Protein Language Representation Learning to predict SARS-CoV-2 Mutational Landscape



Hunar Batra

Department Of Computer Science

University of Oxford

A Dissertation submitted for the degree of
Master of Science in Advanced Computer Science
under the supervision of *Prof. Peter Minary*

Michaelmas Term 2022

Abstract

With the proliferation of SARS-CoV-2 pandemic globally, numerous variants have been emerging on a daily basis containing distinct transmission and infection rates, risks and impact over evasion of antibody neutralisation. Early discovery of high-risk mutations is critical towards undertaking data-informed therapeutic design decisions and effective pandemic management. This dissertation explores the application of Language Models, commonly used for textual processing, to decipher SARS-CoV-2 spike protein sequences which are an amalgamation of amino acids represented as alphabets. Deep protein language models are revolutionising protein biology, and with the introduction of two novel models: transformer encoder-based sequence only *CoVBERT* for predicting point mutations, and *MuFormer* which leverages the sequence and structural space to design mutational protein sequences iteratively. CoVBERT has been able to predict highly transmissible mutations including $D614G$ with a masked marginal log likelihood of 0.95, surpassing state-of-the-art large protein language models. This reflects over large language models ability to encapture in vitro mutagenesis by learning the language of evolution.

MuFormer is capable of generating *de novo* protein sequences using AlphaFold2 for fixed backbone design, and curates evolutionary novel mutational sequences by injecting the representation derived state-of-the-art protein language models. The generated mutational sequences have been validated with historical data which exemplified the ability of MuFormer to capture phylogenetic properties for generating mutations such as Omicron and Delta variant, given Alpha variant as the input. MuFormer conditions not only over the sequence, but also the structure to generate end-to-end protein sequences and structure by optimising using two strategies of fixed backbone design (MuFormer-fixbb) and backbone atom optimisation (MuFormer-bba). Both these variants of MuFormer outperformed AlphaFold2 over the mutational sequence generation task for several structure and sequence likelihood metrics. These models ascertain over the potential of large language models, termed as foundational models, towards learning the representational language of biology which can assist in controlling pandemics by predicting mutations with higher infectivity in advance.

Contents

List of Figures **vi**

List of Tables **viii**

1 Overview **1**

1.1 Motivation . . . 1
1.2 Contribution . . . 1
1.3 Thesis Overview . . . 2

2 Related Work **4**

3 Background **7**

3.1 Neural Language Models . . . 7
3.1.1 Feedforward Neural Language Models . . . 8
3.1.2 Word Embeddings . . . 10
3.1.3 Recurrent Neural Networks . . . 11
3.1.4 Gated RNNs . . . 13
3.1.4.1 Long Short-Term Memory . . . 13
3.1.4.2 Gated Recurrent Units . . . 15
3.1.5 Seq2Seq Models . . . 16
3.1.6 Attention Mechanism . . . 17
3.1.7 Masked Autoencoders . . . 18
3.1.8 Autoregressive Decoders . . . 18
3.2 Protein Language Modelling . . . 19
3.2.1 Proteins and Natural Language . . . 19
3.2.2 Protein Engineering . . . 21
3.2.3 Protein Embeddings . . . 22
3.2.3.1 Proteins Tokenization . . . 22
3.2.3.2 Bag of Words Representation . . . 22
3.2.3.3 Word Embeddings . . . 23

I Large Protein Language Models are Zero-shot Mutagenesis Learners **24**

4 CoVBERT - Mutation Prediction Language Model **25**
4.1 Introduction . . . 25
4.2 Proposed Model Overview . . . 27
4.3 Model Architecture . . . 29
4.4 Dataset Preprocessing . . . 34
4.5 Tokenizer Training . . . 35
4.6 Positional Embeddings . . . 37
4.7 Masking Objective . . . 40
4.8 Self-supervised training . . . 41

5 CoVBERT - Experimental Results **45**
5.1 Zero-shot SARS-CoV-2 mutation prediction . . . 45
5.1.1 Baseline Protein Language Models . . . 46
5.1.1.1 ProtBERT-BFD . . . 47
5.1.1.2 DistilProtBERT . . . 48
5.1.1.3 ProtElectra . . . 48
5.1.2 Results . . . 49
5.2 Interpreting Self-Attention in CoVBERT . . . 51

II Large Protein Language Models are Self-supervised Evolutionary Learners **58**

6 MuFormer - Structure and Sequence based Mutation Generation Model **59**
6.1 Introduction . . . 59
6.2 MuFormer - Proposed Model Architecture . . . 61
6.2.1 Inductive Bias Encoder . . . 61
6.2.1.1 Input Embeddings . . . 61
6.2.1.2 MuFormer1 Encoder . . . 64
6.2.1.3 MuFormer2 Encoder . . . 64
6.2.1.4 Pre-training objective . . . 64
6.2.1.5 Injecting Inductive Bias for Sequence Design . . . 67
6.2.2 Fixed Backbone Sequence Design . . . 69
6.2.2.1 Structure Prediction Oracle $\sim$ AlphaFold2 . . . 71
6.3 Training Objective . . . 72
6.3.1 FixBB Optimisation . . . 74
6.3.1.1 Masked MSA Prediction Loss . . . 75

6.3.1.2 Distogram CCE Prediction Loss 75
6.3.1.3 pLDDT Prediction Loss 76
6.3.1.4 PAE Prediction Loss 76
6.3.1.5 Root mean square distance loss 77
6.3.2 Backbone Atom Optimisation 77

7 Experimental Results **78**
7.1 Fitness Prediction 79
7.2 Structure likelihood Prediction 82
7.3 In vitro Mutagenesis Generation 84

8 Conclusion **90**
8.1 Discussion and Future Work 90

Appendices
.1 AdamW Optimisation Algorithm 93

References **94**

List of Figures

3.1 Flow of MLM applied over natural language texts, taken from [57] 18
3.2 Comparison between Alignment-based and Autoregressive modelling, taken from [61] 19
3.3 Contrast between language processing tasks performed over text and protein sequences, taken from [57] 20
3.4 Tokenization performed over text vs. protein sequences, taken from [57] . . 22
3.5 Bag of Words (BoW) representation for text vs. protein sequences, taken from [57] 23

4.1 Proposed model architecture - CoVBERT 30
4.2 Self-attention scaled dot product computation 31
4.3 Input Embeddings 38
4.4 CoVBERTs Sinusoidal Positional Embeddings 39
4.5 CoVBERT Training Loss Curve 44

5.1 Point Mutation Prediction Log Likelihood 51
5.2 CoVBERT Layer-wise Attention head visualisation 53
5.3 CoVBERT Attention head (mean) visualisation for layers 1-3 54
5.4 CoVBERT Attention head (mean) visualisation for layers 4-6 54
5.5 (a) Penultimate layer attention head visualisation for token L (b) Final layer attention head visualisation for token S 55
5.6 (a) CoVBERT Attention head interpretation for Layer 4, head 5 (b) CoVBERT Attention head interpretation for Layer 2, head 9 56
5.7 CoVBERT model Attention interpretation for protein structure 57

6.1 Proposed architecture for MuFormer de novo sequence designing model . . 62
6.2 ESM-1b Inductive Bias for SARS-CoV-2 Alpha Variant (B.1.1.7) Sequence (PDB: 7LWV) 68
6.3 ESM-2 Inductive Bias for SARS-CoV-2 Alpha Variant (B.1.1.7) Sequence (PDB: 7LWV) 69
6.4 AlphaFold2 model architecture 72

7.1 Sequence Alignment Chart for MuFormer2-fixbb designed protein sequences for SARS-CoV-2 Alpha Vairant (B.1.1.7), PDB: 7LWV 79

7.2 Log Likelihood and Perplexity for MuFormer models and AlphaFold2 Baseline 82
7.3 Sequence Recovery and Pairwise Alignment Score for MuFormer models and AlphaFold2 Baseline 82
7.4 Structural likelihood metrics vs epochs- (a) plDDT, (b) PAE, (c) RMSD, and (d) Distogram loss for MuFormer models and AlphaFold2 Baseline 84
7.5 Structural likelihood metrics- (a) plDDT, (b) PAE, (c) RMSD, and (d) Distogram loss for MuFormer models and AlphaFold2 Baseline 85
7.6 Alpha Variant Mutational Protein with (a) MuFormer's Fixed Backbone design (b) MuFormer's Backbone atom design (c) AlphaFold2 86
7.7 SARS-CoV-2 closest genomic lineage detected for sequences designed by (a) MuFormer2-fixbb and (b) MuFormer2-bba 88
7.8 SARS-CoV-2 closest genomic lineage detected for sequences designed by (a) MuFormer1-fixbb and (b) MuFormer1-bba 89
7.9 SARS-CoV-2 closest genomic lineage detected for the sequence designed by CoVBERT-MuFormer (with fixbb optimisation) 89

List of Tables

4.1 Proposed CoVBERT model hyperparameters 43

5.1 Proposed CoVBERT model and pre-trained state-of-the-art baseline models hyperparameters 47

List of Algorithms

1 CoVBERT($x|\theta$): encoder-only transformer, forward pass [Adapted from [110]] 34
2 Token embeddings [110] 36
3 Byte-pair encoding [104] 37
4 Positional embeddings [110] 38
5 Training: Masked Language Modelling [110] 43
6 MuFormer Encoder forward pass 67
7 AdamW algorithm with decoupled weight decay 93

1 Overview

1.1 Motivation

Protein–protein interactions between the SARS-CoV-2 spike protein, human receptors, and antibodies are important factors in the virus's virulence and ability to escape the human immune system. SARS-CoV-2 has evolved into a number of mutations, each with a different level of virulence. The Delta (B.1.617.2) mutation, which has a high transmissibility, resulted in higher mortality [1] [2] and the highly mutated Omicron (B.1.1.529) variant proliferated quickly resulting in a third wave in numerous countries [3]. The ability of coronaviruses to mutate and the possibility of riskier mutations to emerge, such as the recent Deltacron variant [3], calls for the need to leverage SARS-CoV-2 proteomic data for anticipating viral features and future alterations to considerably improve disease control, prevention and drug development. It is critical to understand the patterns of emerging VOCs and to predict alterations that may emanate in gain of function or immunological evasion.

1.2 Contribution

This work introduces two novel state of the art protein language models namely, CoVBERT and MuFormer which are capable of generating sequences with mutations for SARS-CoV-2 specifically, given the context. CoVBERT, which is a sequence based protein language model generates point mutations, and has been able to predict the most transmissible mutations like *D614G* and *E484K* which were responsible for the outbreak of SARS-CoV-2 variants like Alpha and Delta. For the task of zero shot prediction of point mutations, CoVBERT outperformed other state-of-the-art protein language models. This sheds light over the ability of language

models to learn the language of evolution for proteins and their capability of predicting transmissible mutations, which can assist therapeutic design decisions.

The MuFormer model, which leverages both sequence and the protein structure to design sequences in an iterative setting, has been able to design de novo protein sequences for SARS-CoV-2 Delta and Omicron variants, given the Alpha variant protein as input. This model introduces a novel architecture of combining encoded representations from large pre-trained protein language models with AlphaFold2 [4] for the task of fixed backbone sequence design. A new optimisation objective function called backbone atom loss has been introduced here which optimises the sequence to adapt to the protein's structural space. The mutational sequence generation capability of MuFormer which conditions over both the protein structure and the sequence, highlights the ability of transformer based models to explore the evolutionary landscape of protein sequences.

Till now, various researchers have conducted studies about mutational fitness prediction of SARS-CoV-2 sequences [5] that have been deposited to large genome repositories like Global Initiative for Sharing All Influenza database (GISAID [6]). This dissertation marks the first work to introduce two models: CoVBERT and MuFormer capable of performing point and sequence level in vitro mutagenesis for SARS-CoV-2, by learning the language of evolution derived from the masked language modelling based protein language models and the MSA profile generated by AlphaFold2 in case of MuFormer.

1.3 Thesis Overview

The dissertation is divided into two main components where the first part introduces the sequence based CoVBERT protein language model for generation of point mutations in SARS-CoV-2 spike glycoprotein sequences. The second part introduces the sequence and structure -based MuFormer suite of models which performs fixed backbone sequence design for generation of mutational sequences for SARS-CoV-2.
Section 2 reflects over related work in the space of protein language models, their mutational sequence fitness prediction, and fixed backbone sequence design regime, followed by *Section 3* which provides concise background material.
Section 4 introduces CoVBERT mutation prediction model including the proposed model architecture, data preprocessing step, training of the tokenizer used for CoVBERT, positional embeddings, masking objective and models training.
Section 5 showcases the experimental results of CoVBERT for the task of Zero-shot SARS-CoV-2 point mutation prediction and its comparison to other state of the art protein language

models. Interpretation of self-attention mechanism in the CoVBERT model is also discussed in this section.

Section 6 introduces the MuFormer model including its architecture consisting of the inductive bias encoder, and AlphaFold2 for fixed backbone sequence design, and the training objectives used along with the newly introduced backbone atom loss function.

Section 7 discusses the experimental results of MuFormer suite of models for their ability to generate *de novo* protein mutations for SARS-CoV-2, along with their fitness prediction and structure likelihood.

Section 8 provides a conclusion for both the newly introduced protein language models for generating mutations.

2
Related Work

Sequence based Protein Language Models The use of language models for sequence-only protein generation as well as representation learning has recently been the subject of extensive research [7], [8] [9] [10]. Alley et al. [11] and Heinzinger et al. [12] show how an LSTM-based [13] autoregressive protein language model trained on a sizable protein sequence dataset such as UniRef 50 [14] may be used for a number of downstream tasks, such as stability, fluorescence, and secondary structure prediction. This is improved by ESM-1b introduced by Rives et al. [15], TAPE by Rao et al. [16], ProtTrans suite of models by Elnaggar et al. [17], and ProteinBERT by Brandes et al. [18] which utilises the transformer based model [19], swapping out the autoregressive pretraining task for a bidirectional denoising task, and enlarging the model and dataset sizes. By enabling protein language models to take into account multiple sequence alignments, Rao et al. significantly increased their functionality [20]. DeepSequence [21] and ESM-1v [22] have shown that these representations can be used to predict variant effects, and RGN2 [23] has recently demonstrated their utility for quick and accurate single sequence tertiary structure prediction. UniRep [11] has been used successfully for protein engineering, and the more current transformer-based ProGen [7] can generate proteins with a range of desirable properties by conditioning the model on a variety of sequence metadata.

Self-supervised learning on large datasets of unlabeled protein sequences has proven to be quite useful in a variety of protein modelling tasks [15] [17] [8]. Embeddings from transformer encoder models trained for masked language modelling have been used for variant prediction, evolutionary analysis and as features for protein structure prediction [24] [25] [15]. It has proven possible to produce functional proteins only by learning from sequences alone using transformer based models [26].

Mutation Effect Prediction To predict mutations that could culminate in viral escape, Hie et. al, applied protein language modelling to gauge the fitness of surface protein mutants for influenza A hemagglutinin, HIV-1 envelope glycoprotein, and SARS-CoV-2 spike glycoprotein [27]. Unsupervised language models allow mutational effects and secondary structure prediction which has improved long-range contact prediction [15] [16] [28], binding site targeting [28], and the capture of crucial biophysical features influencing protein shape [28] [17]. Schwaller et al. [29] demonstrated that language models trained using masked language modelling (MLM) on organic chemical reaction data, represented using SMILES [30], can leverage attention to accurately map atoms between precursors and products with no supervision for small molecules.

Fixed Backbone Sequence Design The protein backbone was first conceptualised as a proximity graph that could be encoded and decoded using a graph transformer model [31]. This method outperformed the well-known Rosetta protein design software [32] in terms of sequence recovery (the percentage of created residues that match the original protein sequence), while also requiring orders of magnitude less computing power. Inspired by this research, ESM-IF1 switched the encoder network for a rotation-equivariant model and scaled the network to 142M parameters [33]. As a result, the sequence recovery performance continued to improve. Finally, ProteinMPNN [34] used a novel order-agnostic decoding method to decode sequences after encoding structures using a rotation-invariant message passing network. The sequences produced by ProteinMPNN significantly enhanced soluble expression and folded into the expected structures, according to extensive experimental validation [35].

Shin et al. demonstrated that protein-specific autoregressive sequence models trained using related proteins can anticipate point mutation, indel effects and functional nanobodies design [8]. Protein-specific autoregressive models for sequence creation were also investigated by Trinquier et al [36]. Meier et al. discovered that big protein language models' log-likelihoods predict mutational effects [22]. Madani et al. investigated a functional annotation-conditioned autoregressive sequence model and demonstrated its ability to construct functional proteins [7]. AntiBERTa [37], a 12-layer transformer model pre-trained over 57M human BCR sequences can predict an antibody's binding site. Hie et al. recently employed language models to learn about the evolutionary fitness of viral envelope proteins and were able to identify variants that could allow the SARS-CoV-2 spike protein to evade neutralising antibodies [38] [27]. The RITA family of models showed that scaling model capacity can further enhance performance in addition to the effectiveness of autoregressive PLMs in estimating protein fitness [10].

Tranception showed that adding retrieval capabilities to autoregressive language models can improve a generalist model by adding family-specific data from MSAs at inference [39]. Thus, Language models are proving to be an excellent technique for learning elusive signals from protein and molecular language representations [40].

For the rational design of proteins, it is essential to comprehend the functional repercussions of sequence changes. Deep mutational scanning (DMS) investigations, which systematically assess the impact of mutations on a given measurable property, might be used to quantify such impacts [41]. However, the complete combinatorial space of potentially helpful mutations cannot be explored by DMS experiments since they often need the characterisation of several thousands of alterations, which is expensive and time-consuming. This work introduces two new state-of-the-art models- *CoVBERT* and *MuFormer*, which are able to perform in vitro mutagenesis by learning evolutionary relationships amongst protein families to perform point mutations and fixed backbone sequence design respectively. This is the first work involving generation and validation of sequence SARS-CoV-2 mutational sequence generation using large pre-trained protein language models.

3 Background

3.1 Neural Language Models

Neural language models compute probability for a sequence of words $w = (w_1, w_2, \ldots, w_N)$ in a particular sequence, such that $\sum_{w \in \sum^*} p(w) = 1$. These models are trained on a large corpora of textual data with the intuition of determining the probability of a new utterance , given the training dataset. Majority of the language models employ the chain rule for decomposing the joint probability $p(w_1, w_2, \ldots, w_n)$ into a sequence of conditional probabilities such that:

$$p(w_1, w_2, w_3, \ldots, w_n) = p(w_1)p(w_2|w_1)p(w_3|w_1, w_2) \times \ldots \times p(w_n|w_1, w_2, \ldots, w_{n-1}) \quad (3.1)$$

Language models seek to learn the conditional distribution for the next word (w_n) given the contextual history of words observed $(w_1, w_2, \ldots, w_{n-1})$, using an autoregressive strategy of decomposing the joint distribution into conditional distributions [42]. As the number of words occurring before a word w_i in a given sequence differs based on its position, $P(w_1, \ldots, w_n)$ is usually conditioned over a window of n previous words rather than all previously occurred words as in Equation 3.1 where the c is the context window size (conditioning context i.e history). For speech and neural machine translation systems, Equation 3.2 is particularly helpful in detecting whether a word sequence is a correct translation of an input sentence.

$$P(w_1, \ldots, w_n) = \Pi_{i=1}^{n} P(w_i|w_1, \ldots, w_{i-1}) \approx \Pi_{i=1}^{n} P(w_i|w_{i-c}, \ldots, w_{i-1}) \quad (3.2)$$

The markov chain assumption [43] states that only the most recent context in a sequence is pivotal for predicting the next word's probability, and neural language models based on this assumption are known as n-gram models. These n-gram models are known as unigram when

the next words predicted probability is independent of the context, bigram when it depends over only the previous token (i.e $|c| = 1$), trigram when it depends on the previous two tokens (i.e $|c| = 2$), and so on. Equation 3.3 represents a bigram (2-gram) language model.

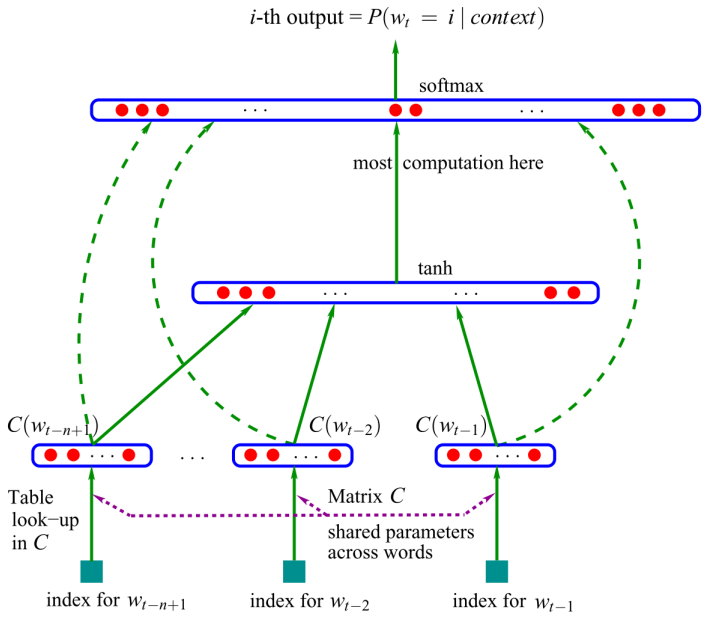
$$P(w_1, w_2, w_3, \ldots, w_n) = p(w_1)p(w_2|w_1)p(w_3|w_2) \times \ldots \times p(w_n|w_{n-1}) \tag{3.3}$$

3.1.1 Feedforward Neural Language Models

At the core of language models, neural networks are primarily used for estimating the conditional distributions over a word space. The straightforward neural network approach for language models leverages the Markov Assumption to directly model the conditional distribution $P(w_t = i|context)$ using feed forward neural networks. Feedforward neural network language models (NNLM) [42] learns to predict the next word based on the context of the preceding words in a given sequence. This model learns a distributed representation of words as well as the probability function for word sequences stated in terms of these representations. The model first learns the word embeddings using a straightforward feedforward neural network, and in a subsequent step it learns the probability function for word sequences. These learned word representations can be used as input for a variety of additional downstream tasks.

