforming the DAPTEV Model. The hill climber method continues to perform poorly in comparison.

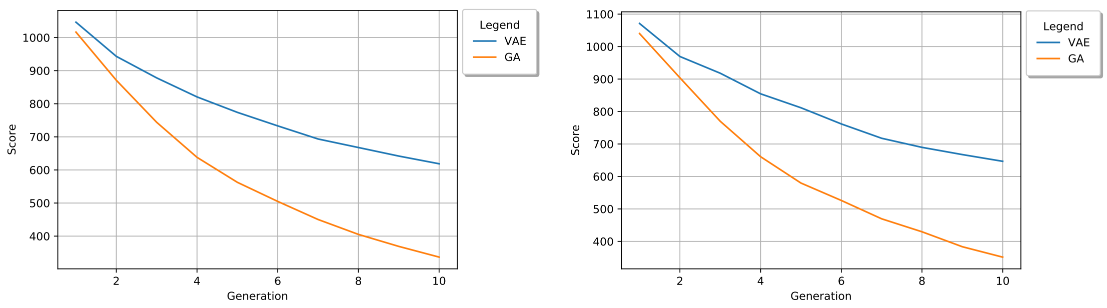


Figure 3.9: The DAPTEV Model and the GA experiment's scores plotted per generation on the same linearly scaled graph. The left graph shows the mean score per generation while the right graph shows the median score.

It also seemed prudent to consider the best five and worst five mean scores for each experiment. The absolute best and worst can send an extreme message and may not convey the true performance of each experiment. This is because the absolute best and worst do not include neighbouring sequence performance. If these models are indeed performing well, then one would expect to see the sequences near the best and worst improving overall. Note, however, median values were not provided as the best and worst five is already a fairly small sample size. Thus, providing a median calculation does not yield much information. Figures 3.10 and 3.11 depict the best and worst five score means per generation.

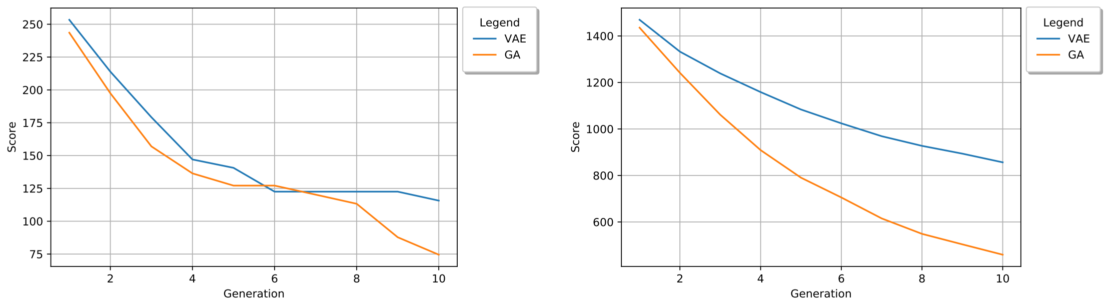
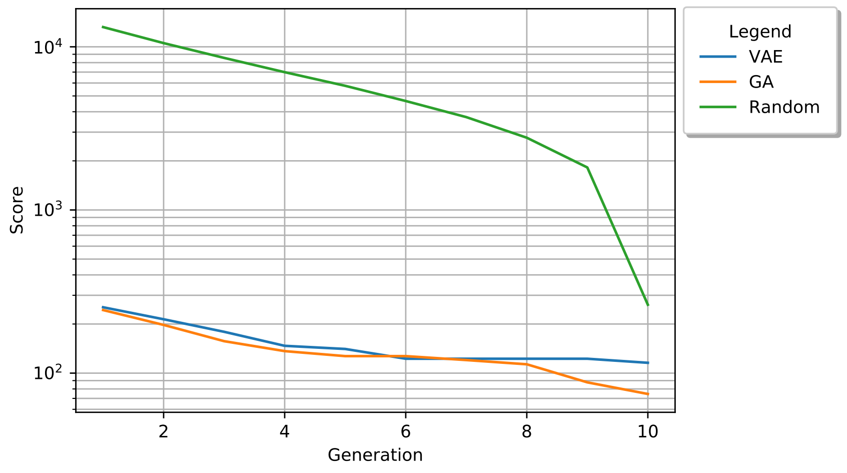


Figure 3.10: The DAPTEV Model and the GA experiment's scores plotted per generation on the same linearly scaled graph. The left graph shows the mean score for the 5 best performing sequences per generation while the right graph shows the mean score for the 5 worst performing sequences.

Here, one can see that the GA is indeed performing better than the DAPTEV Model. However, these graphs indicates that the GA and the DAPTEV Model are performing similarly. This observation is further strengthened when comparing the two models to the hill climber. It is starting to seem like the DAPTEV Model and the GA would converge to similar scores if they were allowed to continue for more generations as not a single graph has displayed convergence yet.

Lastly, two additional metrics were calculated to assess model performance. These metrics, expressed as ratios, include ***Novelty*** and ***Diversity***. *Novelty* is calculated as the number of generated sequences that do not exist in the initial training data versus the total number of generated sequences in a generation (the "population size"). *Diversity* is calculated as the number of generated unique sequences versus the total number of generated sequences in a generation. The novelty values calculated for the DAPTEV Model and the GA's last generation was 0.8475 and 0.9962 respectively. Similarly, the diversity values were 0.8487 and 0.9937 respectively.



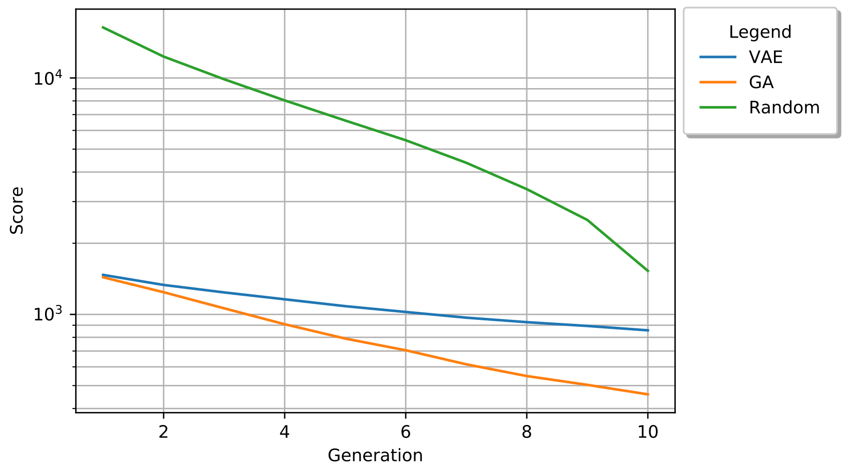


Figure 3.11: The three experiment's scores plotted per generation on the same logarithmically scaled graph. The top graph shows the mean score for the 5 best performing sequences per generation while the bottom graph shows the mean score for the 5 worst performing sequences.

3.2.3 Secondary Goal

Thus far, it has seemed as though the GA is the best suited model for this application. However, there are two goals for this experiment. While the first goal is to optimize the docking scores, the second goal is to produce sequences with at least one connection or nt pairing in the secondary structure. In other words, these models were also supposed to learn the key features of well-performing structural motifs applied to the SARS-CoV-2 Spike protein RBD.

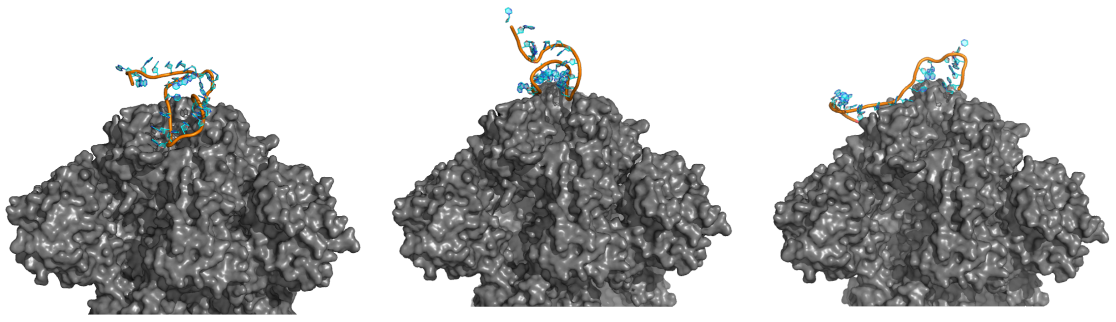
Table 3.9: Information relating to the folded RNAs in each model out of 800 total produced sequences. Note: all best, mean, median, and worst values relate only to the folded RNAs in the last generation, not the entire last generation's output.

			Scores				*Base Pairs*			
	Folded RNAs	*Rate (%)*	*Best*	*Mean*	*Med.*	*Worst*	*Min.*	*Mean*	*Med.*	*Max.*
VAE	310	39	128	694	671	875	1	2.4	2	5
GA	65	8	128	392	374	459	1	2.2	2	4
Hill	800	100	128	1,122	1,098	1,530	1	2.6	3	7

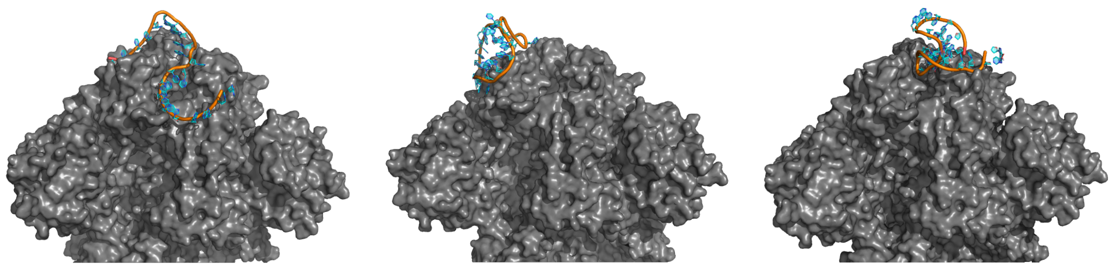
Table 3.9 provides further details on the produced folded sequences. Clearly, the GA performed rather poorly in this task, only producing 65 folded structures out of 800 total sequences, achieving an 8% fold rate. Conversely, the DAPTEV Model produced 310 folded structures out of 800 total sequences, achieving a 39% fold rate. It is important to note that the hill climber algorithm is entirely constructed of folded RNAs as the dataset creation script does not allow for unfolded structures in its random search. Thus, some of the metrics in the table seem significantly better.

For a visual of the best, median, and worst RNAs docked to the SARS-CoV-2 RBD, see Figure 3.12. Figure 3.12a shows renderings of these docked structures as produced by the DAPTEV Model (the VAE). Figure 3.12b shows the structures produced by the GA. Figure 3.12c shows the structures produced by the hill climber. Figure 3.12d shows the best scoring RNA that contains at least one nt connection in its secondary structure docked to the RBD.

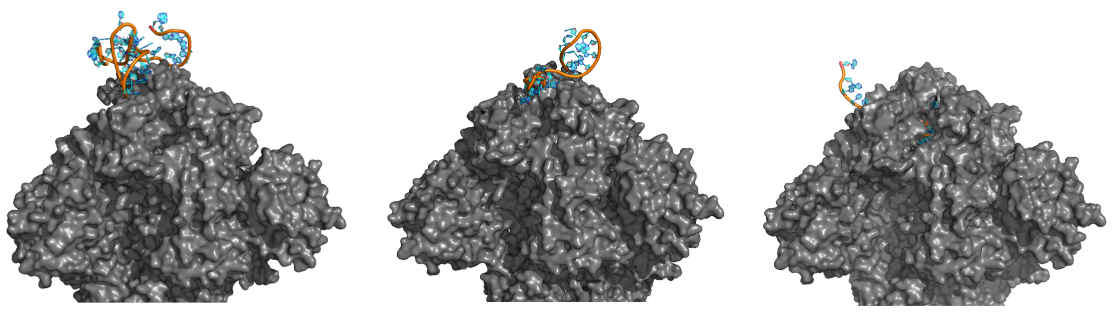
In Figure 3.12a, the best and worst complexes have unfolded RNAs. However, the median complex has a folded RNA with four base parings in its secondary structure. Figure 3.12b is similar, except its median complex has an unfolded RNA and, instead, its worst complex has the folded RNA with three base pairings. Every complex in Figure 3.12c has folded RNAs due to the nature of the dataset creation script. Every model produced the same best performing folded RNA as seen in Figure 3.12d.



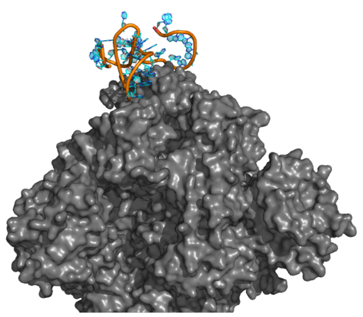
(a) Best, median, and worst docked RNAs produced by the VAE model.



(b) Best, median, and worst docked RNAs produced by the GA.



(c) Best, median, and worst Randomly-produced, folded, docked RNAs.



(d) Global best folded complex.

Figure 3.12: Best, median, and worst RNAs docked to the SARS-CoV-2 RBD based on Rosetta-returned scores. Figure 3.12d shows the best folded RNA produced by every model that has been docked to the spike protein.

3.2.4 Statistical Analysis

Table 3.10: Statistical Analysis of the Last Generation Scores for Each Model.

	P-Values		
Tests	***VAE***	***GA***	***Random***
Shapiro-Wilk	4.445250083e-16	1.87673511408e-15	1.43088210551e-12
Kruskal-Wallis	0		
	Group 1	**Group 2**	**Group 3**
	VAE, GA	***VAE, Rand***	***GA, Rand***
Mann-Whitney U	3.078407570e-182	1.694538918e-186	3.657219807e-255

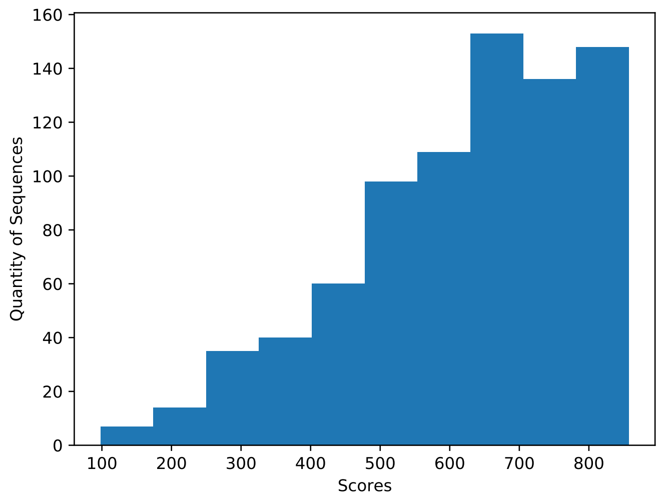
Note: The p-value for the Kruskal-Wallis test was too small to display.

Table 3.10 shows the results of statistical testing performed on the yielded scores from each experiment. The ANOVA test and t-test are used to determine if samples are statistically different from one another. The ANOVA test is used on three or more samples. If the returned p-value from the ANOVA test is less than a specified confidence interval (usually 0.05), it means one can be 95% confident that at least one of the samples are statistically different. Then, the t-test pairs these samples and determines which one or if all samples are statistically divergent, again within a confidence interval. However, the ANOVA and t-test assumes the data is from a normal distribution, using the means of the data when performing the test.

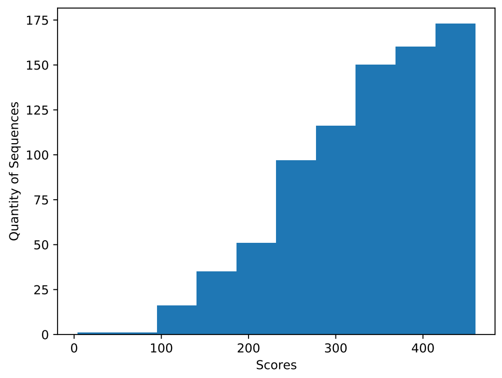
The Shapiro-Wilk test is used to check if the data is ***not*** from a normal distribution, typically with a 95% certainty. Thus, the "null hypothesis" of the Shapiro-Wilk test is that the sample comes from a normal distribution, and the goal is to reject this hypothesis. Note that the Shapiro-Wilk test does not prove the normality of the data. Samples that return p-values greater than 0.05, failing to reject the null hypothesis, only allows one to say that there is no significant deviation from normality. In other words, the Shapiro-Wilk test allows one to confidently say a sample ***does not*** come from a normal distribution, but it does ***not*** prove normality.

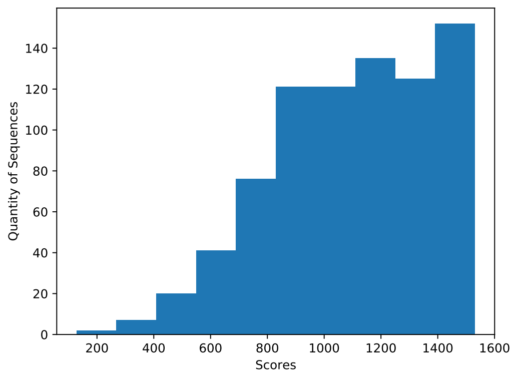
As the returned p-values from the Shapiro-Wilk test were all under 0.05, each experiment rejected the null hypothesis that their sample originated from a normal distribution. Thus, it is safe to assume the data is not normally distributed and the medians of the scores should be tested rather than the means. This finding is supported by the three histograms seen in Figure 3.13, each exhibiting a right skew in the results.

The next step is to determine if the median scores from each test are significantly similar or different. This can be determined in one Kruskal-Wallis test, similar to the ANOVA test, which checks to see if any provided sample sets are statistically different from each other. If the returned p-value is less than 0.05, then at least one sample set is considered to be different from the other sample sets in the test. In the test above, the returned p-value was so small that the computer could not represent it and simply returned a 0. This indicates that at least one of the sets are indeed statistically different from the other two.



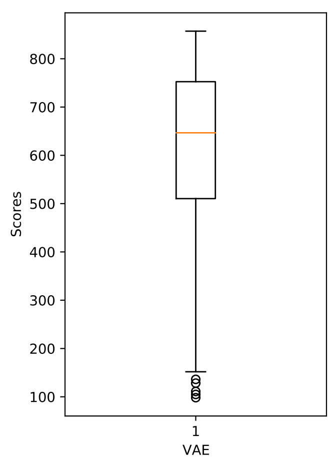
(a) VAE performance histogram.



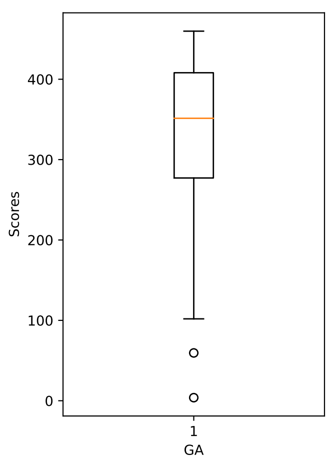


(b) GA performance histogram. (c) Hill climber performance histogram.

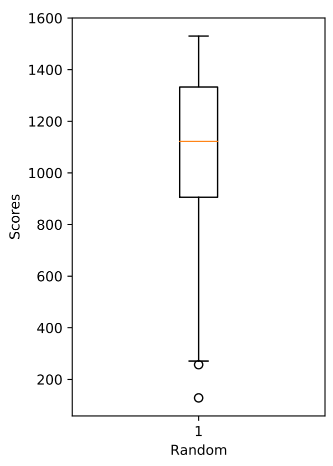
Figure 3.13: Histograms of the VAE, the GA, and the hill climber score results for the last generation.



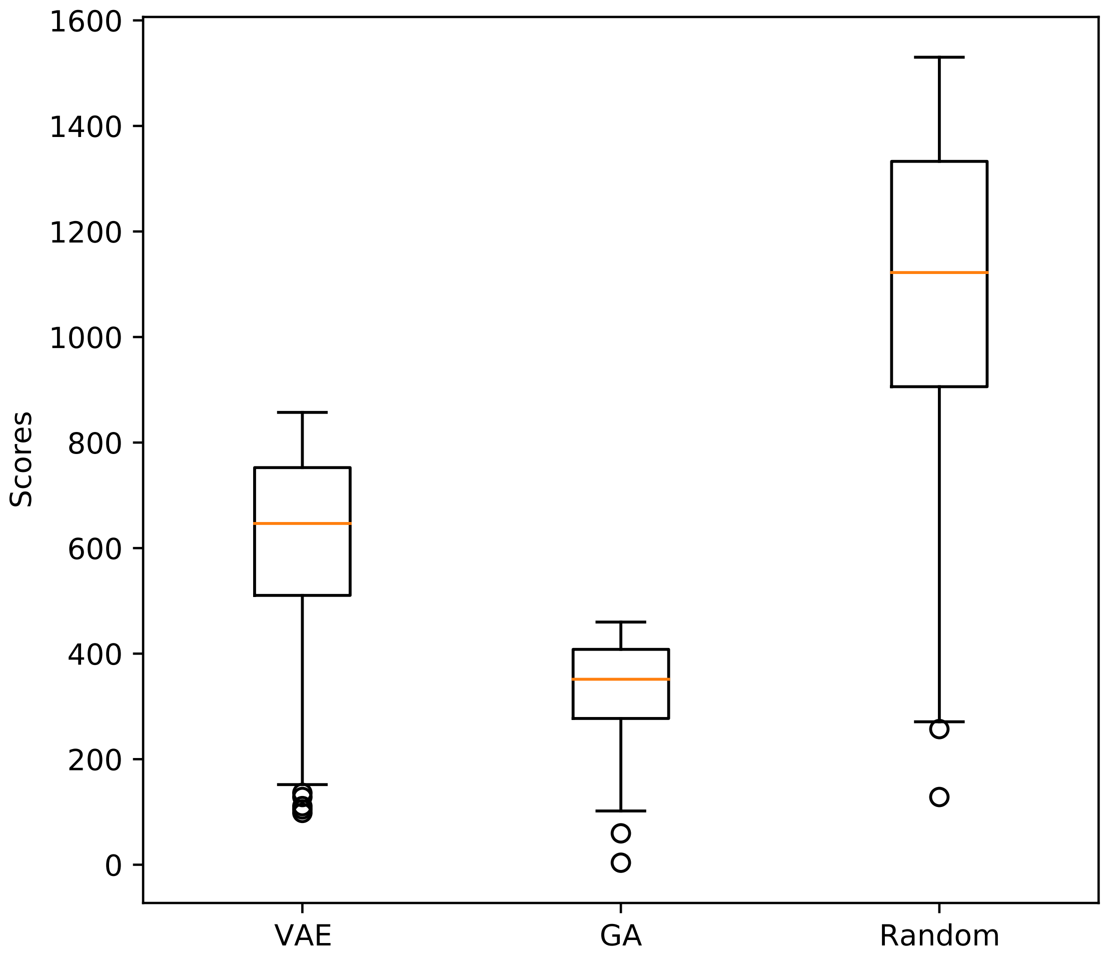
(a) VAE box plot.



(b) GA box plot.



(c) Random box plot.



(d) All box plots on the same scale.

Figure 3.14: Box plots of the VAE, the GA, and the random (hill climber) score results for the last generation.

To determine which sample set's median deviates from the others, pairs will have to be tested individually. The Mann-Whitney U test achieves exactly this. The median of the VAE's last score set was tested against the GA's median and returned a p-value below 0.05. While this shows that these medians are statistically divergent, it does not provide any information regarding which set is different from the others, nor does it yield any insight with regard to the random hill climber set's median. It is possible that the hill climber set's median could be similar to either the VAE or the GA's median. Thus, the VAE and the GA were also tested against the random set. In all three cases, each returned p-value was less than 0.05. This indicates that each set's median is statistically unique. To determine a ranking, box plots displaying outliers, minimums and maximums, lower and upper quartiles, and medians have been provided in Figure 3.14. From these figures, it can be seen that the hill climber set performed the worst, followed by the VAE and the GA as each median line is outside of the other set's quartile ranges. However, the VAE is much closer to the GA than it is to the random set.

Chapter 4

Discussion

4.1 Observations

While it may seem like the GA performed the best, this is not actually the case. As previously mentioned in “Additional Considerations” and “Native Rosetta Score Function”, the docking score can be artificially improved by producing an unfolded RNA secondary structure which, in turn, will incur fewer penalties during the RNA tertiary structure prediction and the docking simulation. If a model prioritizes only unconnected structures, it stands to reason that it would seem to perform better when only considering score output. However, the goal of this research was three-fold.

Firstly, the scores had to be optimized. This was indeed accomplished best by the GA as substantiated by the returned scores, statistical analysis, and box plots in “Results”. That being said, the graphs in “Score Comparisons” do not suggest that a convergence has occurred for any model. It is suspected that the DAPTEV Model could have further reduced its scores if it was allowed to continue iterating for more generations (perhaps 30 or more).

Secondly, some learning of well-performing structural motifs in the provided RNA secondary structures was required. Producing unfolded RNAs is not overly helpful when attempting to develop aptamer-based drugs. Even if this is a desirable trait, industry professionals should have the ability to produce more complex structures from ML models and not be forced to sacrifice these features.

Based on the percentage of folded structures in the last generation, it is clear that the DAPTEV Model performs significantly better than the GA. This points to the conclusion that the GA solely prioritized the optimization of scores to the detriment of structural complexity. The GA likely only obtained 65 folded structures due to these structural features not yet being filtered out. If the GA were allowed to iterate for more generations, there is a very good chance that the resulting output would be entirely unfolded RNAs. These results also indicate that the DAPTEV Model is indeed able to learn structural motif patterns from the training data. The percentage of folded RNAs could likely be increased even further by performing some additional modifications to the VAE parameters. For example, an increase in the vocabulary size would allow the VAE to learn more detailed motif information.

Thirdly, the DL model had to possess the ability to be queried for new, well-performing, structurally connected RNAs to explore aptamer-based drug development. This third point implies the persistence of a trained model and a way to request new data. Unfortunately, a GA must be trained every time new sequences are required. This would mean, if the computation is performed on just one system that is similar to the operating environment used in these experiments (Linux, 18 CPUs, a top-of-the-line NVIDIA graphics card), that researchers and industry professionals would have to wait for at least 5.5 days before obtaining new sequences. A VAE, conversely, is a DL system that can have its current state of learning saved and reloaded near immediately. Furthermore, researchers and industry professionals could obtain new sequences with a simple click of a button. The process of obtaining new sequences would take less than a couple seconds.