NNLM architecture consists of an input layer with words encoded using one-hot encoding, a linear projection layer for word embeddings, and a hidden layer which leverages a hyperbolic tangent function (where majority of the computation is done), culminating with a softmax classifier output layer. This architecture is demonstrated in Figure 3.1.1, wherein the input word vectors are used by the hidden and output layers. Equation 3.4 represents the model architecture where the weight matrix $W^{(1)}$ is applied over the word vectors and $W^{(2)}$ is applied to the hidden layer. It leverages $tanh()$ non-linearity in the hidden layers as well as the linear function ($W^{(3)} \cdot x + b^{(3)}$), which captures the previous n word vector inputs.

$$\hat{y} = softmax(W^{(2)}tanh(W^{(1)} \cdot x + b^{(1)}) + W^{(3)} \cdot x + b^{(3)}) \tag{3.4}$$



A vector representing the probability conditional distribution across all words in a particular context is the model's output, where the $i-th$ element represents the probability estimated $P(w_t = i|context)$, y_i is the unnormalised log probability for each output word i which were computed in the previous layer, and c is the context window size as shown in Equation 3.5.

$$\hat{P}(w_t|w_{t-1},\ldots,w_{t-c+1}) = softmax(y) \tag{3.5}$$

$$where\ softmax(y) = \frac{e^{Y_{w_t}}}{\sum_i e^{y_i}}$$

NNLM uses a cross-entropy loss function (defined in Equation 3.6) to optimise the weights for the embedding vector iteratively to minimise the loss L over the training samples using stochastic gradient descent (SGD) [44] optimiser which obtains the gradient via backpropagation (Equation 3.7). SGD tries to find the strongest descent direction through the partial derivatives and applies these to update the parameters weights' in accordance. The learning rate ϵ determines the size of the step in each of these gradient steps in a particular direction [45].

$$\mathbb{L} = -\frac{1}{N}\sum_n w_n^T \cdot log(\hat{P}) \tag{3.6}$$

$$\theta \leftarrow \theta + \epsilon \frac{\partial log(\hat{P}(w_t|w_{t-1},\ldots,w_{t-c+1}))}{\partial \theta} \tag{3.7}$$

3.1.2 Word Embeddings

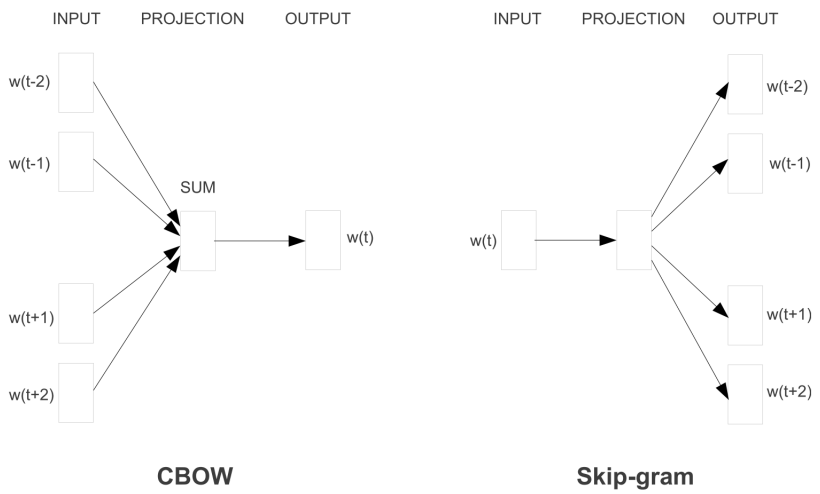
To represent each word w_i in a vocabulary, word embeddings make use of continuous vectors of n dimensions, which help in representing different aspects of the word. These word representations assist in mapping the words to a continuous vector space called the semantic space, which is capable of capturing semantic similarity such that semantically similar words exist in closer vicinity, whereas the dissimilar words are far apart from each other. Dense vector representations are used to embed words in a sequence by mapping each word to the continuous vector of n dimensions. Generally, word embeddings are learned by neural networks, and once trained, these can be reused across distinct downstream tasks. Majority of natural language processing algorithms at present make use of word embeddings, owing to their advantages of capturing semantic relationships unlike one-hot encoding or bag-of-words representations. The intuition behind learning word embeddings comes from the distributional hypothesis [46], which expresses that words occurring in the same context tend to have similar meanings.

Word embeddings can be computed using Word2Vec [47] and GloVe [48]. Word2Vec models include Continuous Bag-of-words (CBOW) and Skip-gram which seeks to predict a target word given the context, and predicting context words for a given target word using a feed forward neural network, respectively. These models make use of shallow neural networks including input, projection and output layers. Unlike feedforward neural language models, Word2Vec models do not have a non-linear hidden layer.

CBOW architecture consists of input words which are one-hot encoded vectors of size V, where V is the size of the vocabulary. CBOW makes use of both the previous and the following words in the context space, instead of only the previous words as in neural network language models. The projection layer which is shared for all words, is a fully connected dense layer with $1 \times D$ dimensionality (where D is the size of the word embedding dimension). This reflects that all words get projected in a linear manner into the same position with the vectors averaged. CBOW uses a log-linear hierarchical softmax classifier for computing the output probabilities such that the output has a dimension of V, including probability distributions for the target words spanning over each word in the vocabulary space.

Similar to CBOW, the continuous skip-gram architecture also makes use of a log-linear hierarchical softmax classifier with a continuous projection layer, but the input consists of only one target word, and the output includes of as many probability distribution vectors over all words as the chosen number of context words. Skip-gram model provides greater weight to local context words, than to the ones which are farther, by selecting less from the

farther away words in the training samples as they are typically less relevant to the target word.



GloVe, the other approach for computing word embeddings stands for Global Vector word representations, emphasising over the global character of this model. GloVe relies on the local context as well as the global co-occurrence statistics, to derive semantic relationships amongst the words from the co-occurrence matrix. The comparison of co-occurrence probabilities with a third word is more indicative of the semantic association than a direct co-occurrence probability [**empty citation**]. GloVe tries to fulfil the objective described in Equation 3.8, wherein b_i and b_k are the bias terms for word w_i and the probe word w_k, and X_{ik} is the number of times w_i co-occurs with w_k.

$$w_i^T w_k + b_i + b_k = log(X_{ik}) \tag{3.8}$$

3.1.3 Recurrent Neural Networks

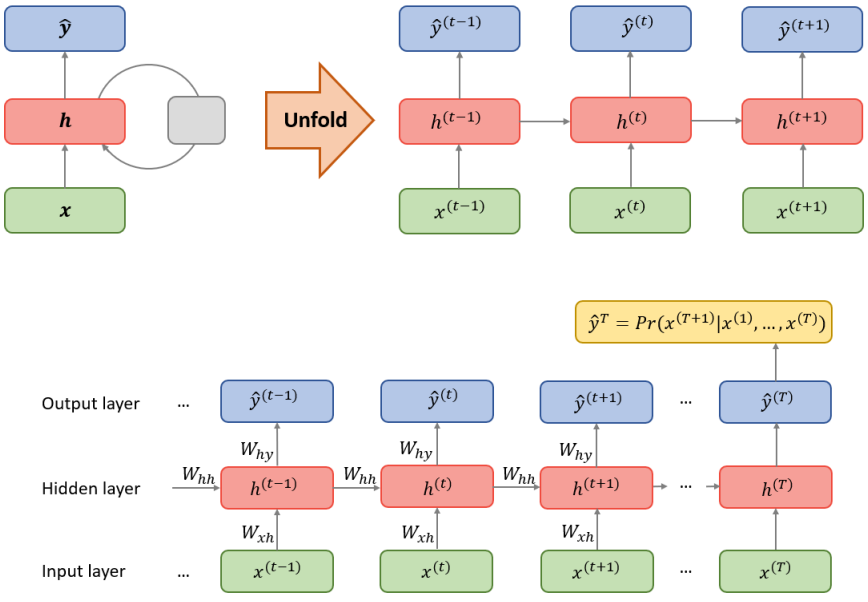
Conditional probabilities generated by n-gram models which make a markov assumption, inculcate only the previous $n-1$ tokens, thus ignoring any long-range dependencies that might exist in the data. Recurrent Neural Networks (RNNs), which are deprived of the markov assumption, resolve this long-range dependency problem by modelling the conditional probabilities dependent over the entire history of the string being generated up till that token [49] [50]İntuitively, recurrent network is a deep feed forward neural network wherein the hidden layer is a function of both the input x_t and of the previous hidden layer h_{t-1}, at each time-step t, and this is further transmitted to the next hidden state.

RNNs are able to process a sequence of tokens (words or characters) $x^{(1)}, \ldots, x^{(t)}$ by going over each element in turn and capturing information which is stored in hidden states $h^{(t)}$ as

network memory, based on the previous elements. Instinctively, a simple RNN can be viewed as a for-loop which reuses values calculated in the previous iterations [51].

The unfolded directed acyclic graph for RNNs computation (unrolled) is depicted in Figure 3.1.3 and 3.1.3, which showcases the structure of a classical RNN, where the grey square reflects the flow of information at time-steps and each node is associated with a network layer at a particular time instance [52]. The probability distribution for the next input is simply $Pr(x^{(t+1)|y^{(t)}}$. The network includes the following weight matrices: $W_{(xh)}$: weight matrix connecting inputs to hidden layers, $W_{(hh)}$: weight matrix connecting recurrent connections between hidden states, and $W_{(hy)}$: weight matrix connecting hidden states to outputs. The input x_t is multiplied to the weight matrix $W^{(xh)}$, and the output of the previous layer h_{t-1} is multiplied to the weight matrix $W^{(hh)}$ to produce the output features h_t which are multiplied by the $W^{(hy)}$ weight matrix, followed by a softmax function over the vocabulary to obtain prediction output $\hat{y}$ for the next word (Equation 3.9).

$$h_t = tanh(W^{(hh)} \cdot h_{t-1} + W^{(xh)} \cdot x_t) \tag{3.9}$$
$$\hat{y} = softmax(W^{(hy)} \cdot h_t)$$



Weight matrices in RNNs ($W_{(xh)}$, $W_{(hh)}$, $W_{(hy)}$), remain the same throughout the loop and are used repeatedly. This parameter sharing ability of RNNs makes them apt for working with sequential data as the model size doesn't grow for longer input sequences, and this allows application of models to sequence inputs of different lengths, thus enabling generalisation

across distinct positions [52].

Backpropagation for RNNs necessitates recursive computations of gradients through time-steps t, because each part of the output h_t is a function of the preceding parts of the output h_{t-1}. This backpropagation through time (BPTT) [53], allows RNNs to leverage information from preceding computation time-steps network memory [54].

As RNNs store hidden states h_t for each layer (time-step), the computation and memory required grows linearly with the length of the sequence, making BPTT impractical to apply for longer sequences. To overcome these difficulties, Truncated Backpropagation through Time (TBPTT) splits a long sequence into chunks of fixed length sequences of length k, followed by a forward and backward pass over each of these subsequences by initialising the forward pass with the last hidden state of the previous subsequence. The backward pass in TBPTT discards gradient information across subsequences. TBPTT boasts a complexity linear in k, instead of the length of the longest training sequence, making it effective for learning medium range dependencies over long sequences.

Although RNNs offer the advantage of modelling long-range dependencies, they are prone to two big problems of exploding gradients (and vanishing gradients. As the gradients are computed recursively, they could become extremely small or extremely huge with the repeated application of non-linearities. This emanates in obscuring information regarding long-term interdependence. Further, the computation is slow for RNNs owing to its sequential nature which cannot be parallelized.

3.1.4 Gated RNNs

Gated RNNs assist in overcoming the problems of exploding and vanishing gradients observed in RNNs, making accumulation of information about specific features over a long range possible. Two most popular types of Gated RNNs include Long Short-Term Memory [13] and Gated Recurrent Units models [55].

3.1.4.1 Long Short-Term Memory

Long Short-Term Memory [13] networks (LSTMs) improve over RNNs with the purpose of dealing with the long-term dependency problem. Instead of being a straightforward hidden unit that mixes inputs and previous hidden states linearly and outputs their nonlinear transformation to the next hidden step, in LSTMs hidden units are augmented by specialised input, forget, and output gates that assist in regulating the flow of information. Such complex

units are known as memory cells. Figure 3.1.4.1 and Equation 3.10 depict the architecture of LSTM.

New memory generation: At this stage, the input word x_t and the previous hidden state h_{t-1} are used to generate a new memory unit $\hat{c}_t$ which includes aspects of the new word x_{t+1}.

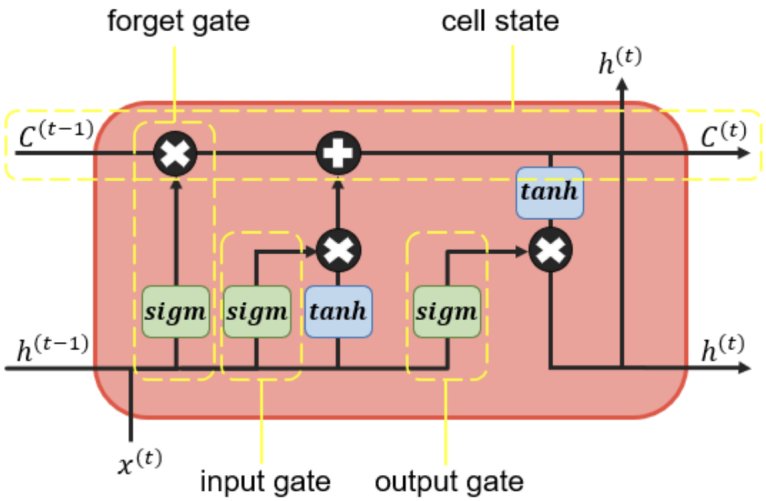
Input Gate: The input gate i_t acts as a gate for the new memory cell $\hat{c}_t$ such that it uses the input word x_t and the previous hidden state h_{t-1} for determining whether or not the input is worth storing in the memory history. The input gate i_t is the indicator of this information as the new memory generation stage doesn't check if the word is even useful before generating the memory unit.

Forget Gate: Similar to the input gate i_t, the forget gate f_t determines if the past memory cell c_{t-1} is useful for the computation of the current memory cell or not. If the past memory cell c_{t-1} is not useful, then the forget gate completely discards (forgets) this information. Thus, with the input word x_t and the past hidden state h_{t-1}, the forget gate generates f_t.

Final Memory Generation: This stage leverages the advice of the forget gate f_t to forget the past memory c_{t-1} if applicable. It also takes the advice of the input gate i_t similarly, and in accordance to that it gates the new memory $\hat{c}_t$. These two are then summed to compute the final memory c_t.

Output Gate: The output gate separates the final memory c_t from the hidden state h_t. There is a lot of information in the final memory c_t that does not necessarily need to be kept in the hidden state. Every gate of an LSTM uses hidden states, hence this gate determines which portions of the memory should be exposed or present in the hidden state h_t. The signal it generates to communicate this is o_t which is used to gate the memory's point-wise $tanh$.

$$
\begin{aligned}
i_t &= \sigma(W_{i_1} \cdot h_{t-1} + W_{i_2} \cdot x_t) - Input\ Gate \\
f_t &= \sigma(W_{f_1} \cdot h_{t-1} + W_{f_2} \cdot x_t) - Forget\ Gate \\
o_t &= \sigma(W_{o_1} \cdot h_{t-1} + W_{f_2} \cdot o_t) - Output\ Gate \\
\hat{c}_t &= tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t) - New\ Memory\ Cell \\
c_t &= f_t \cdot c_{t-1} + i_t \cdot \hat{c}_t - Final\ Memory\ Cell \\
h_t &= o_t \cdot tanh(c_t) - Hidden\ State \\
\hat{y}_t &= softmax(W_{hy} \cdot h_t)
\end{aligned}
\tag{3.10}
$$



3.1.4.2 Gated Recurrent Units

Gated Recurrent Units (GRUs) [55] are a modified version of RNNs, designed in a manner to have greater persistent memory, thereby making it easier for RNNs to capture long-term dependencies. Compared to LSTM, it is much simpler as it consists of only two gates, namely reset r_t and update gates z_t (Figure 3.1.4.2).

New memory generation: A new memory $\hat{h}_t$ is generated by applying tanh non-linearity over the sum of two products: (1) the weight matrix connecting inputs to hidden layer W_{xh} and the current input x_t and (2) the weight matrix connecting recurrent connections between hidden layers W_{hh} and element-wise dot product of the reset gate r_t with the previous hidden state h_{t-1} as shown in Equation TODO. This step assists in combining a newly observed word with the past hidden state h_{t-1} to summarise it given the past context of $\hat{h}_t$.

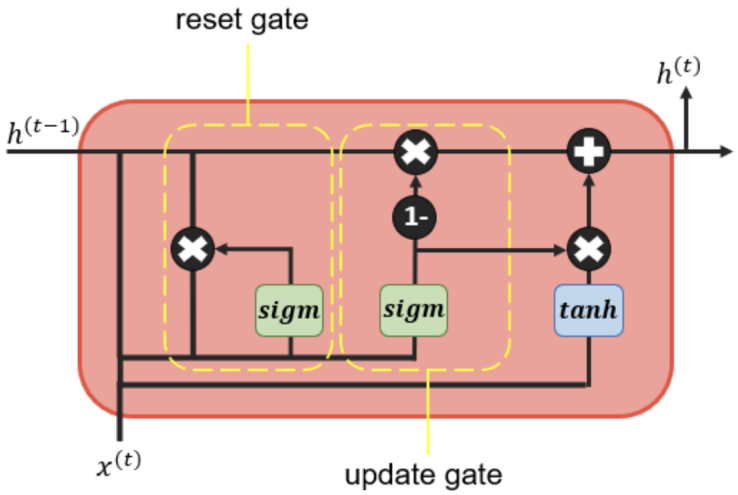
Reset Gate: The reset gate r_t is responsible for determining the importance of the previous hidden state h_{t-1} and if it should be ignored. Considering if r_t is close to 0 (i.e $r_t \approx 0$) or completely irrelevant to the computation of the new memory, the hidden state is forced to drop any irrelevant information stored from the tokens traversed in the past and it is reset with the current input x_t.

Update Gate: The update gate z_t is responsible for analysing how much of the past hidden state h_{t-1} has to be carried forward to the next hidden state h_t. If the update gate $z_t \approx 1$, nearly the entire memory from the previous hidden state h_{t-1} is copied to the next hidden state h_t. But if the update gate $z_t \approx 0$, then mostly the newly generated memory $\hat{h}_t$ is forward to the next hidden state h_t. As a result, the update gate combines the abilities of

the LSTM input and forget gates and aids in the retention of long-term memory, while being computationally cheaper than LSTM [55].

Hidden State: The hidden state h_t is finally computed with the past hidden input h_{t-1} and the new memory generated $\hat{h}_t$, along with the assistance of the update gate z_t as shown in Equation 3.11.

$$\begin{aligned} z_t &= \sigma(W_{z_1} \cdot h_{t-1} + W_{z_2} \cdot x_t) - Update\ Gate \\ r_t &= \sigma(W_{r_1} \cdot h_{t-1} + W_{r_2} \cdot x_t) - Reset\ Gate \\ \hat{h}_t &= tanh(W_{hh}(r_t \cdot h_{t-1})) + W_{xh}x_t) - New\ Memory \\ h_t &= ((1 - z_t) \cdot \hat{h}_t) + (z_t \cdot h_{t-1})) - Hidden\ State \\ \hat{y}_t &= softmax(W_{hy} \cdot h_t) \end{aligned} \tag{3.11}$$



3.1.5 Seq2Seq Models

The task of mapping variable-length input sequences to variable-length output sequences is called Sequence-to-Sequence or seq2seq learning in language modelling [55]. The seq2seq model is composed of two blocks known as the encoder and decoder segment (Figure 3.1.5).

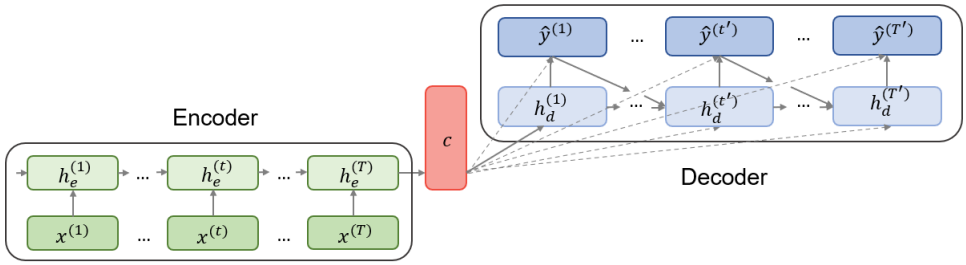
The first part, which is an encoder block, is an RNN trained over input sequences to compute a vector c of encoded intermediate representation of the inputs summary with a fixed dimension. This vector which is a simple function of the last hidden state unit $c = h_e^{(T)}$ is called the context vector [56].

The second part includes a decoder block, which is another RNN trained to generate predictions for a given context c and all the previous hidden state outputs y. Contrary to a

vanilla RNN described above, the decoder hidden states $h_d^{(t)}$ are conditioned on the previous outputs $y^{(t)}$, preceding hidden states $h_d^{(t)}$ and the summary vector c which is the intermediate representation from the encoder segment.

For a single step of prediction, the conditional distribution for Seq2Seq models is obtained by training both the components (encoder and decoder) simultaneously in order to maximise the conditional log-likelihood (Equation 3.12), where θ denotes the set of model parameters, x_n is the input sequence, and y_n is the output sequence.

$$\frac{1}{N}\sum_{n=1}^{N}\log p_\theta(y_n|x_n) \tag{3.12}$$



3.1.6 Attention Mechanism

The self-attention mechanism learns the representation of each token by attending to each position in the sequence [19]. Given an input x = $(x_1, \ldots, x_n)$, self-attention assigns an attention weight $\alpha_{i,j} > 0$ to each token pair i, j where $\sum_j \alpha_{i,j} = 1$. The inputs are processed by the transformer via a series of blocks alternating between multi-head attention and feed-forward layers. Transformers view the encoded representation as a set of query-key-value triples (Q, K, V) in each head of a self-attention layer. The attention weights $\alpha_{i,j}$ are computed with a scaled-dot product of i's query vector and j's key vector, which is followed by a softmax operation. These attention weights are then used to generate a weighted sum of the value vectors:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.13}$$

where Q is the query matrix, K is the key matrix, V is the value matrix, d_k is the dimension of K i.e Key.

In the multi-head setting, queries, keys and values are projected linearly h times and the attention operation is performed in parallel for each representation, with the results concatenated.

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O \tag{3.14}$$

where,

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{3.15}$$

Attention mechanism allows transformer models to learn relationships between tokens that are far apart in the sequence with ease. As self-attention creates pairwise interactions (QKT) between all places in the sequence, the model can capture residue-residue interactions directly.