As a reminder, the results from this experiment with the VAE were obtained using previous generations of training data. The data from these previous generations are sent through the encoder and decoder models of the VAE to produce output sequences. The final sequences returned, in this case, are the ones produced in the last generation of the DAPTEV Model. This final generation occurs after the encoder and decoder models have learned how to distil and recreate the training material to an almost exact match as indicated by the VAE graphs in "VAE Performance". As such, the sequences produced in the final generation are heavily influenced by previous learning and are, thus, subject to duplication. Hence the need to replace these sequences. The same is not true, however, for producing new sequences. The user simply has to give the VAE's decoder model a new latent vector (a sample) based on a random normal distribution (or any other kind of distribution). The VAE will then produce a unique sequence that should still dock well to the specified target and be structurally complex (assuming proper training has taken place).

With everything mentioned above, one could posit that the DAPTEV Model performs admirably in all three requirements. It was able to achieve relatively low docking scores, especially when weighed against its starting generation's worst score and the worst score in the initial dataset. Additionally, every score produced in the last generation of the DAPTEV Model was below the score threshold of 3,500, which was the docking score limit set initially to tell the VAE that a sequence was performing well. The DAPTEV Model was able to learn structural features and other characteristics that make RNA perform well when docking to a specific target. Finally, the model can be saved and queried for new sequences that should perform well at docking to a specific target's RBD while also maintaining some structural complexity. In contrast, the GA and the hill climber algorithm must both be retrained every time, the GA prioritized score optimization at the cost of motif preservation, and the hill climber is likely to yield poor results due to premature convergence at a local minimum.

4.2 Limitations

There are some noteworthy limitations of this research. Firstly, the DAPTEV Model did not have enough starting aptamer data in the initial dataset. As previously mentioned in Dataset Preparation Script Parameters, only 849 unique, known, aptamer sequences were found during the literature review stage, but that was narrowed down further to only 344 out of 12,000 due to parameter and computation time constraints. Ideally, at least 30% of the data should have been existing aptamer data. Iwano et al.'s dataset could have helped significantly, being roughly 110,000 sequences [38]. However, the sequences are a minimum of 89 nt in length. This means every sequence would have been filtered out due to the parameter choices of 20 to 40 nt chosen to keep the computation time within an acceptable range. Moreover, these sequences seem to be paired with other target sequences, meaning there could be repetitions among the sequence entries, and this dataset only provides DNA sequences.

Secondly, the DAPTEV Model does not work with DNA unless the user converts DNA to RNA, via a base change of all thymine to uracil, due to the limitations of utilized software. While this research was mainly focused on RNA, plenty of other research has shown that DNA can be implemented. It would have been preferable to include DNA in the capabilities of this model without having to perform a conversion.

Right now, the randomly generated sequences must have at least one connection. However, more control over the secondary structure could have been desirable. Given enough time, the specification of connection amounts or a range of acceptable connections could have been implemented. To this end, perhaps the custom parabolic penalty function should have been implemented in the random sequence generator to penalized sequences for having too many or too few connections.

Currently, the DAPTEV Model assumes that the latent space follows a Gaussian distribution. This distribution may not be robust enough to capture the complexity of the task. Testing different distributions could have yielded further insight into the VAE's ability to operate in this problem space.

Additionally, there is no real way to validate or test this system as the DAPTEV Model is producing new sequences. How would one measure the accuracy for predicted sequences outside of the KL divergence and reconstruction values? What would that comparison look like? Would one measure the percentage of sequence similarity, perhaps utilizing sequence alignment? What if one small change in the RNA drastically affects the secondary and tertiary structures? Does one compare their similarities too? How would one compare a tertiary structure's similarity? Perhaps given more time, different assessment methods could be further explored and implemented.

Lastly, for anyone downloading the DAPTEV Model code, they may notice that the data class only allows for a vocab size of 3 maximum. This was chosen due to the simplicity of implementation and because this number was not going to increase for the duration of the research. However, it would be very simple to implement additional vocabulary amounts.

Chapter 5

Conclusion

The goal of this research was to see if a deep generative model (DGM) would be efficient at accelerating the ribonucleic acid (RNA) aptamer drug development process. While this research was applied to the SARS-CoV-2 spike protein, careful consideration was placed into the universal design of nearly any protein target. To prove the DGM's efficacy, the model had to display three things. The first was strong docking scores to the SARS-CoV-2 spike protein's receptor-binding domain (RBD), suggesting a strong target affinity. The second was structurally complex sequences, suggesting a strong target specificity. The third was the ability of the model to produce new sequences immediately rather than having to retrain every time. Furthermore, it would be ideal if the training for all this could be achieved in a reasonable amount of time. To test these capabilities, two other comparison models were included. One of which is a GA, and the other being a random sequence generator.

With regard to target affinity, one could conclude that both the DAPTEV Model and the GA performed well at this task. While the GA did outperform the DAPTEV Model in this regard, the difference between these two models was not very large when considering worst score values to best score values. For example, a docking score of 98 (DAPTEV Model) compared to 4 (GA) seems insignificant when compared to the worst value of 60,678 in the starting dataset. Especially when one considers that the score threshold was set to 3,500, meaning the DAPTEV Model considered 3,500 as a "good" docking score and was still able to produce scores significantly lower than that.

Regarding target specificity, the DAPTEV Model certainly shows some promising results. This is especially true when compared to the GA. The DAPTEV Model had an output with 39% of its produced sequences containing at least one connection in the secondary structure. This number could have been even higher if more vocabulary combinations had been implemented and some additional parameter tests were run. Also, only a *one base-pair* rule was enforced for randomly generated RNA. If additional customization options for the RNA structures were added, it is suspected that both the fold rate and fold complexity would increase. Conversely, the GA only produced an 8% fold rate. This indicates that the VAE was indeed able to learn some structural features of the data and the multilayer perceptron (MLP) regularizer model performed well at its task.

The last goal did not require testing. By its very nature, a variational autoencoder (VAE) can have its state saved and reloaded. This means the model can be trained, saved, and then loaded up again to produce new sequences. It can even be trained further should new information or more aptamer data be released/discovered. To obtain new sequences from a trained model, all one has to do is generate random samples from a normal distribution and provide these samples to the VAE. The VAE will then reconstruct the random samples based on its previous training. The returned sequence would then be something that could exist within a similar output space.

Furthermore, the computation time was reduced significantly. The original computation time estimate was 4+ years. This was due to both the size of the SARS-CoV-2 spike protein, and performing calculations sequentially. After reducing the size of the protein file by removing unnecessary residues and distributing the docking simulations over multiple CPUs, the time was reduced to just 16.5 days for three runs of one experiment. This would equate to 5.5 days to train the model. This time can be further reduced if the user has access to a cluster.

In summary, one could say this DAPTEV Model would indeed be useful in the aptamer drug development sector. All three conditions of producing sequences with high target affinity, producing sequences with high target specificity, and model reusability were met to some significant extent. Furthermore, the model can be trained in as little as 5.5 days or even less depending on the resources available to the user. At the moment, the model may require some fine-tuning, but it already shows a tremendous amount of promise. The code for this model can be obtained at `https://github.com/candress/DAPTEV_Model`.

5.1 Future Work

Several additional feature and considerations could improve this research. One such consideration, as previously mentioned in "Limitations", is the fact that there was not enough starting aptamer data. It would have been preferable to add more to the starting data set. Unfortunately, at the time of searching, it seemed most of the aptamer datasets and databases were taken down. For every paper found referencing an aptamer dataset/database, the data was no longer accessible. The aptamers found was provided in papers or found on the PDB website [10, 50, 51]. Perhaps a larger sequence length range than 20 to 40 nts could have been used to reduce the amount of data filtering. This way, far more of the 849 known aptamers (minus 44 for the unconnected structures) could have been utilized rather than just the 344. But even if the full list was used, it would still only equate to 849 out of 12,000 sequences.

Currently, this model only accepts RNA data. However, DNA aptamers are far more common. It would be beneficial to include the capability to work with DNA data. However, this would require either the conversion of DNA data to RNA data via base changing (which can be easily accomplished in a dataset preprocessing step) so the data can still work through Rosetta, or the integration of an additional system. The second option may be more or less accurate, which could affect the model's ability to learn due to inconsistencies.

Now that the research has come to an end, it is clear that allowing the user to perform tertiary structure prediction separately, or allowing the user to provide pre-calculated tertiary structure could be beneficial. While performing an additional tertiary structure prediction would almost double the computation time, it would allow the user to select more structurally complex RNAs. This would also allow the DAPTEV Model to optimize the RNA tertiary conformation score separately from the docking score, removing any influence from the lack of penalties during structure prediction.

Perhaps statically choosing the protein atom of CA and the RNA atom of C5 was negatively affecting the performance as there is no determination of which atoms are more likely to interact. Instead, each atom present in the protein residue and the possible RNA bases could have been noted. Then the DAPTEV Model could update these values during iteration depending on pre-calculated atom interactions.

It would be beneficial to increase the vocabulary combination amount. This DAPTEV Model only allows for combinations up to three characters long to reduce computation time. However, $4^3 = 64$ combinations and calculations only take the ML process a few minutes to perform per generation, whereas the scoring procedure per generation takes 800 minutes for a population size of 800. Moreover, vocabulary lengths of size three impede the model's ability to learn more complex structural motifs. This combination length could easily be increased to 6 ($4^6 = 4,096$ calculations), likely increasing computation time, but still far below 800 minutes, thus allowing the model to capture more information pertaining to these motifs.

The sequence generator could be updated to create a sequence with a specific GC percentage and length, while also yielding sequences with a specified range of connections rather than having to keep producing random sequences until one with a connection is discovered. For example, a palindrome system could be implemented to encourage the likelihood of hairpin loops [1]. This would allow the user to have better control over the quality of the produced RNA.

The VAE assumes the data comes from a normal distribution. Perhaps this distribution is not robust enough to represent the problem space. As such, the ability for the user to choose a default distribution could be implemented.

Some additional considerations and implementations could be the following: a learnable KL/reconstruction loss balancing component could be implemented so the model regulates the weights of the KL and the reconstruction loss [2]. A Transformer is another DL system that relies entirely on self-attention and is well-suited for Seq2Seq tasks [80]. It would be interesting to switch out the VAE with a Transformer to see how the results are affected. The addition of a molecular dynamic calculation/simulation capability would likely aid in assessing the quality of structure prediction and docking. Residue selection could be performed for the constraint file after performing tertiary structure prediction, but before docking, to get a better idea of what residue(s) to constrain.

Lastly, it is difficult to make conclusive observations at this moment due to the few number of experiments run. Ideally, at least 30 runs per experiment per model should be performed. However, this would take an inordinate amount of time to complete given the current computing restrictions for this thesis. The only way this would be feasible at this juncture is if one had access to a cluster to distribute the workload over.

Bibliography

[1] Rajesh Ahirwar, Smita Nahar, Shikha Aggarwal, Srinivasan Ramachandran, Souvik Maiti, and Pradip Nahar. In silico selection of an aptamer to estrogen receptor alpha using computational docking employing estrogen response elements as aptamer-alike molecules. *Scientific Reports*, 6(1):21285, February 2016.

[2] Andrea Asperti and Matteo Trentin. Balancing reconstruction error and kullback-leibler divergence in variational autoencoders, 2020.

[3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, December 2017.

[4] Nicholas Baker, Hongjing Lu, Gennady Erlikhman, and Philip J. Kellman. Deep convolutional networks do not classify based on global object shape. *PLOS Computational Biology*, 14(12):e1006613, December 2018.

[5] RCSB Protein Data Bank. Rcsb pdb - 6vxx: Structure of the sars-cov-2 spike glycoprotein (closed state).

[6] Florent Barbault, Bo Ren, Joseph Rebehmed, Catia Teixeira, Yun Luo, Ornella Smila-Castro, François Maurel, BoTao Fan, Liangren Zhang, and Lihe Zhang. Flexible computational docking studies of new aminoglycosides targeting rna 16s bacterial ribosome site. *European Journal of Medicinal Chemistry*, 43(8):1648–1656, August 2008.

[7] Florent Barbault, Liangren Zhang, Lihe Zhang, and Bo Tao Fan. Parametrization of a specific free energy function for automated docking against rna targets using neural networks. *Chemometrics and Intelligent Laboratory Systems*, 82(1):269–275, May 2006.

[8] Ali Bashir, Qin Yang, Jinpeng Wang, Stephan Hoyer, Wenchuan Chou, Cory McLean, Geoff Davis, Qiang Gong, Zan Armstrong, Junghoon Jang, Hui Kang, Annalisa Pawlosky, Alexander Scott, George E. Dahl, Marc Berndl, Michelle Dimon, and B. Scott Ferguson. Machine learning guided aptamer refinement and discovery. *Nature Communications*, 12(1):2366, April 2021.

[9] Kristian C. D. Becker and Richard C. Becker. Nucleic acid aptamers as adjuncts to vaccine development. *Current Opinion in Molecular Therapeutics*, 8(2):122–129, April 2006.

[10] Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, January 2000.

[11] Christopher Bishop. *Pattern Recognition and Machine Learning.* Information Science and Statistics. Springer-Verlag, New York, 2006.

[12] Javier Delgado Blanco, Leandro G. Radusky, Damiano Cianferoni, and Luis Serrano. Protein-assisted rna fragment docking (rnax) for modeling rna–protein interactions using modelx. *Proceedings of the National Academy of Sciences*, 116(49):24568–24573, December 2019.

[13] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. November 2015.

[14] Jeffrey A. Chao, Yury Patskovsky, Steven C. Almo, and Robert H. Singer. Structural basis for the coevolution of a viral rna–protein complex. *Nature Structural & Molecular Biology*, 15(1):103–105, January 2008.

[15] Zihao Chen, Long Hu, Bao-Ting Zhang, Aiping Lu, Yaofeng Wang, Yuanyuan Yu, and Ge Zhang. Artificial intelligence in aptamer–target binding prediction. *International Journal of Molecular Sciences*, 22(7):3605, January 2021.

[16] Kizzmekia S. Corbett, Darin K. Edwards, Sarah R. Leist, Olubukola M. Abiona, Seyhan Boyoglu-Barnum, Rebecca A. Gillespie, Sunny Himansu, Alexandra Schäfer, Cynthia T. Ziwawo, Anthony T. DiPiazza, Kenneth H. Dinnon, Sayda M. Elbashir, Christine A. Shaw, Angela Woods, Ethan J. Fritch, David R. Martinez, Kevin W. Bock, Mahnaz Minai, Bianca M. Nagata, Geoffrey B. Hutchinson, Kai Wu, Carole Henry, Kapil Bahl, Dario Garcia-Dominguez,

LingZhi Ma, Isabella Renzi, Wing-Pui Kong, Stephen D. Schmidt, Lingshu Wang, Yi Zhang, Emily Phung, Lauren A. Chang, Rebecca J. Loomis, Nedim Emil Altaras, Elisabeth Narayanan, Mihir Metkar, Vlad Presnyak, Cuiping Liu, Mark K. Louder, Wei Shi, Kwanyee Leung, Eun Sung Yang, Ande West, Kendra L. Gully, Laura J. Stevens, Nianshuang Wang, Daniel Wrapp, Nicole A. Doria-Rose, Guillaume Stewart-Jones, Hamilton Bennett, Gabriela S. Alvarado, Martha C. Nason, Tracy J. Ruckwardt, Jason S. McLellan, Mark R. Denison, James D. Chappell, Ian N. Moore, Kaitlyn M. Morabito, John R. Mascola, Ralph S. Baric, Andrea Carfi, and Barney S. Graham. Sars-cov-2 mrna vaccine design enabled by prototype pathogen preparedness. *Nature*, 586(7830):567–571, October 2020.

[17] Jose Cruz-Toledo, Maureen McKeague, Xueru Zhang, Amanda Giamberardino, Erin McConnell, Tariq Francis, Maria C. DeRosa, and Michel Dumontier. Aptamer base: a collaborative knowledge base to describe aptamers and selex experiments. *Database*, 2012(bas006), January 2012.

[18] Benedette Cuffari. What are spike proteins?, June 2020.

[19] Xinqiang Ding and Bin Zhang. Deepbar: A fast and exact method for binding free energy computation. *The Journal of Physical Chemistry Letters*, pages 2509–2515, March 2021.

[20] Maria Duca, Vincent Malnuit, Florent Barbault, and Rachid Benhida. Design of novel rna ligands that bind stem–bulge hiv-1 tar rna. *Chemical Communications*, 46(33):6162–6164, September 2010.

[21] Kurt Fredrick and Michael Ibba. Protein synthesis. *Nature*, 457(7226):157–158, January 2009.

[22] Hao Fu, Chunyuan Li, Xiaodong Liu, Jianfeng Gao, Asli Celikyilmaz, and Lawrence Carin. Cyclical annealing schedule: A simple approach to mitigating kl vanishing. March 2019.

[23] Nicolas Galtier and J.R. Lobry. Relationships between genomic g+c content, rna secondary structures, and optimal growth temperature in prokaryotes. *Journal of Molecular Evolution*, 44(6):632–636, June 1997.

[24] Rafael C. Gonzalez. Deep convolutional neural networks [lecture notes]. *IEEE Signal Processing Magazine*, 35(6):79–87, November 2018.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[26] Karl Grantham, Muhetaer Mukaidaisi, Hsu Kiang Ooi, Mohammad Sajjad Ghaemi, Alain Tchagang, and Yifeng Li. Deep evolutionary learning for molecular design. *IEEE Computational Intelligence Magazine*, 17(2):14–28, May 2022.

[27] Adrien Guilhot-Gaudeffroy, Christine Froidevaux, Jérôme Azé, and Julie Bernauer. Protein-rna complexes and efficient automatic docking: Expanding rosettadock possibilities. *PLOS ONE*, 9(9):e108928, September 2014.

[28] Michiaki Hamada. In silico approaches to rna aptamer design. *Biochimie*, 145:8–14, February 2018.

[29] Jiahua He, Jun Wang, Huanyu Tao, Yi Xiao, and Sheng-You Huang. Hnadock: a nucleic acid docking server for modeling rna/dna–rna/dna 3d complex structures. *Nucleic Acids Research*, 47(W1):W35–W42, July 2019.

[30] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *arXiv:1703.06870 [cs]*, January 2018.

[31] Frances L. Heredia, Abiel Roche-Lima, and Elsie I. Parés-Matos. A novel artificial intelligence-based approach for identification of deoxynucleotide aptamers. *PLOS Computational Biology*, 17(8):e1009247, August 2021.

[32] Yangyu Huang, Haotian Li, and Yi Xiao. Using 3drpc for rna–protein complex structure prediction. *Biophysics Reports*, 2(5):95–99, 2016.

[33] Yangyu Huang, Haotian Li, and Yi Xiao. 3drpc: a web server for 3d rna–protein structure prediction. *Bioinformatics*, 34(7):1238–1240, April 2018.

[34] Yangyu Huang, Shiyong Liu, Dachuan Guo, Lin Li, and Yi Xiao. A novel protocol for three-dimensional structure prediction of rna-protein complexes. *Scientific Reports*, 3(1):1887, May 2013. Number: 1 Publisher: Nature Publishing Group.

[35] Ian Fare, Eva Liu, Maggie Hou, Judy Chen, Siddharth Reed, and Biren Dave. Team:mcmasteru/genetic - 2017.igem.org.

[36] Jinho Im, Byungkyu Park, and Kyungsook Han. A generative model for constructing nucleic acid sequences binding to a protein. *BMC Genomics*, 20(13):967, December 2019.

[37] Junichi Iwakiri, Michiaki Hamada, Kiyoshi Asai, and Tomoshi Kameda. Improved accuracy in rna–protein rigid body docking by incorporating force field for molecular dynamics simulation into the scoring function. *Journal of Chemical Theory and Computation*, 12(9):4688–4697, September 2016.

[38] Natsuki Iwano, Tatsuoc Adachi, Kazuteru Aoki, Yoshikazuc Nakamura, and Michiaki Hamada. Raptgen: A variational autoencoder with profile hidden markov model for generative aptamer discovery. Technical report, February 2021.

[39] Myeongjun Jang, Seungwan Seo, and Pilsung Kang. Recurrent neural network-based semantic variational autoencoder for sequence-to-sequence learning. *arXiv:1802.03238 [cs]*, June 2018.

[40] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, pages 1–7, July 2021.

[41] Kalli Kappel and Rhiju Das. Sampling native-like structures of rna-protein complexes through rosetta folding and docking. *Structure*, 27(1):140–151.e5, January 2019.

[42] Anthony D. Keefe, Supriya Pai, and Andrew Ellington. Aptamers as therapeutics. *Nature Reviews Drug Discovery*, 9(7):537–550, July 2010.

[43] Tae-Hyeong Kim and Seong-Wook Lee. Aptamers for anti-viral therapeutics and diagnostics. *International Journal of Molecular Sciences*, 22(8):4168, April 2021.

[44] Andrew B. Kinghorn, Lewis A. Fraser, Shaolin Lang, Simon Chi-Chin Shiu, and Julian A. Tanner. Aptamer bioinformatics. *International Journal of Molecular Sciences*, 18(12), 2017.

[45] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes, May 2013.

[46] Nikolay Kolev. Realistic 3d illustration of covid-19 virus structure diagram., April 2020. Stock Illustration ID: 1703444803, Realistic 3D Illustration of COVID-19 Virus Structure Diagram. Corona Virus SARS-CoV-2, 2019 nCoV virus sheme. Full text description with sliced model and RNA on dark background. Omicron.

[47] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951.

[48] Jun Lan, Jiwan Ge, Jinfang Yu, Sisi Shan, Huan Zhou, Shilong Fan, Qi Zhang, Xuanling Shi, Qisheng Wang, Linqi Zhang, and Xinquan Wang. Structure of the sars-cov-2 spike receptor-binding domain bound to the ace2 receptor. *Nature*, 581(7807):215–220, May 2020.

[49] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In David A. Forsyth, Joseph L. Mundy, Vito di Gesú, and Roberto Cipolla, editors, *Shape, Contour and Grouping in Computer Vision*, Lecture Notes in Computer Science, pages 319–345. Springer, Berlin, Heidelberg, 1999.

[50] Gwangho Lee, Gun Hyuk Jang, Ho Young Kang, and Giltae Song. Predicting aptamer sequences that interact with target proteins using an aptamer-protein interaction classifier and a monte carlo tree search approach. *PLOS ONE*, 16(6):e0253760, June 2021.

[51] Bi-Qing Li, Yu-Chao Zhang, Guo-Hua Huang, Wei-Ren Cui, Ning Zhang, and Yu-Dong Cai. Prediction of aptamer-target interacting pairs with pseudo-amino acid composition. *PLOS ONE*, 9(1):e86729, January 2014.

[52] Yaozong Li, Jie Shen, Xianqiang Sun, Weihua Li, Guixia Liu, and Yun Tang. Accuracy assessment of protein-based docking programs against rna targets. *Journal of Chemical Information and Modeling*, 50(6):1134–1146, June 2010.

[53] Ziwei Liu, Xiaoxiao Li, Ping Luo, Chen Change Loy, and Xiaoou Tang. Deep learning markov random field for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(8):1814–1828, August 2018.

[54] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2016.

[55] John L. Markley, Ad Bax, Yoji Arata, C. W. Hilbers, Robert Kaptein, Brian D. Sykes, Peter E. Wright, and Kurt Wüthrich. Recommendations for the presentation of nmr structures of proteins and nucleic acids – iupac-iubmb-iupab inter-union task group on the standardization of data bases of protein and nucleic acid structures determined by nmr spectroscopy. *Journal of Biomolecular NMR*, 12(1):1–23, July 1998.

[56] MATLAB. *version 9.6.0 (R2019a).* The MathWorks Inc., Natick, Massachusetts, 2019.

[57] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, September 2013. arXiv:1301.3781 [cs].

[58] Boaz Musafia, Rony Oren-Banaroya, and Silvia Noiman. Designing anti-influenza aptamers: Novel quantitative structure activity relationship approach gives insights into aptamer - virus interaction. *PLoS ONE*, 9(5), May 2014.

[59] Richard Nagyfi. The differences between artificial and biological neural networks, September 2018.

[60] Michael A. Nielsen. Neural networks and deep learning. 2015.

[61] Chandran Nithin, Pritha Ghosh, and Janusz M. Bujnicki. Bioinformatics tools and benchmarks for computational docking and 3d structure prediction of rna-protein complexes. *Genes*, 9(9):432, September 2018.

[62] Byungkyu Park and Kyungsook Han. Discovering protein-binding rna motifs with a generative model of rna sequences. *Computational Biology and Chemistry*, 84, February 2020.

[63] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[64] Shubham Patel. All you need to know about variational autoencoder, June 2019.

[65] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, 2014. Association for Computational Linguistics.

[66] Stanley Plotkin, James M. Robinson, Gerard Cunningham, Robyn Iqbal, and Shannon Larsen. The complexity and cost of vaccine manufacturing – an overview. *Vaccine*, 35(33):4064–4071, July 2017.

[67] Lakshmanane Premkumar, Bruno Segovia-Chumbez, Ramesh Jadi, David R. Martinez, Rajendra Raut, Alena Markmann, Caleb Cornaby, Luther Bartelt, Susan Weiss, Yara Park, Caitlin E. Edwards, Eric Weimer, Erin M. Scherer, Nadine Rouphael, Srilatha Edupuganti, Daniela Weiskopf, Longping V. Tse, Yixuan J. Hou, David Margolis, Alessandro Sette, Matthew H. Collins, John Schmitz, Ralph S. Baric, and Aravinda M. de Silva. The receptor binding domain of the viral spike protein is an immunodominant and highly specific target of antibodies in sars-cov-2 patients. *Science Immunology*, 5(48):eabc8413, June 2020.

[68] Tyler J. Ripperger, Jennifer L. Uhrlaub, Makiko Watanabe, Rachel Wong, Yvonne Castaneda, Hannah A. Pizzato, Mallory R. Thompson, Christine Bradshaw, Craig C. Weinkauf, Christian Bime, Heidi L. Erickson, Kenneth Knox, Billie Bixby, Sairam Parthasarathy, Sachin Chaudhary, Bhupinder Natt, Elaine Cristan, Tammer El Aini, Franz Rischard, Janet Campion, Madhav Chopra, Michael Insel, Afshin Sam, James L. Knepler, Andrew P. Capaldi, Catherine M. Spier, Michael D. Dake, Taylor Edwards, Matthew E. Kaplan, Serena Jain Scott, Cameron Hypes, Jarrod Mosier, David T. Harris, Bonnie J. LaFleur, Ryan Sprissler, Janko Nikolich-Žugich, and Deepta Bhattacharya. Orthogonal sars-cov-2 serological assays enable surveillance of low-prevalence communities and reveal durable humoral immunity. *Immunity*, 53(5):925–933.e4, November 2020.