3.1.7 Masked Autoencoders

Masked language modelling, popularly known as autoencoder-based models, seek to model probability of each token appearing in a masked position, given all other tokens and the surrounding sequence as the context. To ensure computational efficiency, a random set of tokens are masked in order to achieve this masked objective. This masked position prediction formulation overcomes the problem of learned representations being dependent over the preceding tokens in the autoregressive models. However, the masked language modelling objective does not allow calculation of normalised probabilities for the entire sequence correctly, but it is suitable when the learned representations are the outcomes of primary interest. For a sequence $x \in X^N$, wherein X is a set of amino acids and N is the length of the sequence, the masked language modelling approach implicitly tries to model a distribution over the protein sequences using conditional likelihoods $p(x_i|x_{[N]\setminus\{i\}})$ where, $x_{[N]\setminus\{i\}}$ is the sequence without residue at position i which is also referred to as the sequence context. Usually, protein language models with masked language objective also learn a latent variable $z_i \in \mathbb{R}^D$ by learning a function $f : X^{N-1} \to \mathbb{R}^D$ where $z_i \stackrel{\text{def}}{=} f(x_{[N]\setminus\{i\}})$ such that $p(x_i|x_{[N]\setminus\{i\}}, z_i) = p(x_i|z_i)$

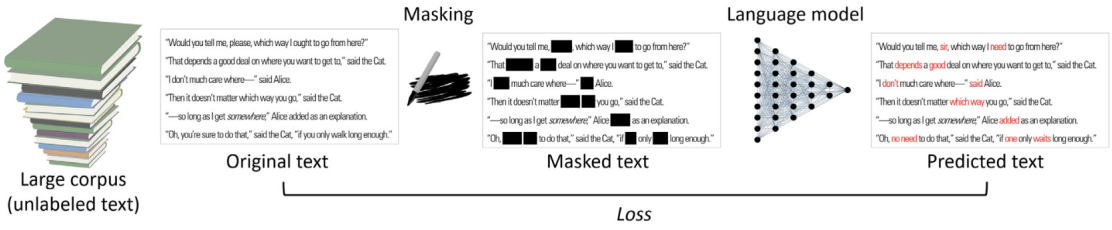


Figure 3.1: Flow of MLM applied over natural language texts, taken from [57]

3.1.8 Autoregressive Decoders

Causal Language Modelling, popularly known as autoregressive models, factorises the probability of a sequence into a product of conditional probabilities wherein the probability for each token is conditioned over the preceding tokens. This factorisation comes beneficial while sampling from the distribution or when evaluating the probabilities themselves is of primary

interest. In other words, autoregressive models aim at modelling the probability of a token appearing after another in order. The disadvantage of this approach is that the representations learned for each position are only dependent on previous positions, making them potentially less effective as contextual representations.
K-mer (n-gram) models, Hidden Markov Models, and standard autoregressive recurrent neural network or generative transformer language models like GPT [58], XLNet [59] are examples of autoregressive language models. Comparing autoregressive models for biological sequences to typical models in bioinformatics, the autoregressive models predict the residue given all the preceding positions (Fig 3.2), instead of finding correlations between columns as performed in multiple sequence alignment [60] (Figure 3.2).
The probability distribution $p(x|\theta)$ can be broken down into the product of conditional probabilities over the previous tokens in a sequence of length L through an autoregressive likelihood:

$$p(x|\theta) = p(x_1|\theta) \prod_{i=2}^{L} p(x_i|x_1, \ldots, x_{i-1}; \theta) \tag{3.16}$$

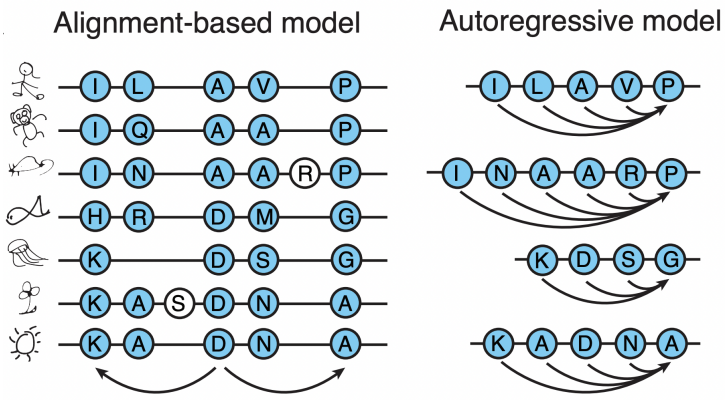


Figure 3.2: Comparison between Alignment-based and Autoregressive modelling, taken from [61]

3.2 Protein Language Modelling

3.2.1 Proteins and Natural Language

Protein sequences are linear chains of amino acids (20 standard amino acids) combined with covalent bonds and they're limited to take on specific 3D shapes (known as protein 3D structure) that are optimised for specific functions [62]. Common protein motifs and domains, which are the basic functional building blocks of proteins, are analogous to human language words, phrases, and sentences [63] [64], allowing us to leverage natural language processing (NLP) methods directly over the protein sequences. Whilst training, the language models

learn to extract these rules from millions of instances in growing protein sequence databases and save the acquired knowledge to the weights [17]. Several existing solutions in protein bioinformatics commonly conduct search for evolutionary-related proteins in exponentially growing databases [65] [66] [67]. For instance, Deep Mutational Scanning (DMS) [41] involves replacing every possible amino acid to a protein and evaluates the effect of each mutation by assessing a property of that protein, such as cellular binding or infectivity. While DMS is useful for assessing single-amino-acid changes, analysing the escape potential of combinatorial mutations becomes infeasible and computationally expensive.
Language models offer the benefit of extracting features directly from protein sequences efficiently and can be used to gauge single-residue mutation effects on biological fitness [68]. Moreover, protein language models have recently been proven to boost performance while dealing with situations that have minimal annotated data [69]. Furthermore, language modelling of protein representations directly captures a sequence's evolutionary context [68] [17] [11]. These models learn from the context of the entire sequence to predict missing or masked amino acids. The current computational models based on language modelling [70] [16] focus on their fitness or the protein function (semantic) similarity. Protein language modelling utilises the genomic sequences data alone and requires no explicit viral escape information (i.e unsupervised learning), which is easier to obtain than the protein structures or the annotated sequences. By leveraging language models, amino-acid sequences can be encoded into distributed vector representations capturing their structural and functional properties and evaluating the fitness of sequence mutations [71].

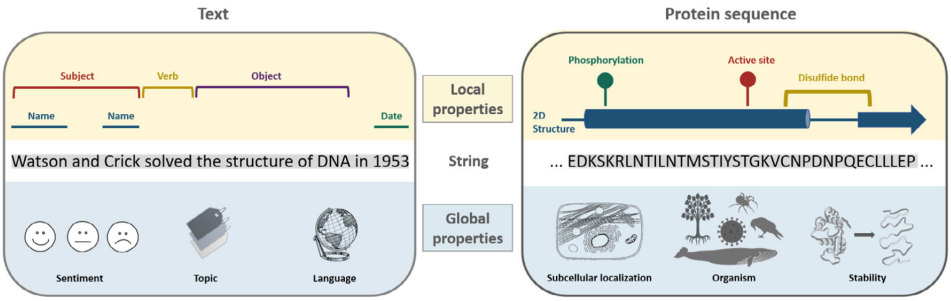


Figure 3.3: Contrast between language processing tasks performed over text and protein sequences, taken from [57]

Whilst modelling the language of protein sequences, we view proteins as a language containing a vocabulary of amino acids strung together to form the sequence. Language models can be used to predict the fitness of sequence mutations by learning the likelihood of a sequence occurring [21] [38] [27]. They also learn summary representations and features for better capturing sequence relationships and connecting sequence to function through transfer learning [11] [70] [16] [15] [72] [73]. Furthermore, by conditioning the language model on

structural [31] or functional [7] specifications, they can generate controlled sequences. The language model can develop an implicit understanding of how proteins are created by analysing a large number of sequences including any repetitive sequence segments, interactions between pairs, and evolutionarily conserved regions.

Several groups have looked into protein language modelling with an unsupervised training goal [15] [11] [12] [16] [7]. Fitting linear models to focussed groups of evolutionarily related and aligned sequences has been a long-standing practice in bioinformatics. Protein language modelling, on the other hand, trains nonlinear deep neural networks on enormous databases of evolutionarily different and unaligned sequences. Protein language models with high capacity have been proven to learn underlying intrinsic aspects of proteins like structure and function from sequence data [15]. With the amalgamation of NLP models to biology, we now have the ability to learn from naturally observed sequences [11] [70], as well as incorporate existing structural and functional information through multi-task learning.

3.2.2 Protein Engineering

The emerging field of Protein Engineering involves designing functional protein sequences through techniques such as directed evolution and de novo protein designing [74] [75], predicting effects of mutations, and determining the modifications to make to a protein for optimising it for a certain function and with desired properties. It possesses various useful applications such as optimising antibodies effectiveness, anticipating future mutations of a virus etc.

Protein sequence generative models are effective tools for predicting protein structure, mutation effects, and residue-residue interactions [76] [77] [21]. However, until yet, these methods have relied on homologous sequence alignments, which impose a forced structure on the data, which can introduce artefacts and ignore crucial information, especially due to alignment gaps created by insertions and deletions (indels) [61]. Autoregressive models have recently been used to learn general features and predict mutation effects in unaligned protein sequences across a wide range of protein families [11] [15] and these have started to take advantage of the much larger set of raw protein sequences by adapting state-of-the-art representation learning techniques [78] from natural language processing to protein property classification. On a small number of proteins, Costello et al. [79] achieved conditional sequence creation. ProGen [7] can generate novel proteins by iteratively predicting the next likely amino acid and generating proteins it has never seen before.

3.2.3 Protein Embeddings

Language models for protein sequence representation are based on the idea that distributed vector representations of proteins can be extracted from generative models of protein sequences learned from a large and diverse database of sequences across natural protein space, capturing the semantics, or function, of a given sequence.

3.2.3.1 Proteins Tokenization

Tokenization, or the division of text into individual tokens, is required for computational text analysis. Tokens are the atomic units of information in a given language representation. For proteins, individual amino acids are treated as character-level tokens, which is the simplest and most popular tokenization method. Since proteins lack a well-defined vocabulary of words, word-level tokenization is not a viable choice. Subword segmentation, on the other hand, is a potentially promising strategy for detecting "words" or motifs in proteins because it does not require any predefined knowledge of words in the target language [80] [81] [82] [83]. Fig 3.4 shows the difference between tokenization performed over natural text vs. protein sequences.

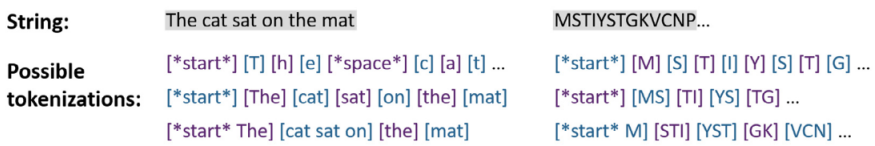


Figure 3.4: Tokenization performed over text vs. protein sequences, taken from [57]

3.2.3.2 Bag of Words Representation

The Bag-of-Words (BoW) feature extraction approach is the most basic and widely used in text processing. To produce token frequencies, BoW can normalise the resulting vector with respect to the entire number of tokens in the same text, or with regard to their normalised counts in all texts in a dataset, resulting in TF-IDF vectors. Additionally, BoW can count multi-token combinations with overlap (k-mers) or without (n-gram). BoW has been widely used in bioinformatics research on proteins [84] [85]. Protein Character-level tokens (i.e amino acids) are commonly counted in BoW. However, alternate sequence representations, such as 2D structure or reduced amino acid alphabets for hydrophobic or polar binary tokens, can be used to extract BoW properties [86] [87].

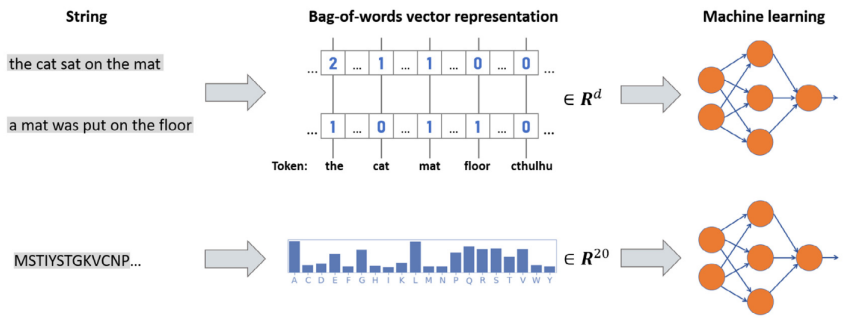


Figure 3.5: Bag of Words (BoW) representation for text vs. protein sequences, taken from [57]

3.2.3.3 Word Embeddings

Word embeddings represent tokens (eg: amino acids or words) in low-dimensional with fixed-size dense vector embeddings such that similar or neighbouring tokens acquire a similar vector representation and distant or unrelated tokens obtain a distinct vector representation. In proteins, we have 20 amino acids. While these amino acids can be embedded over a low-dimensional space, it is clearly not beneficial to do so [81]. While dimensionality reduction is limited when working with single amino acid tokens, it can be advantageous when considering longer amino acid combinations. For instance, ProtVec leverages word2vec over 3-mers amino acids for extracting a 300-dimensional vector representation instead of 8 k distinct trigrams [88]. It's worth noting that the amino acid embeddings learned by ML models are similar to those resulting from deconstructing AA substitution matrices in terms of the functional clusters they generate [15] [89] [90].

Part I

Large Protein Language Models are Zero-shot Mutagenesis Learners

4

CoVBERT - Mutation Prediction Language Model

4.1 Introduction

Language models are statistical models that give probability to words and sentences. Protein sequences are an amalgamation of alphabets with a vocabulary size of 20 amino acids, and this akin nature of proteins sequences to language has emanated in the recent rise of state of the art protein language models [15] [22] [17]. Language models for protein sequences seek to assign probability to sequences that are semantically and syntactically correct, fit and functional in the biological sense. Viruses such as the SARS-CoV-2 are prone to mutations which are in the form of substitution or indels in a protein sequence which affects the properties and function of a protein. This section explores the proposed model for curating de novo positional mutations in a zero-shot setting and analyses the representations learnt by large protein language models for the task of in vitro mutagenesis.

Here, CoVBERT, a transformer based [19] state-of-the-art protein language model has been proposed which has been trained with the masked language modelling objective [78] to perform in vitro mutagenesis over SARS-CoV-2 mutational hotspots, and for predicting novel mutations. Transformer based models handle all sequence elements concurrently by creating direct connections between individual elements through an attention mechanism, in contrast to conventional recurrent networks like LSTMs that process each sequence element in turn. This not only makes it possible to perform more tasks simultaneously, but it also improves accuracy for a wide range of tasks.

4. CoVBERT - Mutation Prediction Language Model

Transformer's capacity to simulate long-range relationships, which are poorly captured by RNNs or LSTMs, is particularly pertinent to protein sequences [91]. The network's path lengths for forward activation and backward gradient signals are a significant element influencing how well LSTMs perform on certain tasks [92]. It is generally recognised that long-distance dependence reflects structural characteristics of protein sequences. A Markov Random Field (Potts Model) that models the entire sequence with pairwise coupling parameters is used in direct coupling analysis [93] [94] to find pairwise dependencies in multiple sequence alignments. Similar to this, the Transformer builds up a representation of a sequence by alternating between self-attention and non-linear projections.

With the self-attention mechanism, each amino acid position is represented as a weighted sum of the other amino acid positions in a given sequence. The attention weights are dynamically calculated, allowing each position to select which data from the rest of the sequence to integrate at each calculation step. Self-attention architectures provide cutting-edge performance on a variety of natural language tasks, largely because of the Transformer's scalability in parameters and the amount of context it can integrate. These architectures are able to model large contexts and long-range dependencies in language data [78].

To this end, self-supervised learning has been used to train the proposed model, CoVBERT over 40,000 SARS-CoV-2 spike protein sequences spanning evolutionary diversity of lineages. The generated model includes representations for information about biological characteristics, which are learnt using only the sequence data. The bidirectional self-attention allows the model to make use of the local and global contexts to predict mutated amino acids. This mechanism makes use of a two-dimensional vector for computing and modelling context amongst all pairs of amino acids to enable learning of long-range dependencies between a protein sequence.

Owing to the multi-headed self-attention mechanism, this section elucidates over the models capability to learn distinct representations and interactions between amino acids for generation of novel mutations. CoVBERT has has been able to predict the most prevalent mutation *D614G* with a masked marginal log likelihood of 0.95, outperforming several state of the art baseline protein language models such as ProtBERT, ProtElectra [17] and DistilProtBERT [95]. *D614G* mutation led to the genesis of several SARS-CoV-2 mutations with increased infectivity such as Alpha (B.1.1.7), Beta (B.1.351), Gamma (P.1), Delta (B.1.617.2) and Omicron (BA.x, B.1.1.529) lineages [96]. CoVBERT has been trained on longer sequences with a context window length of 2,048. It utilises the full spike protein sequence with an average length of 1273 amino acids as input and generates output mutation log likelihoods

for mutational hotspot positions to predict substitution indels for in vitro mutagenesis of SARS-CoV-2 spike protein.

The learnt phylogenetic inductive biases and prowess of CoVBERT to predict *E484K* and *D614G* mutational hotspots in a zero-shot setting paves the path towards the outset of large foundational language models [97] for deciphering protein-protein interactions and generating novel viral mutations that may emanate in fitness advantages of SARS-CoV-2 variants prior to their onset. These findings have further been corroborated with an analysis of the interpretation of CoVBERTs self-attention over protein sequences, which showed its propensity to learn the representation of interactions amongst amino acids.

4.2 Proposed Model Overview

CoVBERT is a transformer-based discriminative encoder model which leverages a multi-head self-attention mechanism for modelling genetic mutations of viruses, specifically SARS-CoV-2. CoVBERT has been trained over 40,000 raw SARS-CoV-2 protein sequences space consisting of sequences of amino acids, allowing it to generate natural sequences with in vitro mutagenesis and predicting substitution indels over SARS-CoV-2 spike glycoprotein which is the region responsible for genesis of new variants with greatly enhanced immune fitness [98]. The model uses the full spike protein sequence of 1273 amino acids (i.e characters) as input and generates output mutation probabilities based on the surrounding bidirectional sequence context for the selected mutation hotspot. As CoVBERT is not trained over MSAs (multiple sequence alignment), the performance is less sensitive to their characteristics such as depth, thus, augmenting flexibility to work without them.

Opposing recurrent and convolutional neural networks, transformer-based models do not make any assumptions about the ordering of the input sequence, and instead make use of positional embedding to inculcate the information of tokens positions and for stripping off the permutation invariant nature. Transformer's capability to simulate long-range dependencies, which are inadequately captured by RNNs [49] or LSTMs [13], is especially pertinent to protein sequences. Transformers construct a representation of a sequence by alternating non-linear projections through the self-attention mechanism. The dynamic computation of attention weights enables each position to select what information to integrate from the rest of the sequence, at each computation step. Self-attention architectures, which were created to model large contexts and long-range dependencies, proffer state of the art performance over a variety of natural language tasks, largely because of Transformer's scalability in terms of

parameters and the amount of context it can integrate.

For identifying complex dependencies in genomic sequences and mutation hotspots, CoVBERT utilises the masked language modelling (MLM) training objective from Bidirectional Encoder Representations from Transformers (BERT) [78] [19] which allows the model to pre-train deep bidirectional representations from unlabeled sequence data only by jointly conditioning over both left and right context for all layers. Use of bidirectional encoder representations which leverages the masked language modelling objective inspired by the Cloze task [99] to fuse both the left and right context of a sequence, alleviates the unidirectionality constraint of ELMo and GPT models [100] [101].

The masked language modelling training objective randomly masks some of the tokens of an input sequence with the objective to reconstruct the sequence by predicting the masked positions given the unmasked sequence as the context. Unlike denoising auto-encoders [102] which reconstructs the entire input, only the masked words are predicted here. According to the masking probability which is set to 0.15 for CoVBERT, a set of indices k to be masked are sampled from a sequence s_i, substituting the true token with a mask token at <*mask*> each index $i \in k$.

For every masked token, the negative log likelihood is minimised for the true amino acid (i.e character), given the unmasked sequence as context. Instinctively, CoVBERT must deduce dependencies between the masked mutational hotspot positions, and the unmasked sequence space. CoVBERT omits the next sentence prediction objective of BERT [78] and employs dynamic masking for curating a dynamic masking pattern each time a sequence is fed into the model. This masking objective assists CoVBERT to predict mutations at a masked hotspot position.

CoVBERT is able to generate mutated sequences with a given masked position in a matter of seconds and its pre-trained model has been uploaded on HuggingFace hub [103] to facilitate easy usage of CoVBERT, allowing free public access through HuggingFace transformers Python package which allows downloading of the pre-trained CoVBERT model state, fine-tuning and evaluating it on other datasets.

4.3 Model Architecture

The final CoVBERT model is a 6 layer transformer encoder-based model consisting of 12 attention heads per layer with a hidden dimension of 768 for all encoder layers, leading to a 44 million parameters model as shown in Figure 4.1. The 6 transformer blocks manipulate both the local and global representations (left- and right-hand side respectively). With this bidirectional self-attention, CoVBERT is capable of learning long-range dependencies between tokens (amino acid characters) by calculating context between all positions in a given sequence.

Learned token embeddings trained using Byte-Pair encoding [104] have been employed. The model uses tied embeddings wherein the token embeddings are tied with the embeddings of the final output layer [105] [106]. These coupled embeddings improves language models performance via tying (i.e sharing) the weights of the embedding and softmax layers which helps in massively reducing the total number of parameters of the model.

Similar to the original transformer [19], sinusoidal absolute positional embeddings have been used. The feed-forward network in the encoder block leverages Gaussian Error [107] instead of the standard ReLU non-linearity [108]. All transformer layers were initialised from a zero centred normal distribution with a standard deviation of 0.02 s.t $L \sim \mathcal{N}(0, 0.0004)$, where $\mu = 0$ and $\sigma^2 = 0.0004$. All biases were initialised to zero and every projection includes biases, apart from the token and positional embeddings.

Transformer processes inputs using a series of stacked encoder blocks reflecting self-attention amongst pairs of tokens followed by feed-forward neural network and residual connections which help mitigate the vanishing gradient problem. Each input token embedding is transformed by the self-attention layer using a weighted combination of all the token embeddings in the input sequence, where the weights are computed by a pairwise dot-product between the input token embeddings. These modified input token embeddings are then independently processed by the token-wise feed-forward layer without any interaction between them.