[69] LLC Schrödinger and Warren DeLano. The pymol molecular graphics system, version 1.8, November 2015.

[70] Osman Shabir. What is a receptor-binding domain (rbd)?, July 2020.

[71] Jia Song, Yuan Zheng, Mengjiao Huang, Lingling Wu, Wei Wang, Zhi Zhu, Yanling Song, and Chaoyong Yang. A sequential multidimensional analysis algorithm

for aptamer identification based on structure analysis and machine learning. *Analytical Chemistry*, 92(4):3307–3314, February 2020.

[72] Yanling Song, Jia Song, Xinyu Wei, Mengjiao Huang, Miao Sun, Lin Zhu, Bingqian Lin, Haicong Shen, Zhi Zhu, and Chaoyong Yang. Discovery of aptamers targeting the receptor-binding domain of the sars-cov-2 spike glycoprotein. *Analytical Chemistry*, 92(14):9895–9900, July 2020.

[73] Filip Stefaniak and Janusz M. Bujnicki. Annapurna: a scoring function for predicting rna-small molecule interactions. *bioRxiv*, page 2020.09.08.287136, September 2020.

[74] Eli Stevens, Luca Antiga, and Thomas Viehmann. *Deep Learning with PyTorch*. Manning Publications, Shelter Island, 1 edition, July 2020.

[75] Gwendolyn M. Stovall, Vincent Huynh, Shelly Engelman, and Andrew D. Ellington. Aptamers in education: Undergraduates make aptamers and acquire 21st century skills along the way. *Sensors (Basel, Switzerland)*, 19(15):3270, July 2019.

[76] Wanbo Tai, Lei He, Xiujuan Zhang, Jing Pu, Denis Voronin, Shibo Jiang, Yusen Zhou, and Lanying Du. Characterization of the receptor-binding domain (rbd) of 2019 novel coronavirus: implication for development of rbd protein as a viral attachment inhibitor and vaccine. *Cellular and Molecular Immunology*, 17(6):613–620, June 2020.

[77] Bernhard C. Thiel, C. Flamm, and I. Hofacker. Rna structure prediction: from 2d to 3d. 2017.

[78] Irina Tuszynska, Marcin Magnus, Katarzyna Jonak, Wayne Dawson, and Janusz M. Bujnicki. Npdock: a web server for protein–nucleic acid docking. *Nucleic Acids Research*, 43(W1):W425–W430, July 2015.

[79] UW Video. Application of aptamers in the clinical laboratory - geoffrey baird, md, phd.

[80] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, December 2017.

[81] Alexandra C. Walls, Young-Jun Park, M. Alejandra Tortorici, Abigail Wall, Andrew T. McGuire, and David Veesler. Structure, function, and antigenicity of the sars-cov-2 spike glycoprotein. *Cell*, 181(2):281–292.e6, April 2020.

[82] Sheng Wang, Jian Peng, Jianzhu Ma, and Jinbo Xu. Protein secondary structure prediction using deep convolutional neural fields. *Scientific Reports*, 6, January 2016.

[83] Hannah K. Wayment-Steele, Wipapat Kladwang, Alexandra I. Strom, Jeehyung Lee, Adrien Treuille, Eterna Participants, and Rhiju Das. Rna secondary structure packages evaluated and improved by high-throughput experiments. preprint, Biophysics, May 2020.

[84] Brian C. Wengerter, Joseph A. Katakowski, Jacob M. Rosenberg, Chae Gyu Park, Steven C. Almo, Deborah Palliser, and Matthew Levy. Aptamer-targeted antigen delivery. *Molecular Therapy: The Journal of the American Society of Gene Therapy*, 22(7):1375–1387, July 2014.

[85] Michael Wornow. Applying deep learning to discover highly functionalized nucleic acid polymers that bind to small molecules. June 2020.

[86] Chunyan Yi, Xiaoyu Sun, Jing Ye, Longfei Ding, Meiqin Liu, Zhuo Yang, Xiao Lu, Yaguang Zhang, Liyang Ma, Wangpeng Gu, Aidong Qu, Jianqing Xu, Zhengli Shi, Zhiyang Ling, and Bing Sun. Key residues of the receptor binding motif in the spike protein of sars-cov-2 that interact with ace2 and neutralizing antibodies. *Cellular & Molecular Immunology*, 17(6):621–630, June 2020.

[87] Soo Young Yoon, Grace Gee, Kee-Jong Hong, and Sang-Hwan Seo. Application of aptamers for assessment of vaccine efficacy. *Clinical and Experimental Vaccine Research*, 6(2):160–163, July 2017.

[88] Xinran Zou, Jing Wu, Jiaqi Gu, Li Shen, and Lingxiang Mao. Application of aptamers in virus detection and antiviral therapy. *Frontiers in Microbiology*, 0, 2019.

Appendix A

Installation

The following outlines system requirements and build instructions. Note: Before installing, make sure you can see hidden files and folders on your system and that you have admin privileges. You will need to access and modify your ".bashrc" file and perform other installation actions which require admin access. your system will need:

1. Nvidia graphics card (to fully utilize PyTorch)
2. Python 2.7 to run the Rosetta version
3. Python 3.8.10 should be your system default. I.e. When you type "python" in the terminal, python 3.8.10 should load up. Otherwise, you should run the code from Visual Studios Code editor with python 3.8.10 as your interpreter.
4. Packages for python3:
    (a) Numpy
    (b) Pandas
    (c) tqdm
    (d) Pytorch - no instructions for this yet
5. Cmake version 2.8.8 or greater (`https://cmake.org/download/`) and Ninja build (`https://github.com/martine/ninja.git`) to compile Rosetta.
6. Rosetta version 3.12
    (a) Go to the Rosetta Commons website and follow the instructions to obtain an academic licence (`https://www.rosettacommons.org/software`)

(b) Complete the licence process, then download Rosetta version 3.12 (the "**Rosetta 3.12 source + binaries for Linux**" option)

(c) Exact the Rosetta folder from the zip file.

(d) Rename the extracted folder to "**Rosetta**", then move it to your desired location (I chose "`/home/user1/`")

(e) In your "`/home/user1/`" directory (where "user1" is your computer user name), open your "**.bashrc**" file and enter the following:

```
export ROSETTA=/home/user1/Rosetta/
export ROSETTA_TOOLS=/home/user1/Rosetta/main/tools
export RNA_TOOLS=$ROSETTA/tools/rna_tools/
export PATH=$RNA_TOOLS/bin/:$PATH
export PYTHONPATH=$PYTHONPATH:$RNA_TOOLS/bin/
```

(f) Then save and close the ".bashrc" file.

(g) Open a terminal in "`/Rosetta/main/source`" and run the following command:

```
python2 ninja_build.py r -remake
```

(h) Once compilation finishes, close that terminal. Then open a new terminal in "`Rosetta/tools/rna_tools/`".

(i) Run this command:

```
python $RNA_TOOLS/sym_link.py
```

(j) Verify everything has been done correctly up to this point by running the following in your terminal:

```
rna_helix.py -h
```

You should see a printout of usage instructions for "`rna_helix.py`". If you do not see this, either a mistake was made in the previous steps (e.g. a spelling error), or you do not have the prerequisite items mentioned above.

(k) Now, go into your "`Rosetta/tools/rna_tools/bin/`" directory and confirm there is an "`rna_denovo`" symlink there. If it is not present, you have to create it yourself:

i. Go into the "`/Rosetta/main/source/bin/`" directory and find the "`rna_denovo`" symlink.

ii. Copy and paste the "`rna_denovo`" symlink from "`/Rosetta/main/source/bin/`" into "`Rosetta/tools/rna_tools/bin/`".

(l) To confirm the required command (`rna_denovo`) is working correctly, navigate to "`/Rosetta/demos/public/rnp_structure_prediction/`", open the "flags" file, add the following line to the bottom of the file:

```
-bps_moves false
```

Then save and close the flags file. Open a terminal in this directory and run the following command:

```
rna_denovo @flags | tee terminal_output.txt
```

You should see a lot of text output to the terminal, with a table showing scores near the end (once the command finishes). A text file named "`terminal_output`" will also be saved to this folder location which will contain the same information that was printed to the terminal. This file is created so you can obtain the scores associated with each docking attempt.

- Note: This version of Rosetta does not seem to write its scores to the "out" file when running the "`rna_denovo`" command (even though the documentation says it does). Thus, you must save the terminal output while running the "`rna_denovo`" command and then perform a "`grep`" command on the terminal output file.

(m) You can then, in the same terminal, run:

```
extract_lowscore_decoys.py 2qux_fold_and_dock.out 5
```

To get the 5 pdb files for the docked RNA-Protein complexes. Notice that the number 5 here corresponds with the number 5 in the flags file, indicating 5 structures are to be predicted.

- Note: you will have to use something like PyMol (`https://pymol.org/2/`) to view/render the pdb files.

7. Eternafold and Arnie for secondary structure prediction

(a) You can go to `https://github.com/DasLab/arnie` and follow the instructions in the "`ReadMe`" file to install Arnie and Eternafold. However, the instructions on this page assume you already well versed, and know how to do certain things, in a Linux environment. If you are new to Linux, you can follow my steps as outlined below.

(b) Install Eternafold (more accurate thermodynamic predictions) first:

i. Request a licence and Download EternaFold from `https://eternagame.org/software`

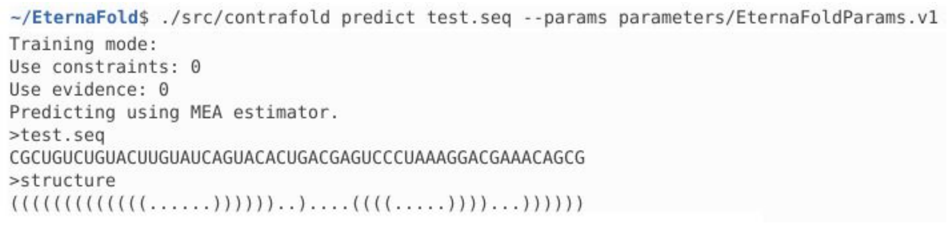
ii. Extract the zip file, rename the extracted folder to "EternaFold", then move the EternaFold folder to your desired destination. I chose "`/home/user1`" as my desired location.

- The Arnie github will tell you to simply download the package, then check your build by running a certain command (see below). This may return an error. If so, open a terminal in the "`EternaFold/src`" folder and type "make" (without quotes).

iii. Once the "make" command is finished, confirm the EternaFold build is working: within the terminal you typed "make", change directory to the EternaFold directory (type "cd .." without the quotation marks). If you did not need to run the "make" command, open a terminal in the EternaFold directory. Then type:

```
./src/contrafold predict test.seq
--params parameters/EternaFoldParams.v1
```

(the command should be all on one line with a space between ".seq" and the first hyphen). You should see this as your output:



```
~/EternaFold$ ./src/contrafold predict test.seq --params parameters/EternaFoldParams.v1
Training mode:
Use constraints: 0
Use evidence: 0
Predicting using MEA estimator.
>test.seq
CGCUGUCUGUACUUGUAUCAGUACACUGACGAGUCCCUAAAGGACGAAACAGCG
>structure
(((((((((((((......))))))..)....((((.....))))...))))))
```

iv. Now, you can close this terminal

(c) Then install Arnie

i. First, ensure that you have a tmp folder in your "`/home/user1`" directory, where "user1" is your computer username. If you do not have this directory, go ahead and create one.

ii. Download Arnie from Github. I just downloaded the code as a zip file, unzipped it, renamed the folder to "Arnie", then moved the Arnie folder to my "`/home/user1/`" directory. I do not think the EternaFold and Arnie folders need to be in the same parent directory. I just chose to do this in an effort to reduce potential complications.

iii. Go into your Arnie folder and create a file called "arnie.txt".

iv. Open this file and enter the following:

```
eternafold: /home/user1/EternaFold/src
TMP: /home/user1/tmp
```

- This will point the Arnie program to EternaFold and the temp folder.

v. Open your ".bashrc" file and add these lines to the very bottom of the document:

```
export PYTHONPATH=$PYTHONPATH:/home/user1
export PYTHONPATH=$PYTHONPATH:/home/user1/arnie
export ARNIEFILE=/home/user1/arnie/arnie.txt
```

vi. Save your ".bashrc" document and close it.

vii. You should now be able to confirm if Arnie is set up correctly. Open up a terminal (where you open the terminal should not matter). Type the following:

```
python
>>> from arnie.mfe import mfe
>>> sequence = "CGCUGUCUGUACUUGUAUCAGUACACUGACGAGUCCCU
AAAGGACGAAACAGCG"
>>> mfe_structure = mfe(sequence,package='eternafold')
>>> print(mfe_structure)
```

(the sequence should all be on one line)

viii. Your output should look like this:

```
((((((((((((......))))))..)....((((.....))))...))))))
```

ix. If you received an error, then you may have made a mistake in one of the previous steps or something is misspelled (the python commands, the folder names, the arnie.txt file name, the lines inside the arnie.txt file, etc.)

Appendix B

Rosetta File Settings

Flag File

```
-fasta fasta.txt
-secstruct_file secstruct.txt
-constraints:cst_file constraint.cst
-6vxx_rbd.pdb
-nstruct 5
-cycles 1000
-docking_move_size 1.0
-out:file:silent /dev/null
-new_fold_tree_initializer true
-minimize_rna false
-rna:denovo:lores_scorefxn rna/denovo/rna_lores_with_rnp_aug.wts
-rna_protein_docking true
-ignore_zero_occupancy false
-convert_protein_CEN false
-FA_low_res_rnp_scoring true
-ramp_rnp_vdw true
-mute protocols.moves.RigidBodyMover
-no_filters
-bps_moves false
```

Fasta File

```
>6vxx_ABC_covid_align5.pdb  A:1-266 B:267-557 C:558-828
LPFNDGVIFGTTLDSKTQSLLIIKVCEFNCTFEYVFKNIDGYFKIYLVDLPIGINITRFESIVRFPNITNCPF
GEVFNATRFASVYAWNRKRISCVADYSVLYNSASFSTFKCYGVSPTKLNDLFTVYADSFVIRGDEVRQIAPGQ
TGKIADYNYKLPDDFTCVIAWNSNNLDSKGNYNYLYRKPFERDIYFPLQSYGFQPTNVGYQPYRVVVLSFELL
PATVCGPKKSTNLNNFNGLTPQRDPQTLEEAISDILSRLDPPEAEVQKDLPFNDGVYWIFGTTLDSKTQSLLI
VVIKVCEFQNCTFEYSFVFKNIDGYFKIYSPLVDLPIGINITRFGYLQESIVRFPNITNLCPFGEVFNATRFA
SVYAWNRKRISNCVADYSVLYNSASFSTFKCYGVSPTKLNDLCFTNVYADSFVIRGDEVRQIAPGQTGKIADY
NYKLPDDFTGCVIAWNSNNLDSKGNYNYLYRKPFERDIYFPLQSYGFQPTNVGYQPYRVVVLSFELLHAPATV
CGPKKSTNLVNNFNGLTGPFQRDPQTLEEAINDILSRLDPPEAEVQKPFNDGVIFGTTLDSKTQSLLIVIKVC
EFQNCTFEYVSVFKNIDGYFKIYSPLVDLPIGINITRFGLSVRFPNITNLCPFGEVFNATRFASVYAWNRKRI
SNCVADYSVLYNSASFSTFKCYGVSPTKLNDLCFTNVYADSFVIRGDEVRQIAPGQTKIADYNYKLPDDFTGC
VIAWNSNNLDSKGNYNYLYRKPFERDIYFPLQSYGFQPTNVGYQPYRVVVLSFELLHAPATVCGPKKSTNNNG
LPQTTEAISNDILSRLDPPEAEVQI
```

Secondary Structure File

```
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................................................................
.........................

LPFNDGVIFGTTLDSKTQSLLIIKVCEFNCTFEYVFKNIDGYFKIYLVDLPIGINITRFESIVRFPNITNCPF
GEVFNATRFASVYAWNRKRISCVADYSVLYNSASFSTFKCYGVSPTKLNDLFTVYADSFVIRGDEVRQIAPGQ
TGKIADYNYKLPDDFTCVIAWNSNNLDSKGNYNYLYRKPFERDIYFPLQSYGFQPTNVGYQPYRVVVLSFELL
PATVCGPKKSTNLNNFNGLTPQRDPQTLEEAISDILSRLDPPEAEVQKDLPFNDGVYWIFGTTLDSKTQSLLI
VVIKVCEFQNCTFEYSFVFKNIDGYFKIYSPLVDLPIGINITRFGYLQESIVRFPNITNLCPFGEVFNATRFA
SVYAWNRKRISNCVADYSVLYNSASFSTFKCYGVSPTKLNDLCFTNVYADSFVIRGDEVRQIAPGQTGKIADY
NYKLPDDFTGCVIAWNSNNLDSKGNYNYLYRKPFERDIYFPLQSYGFQPTNVGYQPYRVVVLSFELLHAPATV
CGPKKSTNLVNNFNGLTGPFQRDPQTLEEAINDILSRLDPPEAEVQKPFNDGVIFGTTLDSKTQSLLIVIKVC
EFQNCTFEYVSVFKNIDGYFKIYSPLVDLPIGINITRFGLSVRFPNITNLCPFGEVFNATRFASVYAWNRKRI
SNCVADYSVLYNSASFSTFKCYGVSPTKLNDLCFTNVYADSFVIRGDEVRQIAPGQTKIADYNYKLPDDFTGC
VIAWNSNNLDSKGNYNYLYRKPFERDIYFPLQSYGFQPTNVGYQPYRVVVLSFELLHAPATVCGPKKSTNNNG
LPQTTEAISNDILSRLDPPEAEVQI
```

Constraint File

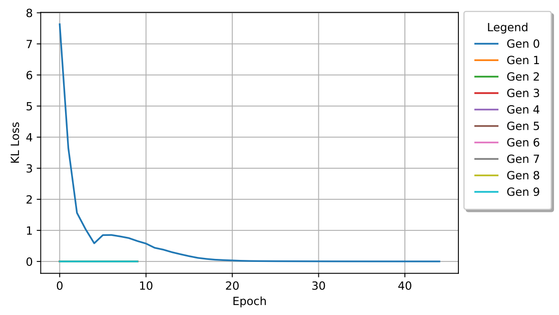
```
AtomPair CA 201A C5 xxx FLAT_HARMONIC 0 0.125 1
```

Note: the “xxx” represents some residue number of your chosen RNA plus the max residue number of the chosen protein. There can be fewer or more than 3 digits. It all depends on the conditions of your protein sequence and the RNA residue chosen.

Appendix C

Additional Experimental Analysis

Run 2 Results



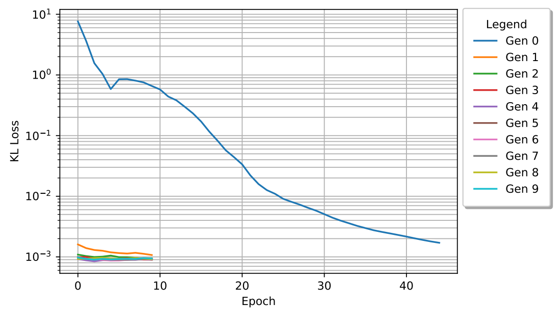
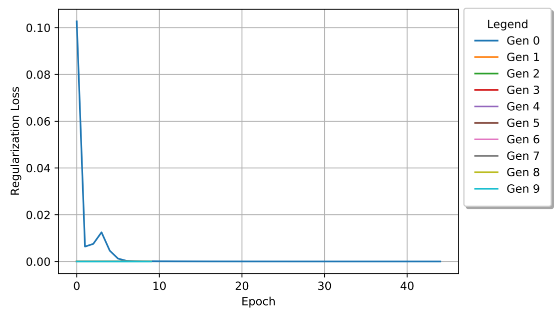


Figure C.1: The KL loss values per epoch for each generation plotted over a linear scale (left) and a logarithmic scale (right).



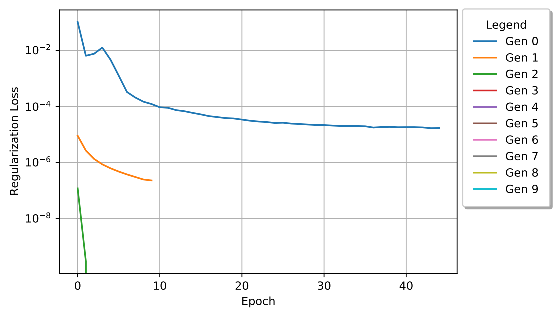


Figure C.2: The regularization BCE loss values per epoch for each generation plotted over a linear scale (left) and a logarithmic scale (right).

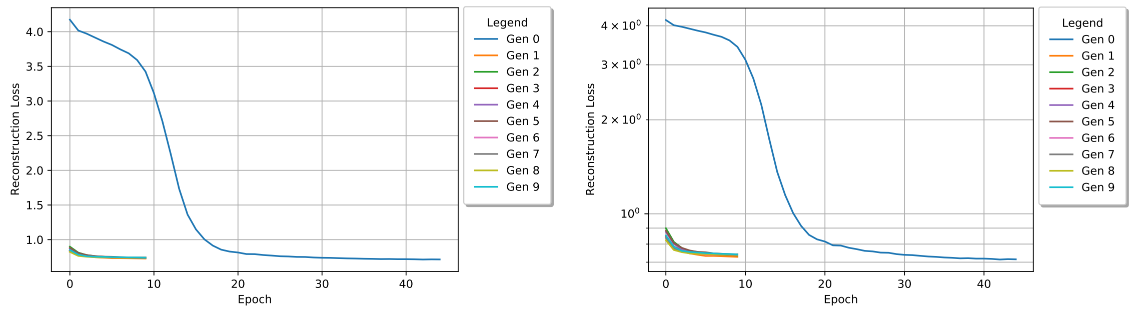


Figure C.3: The reconstruction loss values per epoch for each generation plotted over a linear scale (left) and a logarithmic scale (right).

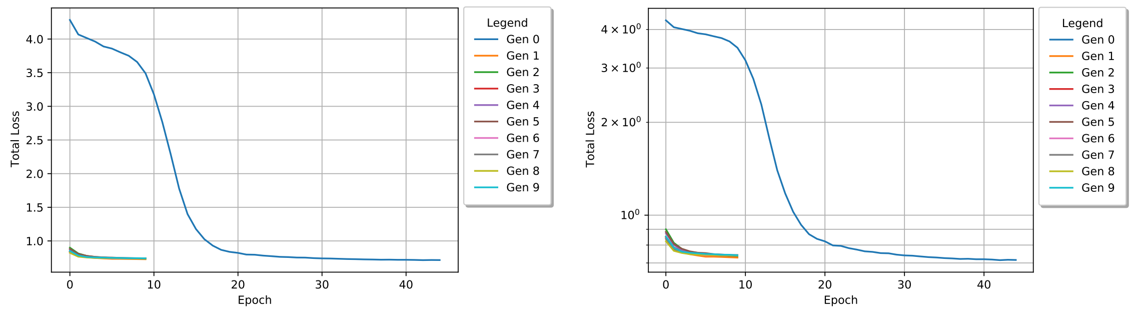


Figure C.4: The total loss values per epoch for each generation plotted over a linear scale (left) and a logarithmic scale (right). Note: total loss values include the KL weight factor over time.

Table C.1: The score performance in the first and last generation for each experiment.

		DAPTEV	**GA**
First Gen	*Best Score*	98.205	128.304
	Mean Score	605.057	1023.306
	Median Score	635.575	1047.007
	Worst Score	836.791	1447.986
Last Gen	*Best Score*	27.941	61.949
	Mean Score	510.197	327.772
	Median Score	534.332	337.751
	Worst Score	687.297	439.602

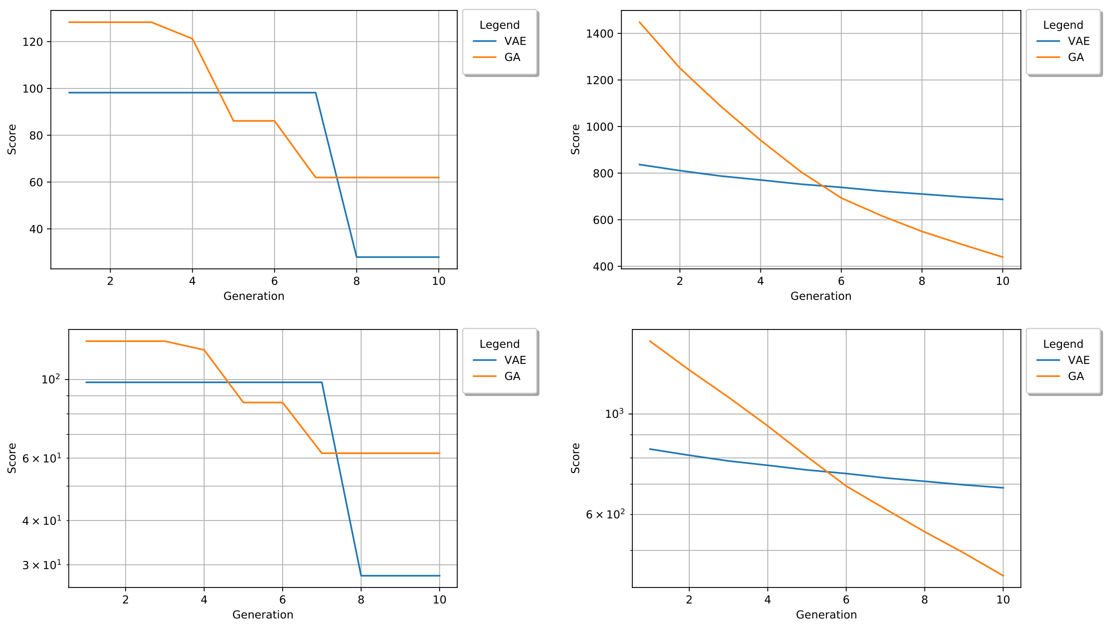


Figure C.5: The DAPTEV Model and the GA experiment's scores plotted per generation on the same linearly scaled graph (top) and logarithmically scaled graph (bottom). The left graphs show the best score per generation while the right graphs show the worst score.