Let S = $\{s_1, s_2, \ldots, s_N\}$, where N is the length of the sequence, be the reference sequence wherein s_i denotes the amino acid at position i in a given spike protein sequence. First, trainable projections are applied for mapping each of these s_i with its tokenization and positional information to three embedding vectors namely, query (q_i), key (k_i), and value (v_i), for the computation of self-attention. The output of the self-attention layer is computed by the following Equation 4.1:

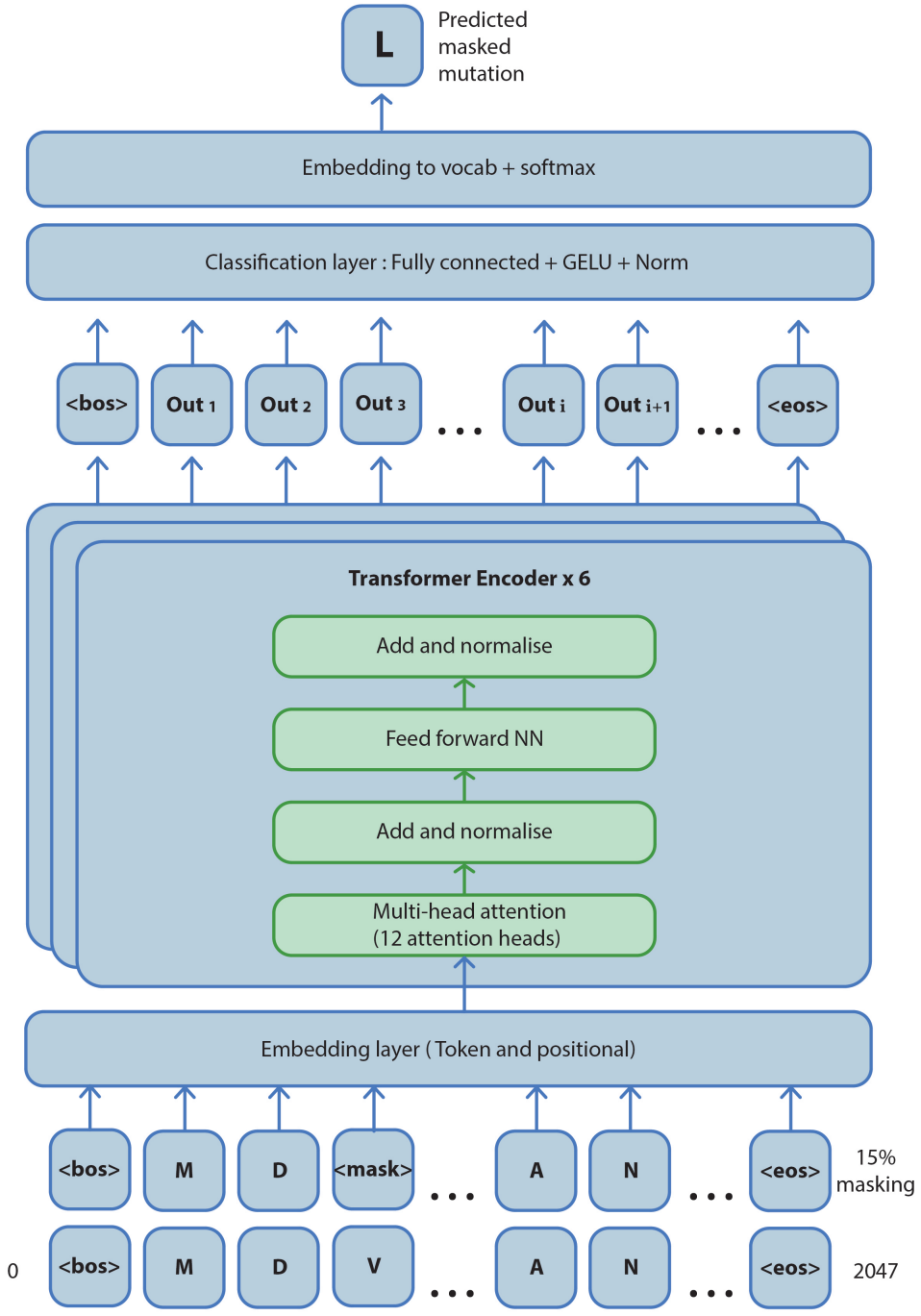


Figure 4.1: Proposed model architecture - CoVBERT

$$Attention(Q, K, V) = A \cdot V = softmax(\frac{QK^N}{\sqrt{d}}) \cdot V \tag{4.1}$$

where the dimension of each embedding is d, $A \in \mathbb{R}^{N \times N}$ is the attention matrix, $Q = [q_1; q_2, \ldots, q_N]$, $K = [k_1; k_2, \ldots, k_N]$, and $V = [v_1; v_2, \ldots, v_N]$ are the embedding matrices in $\mathbb{R}^{N \times d}$, where (q_i), (k_i), and (v_i) are row vectors representing the three embedding vectors.

The matrix multiplication represented by QK^N performs a dot product for each possible pair of queries and keys, resulting in a matrix of shape $N \times N$. The dot product between the query and key vectors as shown in Figure 4.2 (a), is performed independently for each pair of amino acid (i.e token) combinations as in Figure 4.2 (b). Each row depicts attention logits for a given amino acid (token) to all other amino acids (tokens) in the sequence. Over these, a softmax function is applied to normalise the $q_i \cdot k_j$ scores scaled by $\frac{1}{\sqrt{d}}$, which represents the final attention weights (Figure 4.2 (c)). These attention weights are then multiplied with the value vector v_i, resulting in a weighted mean with the weights determined by the self-attention.

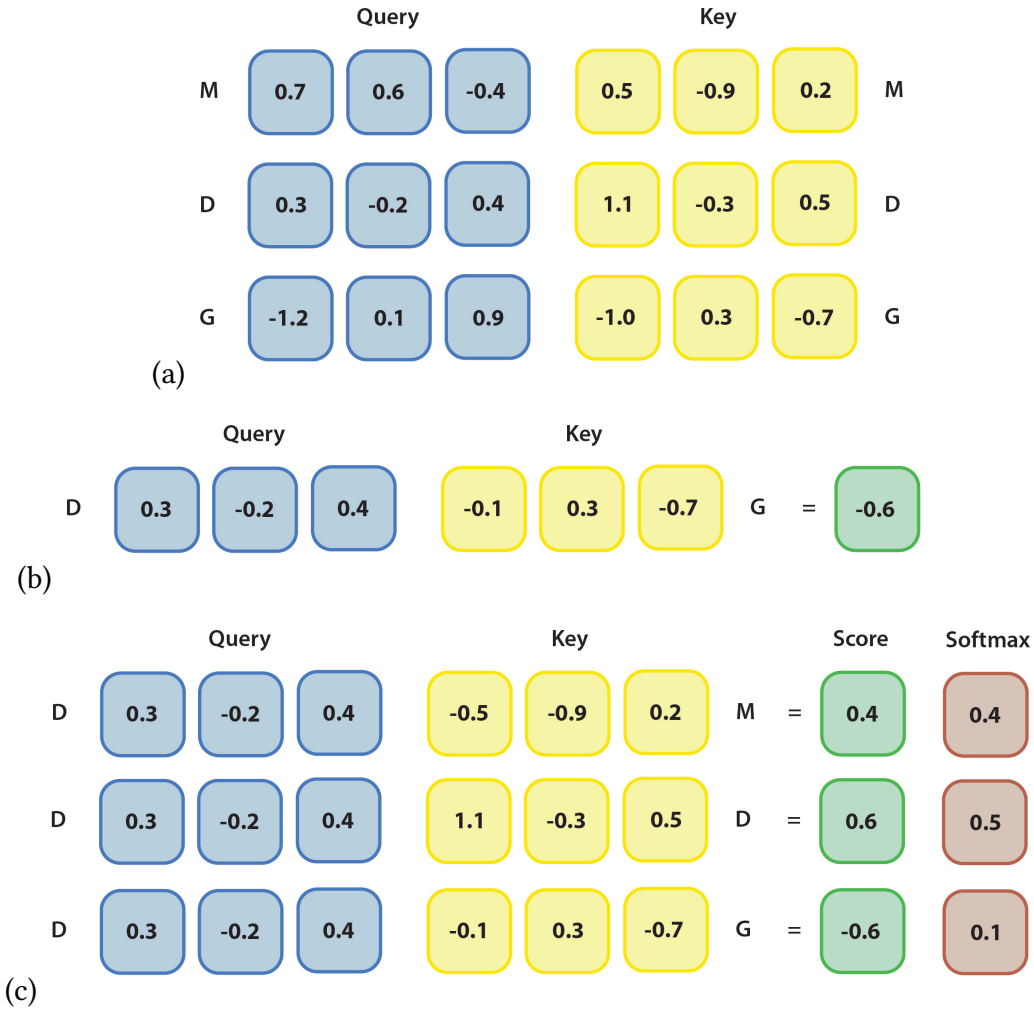


Figure 4.2: Self-attention scaled dot product computation

The scaling factor of $\frac{1}{\sqrt{d}}$ is critical for maintaining an appropriate variance of attention values post the initialisation and for adjusting the fact that long vectors might result in very high dot products. Applying the scaled dot product over query (q_i) and key (k_i) vectors with a variance $\sigma^2 = 0.0004$ results in a scalar having d times higher variance as shown in Equation

4.2, and this calls for the need to scale down the variance back to σ^2 with the scaling factor of $\frac{1}{\sqrt{d}}$ to avoid softmax over the logits from saturating to 1 for one random element and 0 for all others which would have resulted in vanishing gradients.

$$q_i \sim \mathcal{N}\left(0, \sigma^2\right), k_i \sim \mathcal{N}\left(0, \sigma^2\right) \rightarrow \mathrm{Var}\left(\sum_{i=1}^{d} q_i \cdot k_i\right) = \sigma^2 \cdot d \tag{4.2}$$

A single transformer encoder block is composed of the multi-head self-attention component followed by a feed-forward neural network with residual connections (skip connections) and post layer normalisation. CoVBERT's architecture comprises six of such transformer encoder blocks interleaving two functions involving multi-headed self-attention computation of residue-residue interactions throughout the sequence, and the feed-forward network, with skip connections present in both. Each self-attention layer has a computational complexity of $O(N^2 \cdot d)$, which is faster than recurrent layers [49] when the sequence length N is smaller than the representation dimension $\mathbb{R}^d$. Self-attention scales quadratically with respect to the sequence length N.

With 12 attention heads in each of the transformer encoder layers, the self-attention mechanism in Equation **??** is computed twelve times in parallel with their outputs concatenated, where W denotes the entire set of parameters including query, key, value and output linear projections required by a multi-head attention layer. Each position in the encoder layer can attend to those in the layer below them. The feed-forward network is applied independently and identically over each amino acid position. Another way of describing the feed-forward neural network is as two convolutions with a kernel size of 1. This is followed by a post layer normalisation (Equation 4.3) which enables faster training and provides small regularisation, ensuring that the features are in similar magnitude across the tokens in the sequence. For post layer normalisation in a transformer with L layers, the scale of gradients of the parameters for the last fully connected network layer is of order $\mathcal{O}(d\sqrt{\log d})$ which is independent of L [109]. Equation 4.3 depicts the complete transformation of the fully connected network with the residual connections where scale γ and bias vector β are parameters.

$$FFN(x) = GELU\left(x \cdot W_1 + b_1\right) W_2 + b_2 \tag{4.3}$$

$$x = PostLayerNorm(x + FFN(x))$$

$$where\ \boldsymbol{W} := \begin{pmatrix} W_{\boldsymbol{q}}^h \in \mathbb{R}^{d_{\text{attn}} \times d_{\text{x}}}, & \boldsymbol{b}_q^h \in \mathbb{R}^{d_{\text{attn}}}, & h \in [H] \\ W_k^h \in \mathbb{R}^{d_{\text{attn}} \times d_z}, & \boldsymbol{b}_k^h \in \mathbb{R}^{d_{\text{attn}}}, & h \in [H] \\ W_v^h \in \mathbb{R}^{d_{\text{mid}} \times d_z}, & \boldsymbol{b}_v^h \in \mathbb{R}^{d_{\text{mid}}}, & h \in [H] \\ W_{\boldsymbol{o}} \in \mathbb{R}^{d_{\text{out}} \times H d_{\text{mid}}}, & \boldsymbol{b}_{\boldsymbol{o}} \in \mathbb{R}^{d_{\text{out}}} & \end{pmatrix}$$

$$PostLayerNorm(v) = \gamma \frac{v - \mu}{\sigma} + \beta \tag{4.4}$$

$$where\ \mu = \frac{1}{d}\sum_{k=1}^{d} v_k\ and\ \sigma^2 = \frac{1}{d}\sum_{k=1}^{d}(v_k - \mu)^2$$

Through self-attention, the network is able to construct intricate representations that include context from both sides of the sequence. In the Transformer design, residue-residue interactions are directly represented since self-attention explicitly generates pairwise interactions between all places in the sequence. Algorithm 22 defines the pseudocode covering the forward pass for the encoder-only transformer used in CoVBERT.

CoVBERT takes in a sequence of amino acids i.e tokens and outputs a sequence of log probabilities optimised using masked language modelling objective from the bidirectional encoder representations from transformers [78] wherein each sequence is corrupted by substituting a selection of sampled amino acid indices with a special mask token. The model is trained to predict the masked token from the corrupted sequence which is similar to a fill in the blank task given the surrounding context (local and global), by minimising the negative log likelihood in Equation 4.5 which is stochastic in nature. Over a series of residual blocks, hidden states are computed, each having a sequence of vectors $H = \{h_1, h_2, \ldots, h_N\}$ with embedding dimension d.

$$\mathcal{L}_{\text{MLM}} = \mathbb{E}_{s\sim S}\mathbb{E}_{masked} \sum_{i\in masked} -\log p\left(s_i \mid s_{/masked}\right) \tag{4.5}$$

Self-supervised learning has been used to train CoVBERT, a deep contextual language model, over a curated training dataset of 40,000 SARS-CoV-2 spike glycoprotein sequences fetched from the Global Initiative for Sharing All Influenza Data (GISAID) [6] with only the raw sequences data. The training was performed for 5000 optimisation steps using AdamW optimiser with a learning rate of $\eta = 5e-5$, decoupled weight decay of 1e-2, context window of size $|n_{ctx}| = 2048$ tokens, and a dropout probability of 0.1 which follows the residual connections for each fully connected layer in the encoders.

Algorithm 1 CoVBERT$(x|\theta)$: encoder-only transformer, forward pass [Adapted from [110]]

Input: $x \in V^*$, a sequence of token IDs
Output: P $\in (0,1)^{N_V \times length(x)}$, where each column P is a distribution over the vocabulary
Hyperparameters: $\ell_{max}, L, H, d_e, d_{mlp}, d_f \in \mathbb{N}$
Parameters: θ includes the following parameters:
$W_e \in \mathbb{R}^{d_e \times N_V}, W_p \in \mathbb{R}^{d_e \times \ell_{max}}, W_s \in \mathbb{R}^{d_e \times |S|}$ - token, pos & segment embedding vectors
For $l \in [L]$:
W_ℓ, multi-headed attention parameters for layer ℓ
$y^1_\ell, \beta^1_\ell, \gamma^2_\ell, \beta^2_\ell \in \mathbb{R}^d$, two set of layer normalisation parameters
$W^\ell_{mlp1} \in \mathbb{R}^{d_{mlp} \times d_e}, b^\ell_{mlp1} \in \mathbb{R}^{d_{mlp}}, W^\ell_{mlp2} \in \mathbb{R}^{d_e d_{mlp}}, b^\ell_{mlp2} \in \mathbb{R}^{d_e}$, MLP parameters

$\ell \leftarrow length(x)$
for $t \in [\ell]$ **do**
  $e_t \leftarrow W_e[:, s[t]] + W_s[:, seg[t]] + W_p[:, t]$
end for
$X \leftarrow [e_1, e_2, \ldots, e_\ell]$
for $\ell = 1, 2, \ldots, L$ **do**
  $X \leftarrow X + MultiHeadAttention(X|W_\ell, Mask \equiv 1)$
  for $t \in [\ell]$ **do**
    $X[:, t] \leftarrow layer_norm(X[:, t]|\gamma^1_\ell, \beta^1_\ell$
  end for
  $X \leftarrow X + W^\ell_{mlp2} \cdot GELU(W^\ell_{mlp1} \cdot X + b^\ell_{mlp1}) + b^\ell_{mlp2}$
  for $t \in [\ell]$ **do**
    $X[:, t] \leftarrow layer_norm(X[:, t]|\gamma^2_\ell, \beta^2_\ell)$
  end for
end for
$X \leftarrow GELU(W_f \cdot X + b_f)$
for $t \in [\ell]$ **do**
  $X[:, t] \leftarrow layer_norm(X[:, t]|\gamma, \beta)$
end for
$logits \leftarrow W_u X$
$probs \leftarrow softmax(logits)$
$return\ probs$

4.4 Dataset Preprocessing

The pre-training dataset for CoVBERT was curated from the SARS-CoV-2 protein sequences data retrieved from Global Initiative for Sharing All Influenza Data (GISAID) database [6], which is the largest repository for SARS-CoV-2 sequences. To understand the effect of distinct mutations, these protein sequences were filtered to include only the spike glycoprotein region (S gene). For preserving all information contained in the pre-training data, clustering of sequences was not performed for training. Spike glycoprotein region was chosen because it is the primary component of the SARS-CoV-2 virus that is responsible for inducing neutralising

antibody responses in the host. This spike protein on the antigen is what allows it to engage with and penetrate into the host human cells. As a result, the spike protein has received increasing attention in the investigation of SARS-CoV-2 virus mutations and to gauge the infectivity levels of a variant [111]. Therefore, the predictions here have been focused on the SARS-CoV-2 spike protein.

Over 230,623 unclustered sets of sequences were downloaded from GISAID and filtered to include the S region. Sequences consisting of unknown amino acids such as U, Z, O, B and X, and sequences of length less than 1000 amino acids were eliminated. Further, duplicate sequences were removed. The resulting corpus of 50,000 unique sequences contained sequences spanning SARS-CoV-1 virus to the most recent SARS-CoV-2 variant sequences till August 2022, in order to curate an evolutionary dataset for equipping the model with phylogenetic context. Majority of these sequences contained 1273 amino acids i.e for $s \in S, |s| = 1273$, with the maximum length being 1332 amino acids. The final dataset of 50,000 sequences was generated with a block size of 1332 tokens i.e the maximum sequence length ($block_size := max_seq_len$), and further it was randomly split in 80:20, resulting in 40,000 random and unique sequences for training and 10,000 for validation. Further, this dataset was tokenized using Byte-Pair Encoding (BPE) [104].

4.5 Tokenizer Training

Prior to splitting an input sequence s_i into sub-tokens, the tokenizer first performs pre-tokenization to count the frequency of each token in the training corpus. The base vocabulary V for CoVBERT's tokenizer $N_V := |V| = 45$ is curated after the character-level pre-tokenization with a set of 45 unique tokens representing 20 amino acids (AAs) in upper as well as lower case, along with 5 special tokens ($|special(V)| = 5$), including $< bos_token >$, $< eos_token >$, $< pad >$, $< unk >$, and $< mask >$. The tokenization process assigns and encodes each element in the vocabulary $e \in \mathcal{V}$ into a unique index $v \in \{1, 2, \ldots, N_V\}$ where $V[: N_V] \in \{< bos_token >, < eos_token >, < pad >, < unk >, < mask >\}$. An input sequence s_i is represented as a sequence of token IDs i.e indices corresponding to its sub-tokens.

The $< bos_token >$ and $< eos_token >$ are added to augment information about the beginning and end of a protein sequence for allowing the model to interpret the full-sized sequence and to segregate a complete sequence from a cropped one in case of sequences longer or shorter than the context window with size of $|n_{ctx}| = 2048$ tokens. Whilst processing sequences that are longer than the context window size $|n_{ctx}|$, the sequence is cropped and

processed separately, emanating in leaving out one of its ends. During training, the context window is fully utilised by packing up sequences together. The missing $< bos_token >$ or $< eos_token >$ allows the model to interpret that it received only a portion of the sequence. The $< pad >$ token is used for padding sequences which are shorter than the selected mini-batch sequence length. The $< unk >$ token represents any out-of-vocabulary token observed in the input sequence s_i which could include unknown amino acids such as selenocysteine (U), undefined amino-acid (X), or others. Finally, the $< mask >$ token is leveraged for CoVBERT's masked language modelling objective to corrupt a token in a sequence that the model reconstructs.

The token embedding function $E(s)$ learns to represent each sub-token in this final vocabulary as a vector in $\mathbb{R}^D$ as shown in Algorithm 1. The raw input sequence is represented as a sequence of one-hot vectors of dimension $|V|$, passed to the learned embedding layer $E(x)$ and then forwarded to the transformer encoder blocks for computing multi-head self-attention with a feed-forward neural network. The resultant embedding vector $e \in \mathbb{R}^D$ is of size $(|V|, hidden_dimension)$ where the hidden_dimension is 768 for CoVBERT.

Algorithm 2 Token embeddings [110]

Input: $v \in V[N_V]$, a token ID
Output: $e \in \mathbb{R}^{d_e}$, vector representation of the token
Parameters: $W_e \in \mathbb{R}^{d_e \times N_V}$, token embedding matrix

1: $return\ e = W_e[:, v]$

Byte-Pair Encoding (BPE) [104], which is a sub-word tokenization algorithm assuring better performance to one-hot tokenization and prevention from the out-of-vocabulary problem, has been used for training the vocabulary over the protein sequences corpus. The original BERT model [78] uses WordPiece tokenization algorithm [112] consisting of common sub-tokens in the training corpus. Albeit, it suffers from higher out-of-vocabulary words incase of spelling errors, which hampers the models interpretation ability for a given input sequence s_i. Recent work has shown that vocabulary-free models such as byte-level models including Byte-Pair Encoding, are more resilient to such noise and perform better on tasks that are spelling sensitive like our case wherein the training corpus consists of protein sequences [113].

BPE tokenization takes the vocabulary V consisting of ordered merges which is applied to the training corpus in the same ordering as the vocabulary. It starts with a small vocabulary and learns rules for merging the tokens which are applied during the tokenizers training. The greedy construction strategy of BPE relies over sub-token units (single amino acid i.e

character) with a learned merge operation to obtain a segmentation with rarely any unknown symbols $< unk >$ [114], as shown in Algorithm 10. BPE assists the model to perform well without an enormous vocabulary size owing to the relationary inductive bias it injects for the tokens composed of similar sub-tokens. It learns word representations such that similar sub-tokens are represented by vectors that are closer in terms of their cosine distance. Following the tokenization phase which encodes an input sequence s_i into a vector of token IDs i.e integer tokens, the masking objective is applied with a uniform masking rate of 15% sub-tokens.

Algorithm 3 Byte-pair encoding [104]

Input: set of strings *D*, target vocab size *k*
procedure BPE(D, k)
  $V \leftarrow$ all unique characters in D
  while $|V| < k$ **do** Merge Tokens
    $t_L, t_R \leftarrow$ Most frequent bigram in D
    $t_{NEW} \leftarrow t_L + t_R$ Make new token
    $V \leftarrow V + [t_{NEW}]$
    Replace each occurrence of t_L, t_R in D with t_{NEW}
  end while
  return V
end procedure

4.6 Positional Embeddings

As transformers are deprived of recurrence and convolutions, some information corresponding to the relative or absolute positions of tokens in a sequence is required to be injected. The positional embeddings (PEs) are pivotal for encoding order of words or tokens in a sequence in transformer-based architectures and for allowing the model to make sense of the ordering of words. Position of a token defines the semantics of a sequence and omitting positional embeddings makes the transformer equivalent to a simple "bag of words" representation, emanating in a permutation equivariant model. Positional embeddings learn representation of a tokens position index in a sequence as a d-dimensional vector $\in \mathbb{R}^D$ (Algorithm 1). As shown in Figure 4.3, these positional embeddings are added to the input token embeddings and the segment embeddings (i.e sequence number) at the bottom of the encoder and decoder stack of the transformer, and the positional information is propagated to further layers through the residual connections in transformers. For example, for the i-th token of a sequence s, the cumulative embedding is represented as $e := W_{embedding}[:, s[i]] + W_{segment}[:, seg[s]] + W_{position}[:, i]$.