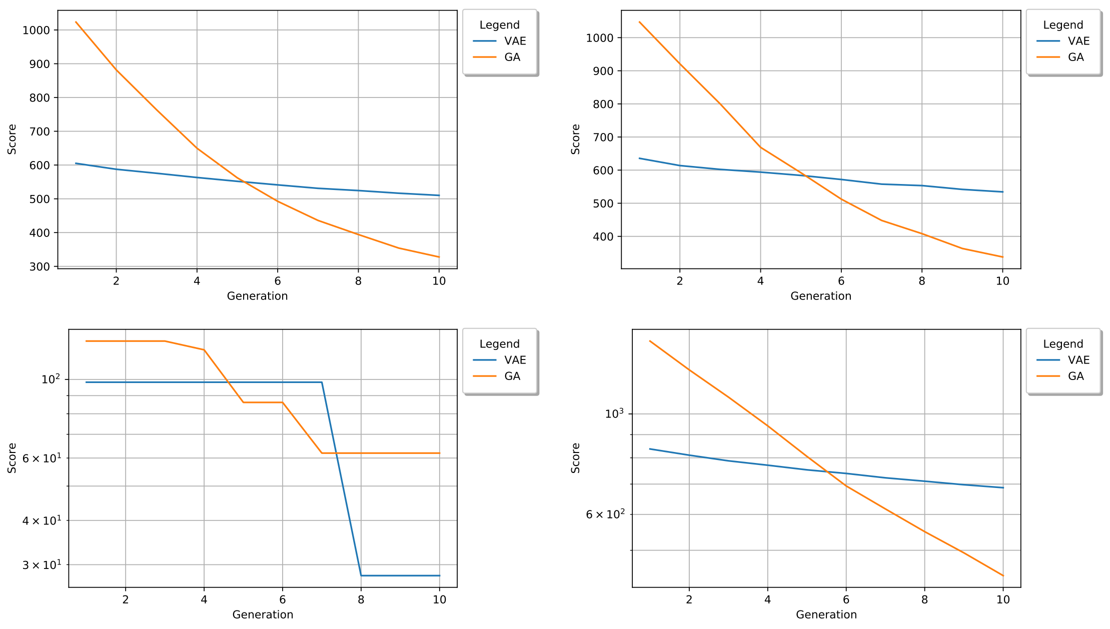


Figure C.6: The DAPTEV Model and the GA experiment's scores plotted per generation on the same linearly scaled graph (top) and logarithmically scaled graph (bottom). The left graphs show the mean score per generation while the right graphs show the median score.

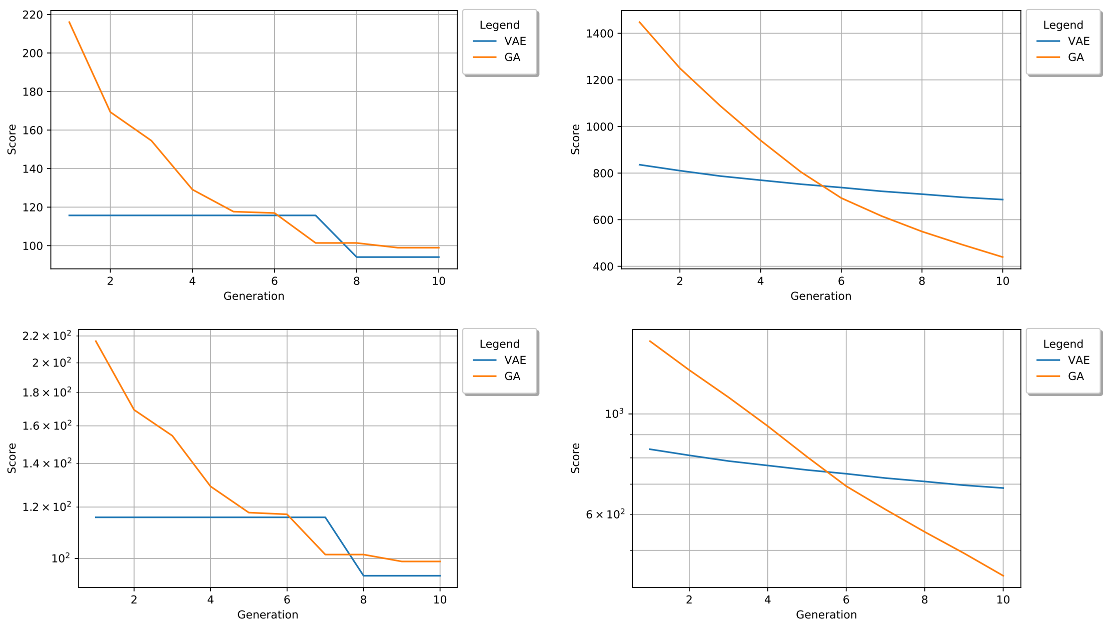


Figure C.7: The DAPTEV Model and the GA experiment's scores plotted per generation on the same linearly scaled graph (top) and logarithmically scaled graph (bottom). The left graphs show the mean score for the 5 best performing sequences per generation while the right graphs show the mean score for the 5 worst performing sequences.

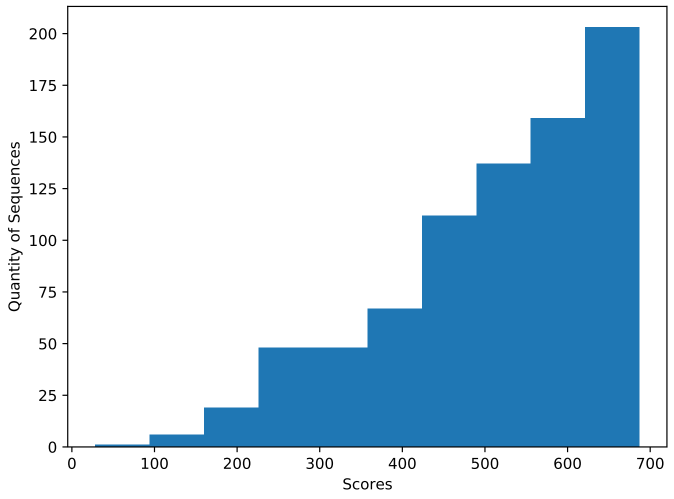
The novelty values calculated for the DAPTEV Model and the GA's last generation was 0.9013 and 0.9987 respectively. Similarly, the diversity values were 0.71 and 0.98 respectively.

Table C.2: Information relating to the folded RNAs in the VAE and the GA models out of 800 total produced sequences. Note: all best, mean, median, and worst values relate only to the folded RNAs in the last generation, not the entire last generation's output.

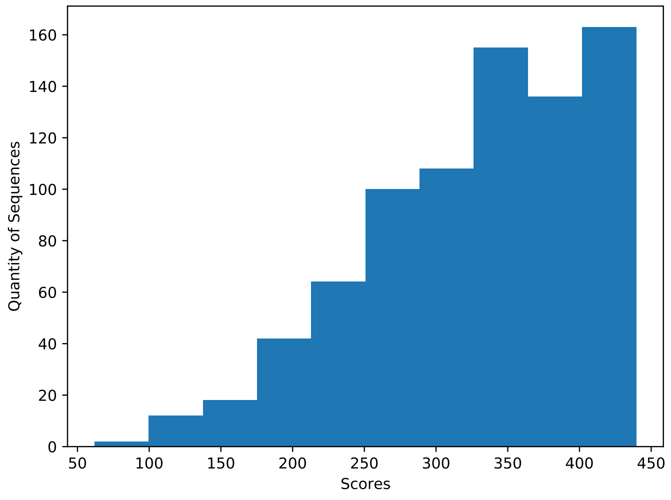
			Scores				*Base Pairs*			
	Folded RNAs	*Rate (%)*	*Best*	*Mean*	*Med.*	*Worst*	*Min.*	*Mean*	*Med.*	*Max.*
VAE	225	28	128	551	583	687	1	2.2	2	5
GA	58	7	62	342	343	440	1	2	2	3

Table C.3: Statistical Analysis of the Last Generation Scores for the VAE and the GA Models.

Tests	**P-Values**	
	VAE	***GA***
Shapiro-Wilk	1.3796984876472367e-18	6.985057199532588e-15
Kruskal-Wallis	1.6920869583651724e-141	
	VAE, GA	
Mann-Whitney U	8.472054368705904e-142	



(a) VAE performance histogram.



(b) GA performance histogram.

Figure C.8: Histograms of the VAE and the GA score results for the last generation.

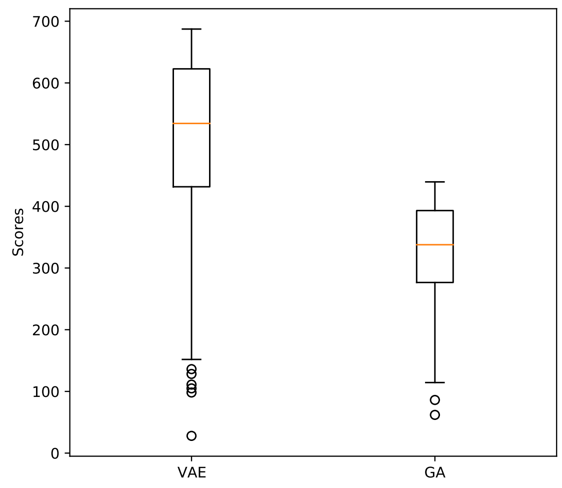


Figure C.9: Box plots of the VAE and the GA score results for the last generation on the same scale.

Run 3 Results

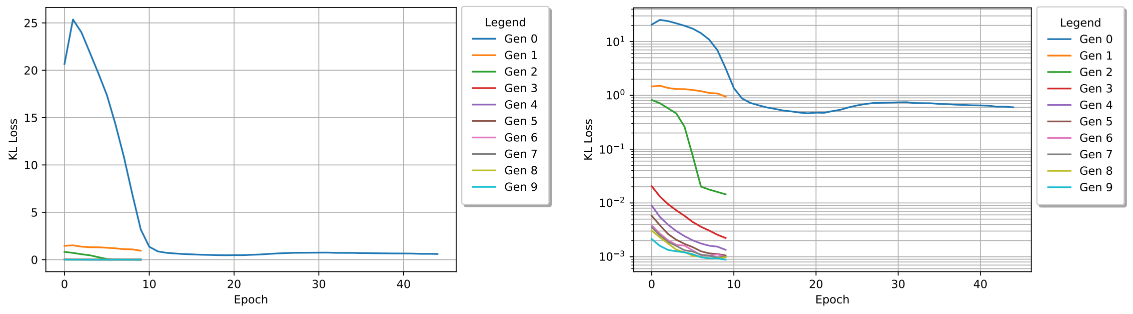


Figure C.10: The KL loss values per epoch for each generation plotted over a linear scale (left) and a logarithmic scale (right).

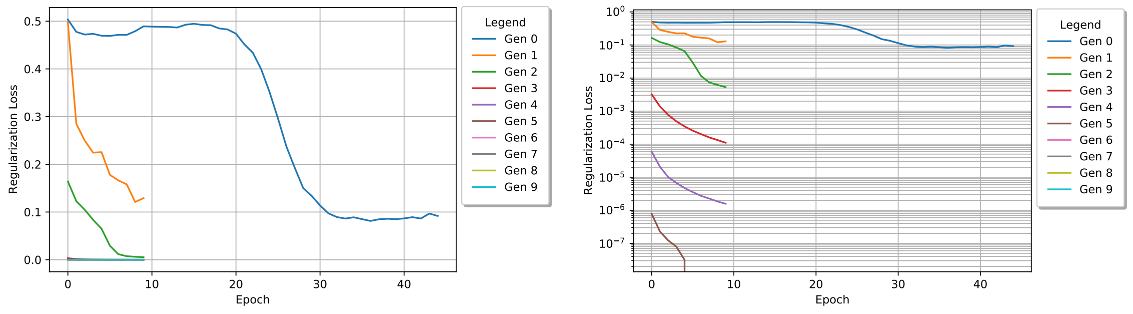


Figure C.11: The regularization BCE loss values per epoch for each generation plotted over a linear scale (left) and a logarithmic scale (right).

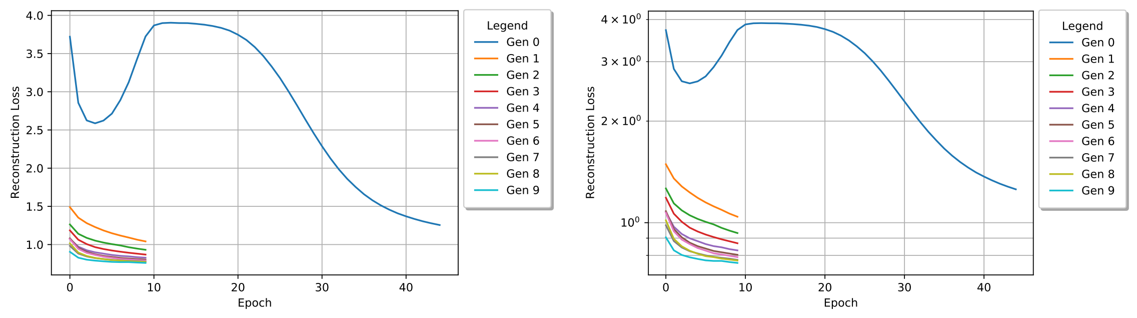


Figure C.12: The reconstruction loss values per epoch for each generation plotted over a linear scale (left) and a logarithmic scale (right).