Algorithm 4 Positional embeddings [110]

Input: $l \in [\ell_{max}]$, position of a token in the sequence
Output: $e_p \in \mathbb{R}^{d_e}$, vector representation of the position
Parameters: $W_p \in \mathbb{R}^{d_e \times \ell_{max}}$, positional embedding matrix
1: $return\ e_p = W_p[:, \ell]$

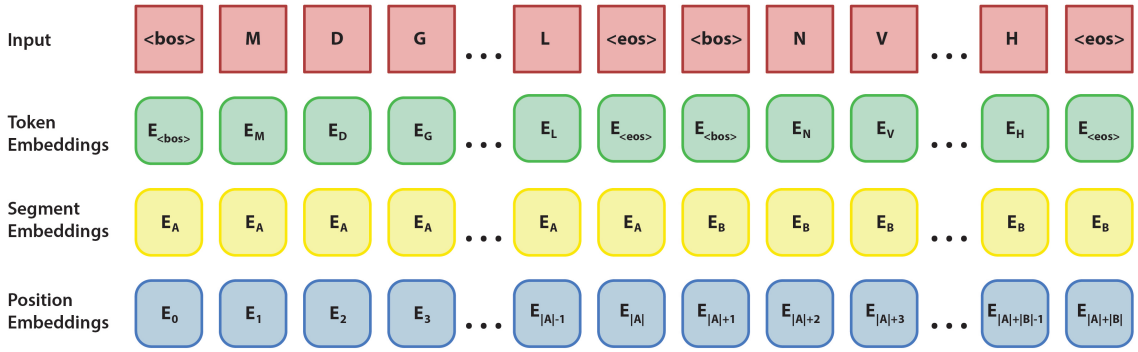


Figure 4.3: Input Embeddings

CoVBERT leverages the sinusoidal absolute positional encoding, similar to the original transformer model [19], due to its ability to generalise to sequences with longer lengths than the ones observed during the training phase which is critical whilst dealing with protein sequences. The following *sine* and *cosine* functions (Equation 4.6 and 4.7) of different frequencies are used where *pos* is the position, *d* is the dimension of embedding and d_{model} is the dimension of the models hidden layer. For any fixed offset $x \in \{1, \ldots, n\}$, PE_{pos+x} can be represented as a linear function of PE_{pos} such that $PE_{pos+x} = T^{(x)} PE_{pos}$ where $T^{(x)} \in \mathbb{R}^{d_{model} \times d_{model}}$ is a linear transformation and PE_{pos} is defined in Equation 4.8. These functions allow the transformer-based model to learn attending over relative positions [19] for $pos \in \{1, \ldots, n - x\}$ in the sequence *s* of length $|s| = n$.

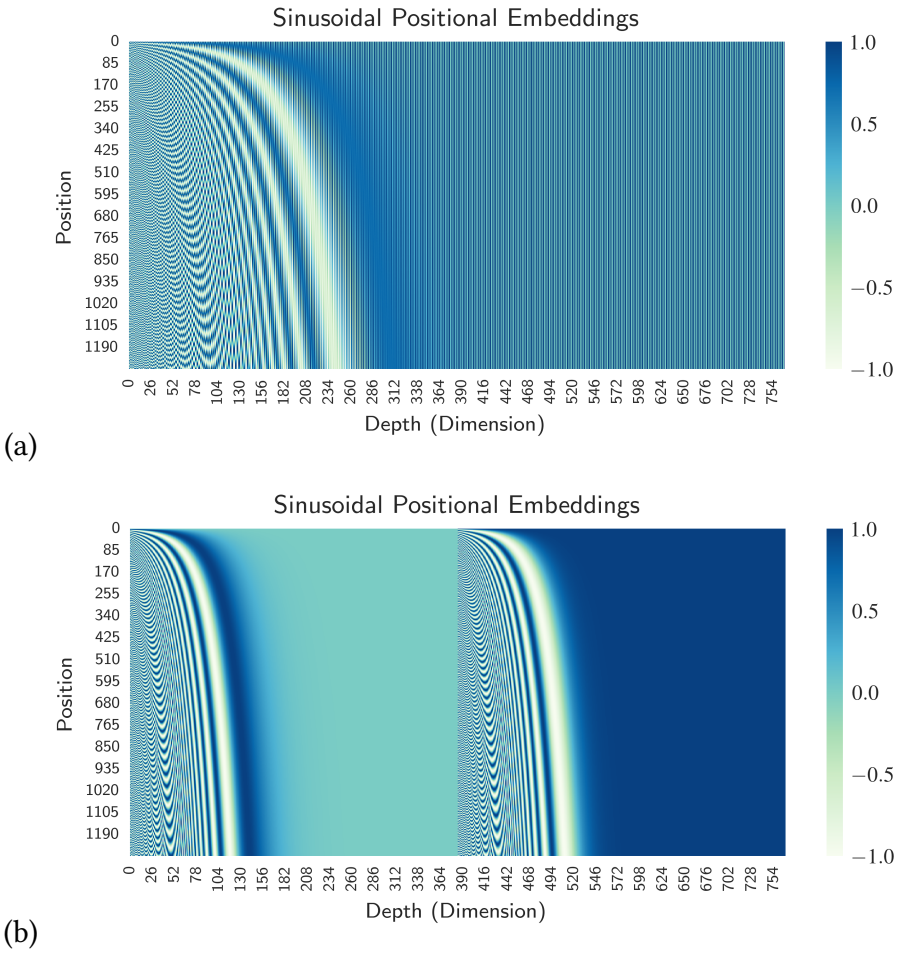
The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$ as shown in Figure 4.4 (a) for a protein sequence of length 1273 and CoVBERTs hidden layer dimension of 768. In Figure 4.4 (b) the values for the left-half are generated by a function which uses $sine$ and for the right-half the generation function uses $cosine$, which are then interweaved to form each of the positional encoding vectors. Each row here corresponds to the positional encoding for a vector such that the first row would be the vector which gets added to the embeddings of the first word s_1 in an input sequence S.

$$PE_{(pos,2d)} = \sin\left(pos/10000^{2d/d_{\text{model}}}\right) \tag{4.6}$$

$$PE_{(pos,2d+1)} = \cos\left(pos/10000^{2d/d_{\text{model}}}\right) \tag{4.7}$$

$$\boldsymbol{PE}_{pos,:} := \begin{bmatrix} \sin\left(\frac{pos}{f_1}\right) \\ \cos\left(\frac{pos}{f_1}\right) \\ \sin\left(\frac{pos}{f_2}\right) \\ \cos\left(\frac{pos}{f_2}\right) \\ \vdots \\ \sin\left(\frac{pos}{f_{d_{\frac{\text{model}}{2}}}}\right) \\ \cos\left(\frac{pos}{f_{d_{\frac{\text{model}}{2}}}}\right) \end{bmatrix} \tag{4.8}$$

$$where\ f_m = \frac{1}{\lambda_m} := 10000^{\frac{2m}{d_{model}}}$$



(a)

(b)

Figure 4.4: CoVBERTs Sinusoidal Positional Embeddings

Although, Rotary Position Embeddings (RoPE) [115] are being widely used in recent research owing to their ability for extrapolating well beyond the context window using

a multiplicative factor [116] [10], I chose not to use them because it has been observed that the performance improvement added by RoPE embeddings deprecates as the model size and training duration increases [116], resulting in minimal to no improvement than the vanilla sinusoidal positional embeddings. Moreover, RoPE substantially increases the computational costs as it multiplies each query and key vector inside self-attention with a sinusoidal embedding.

4.7 Masking Objective

The self-supervised masked language modelling pre-training objective for CoVBERT to reconstruct a corrupted sequence with masked tokens has been adapted from the original implementation introduced by BERT [78], with the addition of the masking procedure being performed dynamically, unlike the static one in BERT. This masked language modelling objective can be viewed as stochastic $maximum\, pseudo\, log\, likelihood\, estimation$ [117]. The masking objective models $p(x_{masked}|x_{unmasked})$ by randomly replacing tokens at numerous positions with masked or alternate tokens. The addition of a dynamic masking strategy wherein the masking pattern is generated each time a sequence is fed into the model allows learning of greater dependency knowledge and minimises overfitting [118].

For a given input sequence s_i, 15% of the tokens are randomly sampled i.e about 208 tokens for a SARS-CoV-2 spike glycoprotein sequence and these are masked according to the masking procedure f_{mask} in Equation 4.10. Setting the masked language modelling probability to 15% ensures that enough context is present for the model for reconstruction. The masking procedure over these sampled tokens changes them to the $< mask >$ token with 80% probability, or to a randomly selected alternate amino acid token with 10% probability, or the token is kept unchanged (i.e original amino acid) with 10% probability. For a given reference sequence $S = \{s_1, s_2, \ldots, s_N)$, wherein S is a set of amino acids and N is the length of the sequence, let $\Pi = \pi_1, \pi_2, \ldots, \pi_K$ denote indices of the masked tokens in a sequence S, where K is the number of masked tokens, S_Π is the set of masked tokens and $S_{-\Pi}$ is the set of context i.e unmasked tokens in S which gives $\frac{S_\Pi \cup S_{-\Pi}}{N}$. The objective of masked language modelling (MLM) assuming that the probability for predicting a masked token $< mask >$ is independent as show in Equation 4.9 and the negative loglikelihood of the true amino acid given the masked sequence is minimised, which is stochastic in nature.

$$L_{\text{MLM}}\left(S_\Pi \mid S_{-\Pi}\right) = \frac{1}{K}\sum_{k=1}^{K} \log p\left(s_{\pi_k} \mid S_{-\Pi}; \theta\right) \tag{4.9}$$

$$f_{mask}\left(s_i\right) = \begin{cases} < \text{mask} > & 80\% \quad \text{of cases} \\ s_i & 10\% \quad \text{of cases} \\ \text{Random}(AA) & 10\% \quad \text{of cases} \end{cases} \tag{4.10}$$

$$where\ AA \in \{L, A, G, V, S, E, R, T, I, D, P, K, Q, N, F, Y, M, H, W, C, X, B, U, Z, O\}$$

The masked language modelling approach implicitly tries to model a distribution over the protein sequence using conditional likelihood $p(s_{\pi_k} \mid S_{-\Pi})$. With the masked language objective, the model learns a latent variable $z_i \in \mathbb{R}^D$ by learning a function $f : S^{N-1} \rightarrow \mathbb{R}^D$ where $z_i \overset{\text{def}}{=} f(s_{[N]\setminus\{i\}})$ such that $p(s_{\pi_k} \mid S_{-\Pi}, z_i) = p(s_i|z_i)$.

The masked pre-training objective allows learning a high-dimensional representation of the sequence which can be used for predicting mutations at specific hotspots. The masked sequence $\widetilde{S} := f_{mask}(S)$ is encoded with the stacked transformer blocks (which are 6 for CoVBERT) and it represents each amino acid i.e character in the sequence as a hidden vector h_i. These final hidden vectors for the masked tokens are fed into an output softmax function. The probability for y_i in a mutated sequence $Y = \{y_1, y_2, \ldots, y_N\}$ at each masking position can be computed as the following:

$$P\left(y_i \mid \widetilde{S}\right) = \text{softmax}\left(W_o h_i\right) \quad \forall i \in S_{\Pi} \cup S_{-\Pi} \tag{4.11}$$

where W_o are the trainable parameters. The probability of all masked amino acids (characters) is:

$$P(Y \mid \widetilde{S}) = \prod_{i \in S_{\Pi} \cup S_{-\Pi}} P\left(y_i \mid \widetilde{S}\right) \tag{4.12}$$

For CoVBERT, additional auxiliary losses such as next sentence prediction objective as in the original BERT model was omitted since protein sequences unlike natural language do not form 'paragraphs' with any logical connections between sentences.

4.8 Self-supervised training

The encoder-based CoVBERT model has been trained in a self-supervised setting using a corpus encoded with BPE strategy, consisting of only raw protein sequences with the masked language modelling pre-training objective with the hyperparameters described in Table 4.1. CoVBERT was implemented in PyTorch [119] and trained with a global batch size of 8 for over 5000 optimisation steps. CoVBERT uses AdamW optimiser [120] (Equation 4.13) (Algorithm: Appendix .1) to minimise the log loss i.e cross-entropy as the update rule with a learning rate of 5e-5, decoupled weight decay of 1e-2, $\beta 1$ = 0.9, $\beta 2$ = 0.999, ϵ = 1e-08, context window of size $|n_{ctx}| = 2048$ tokens, and a 0.1 dropout probability which follows

the residual connections for each fully connected layer in the encoders. The loss is taken as the whole batch average cross entropy loss between the model's predicted token and the true token for the 15% masked tokens in a sequence.

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / \left(\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon \right) + \lambda \boldsymbol{\theta}_{t-1} \right) \tag{4.13}$$

CoVBERT uses Gaussian Error Linear Units non-linearity [107] (Equation 4.14) instead of the standard ReLU, where $x\Phi(x)$ is the standard Gaussian cumulative distribution function and erf is the error function:

$$\mathrm{GELU}(x) = x \cdot \mathbb{P}_{X \sim N(0,1)}[X < x] = x\Phi(x) = x \cdot \frac{1}{2}[1 + \mathrm{erf}(x/\sqrt{2})] \tag{4.14}$$

The CoVBERT model consists of 6 stacked transformer encoder blocks with 12 attention heads in each attention layer, similar to DistilBERT [121] and TinyBERT [122], resulting in a small model with 44 million parameters, which is 66% smaller than the BERT-Base model [78]. The dimensionality of the hidden layers is set to 768 for better representational capability. CoVBERT has a context window of 2048 tokens similar to GPT-3 [58], and during training it utilises the full $|n_{ctx}| = 2048$ tokens context window, packing multiple sequences together in order to increase the computational efficiency. A zero-centered normal distribution with a standard deviation of 0.02 served as the initial distribution for all transformer layers s.t for $\exists l \in L, L \sim \mathcal{N}(0, 0.0004)$, where $\mu = 0$ and $\sigma^2 = 0.0004$. The initial value of each bias was set to 0.

The training over our training set of 40,000 raw SARS-CoV-2 spike protein sequences took nearly 13 hours with a batch size of 8 per device, and was performed using NVIDIA Tesla T4 Cloud GPU and Advanced Research Computing 48-core Xeon CPU (ARC, [123]). Whilst training, no additional labels were given related to the function or structure of the protein, and the training was performed over the raw SARS-CoV-2 protein sequences only in a self-supervised setting. The training pseudocode is defined below in Algorithm 13.

Hyperparameter	CoVBERT
Dataset	GISAID
# of Training samples	40,000
Model Parameters	44 million
# of Hidden Layers	6
# of Attention Heads	12
Hidden Layers Intermediate Size	768
Masking Probability	0.15
Positional Encoding Limits	2,048
Positional Embedding Type	Sinusoidal Abs
Learning Rate	5e-5
Dropout	0.1
Optimiser	AdamW
Weight Decay	1e-2
Training Steps	5,000
Global Batch Size	8
$\beta 1$	0.9
$\beta 2$	0.999
ϵ	1e-08
Initialiser Range σ	2e-2
Hidden Activation	GELU

Table 4.1: Proposed CoVBERT model hyperparameters

Algorithm 5 Training: Masked Language Modelling [110]

Input: $\{x_n\}_{n=1}^{N_{data}}$, dataset of sequences
Input: θ, initial encoder-only transformer parameters
Output: $\hat{\theta}$, the trained parameters
Hyperparameters: $N_{epochs} \in \mathbb{N}, \eta \in (0, \infty), p_{mask} \in (0, 1)$

for $i = 1, 2, \ldots, N_{epochs}$ **do**
  for $n = 1, 2, \ldots, N_{data}$ **do**
    $\ell \leftarrow length(x_n)$
    for $t = 1, 2, \ldots, \ell$ **do**
      $\hat{x}_n[t] \leftarrow mask_token$ or $x_n[t]$ randomly with probability p_{mask} or $1 - p_{mask}$
    end for
    $\hat{T} \leftarrow \{t \in [\ell] : \hat{x}_n[t] = mask_token\}$
    $P(\theta) \leftarrow CoVBERT(\hat{x}_n|\theta)$
    $loss(\theta) = -\sum_{t \in \hat{T}} logP(\theta)[x_n[t], t]$
    $\theta \leftarrow \theta - \eta \cdot \nabla loss(\theta)$
  end for
end for
return $\hat{\theta} = \theta$

For validation, 10,000 unseen sequences were used to gauge the generalisation of the model over this held-out set. The training and validation loss curves for the optimisation steps are shown in Figure 4.5, which reflects that the model has a good fit learning curve as the training and validation loss decrease to a point of stability with a low generalisation gap of $8e-3$ between the final loss values.

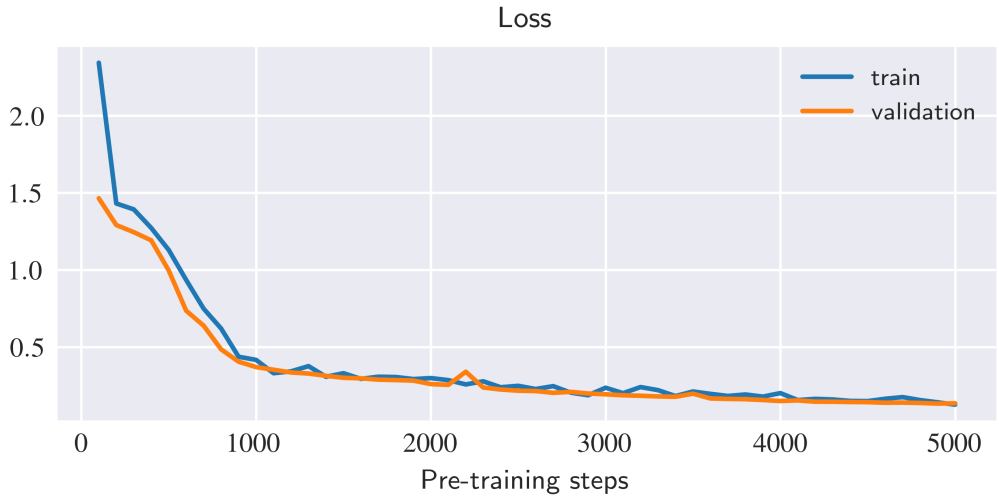


Figure 4.5: CoVBERT Training Loss Curve

5 CoVBERT - Experimental Results

5.1 Zero-shot SARS-CoV-2 mutation prediction

This section aims to show that CoVBERT, a transformer based protein language model is able to capture state of the art mutational effects of sequence variation for SARS-CoV-2 with zero-shot inference, involving no further training or supervision from experimental data. Specifically, the goal is to evaluate the performance of CoVBERT which has been trained over a dataset of diverse and evolutionary spike protein sequences for performing in vitro mutagenesis and predicting indel substitutions at mutational hotspots for SARS-CoV-2, in terms of pseudo log likelihood. An approach similar to zero-shot transfer to that of GPT-3 described in [58], has been undertaken.

The proposed model CoVBERT, a 44 million parameter model, is pre-trained to output the probability for predicting an amino acid for each masked position $S[i]$ for k masked tokens. Herein, CoVBERT protein language model model has been evaluated for the task of generating mutations at given mutational hotspots for SARS-CoV-2 over the validation set of 10,000 samples retrieved from the Global Initiative for Sharing All Influenza Data (GISAID) [6]. Further, CoVBERT's masked marginal log likelihood is compared against extensive baselines including state of the art pre-trained protein language models such as DistilProtBERT [95], ProtBERT-BFD and ProtElectra generator [17]. The list of common mutational hotspots was fetched from Outbreak Info [124] which is an open-source database of COVID-19 epidemiology data.

Zero-shot transfer refers to transfer of a pre-trained model to a specific task with no supervision to specialise the model for that task. In the zero-shot setting, no gradient updates

take place to specialise the model i.e $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1}$. Pre-trained model's frozen weights are utilised to gauge the performance of the protein language model for the specified task of in vitro mutagenesis. The claim is not of an out-of-distribution generalisation, in line with Brown et al. [58]. It is assumed that during the pre-training phase, the model picks up knowledge pertinent to the tasks it would later be applied to. The pre-training dataset for CoVBERT retrieved from the Global Initiative for Sharing All Influenza Data (GISAID) [6] and for other state of the art protein language models trained over millions of sequences from UniRef 50 or UniRef 100 [14], contain sequences from several evolutionary periods for SARS-CoV-2 virus, suggesting that the pre-trained model may have encountered examples of sequences from the protein families on which it would be evaluated.

Trained with the masked language modelling objective, the proposed model CoVBERT and the extensive baseline protein language models are self-supervised to output masked marginal log likelihoods for an amino acid to appear at a position in a given protein sequence, provided with the surrounding bidirectional context in the form of local and global representations. This capability is used to score the mutations in sequences at a given mutational hotspot by evaluating the probabilities which are computed using: $softmax(logits)$ that represents the posterior for the amino acid at position i. For a given mutational hotspot position, the amino acid present at that index in the wildtype protein is considered as the reference state which is used for comparing the probability assigned to the mutated amino acid in the sequence with the probability assigned to the wildtype amino acid.

Zero-shot setting involves use of inference directly on the protein sequence and involves use of frozen pre-trained weights of the models as feature extractors. In the zero-shot setting, only forward passes are made to the model during the inference stage, and no gradient updates are made, which means that the frozen weights of the pre-trained model are leveraged to perform inference over a specified task. CoVBERT model has been trained in a self-supervised setting over raw spike protein sequences for SARS-CoV-2 with a training dataset of size 40,000 sequences with an average length of 1,273 amino acids spanning over evolutionary sequences from SARS-CoV-1 to SARS-CoV-2. No further supervision from experimental measurements of functions or fine-tuning was performed.

5.1.1 Baseline Protein Language Models

For the task of evaluating performance of protein language models to predict in vitro mutagenesis i.e mutational indels for SARS-CoV-2, three state of the art protein language models were taken as baseline models to compare them with CoVBERT. These state of the art protein language models include ProtBERT [17], DistilProtBERT [95] and ProtElectra generator [17]

Hyperparameter	CoVBERT	DistilProtBERT	ProtBERT-BFD	ProtElectra
Dataset	GISAID	UniRef 50	BFD	UniRef 100
# of Training samples	40,000	43 million	2.1 billion	216 million
Model Parameters	44 million	230 million	420 million	420 million
# of Hidden Layers	6	15	30	32
# of Attention Heads	12	16	16	20
Hidden Layers Intermediate Size	768	1,024	1,024	1,280
Masking Probability	0.15	0.15	0.15	0.25
Positional Encoding Limits	2,048	40,000	40,000	40,000
Learning Rate η	5e-5	5e-5	2e-3	2e-3
Dropout	0.1	0.0	0.0	0.0
Optimiser	AdamW	AdamW	Lamb	Lamb
Training Steps	5k	300k	300k	400k

Table 5.1: Proposed CoVBERT model and pre-trained state-of-the-art baseline models hyperparameters

which have been trained over millions of clustered sets of protein sequences spanning multiple viruses and protein families including SARS-CoV-2 protein sequences for all regions. Table 4.1 compares the hyperparameter configurations of these baseline models with that of CoVBERT.

5.1.1.1 ProtBERT-BFD

ProtBERT-BFD [17] which is a BERT-based autoencoder model [78], has been pre-trained with a static masked language modelling objective (15% masking rate) over Big Fat Database (BFD) [125]. BFD is one of the largest databases for protein families including 2.1 billion protein sequences from Uniprot/TrEMBL [126] and Swissprot [127] which includes sequences covering SARS-CoV-2. ProtBERT-BFD was trained over these 2.1 billion sequences consisting of 393 billion amino acids with a learning rate of $\eta = 2e - 3$ using the Lamb optimiser [128]. Lamb, which is a large batch optimisation technique, allows training large batch sizes without any performance degradation. ProtBERT-BFD was first trained for about 800k gradient steps, then for additional 200k steps with protein sequences with a maximum length of 512 and 2k, respectively. The use of larger batch sizes and the models ability to initially extract useful features from shorter sequences, enables the model to train on longer sequences more effectively as a whole. The embeddings from ProtBERT are able to capture biophysical features of protein sequences, which they validate with tasks such as per-residue prediction of protein secondary structure, protein subcellular localisation and membrane vs. water-soluble. On a foundational level, it can detect even more subtle signals of short functional motifs. ProtBERT is only able to work with capital amino acids (alphabets) as it has been trained over uppercase amino acids.

5.1.1.2 DistilProtBERT

model. It has been pre-trained for three epochs over 43 million sequences from UniRef 50 [14] with length of protein sequences ranging between 20 and 512 amino acids. The knowledge distillation approach of DistilProtBERT allowed reduction of the model network size, halved running time and low computational resources requirement for pre-training by 98% relative to the ProtBERT model. The performance of DistilProtBERT is at par with ProtBERT over two published benchmarks including secondary structure prediction and membrane bound vs. water soluble prediction. Further, it has been tested to distinguish between real and random protein sequences. DistilProtBERT leverages the advantage of a large pre-trained model i.e a teacher network in the knowledge distillation mechanism to train a simpler yet robust model. It has been pre-trained with its weights initialised to that of ProtBERT for utilising the knowledge distillation teacher-student network and the training objective of static masked language modelling is set to 0.15. It uses the loss function described by Sanh et. al., [121] which comprises three equally weighted components including L_{mlm} , L_{ce} and L_{cos}, and this loss function is used to impart knowledge from the teacher network i.e ProtBERT to the student network, DistilProtBERT.

5.1.1.3 ProtElectra

ProtElectra [17] is an Electra-based [129] model consisting of a generator and discriminator model including the same number of layers, with the generator being 25% of the discriminator's hidden layer size, hidden layer intermediate size and number of heads. It has been pre-trained over 216 million protein sequences from UniRef 100 [14] which span over numerous viruses and protein families including SARS-CoV-2 protein sequences for all of its regions. The training has been performed in two phases with the Lamb optimiser [128] using a learning rate of 2e-3 ($\eta = 2e - 3$). The first training phase includes protein sequences of length at most 512, trained for over 400k gradient optimisation steps at a 9k global batch size, and the second for proteins having at most 1,024 residues for another 400k optimisation steps with a 3.5k global batch size. Electra trains two networks, a generator and a discriminator, in an effort to increase the pre-training task's sampling efficiency. The generator (BERT) reconstructs masked tokens potentially producing plausible alternatives, in addition to only rebuilding corrupted input tokens, while the discriminator (Electra) determines whether tokens were masked. As a result, the training signal is enhanced because the loss can now be calculated over all tokens rather than just the subset of corrupted tokens (which is 25% here).

5.1.2 Results

For the task of in vitro mutagenesis and evaluation of mutational substitution indels over a list of hotspots positions associated with viral escape, the proposed protein language model CoVBERT and other state of the art protein language models such as DistilProtBERT [95], ProtBERT-BFD and ProtElectra generator [17] have been assessed with their masked marginal log likelihood in a zero-shot setting, over the curated validation set of 10,000 SARS-CoV-2 spike protein sequences from GISAID. Specifically, the analysis focuses over the spike protein sequence mutation hotspots, as mutations in spike protein affect the transmissibility of the virus and its immune escape. A list of common mutation hotspots causing immune escape were retrieved from Outbreak Info [124], which is an open-source database of COVID-19 epidemiology data.

The scoring metric used here for the analysis is masked marginal log likelihood which is defined as the sum of the cross-entropy between the true residue at each selected location (mutation hotspot) and the predicted distribution over probable residues, conditioned on the previous and future characters (local and global representations). The log-ratio of likelihoods of individual sequences can be used to quantify the plausibility of mutant sequence x (Mutant) relative to its wild-type, un-mutated counterpart x, once the model has been fitted to a family of protein sequences (Wild-type). This protein log likelihood estimates the joint probability of observing a given sequence of amino acids (i.e tokens). Intuitively, this estimator approximates the probability of a mutated protein sequence being "natural", and if it can be effectively produced by a cell.

CoVBERT and the baseline protein language models have been pre-trained with the masked modelling objective to predict a "masked" amino acid position in a given sequence S, which emanates in the provision of an estimate of probability to observe an amino acid at the masked positions, given the surrounding context. The mask noise during pre-training influences the extraction of probabilities. This log-ratio has been found to predict the impact of mutations [76] [21]. Meier et al. [22] discovered that the log-likelihoods of large protein language models are capable of predicting mutational effects. Mathematically, the log-ratio of likelihoods of individual sequences is represented as:

$$\log \frac{p\left(\boldsymbol{x}^{(\text{Mutant})} \mid \boldsymbol{\theta}\right)}{p\left(\boldsymbol{x}^{(\text{Wild-type})} \mid \boldsymbol{\theta}\right)} \tag{5.1}$$

The masked marginal log likelihood for two of the most contagious mutations for SARS-CoV-2 namely, $D614G$ and $E484K$ were evaluated using CoVBERT and the baseline protein language models. The D614G and E484K mutations are the two most prevalent mutations

amongst the SARS-CoV-2 virus. The D614G mutation contributes to the pathogenicity of the virus by enhancing its ability to bind to cells, and this mutation has been responsible for the high transmissibility and virulence of variants including Alpha (B.1.1.7), Beta (B.1.351), Gamma (P.1), Delta (B.1.617.2) and Omicron (BA.x, B.1.1.529) lineages [96]. The E484K mutation enhances the replication of SARS-CoV-2 by facilitating an increase in both viral RNA production and protein synthesis, emanating in critical variants such as Beta (B.1.351) and Gamma (P.1) lineages [130].

To perform an evaluation of the models over the selected mutational hotspots $D614G$ and $E484K$, sequences in the validation set having the original amino acid at the mutation position were masked such that $S[pos-1] :=< mask >$ (assuming tokens list to be zero-indexed). These masked sequences were then evaluated with CoVBERT and the baseline models for their log likelihood to generate the correct mutation, given the unmasked context.

Over the validation set of 10,000 sequences, CoVBERT was able to predict the $D614G$ mutation with a masked marginal log likelihood of 0.95 and $E484K$ mutation with 0.12, in a zero-shot setting, as shown in Figure 5.1. Whereas, the baseline models showed lower likelihood scores for both the mutations $D614G$ and $E484K$. DistilProtBERT [95], which is a 230 million parameter model pre-trained over 43 million sequences from UniRef 50 [14] was able to predict $D614G$ mutation with a masked marginal log likelihood of only 0.02 and $E484K$ with 0.07. ProtBERT-BFD, which is pre-trained over 2.1 billion sequences [17] predicted log likelihood for the $D614G$ mutation as 0.10 and for $E484K$ as 0.01. Lastly, ProtElectra generator [17] predicted the $D614G$ mutation to have a log likelihood of 0.09 and $E484K$ mutation with 0.01.

Although the baseline models are bigger in terms of parameters and they've been trained with a greater number of viruses' protein sequences spanning multiple families including SARS-CoV-2 taken from UniProt database [126], they seem to be overfitting when evaluating them over common SARS-CoV-2 mutations in the spike protein region. Comparatively, CoVBERT, a protein language model with 44 million parameters, which is a simpler and smaller model than the baseline models trained over a specific dataset of 40,000 sequences only, outperforms the baseline protein language models for the task of predicting existing mutations for SARS-CoV-2. CoVBERT serves as a model specialising over SARS-CoV-2 spike protein families. CoVBERT's ability to predict $D614G$ with 0.95 marginal log likelihood reflects the capability of large language models to decipher phylogenetic relationships and in vitro mutagenesis for a given protein sequence.

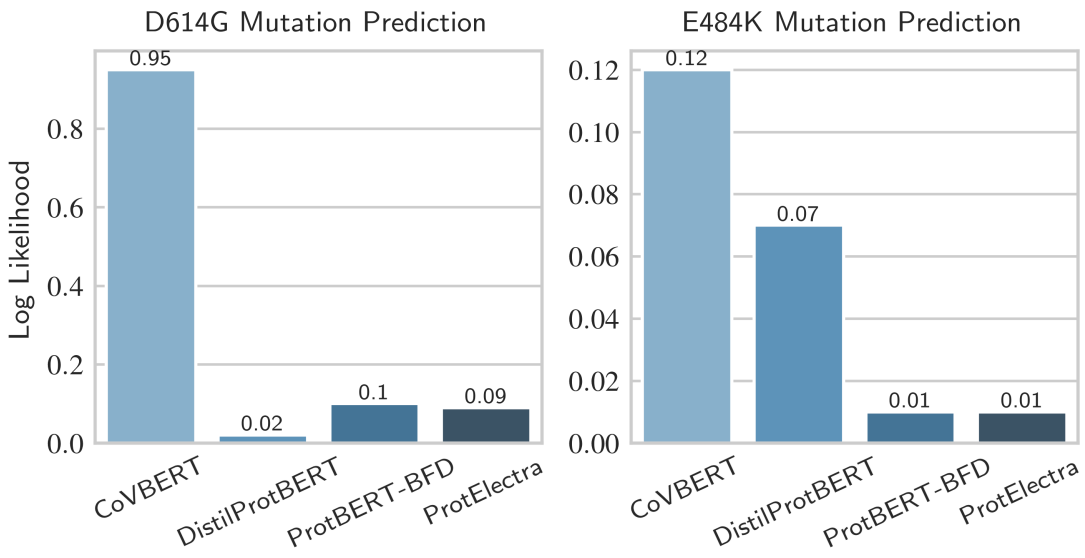


Figure 5.1: Point Mutation Prediction Log Likelihood

This shows that using only zero-shot inference, without the need for any task-specific supervision from additional training, the transformer encoder based model, CoVBERT was able to capture mutational effects of sequence variation and mutational landscape over common SARS-CoV-2 indel substitutions, performing at the state-of-the-art level. This reflects that language models trained over protein sequences are able to capture the language of life, specifically protein sequences, and their capability to deduce the vocabulary of amino acids which are an amalgamation of individual alphabets.

Lastly, CoVBERT is the only protein language model till date specialising over SARS-CoV-2 protein sequence to reason about the mutational landscape by predicting amino acids at mutational hotspots. CoVBERT was able to exceed the performance of bigger protein language models in particular when extrapolating to mutated SARS-CoV-2 spike proteins that are further away in the models embeddings sequence space.

5.2 Interpreting Self-Attention in CoVBERT

The ability of CoVBERT to predict mutational hotspots sheds light over its capability to learn useful representations for grasping phylogenetic relationships to perform in vitro mutagenesis for the SARS-CoV-2 spike protein. In this section, the interactions and representations learnt by the self-attention mechanism through the multi-headed structure in the multiple transformer encoder layers of CoVBERT have been interpreted through the lens of attention, as it is otherwise hard to decipher what the model has learnt that propels it to encapture

the mutational relationship amongst amino acid positions in a SARS-CoV-2 spike protein sequence. A benefit of using attention based models is that they can be interpreted by gauging how the model assigns weights to different tokens in the input sequence [131] [132], albeit its value in explaining distinct predictions could be limited [133].

Recent work has demonstrated analysis of attention in transformer models apart from the NLP domain for proteins and atoms [134] [28]. Although transformer based models perform well, the opaqueness of their interpretability might mask underlying issues like model bias and spurious correlations [135] [136]. To understand the robustness and applications of attention mechanisms specific to protein related tasks with a focus on SARS-CoV-2, herein we explore the ability of attention to decipher properties of proteins and show that attention: **(1) determines identical or related amino acids providing insights over protein contact maps**, **(2) binds amino acids that are physically apart in the underlying sequence but close together in the three-dimensional structure of proteins, capturing the folding structure of proteins.**

CoVBERT takes as input a sequence S of amino acids (i.e characters) s.t $S = \{s_1, s_2, \ldots, s_N\}$ for a sequence of length N ($|S| = N$), and it applies a series of encoder layers (i.e six for CoVBERT) with 12 self-attention heads within each of these blocks. An encoder layer l returns a sequence of continuous embeddings: $H = \{h_1^{(l)}, \ldots, h_n^{(l)}\}$ using multi-headed attention. A set of attention weights α is produced by each attention head in a layer l for an input sequence S, wherein $\alpha_{i,j} > 0$ is the attention from amino acid token i to token j, s.t $\sum_j \alpha_{i,j} = 1$. Transformer-based CoVBERT uses scaled dot-product attention, wherein the distribution of attention at an amino acid position i for a given input sequence S is defined as follows:

$$\alpha_i = softmax(\frac{q_i \cdot k_1}{\sqrt{d}}, \frac{q_i \cdot k_2}{\sqrt{d}}, \ldots, \frac{q_i \cdot k_N}{\sqrt{d}}) \tag{5.2}$$

where q_i is the query vector at amino acid position$i$ that is paying attention, k_j is the key vector at position j which receives attention, d is the dimension of q and k s.t $\{q, k\} \in R^d$, and N is the length of sequence S ($|S| = N$). To perform the analysis, the protein sequence for SARS-CoV-2 Alpha variant (B.1.1.7) A-chain has been used.

CoVBERT with 6 encoder layers and 12 attention heads per layer, consists of 72 distinct attention structures inclusive of attention for each layer and head in the model's architecture. This is elucidated with Figure 5.2 which provides a birds-eye view over the attention spanning all of the models layers and heads for a particular input sequence S. For each layer, all 12 attention heads operate in parallel to each other. Each of these attention heads compute weighted sums for every token over all other tokens in the sequence space, i.e., all residues or

amino acids in a protein are compared to each other. The lines represent the self-attention amongst a pair of tokens $\{i, j\}$, connecting tokens i (left) to tokens j (right) which are being attended. The width of the lines reflect the intensity of attention weights for a particular connection. Figure 5.3 and 5.4 display the weighted attention across all heads for each of the encoder layers, for the first 20 tokens of the sequence.

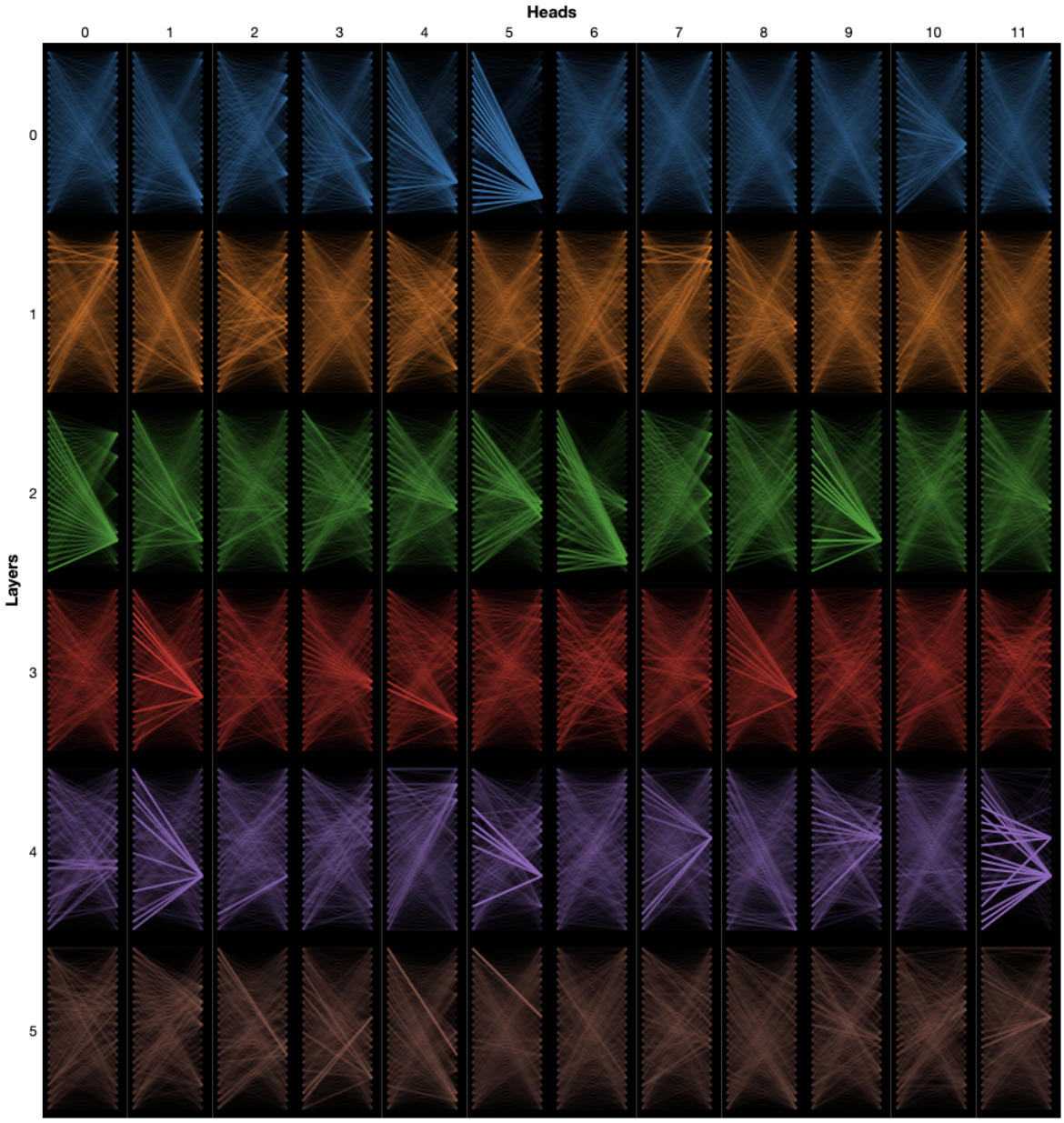


Figure 5.2: CoVBERT Layer-wise Attention head visualisation

Each attention head learns a different attention mechanism since the attention heads do not share parameters, and this pattern can be observed in Figure 5.2. This allows each of

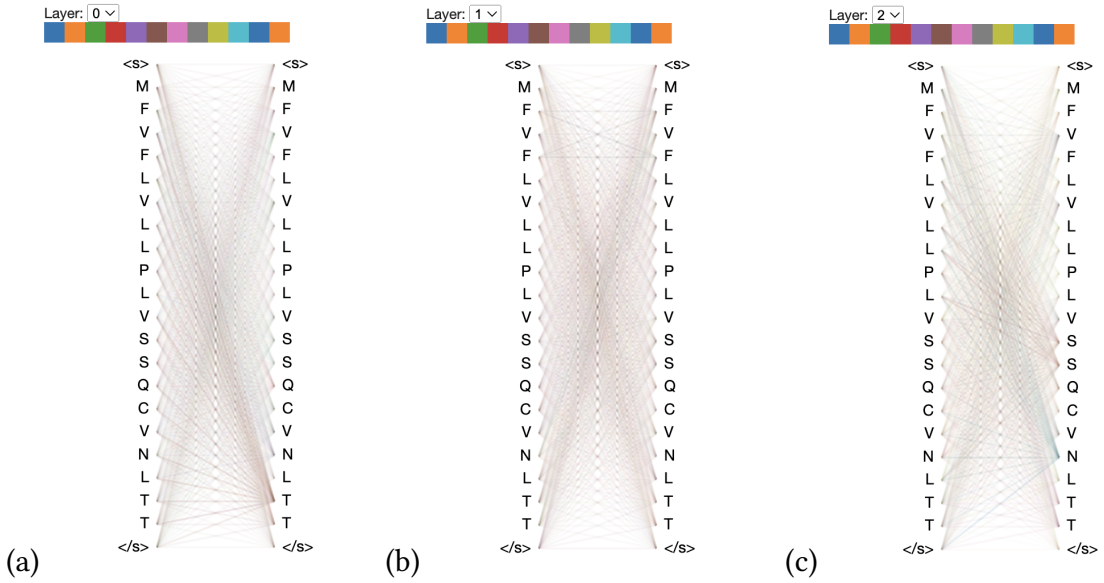


Figure 5.3: CoVBERT Attention head (mean) visualisation for layers 1-3

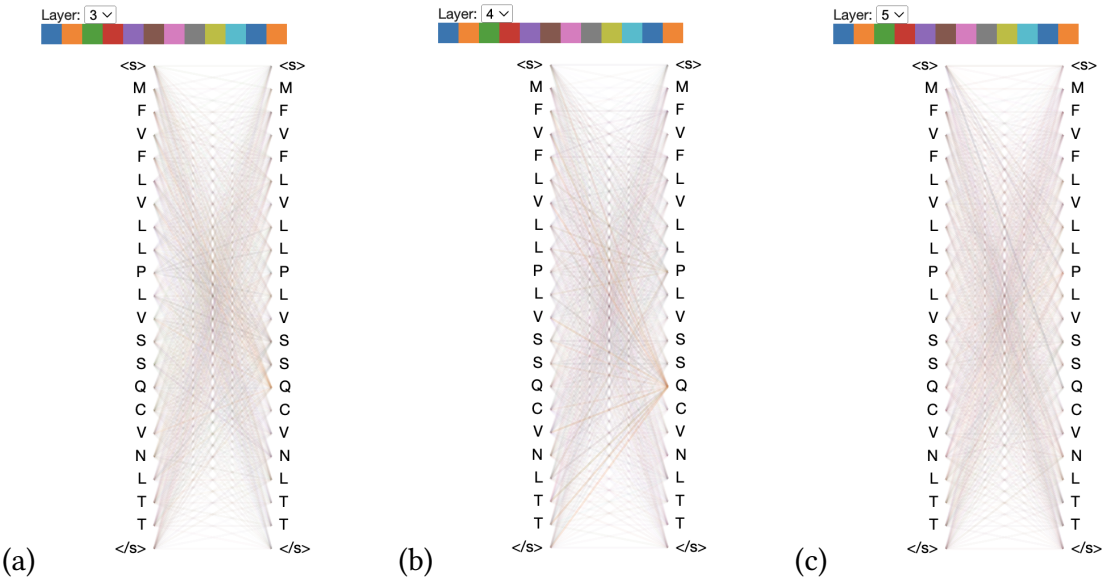


Figure 5.4: CoVBERT Attention head (mean) visualisation for layers 4-6

the heads to specialise in learning a rich array of representations amongst the tokens in a sequence S, which would not have been possible with a single attention head.