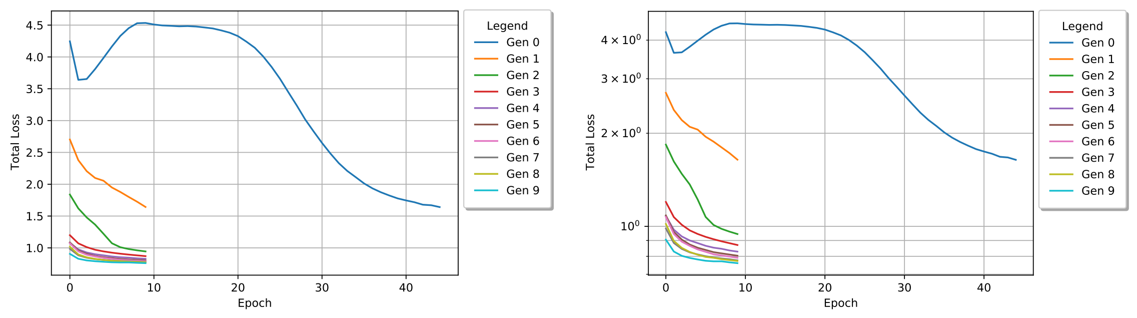


Figure C.13: The total loss values per epoch for each generation plotted over a linear scale (left) and a logarithmic scale (right). Note: total loss values include the KL weight factor over time.

Table C.4: The score performance in the first and last generation for each experiment.

		DAPTEV	**GA**
First Gen	*Best Score*	128.304	42.334
	Mean Score	1,039.674	304.662
	Median Score	1,067.072	317.001
	Worst Score	1,461.798	406.574
Last Gen	*Best Score*	89.413	30.383
	Mean Score	606.588	210.427
	Median Score	624.657	221.048
	Worst Score	843.746	269.184

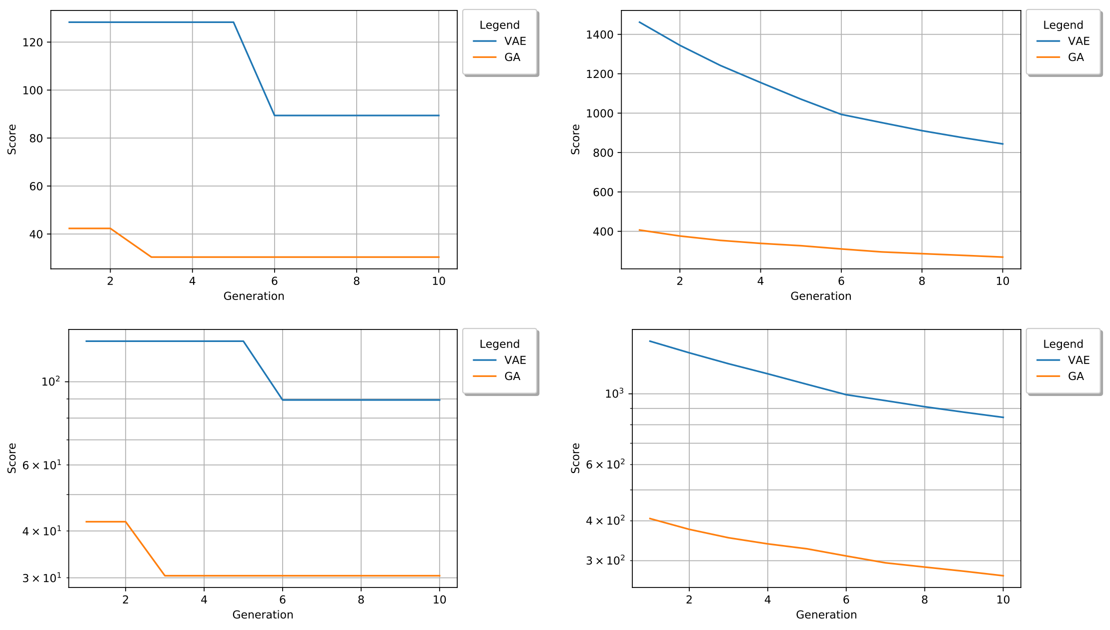


Figure C.14: The DAPTEV Model and the GA experiment's scores plotted per generation on the same linearly scaled graph (top) and logarithmically scaled graph (bottom). The left graphs show the best score per generation while the right graphs show the worst score.

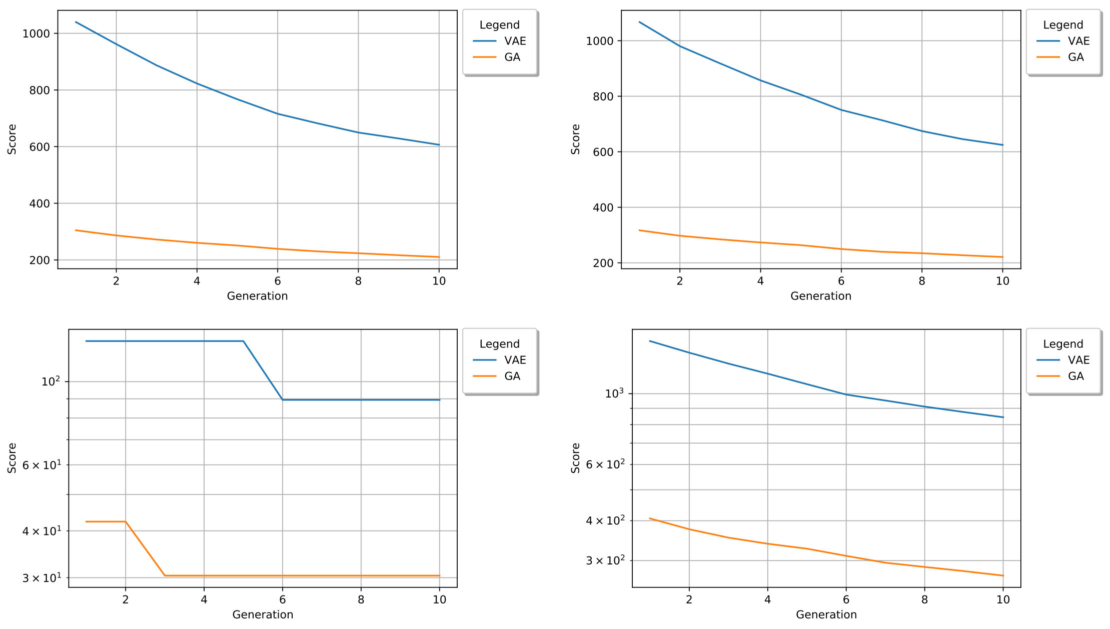


Figure C.15: The DAPTEV Model and the GA experiment's scores plotted per generation on the same linearly scaled graph (top) and logarithmically scaled graph (bottom). The left graphs show the mean score per generation while the right graphs show the median score.

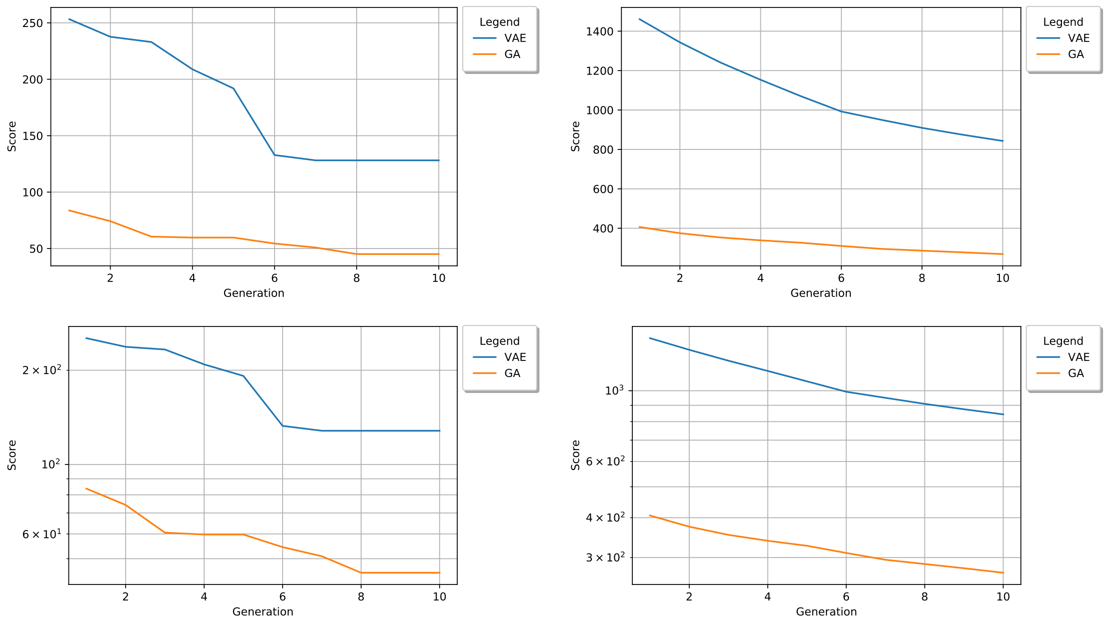


Figure C.16: The DAPTEV Model and the GA experiment's scores plotted per generation on the same linearly scaled graph (top) and logarithmically scaled graph (bottom). The left graphs show the mean score for the 5 best performing sequences per generation while the right graphs show the mean score for the 5 worst performing sequences.

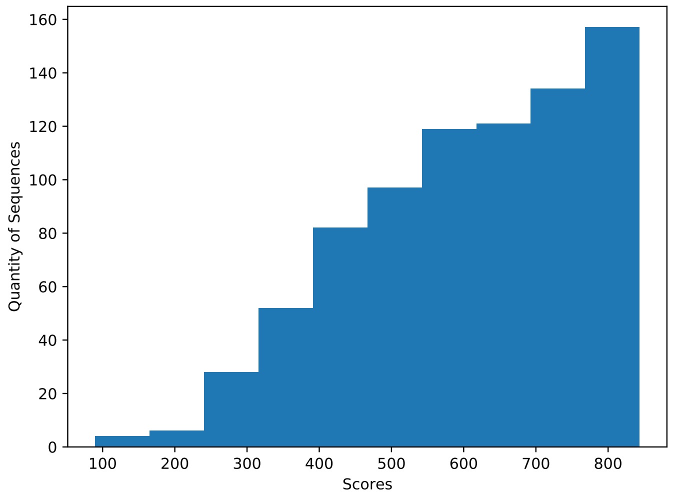
The novelty values calculated for the DAPTEV Model and the GA's last generation was 0.83 and 0.9987 respectively. Similarly, the diversity values were 0.8225 and 0.9887 respectively.

Table C.5: Information relating to the folded RNAs in the VAE and the GA models out of 800 total produced sequences. Note: all best, mean, median, and worst values relate only to the folded RNAs in the last generation, not the entire last generation's output.

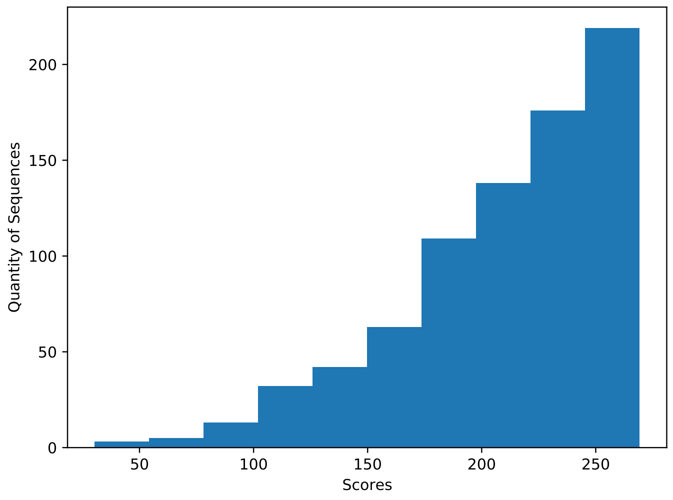
			Scores				*Base Pairs*			
	Folded RNAs	*Rate (%)*	*Best*	*Mean*	*Med.*	*Worst*	*Min.*	*Mean*	*Med.*	*Max.*
VAE	280	35	128	674	713	844	1	2.4	2	5
GA	18	2	128	229	237	264	1	2.2	2	4

Table C.6: Statistical Analysis of the Last Generation Scores for the VAE and the GA Models.

Tests	**P-Values**	
	VAE	***GA***
Shapiro-Wilk	3.488029499563658e-14	3.983942552538331e-20
Kruskal-Wallis	8.669714305334742e-251	
	VAE, GA	
Mann-Whitney U	4.34280509441259e-251	



(a) VAE performance histogram.



(b) GA performance histogram.

Figure C.17: Histograms of the VAE and the GA score results for the last generation.

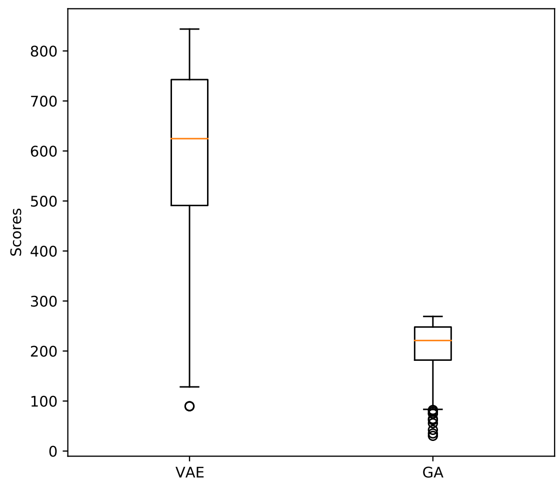


Figure C.18: Box plots of the VAE and the GA score results for the last generation on the same scale.