Each layer operates over the output of the previous layer, which allows CoVBERT to form a rich representation as it gets to the deepest layer of the model. This is corroborated by the observation in Figure 5.2 that deeper layers in the network are able to capture the pattern of dispersing attention over identical or related words in the sequence. The first layer (0) appears to be dispersing attention over the previous tokens, and the other initial layers (1-3,

considering zero indexing) mostly encapture attention over previous and future tokens. The final layers (4-5) display the pattern of encapturing attention amongst identical or related amino acid tokens. Figure 5.5 (a) and (b) show this pattern existing in layer 5 for a single token L and S respectively, which disperses higher attention weights over the same or related tokens in the embedding vicinity.

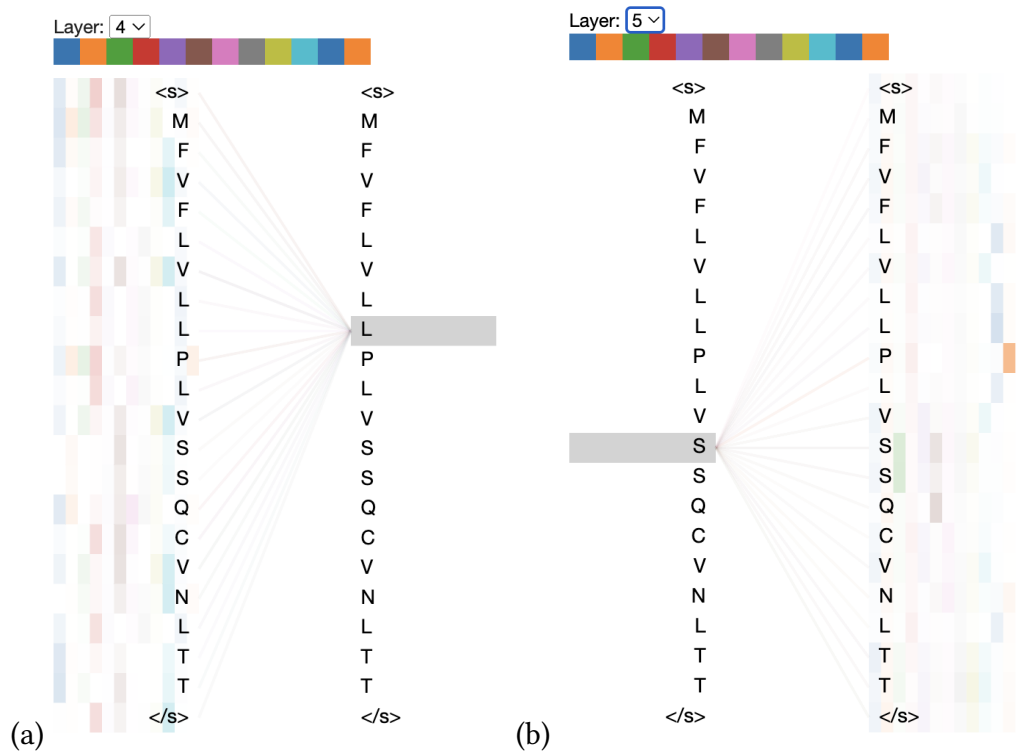
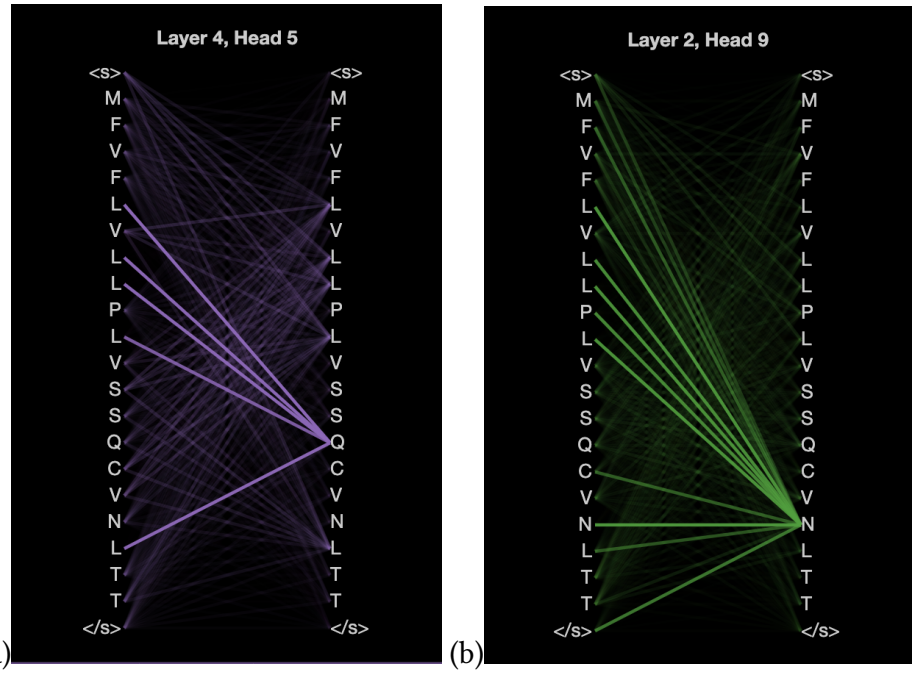


Figure 5.5: (a) Penultimate layer attention head visualisation for token L (b) Final layer attention head visualisation for token S

In the deeper layers, a pattern of attention decay based on distance can be observed as shown in Figure 5.6 (a) and (b) for layer 2 and layer 4 respectively, for a few tokens. This distance decaying pattern occurs when the element-wise product of vectors $q_i \cdot k_j$ decreases as the distance from the source token increases (depicted as faded lines), which occurs due to either a higher decay rate or a shorter context window size.

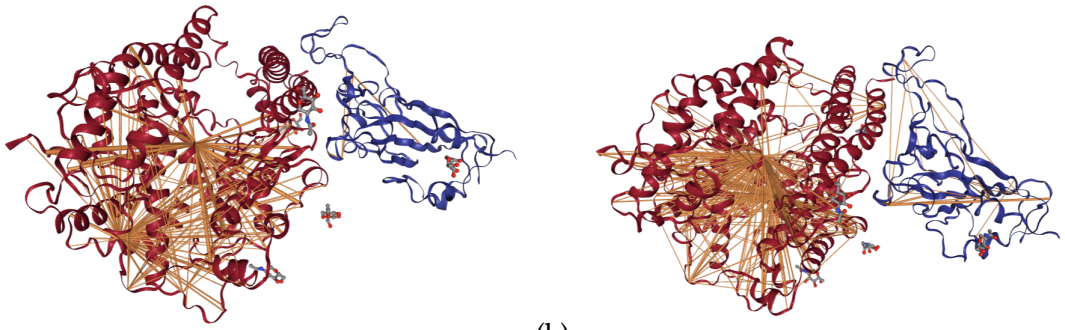
The three-dimensional structures in Figure 5.7 (a) and (b) represent the structure for SARS-CoV-2 spike protein receptor-binding domain bound with ACE2 receptor (PDB:6M0J [137]) which marks the point of entry for the virus into a human's host cells. The attention for this structure is studied specifically in the final layer in the 6th and 12th head respectively, which target the binding sites which is a key functional component of proteins. These



(a) (b)

Figure 5.6: (a) CoVBERT Attention head interpretation for Layer 4, head 5 (b) CoVBERT Attention head interpretation for Layer 2, head 9

examples show how specialised attention heads in transformers can recover the structure and function of a protein, based on the language model pre-training alone. The orange lines depict attention between amino acids wherein the line widths are proportional to the attention weights (values below 0.1 are hidden). This inner working of transformer models attention heads can be utilised for analysing single proteins in detail and the attention scores have been shown to correlate with residue interactions, indicating that they may be indicative of a more fundamental biological truth [138]. Evidently, the attention mechanism is able to capture high-level structure features of proteins by connecting amino acids that are spatially adjacent in the three-dimensional structure but are separated in the underlying sequence. This demonstrates how pre-trained Transformer models may be utilised for inexpensive and quick analysis, perhaps paving the way for novel hypothesis generation methods.



(a) (b)

Figure 5.7: CoVBERT model Attention interpretation for protein structure

Part II

Large Protein Language Models are Self-supervised Evolutionary Learners

6

MuFormer - Structure and Sequence based Mutation Generation Model

6.1 Introduction

Self-supervised pre-training on protein sequences has resulted in cutting-edge performance in protein function and fitness prediction [15] [7]. However, sequence-only techniques overlook the wealth of information contained in actual and predicted protein structures. The proposed model, MuFormer, leverages properties of both the sequences and protein structures to design *de novo* protein sequences through an iterative gradient-based design process of curating mutations. At each design step, MuFormer maximises the likelihood of amino acids appearing in each position in a sequence. The proposed architecture includes a state-of-the-art pre-trained protein language model for generating sequence embedding which is fed as an inductive bias to an inverted AlphaFold2 [4] for the task of protein design in an iterative setting through fixed backbone design. This work shows that MuFormer is able to encapture evolutionary and mutational relationships through the sequence-only protein language model and MSA profile generated by AlphaFold2, for generating *de novo* protein sequences for SARS-CoV-2. The generated mutations are validated with historical data, showcasing the ability of MuFormer to design the sequence for Omicron and Delta variant, given the sequence embedding and structure for just Alpha variant. These findings could help towards therapeutics design for ameliorating the effects of any such probable dangerous mutation with high fitness and stability, in advance.

Evolution amongst protein structures is mostly neutral (i.e conservative) and most of the mutations usually occur within protein sequences. Learning from natural evolutionary

tactics may speed up artificial evolution since natural evolution must search an exponential landscape of potential sequences for desirable but uncommon mutations. Using the best of both, MuFormer seeks to leverage the mutability between the sequences and the protein backbone atom guidance from the structure to allow the model to explore sequences within the evolutionary space. The pre-trained language model as an encoder, extracts potential associations amongst the protein sequence tokens and information about proteins functionality along with structural properties from the sequence data alone. Use of AlphaFold2 as a structure prediction oracle for performing inverse folding aims to reconstruct a protein's amino acid sequence by inculcating mutations in positions with low sequential or structural likelihood, given a single protein structure only. Without any information about target sequence or structure, MuFormer exploits the phylogenetic information from large pre-trained language and structural models to evolve a protein sequence.

MuFormer employs two distinct loss objectives namely, fixed backbone design loss and the newly introduced backbone atom optimisation loss function. Over a set of metrics for gauging the fitness and stability of protein sequences and structures, MuFormer outperforms vanilla AlphaFold2 which suggests the evolutionary relationships injected by the protein language model for the task of mutational sequence design. Specifically, MuFormer is an approach for fixed backbone design using numerous features of AlphaFold2 along with rich information from the protein language model embeddings, for iteratively designing protein sequences that have a high confidence to fold to specified protein backbone atoms. At each iteration of the design process, a sequence is sampled from the predicted protein structure to bias the design towards the backbone conformations and the embedding representation which is rich in information about proteins structural and functional properties [15] [16]. By iteratively mutating the sequence at each design step, we obtain the final sequence design with a predicted structure that fits the backbone atoms configuration with high confidence. It has been shown that AlphaFold2 is capable of learning an energy function to design a low energy conformation of the protein [139], and this serves as a key to the success towards MuFormers success in performing robust sequence design by minimising the folding energy for a backbone conformation.

The existing approaches for inverse folding to generate protein sequence designs usually make use of an energy function and a sampling sequence space for finding low energy conformations and a tremendous amount of work has gone into the creation of parameterized energy functions to serve as a design guideline [140] [141] [142]. However, MuFormer is able to perform fixed backbone design by leveraging the success of AlphaFold2 along with large pre-trained protein language models. This reflects that extracting information from sequence

level data helps in guiding the sequence design process to explore mutational conformations with the embedding representations, proteins structural and functional information along with the MSA profile generated by AlphaFold2.

6.2 MuFormer - Proposed Model Architecture

MuFormer leverages embeddings from an encoder-based protein language model and these embeddings are injected as an inductive bias for the fixed backbone sequence design using AlphaFold2 [4], to directly predict the sequence as well as the atomic coordinates that define the protein structure of the generated mutant. MuFormer conditions over sequence embeddings and a protein structure to generate a mutational sequence along with a full-atom backbone configuration. To do this, MuFormer combines a transformer-based protein language model with the inverted implementation of AlphaFold2 (AFDesign [143] citemirdita2022colabfold) to perform inverse folding for a given protein structure which is guided by the inductive bias from the encoded embeddings captured by the encoder. Augmentation of AlphaFold2 with sequence-only protein language model allows it to process more detailed representations of protein structures guided by the functional properties captured by protein language models [138]. MuFormer takes into account the representation of the protein sequence as well as the geometry of the protein backbone together to guide the protein design process.

MuFormer is composed of a protein language model connecting the amino acid embeddings to an inverted AlphaFold2 model. The proposed architecture as shown in Figure 6.1 includes a protein language model which takes a sequence S as input to compute the embeddings for individual AAs, which are then added as a bias to the fixed backbone design process using AlphaFold2 for designing mutated sequences and their respective structures. The inverse AlphaFold2 network is responsible for learning complex relationships between the structure and functions from the encoded embeddings, with the help of the protein language model. MuFormer employs an end-to-end training process involving backpropagation of gradients to update the weights of the fixed backbone design process. This allows generation of optimised mutational sequences in lines with the guided inductive bias by the encoded embeddings, which enables the model to make use of intrinsic structural, functional and sequence-level properties for *de novo* protein design whilst leveraging the predictive power of AlphaFold [144].

6.2.1 Inductive Bias Encoder

6.2.1.1 Input Embeddings

As the goal of MuFormer is to acquire variants reflecting the evolutionary plausibility and phylogenetic relationships amongst protein sequences, large pre-trained protein language

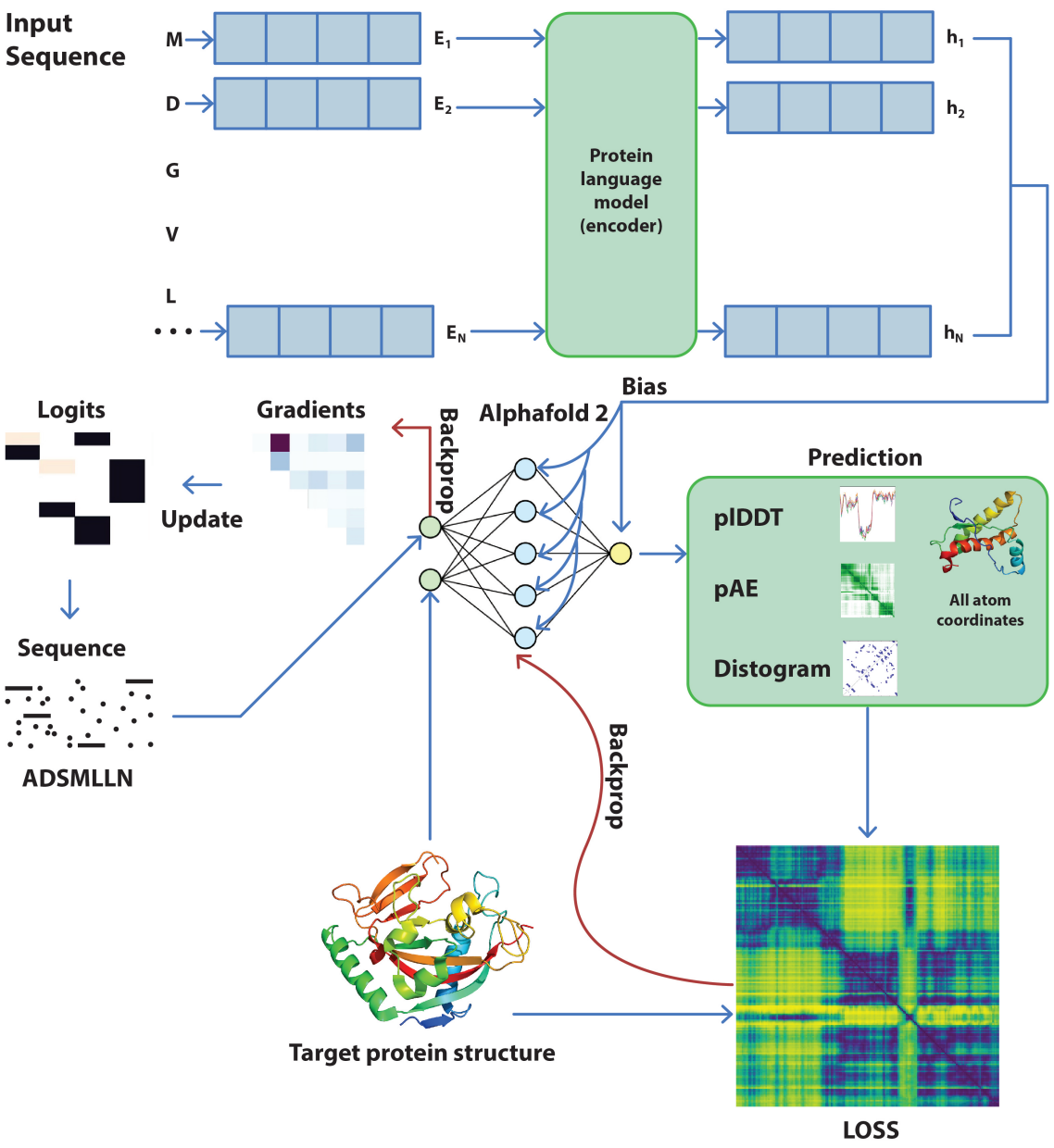


Figure 6.1: Proposed architecture for MuFormer de novo sequence designing model

models have been used here for the encoder to select amino acid substitutions suggested by a consensus of state of-the-art language models. The encoder for MuFormer takes a single wildtype protein sequence S of length n as the input such that $S \in \mathbb{X}^n$ where $\mathbb{X}$ is the set of amino acids $\in 1 \ldots 20$ and generates a vector representation for each position s_i in the in the sequence, $s_i \in \mathbb{R}^d$ wherein d is the dimension of the learned embedding.

By averaging the hidden representation across all of the positions in the sequence, a protein can be represented as a single vector. Sequences are depicted by protein embeddings as points

in a high-dimensional space. Each sequence is represented as a single point, and the network maps similar representations of the sequences to points close by. MuFormer leverages a pre-trained encoder responsible for mapping the amino acid sequence into a vector representation, extracting potential associations between the proteins primary structure for capturing their functional and structural properties from the sequence alone, such that residues in similar contexts culminate in a closer vicinity in the embedding space. Pre-trained word embeddings are a crucial component of state-of-the-art NLP systems providing notable gains over those learned from scratch [145].

The encoder used for MuFormer is based on the masked language modelling objective, and it produces conditional likelihoods for tokens such that $p(s_i|S)$. The encoder consists of 33 stacked encoder layers and leverages a dynamic self-attention mechanism using scaled dot-product amongst the tokens for computing weights. Prior to the computation of the encoder embedding by the transformer stack, the input tokens $s_i \in \{1 \dots n\}$ are embedded into vectors in $\mathbb{R}^d$ such that the learned embedding is represented as:

$$s_i^{aa} = W_{s_i} \tag{6.1}$$

where $s_i^{aa} \in \mathbb{R}^d$, *W* is the matrix of learnable amino acid token embeddings of dimension *d* such that $W \in \mathbb{R}^{|V| \times d}$. V is the vocabulary such that *|V| := 30* consisting of 20 amino acids, 4 rarely occurred amino acids (U, Z, O, B), unknown amino acid (X) and special tokens $< cls >$ representing start of sequence, $< eos >$ representing the end of sequence, $< mask >$ for the masked tokens, $< unk >$ for representing any out of vocabulary tokens, and $< pad >$ for padding sequences shorter than the selected mini-bath sequence length.

These token embeddings are further augmented with positional embeddings at the bottom of the encoder layers stack. The MuFormer1 model which is based on the ESM-1b encoder [15] uses learned sinusoidal positional embeddings. Whereas, MuFormer2 model that leverages ESM-2 [116] as the encoder, uses rotary positional embeddings (RoPE) [115] for allowing the model to extrapolate beyond the context window that it was trained over. Although RoPE positional embeddings [115] improves the model quality for smaller models, they marginally raise the cost of computation of the model, as each query and key vector inside of the self attention are multiplied by a sinusoidal embedding [19]. The final input embedding is formed by learned embeddings for each amino acid in the sequence *S* represented as s_i^{aa} with the positional embeddings resulting in $W_{input_embedding} = s_i^{aa} + W_{pos}[i]$.

6.2.1.2 MuFormer1 Encoder

MuFormer1 leverages the pre-trained ESM-1b [15] protein language model as an encoder which has been pre-trained over UniRef 50 [14] representation sequences with a masked language modelling objective inspired by BERT [78]. Owing to the success of ESM-1b in outperforming state-of-the-art protein language models for the tasks of secondary structure and contact prediction from previous studies [17] [16] [15], ESM-1b as an encoder for MuFormer1 was selected due to its capability to extract functional properties of proteins, given just the sequence. It has been pre-trained over 27 million protein sequences with a learning rate of 10^{-4}, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ using Adam optimiser [146], with GELU non-linearity [107] and with no dropout. The architecture consists of 33 layers involving 20 attention head per layer for multi-headed self-attention, with a hidden embedding size of 1280 and learned positional sinusoidal embeddings.

6.2.1.3 MuFormer2 Encoder

The encoder for MuFormer2 is based on the 650 million parameters variant of the ESM-2 protein language model [116] which is a BERT style encoder only transformer model. ESM-2 consists of 33 encoder layers, including 20 attention heads in each of these layers for multi-headed self-attention, with a hidden embedding dimension of 1280 such that $\mathbb{R}^d$ where $d = 1280$. It has been trained over 43 million sequences from UniRef 50 [14], for over 500K training steps with a learning rate of $4e - 4$ and L_2 weight decay of 0.01 using Adam optimiser[146] with $\beta_1 = 0.9$, $\beta 2 = 0.98$, and $\epsilon = 10^{-8}$, with no dropout. The rotary positional embeddings in ESM-2 allow the model to generalise over sequences longer than the context window it has been trained on, albeit at an additional computational cost. ESM-2 uses self-attention mechanism to learn interactions amongst protein tokens in the input sequence.

The ESM-2 model has proved to outperform ESM-1b over a range of prediction tasks over secondary structure and contact prediction tasks on CASP14 [147] and CAMEO [148] datasets. Moreover, the 150 million parameter model of ESM-2 is capable of surpassing the ESM-1b 650 million model over these protein tasks. This performance improvement is in accordance with data scaling laws developed in the broad language modelling domain [149].

6.2.1.4 Pre-training objective

MuFormers encoder consists of a pre-trained transformer-based discriminative encoder model which employs a multi-head self-attention mechanism to encode properties of the protein sequence into the embedding space. These encoders are pre-trained over the masked language

modelling objective involving sampling of 15% of the tokens out of which of with a 12% probability the true tokens are substituted with the special mask token <*mask*>, with 1.5% probability the true tokens could be replaced with a randomly-chosen alternate token and with the rest of the 1.5% probability, the true tokens remain the same (i.e original amino acid token). The masked language modelling pre-training objective reconstructs the masked tokens, whilst minimising the negative log likelihood for a true amino acid token x_i given the unmasked sequence $x_{/M}$ as the context (Equation 6.2).

$$\mathcal{L}_{\text{MLM}} = \mathbb{E}_{x\sim X}\mathbb{E}_M \sum_{i\in M} -\log p\left(x_i \mid x_{/M}\right) \tag{6.2}$$

To make predictions for a masked position, the model identifies dependencies amongst the masked sites x_mask and the unmasked sequence $x_{/M}$ by computing a weighted mean between positions using self-attention. The encoder processes a sequence of amino acids (tokens) $(x_1, \ldots, x_n)$, and outputs a sequence of logits over which a softmax operation is applied to attain a sequence of log probabilities $(y_1, \ldots, y_n)$ which are optimised with the masked language modelling objective. This computation proceeds over an accumulation of residual blocks stacked together that produces hidden states $(h_1, \ldots, h_n)$ with an embedding dimension d (where $d := 1280$).

The masked language modelling pre-training objective of protein language models can be viewed as a sequence distribution, wherein the sequence distribution is represented as a conditional likelihood $p(s_i|s_{-i})$, where s_{-i} refers to all tokens in a sequence apart from s_i such that $s_{-i} = s_1 \ldots s_{i-1} + s_{i+1} \ldots s_n$. As exact likelihood is computationally expensive to calculate for transformer based models, pseudo-loglikelihood (Equation 6.3) [150] [33] offers computation of embeddings efficiently $\in \mathbb{R}^{n\times embedding_size}$ (where $embedding_size := 1280$).

$$Psuedo_LL(s) = \sum_{i=1}^{n} \log p(s_i|s_{\neg i}) \tag{6.3}$$

However, owing to the requirement of *n* inferences (where $n := |S|$) for the computation of pseudo-loglikelihood, here the difference between conditional likelihoods of the mutated sequence $\hat{S}$ and the wild-type sequence (i.e original sequence *S*) has been used as an approximation for pseudo-loglikelihood for computing encoded embeddings as shown in Equation 6.4. The underlying premise is that wild-type amino acids on mutant backgrounds have conditional log-likelihoods that are roughly equivalent to those of wild-type amino acids on wild-type backgrounds. While this may be correct for mutant sequences that are sufficiently similar to wild-type sequences, there is often no support for this approximation on high-order mutant sequences, and the complete PLLs will probably be more accurate for predictions of the effects of mutation.

$$
\begin{aligned}
&Psuedo_LL(\hat{S}) - Pseudo_LL(S) = \textstyle\sum_{i=1}^{n} \log p(\hat{S}_i|\hat{S_{\neg i}}) - \sum_{i=1}^{n} \log p(S_i|S_{\neg i}) \\
&= \textstyle\sum_{i:\hat{S}_i \neq S_i} (\log p(\hat{S}_i|\hat{S_{\neg i}}) - \sum_{i=1}^{n} \log p(S_i|S_{\neg i})) + \underbrace{\sum_{i:\hat{S}_i = S_i} (\log p(S_i|\hat{S_{\neg i}}) - \log p(S_i|S_{\neg i}))}_{\approx 0}
\end{aligned}
\tag{6.4}
$$

Self-attention mechanism takes the hidden states $(h_1, \ldots, h_n)$ and generates a sequence of vectors $(h'_1, \ldots, h'_n)$ by computing interactions amongst each element in the sequence using scaled dot-product attention (Equation 6.5).

$$
Att(h) = softmax(\frac{Q(h)K(h)^n}{\sqrt{d}}) \cdot V(h) \tag{6.5}
$$

where the query *Q*, key *K*, and value *V*, are projections of the input sequence $\in \ltimes \times$ with n being the length of the sequence and *d* is the inner embedding dimension of the matrix outer product between *Q* and *K*. The outer matric product between *Q* and *K* vectors results in attention logits vector of size $n \times n$, which is then rescaled using $\frac{1}{\sqrt{d}}$ followed by a row-wise softmax operation, which represents the final attention weights. These attention weights are further multiplied with the value vector *V*, resulting in a weighted mean with the weights determined by the self-attention. A single step of self-attention models all possible pairwise interactions amongst each token in the sequence simultaneously. Multi-headed self-attention consists of performing the self-attention mechanism *k* times in each layer, where *k* is the number of independent heads for each layer (Equation 6.6). The encoders used in MuFormer have 12 attention heads ($k := 12$). Utilising multiple heads allows each head to capture distinct properties from the input sequence, resulting in a rich representational embedding.

$$
Att_{MH} = Concat(Att_1(x) \ldots Att_k(x))W^O \tag{6.6}
$$

The stacked encoder layers apply multi-headed self-attention to compute pairwise interaction amongst tokens across the sequence, followed by a feed forward network applied independently at each position and skip connections. The attention unit $U_{Att}(h) = p(Att_{MH}(n(h)))$ applies one step of multi-headed scaled dot-product attention over the pre-normalised input depicted as $n(h)$, which projects the result into the residual path. The following feed forward network $U_{FFN}(h) = FFN(GELU(P(n(h)))$ passes the normalised input through a position-independent feed forward network (FFN), including GELU non-linearity [107]. A single transformer encoder block is represented as:

$$
\begin{aligned}
&Encoder_Block(h): \\
&h \leftarrow h + U_{Att}(h) \\
&h \leftarrow h + U_{FFN}(h)
\end{aligned}
\tag{6.7}
$$

And the encoder model as a whole for all layers $\ell \in \{1 \dots L\}$ consisting of the token embedding step as $Emb(x)$ (which takes the input sequence x as a one-hot encoded vector with a dimension of $|V|$) and the positional embeddings computation as $Pos(x)$, with a projection W to log probabilities is represented as in Algorithm 15 and in the following equation 6.8:

$$\begin{aligned} &Encoder_Model(x): \\ &h \leftarrow Emb(x) + Pos(x) \\ &h \leftarrow Encoder_Block_{\ell}(h) for\ \ell \in 1 \dots L \\ &y \leftarrow W(h) \end{aligned} \tag{6.8}$$

The encoder for MuFormer uses pre-activation blocks [151], which involves application of layer normalisation [152] prior to the activation [153] to enable stable training. Biases are included in all projections apart from the token and positional embeddings, and all biases are initialised to zero. All layers $\ell \in \{1 \dots L\}$ are initialised from a zero centred normal distribution $\ell \sim \mathcal{N}(\mu, \sigma^2)$, where $\mu = 0$ and $\sigma^2 = 0.02$, and the initialisation of the projection into the residual path is rescaled by $\frac{1}{\sqrt{L}}$ where L is the number of residual layers.

Algorithm 6 MuFormer Encoder forward pass

Input: Sequence $x \in X^n$ with tokens $\{x_1, \dots, x_n\}$
$Input_Emb = Token_Emb(x) + Pos_Emb(x)$
$h \leftarrow Input_Emb$
for $\ell \in 1 \dots L$ **do**
    initialise Att_{MH}
    for k $\in 1 \dots K(heads)$ **do**
        $h' \leftarrow LayerNorm(h)$
        $attention(x) = softmax(\frac{Q(x)K(x)^n}{\sqrt{d}}) \cdot V(x)$
        $Att_{MH} \leftarrow Att_{MH} + attention(h')$
    end for
    $h \leftarrow h + Att_{MH}(h')$
    $z \leftarrow MLP(GELU(h)) \in \mathbb{R}^{4d}$
    $h \leftarrow h + z$
end for
$logits \leftarrow W(h)$
return logits

6.2.1.5 Injecting Inductive Bias for Sequence Design

Various studies [15] [138] [28] [7] have observed the linkage between the protein language models attention maps and their structural properties. Protein language models pre-trained over billions of protein sequences with a self-supervised language modelling objective are able to capture protein contact maps in a few shot setting [138], referring to protein language models ability to extract proteins structural information, given just the sequence alone. Therefore, it

has been hypothesised here that augmenting a pre-trained model embeddings for a sequence with the fixed backbone design task for a protein structure could allow the learned embeddings to operate as a soft potential (inductive bias). This soft potential would allow capturing the backbone flexibility, emanating in generation of diverse sequences for a fixed protein backbone to meet our goal of exploring the mutational landscape within and beyond the structural boundaries by leveraging evolutionary relationships.

For protein language models trained with m sequence samples such that sequence $S \sim D^m$, inductive bias states that some realities are less likely than others, and augmenting inductive bias allows the learning rule to bias towards a certain reality, whilst ensuring a good generalisation error [154]. The vector representation generated by the final layer of the stacked-encoder model (i.e the 33rd layer) is used as an inductive bias for AlphaFold2 to bias it towards the given sequence representation embedding which encaptures the proteins structural and functional properties. The encoder takes as input a sequence $(x_i, \ldots, x_n)$, and through the pre-trained model, the final vector representations of the input sequence or in other words, the hidden representation of the final encoder layer are obtained $(h'_1, \ldots, h'_n)$. This representation as an inductive bias guides AlphaFold2 during the inverse folding sequence design process to mutate positions whilst aligning with the backbone structure and the properties derived from the sequence representations. Figure 6.2 and 6.3 depict the inductive bias captured by the MuFormer1 and MuFormer2 encoder respectively for the protein sequence for Alpha Variant of length 993 amino acids (PDB: 7LWV, chain A [155]), reflecting the positions of higher variability.

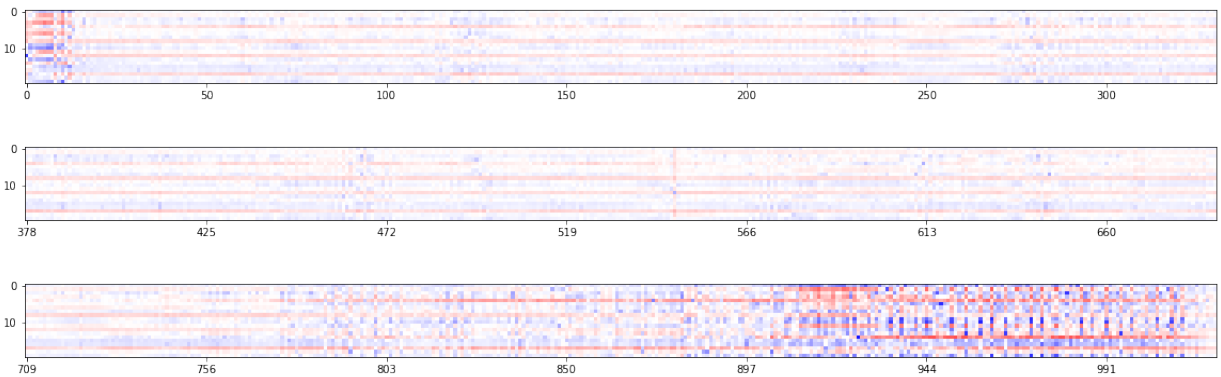


Figure 6.2: ESM-1b Inductive Bias for SARS-CoV-2 Alpha Variant (B.1.1.7) Sequence (PDB: 7LWV)

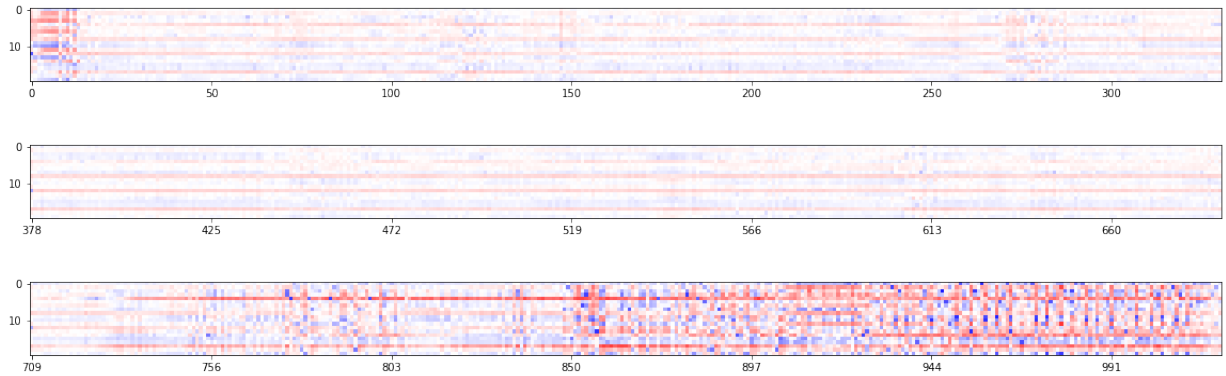


Figure 6.3: ESM-2 Inductive Bias for SARS-CoV-2 Alpha Variant (B.1.1.7) Sequence (PDB: 7LWV)

6.2.2 Fixed Backbone Sequence Design

Fixed backbone design or inverse folding refers to the task of designing a sequence of amino acids that fits the best for a given protein backbone structure conformation. Traditional energy-based approaches have taken a heuristic approach to the problem, concentrating primarily on minimising the energy of the target conformation in the belief that any stable alternative conformation is unlikely to spontaneously occur. However, this tight concentration on a single desirable structure might result in solutions with low-energy alternate states. A better approach is to identify the sequence that maximises the likelihood of the desired structure over all other states.

For the sequence design process, MuFormer uses AlphaFold2 [4] for fixed backbone design, leveraging numerous features of AlphaFold2 to efficiently design *de novo* sequences that are strongly predicted to fold in accordance with a specified backbone and the sequence inductive bias derived from the protein language model embeddings. To perform fixed backbone design, a starting sequence is taken which is biased towards a given structure and sequence embedding. This sequence is iteratively optimised to inculcate mutations by maximising the probability of amino acids at each position, for arriving at a final sequence design which has a predicted structure that fits the desired backbone design and sequence space with high confidence. Using fixed backbone design, for a given desired backbone geometry, the aim is to find an amino acid sequence that folds well in accordance with the structure conformation and the sequence space.

MuFormer embeds AlphaFold2 [4] into the protein sequence design loop as a structure prediction oracle ϵ, with extensive sequence design validation using AlphaFold biased with the protein language model sequence embeddings. Fixed backbone design task involves designing a sequence for a given 3D protein structure, which is also known as the inverse folding problem. MuFormers objective seeks to maximise the probability of designing a new

sequence given the protein backbone coordinates from the 3D structure and the sequence embedding which can be depicted as: $p(sequence|structure, \theta)$ where θ is the . Contrary to the Markov Chain Monte Carlo (MCMC) based search optimisation approaches [156] [157] [140], MuFormer equips gradient descent based optimisation for protein sequence design as it allows to arrive at a better scoring solution within fewer iterations than MCMC [158]. At each iteration, the sequence is mutated by the optimiser to maximise the probability of an amino acid occurring for each position in the sequence to explore the sequence space and querying the structure prediction oracle at each step.

For the sequence design process using MuFormer, sampling is performed over the true distribution of n-length sequences of amino acids $Y \in \{1 \dots 20\}^n$ conditioned over the protein backbone structure and the sequence embeddings. A proteins backbone is represented fully by the positions of each residues four $N - C_\alpha - C - C_\beta$ atoms along with the C-terminal oxygen atom, whose positions for a sequence S are encoded as $S \in \mathbb{R}^{(4n+1)\times 3)}$. MuFormer tries to model the following conditional distribution to design textitde novo protein sequences:

$$P(Y|S) = p(y_1, \dots, y_n|S) \tag{6.9}$$

It is believed that the likelihood of a specific protein folding conformation and the side-chain identity is determined by surrounding residues given the local nature of the physical interactions and contacts amongst proteins [159], including the side chain atoms. Thus, the conditional side chain distribution at a position i amino acid can be represented as in Equation 6.10 where y_{neighb} represents neighbouring amino acids at a given position i, and $\mathcal{X}_i^1, \mathcal{X}_i^2, \mathcal{X}_i^3, \mathcal{X}_i^4, \mathcal{X}_i^5 \in [-180°, 180°]$ depict the angles for the side chain of a protein. Here, $\mathcal{X}_i^1 := \omega_{ij}, \mathcal{X}_i^2 := \theta_{ij}$, and $\mathcal{X}_i^3 := \theta_{ji}$ are the dihedral angles, and $\mathcal{X}_i^4 := \varphi_{ij}, \mathcal{X}_i^5 := \varphi ji$ are the planar angles for every residue pair $\{i, j\}$

$$p(Y|S, Y_{neighb}) = p(y_1, \dots, y_n|S, y_{neighb_i}) \prod_{k=1}^{5} p(\mathcal{X}_i^k|\mathcal{X}_i^{1:k-1}, y_i, \dots, y_n, S, y_{neighb_i}) \tag{6.10}$$

By employing the predictive power of AlphaFold2 [4], MuFormer conditions over the local environment to predict the distributions over residue types $p_\theta(y_1, \dots, y_n|S, y_{neighb_i})$ and the rotamer angles $p_\theta(\mathcal{X}_i^k|\mathcal{X}_i^{1:k-1}, y_i, \dots, y_n, S, y_{neighb_i})$.

6.2.2.1 Structure Prediction Oracle $\sim$ AlphaFold2

The sequence design process involves an iterative procedure wherein: 1) A sequence is sampled from the given protein backbone structure with the addition of an inductive bias from the given sequence embedding representation 2) AlphaFold2 structure prediction oracle ϵ is queried with the input sequence 3) AlphaFold2 generates an internal MSA to encapture evolutionary points to generate a structure, along with the bias for the sequence embedding representation 4) The output consists of all atom coordinate, two protein reliability indices namely, predicted aligned error (PAE), and predicted local distance difference test ($pLDDT$), and a distogram. Over these, gradients are computed to minimise the loss for the true structure and the predicted structure, and the loss is backpropagated to the input sequence logits 4) Distributions for each amino acid position are obtained for a new protein sequence, through a step down the gradient 5) A new amino acid sequence is designed by maximising the probability of an amino acid appearing at each position 6) This newly designed sequence is fed back to the AlphaFold2 structure prediction oracle network for further iterations of sequence design in Step (2), and the process is repeated Step (2-6) until convergence.

Through this iterative procedure, MuFormer seeks to design a sequence that fits the given structural conformation and sequence space with high confidence. AlphaFold2 leverages free modelling, impelling it to ignore similar structures whilst making predictions. As shown in Figure 6.4, the AlphaFold2 network first performs a query on several genetic databases of protein sequences like MMSeqs2 [160] [161] to construct a multiple sequence alignment (MSA) for the input amino acid sequence. Generation of an MSA, allows the model to determine the portions of the sequence that are likely to mutate, and allows the model to deduce correlations amongst the sequences. Since MuFormer does not use any templates, pair representations are not generated. The MSA is then processed by the Evoformer block of AlphaFold2, which consists of 48 stacked layers with invariant point attention to produce a vector $\in \mathbb{R}^{N_{seq} \times N_{res}}$ (where N_{seq} represents the number of sequences in the MSA, and N_{res} represents the number of residues), and another vector $\in \mathbb{R}^{N_res \times N_res}$ representing the residue pairs. The Evoformer block allows refinement of the MSA representation, and exchange of information within the MSA, and the residue-residue pair representations to deduce reasoning about the evolutionary and spatial relationships. The Evoformer model has been pre-trained with the masked language modelling objective from BERT [78] to corrupt a sample of positions from the MSA which it then tries to reconstruct, allowing the model to encapture the sequential space. The final structure module intakes the refined MSA representation to construct a three dimensional protein structure in a single static step, resulting in a list of cartesian coordinates depicting the position of each atom of the protein, including the side chains.

The AlphaFold2 network is inferred N_cycle times for each sequence in the interactive procedure where $N_{cycle} := 2$ (i.e recycles = 1), to improve the prediction quality. The outputs obtained from the former execution are recycled as inputs for the next execution. To optimise over all N_{cycle} recycles, the gradients and losses are averaged across these cycles through the network. All five models of AlphaFold2 are used for optimisation to inculcate diversity in the predictions and during each iteration all five model parameters are evaluated with the gradient and losses being averaged. Each iteration uses a random set of model parameters. Residue-residue contacts per position are optimised to be within the distance of $C_\beta - C_\beta < 14.0\ \mathring{A}$, and sequence separation ≥ 9. As the number of residues that AlphaFold can be trained on is limited by memory and computational considerations, the model has been trained over cropped substrings of the full length protein sequences with a crop length of 128 residues.

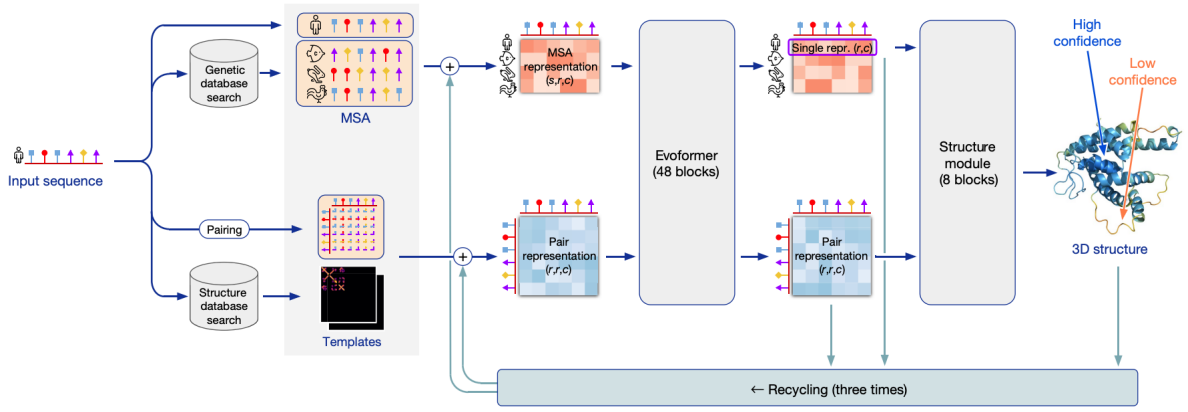


Figure 6.4: AlphaFold2 model architecture

6.3 Training Objective

The fixed backbone protein design problem involves generation of an amino acid sequence that aligns with the given protein structure backbone conformation. Using bayes theorem, a sequence that maximises the conditional probability below is desired:

$$p(sequence|structure) = \frac{p(structure|sequence) \times p(sequence)}{p(structure} \tag{6.11}$$

Since $p(structure)$ remains constant as only a single protein structure backbone is given for the fixed backbone design task, the design problem now comes down to maximising $p(structure|sequence) \times p(sequence)$, where $p(sequence)$ is influenced by the inductive bias using the protein language model encoded representation, which remains constant. For approximating $p(structure|sequence)$, pre-trained AlphaFold2 [4] is used for predicting

residue-residue contact pairwise distances ($C_\beta - C_\beta$) represented as $p(d_{ij})$ for every residue pair $\{i, j\}$. The orientations $\{\mathcal{X}_i^1, \mathcal{X}_i^2, \mathcal{X}_i^3, \mathcal{X}_i^4, \mathcal{X}_i^5\}$ for the dihedral and planar angles are predicted as well. For MuFormer, the contacts per position are optimised to be in the residue-residue distance within $C_\beta - C_\beta < 14.0\mathring{A}$. For the inverse folding problem, Equation 6.12 shows the negative log likelihood quantity that we seek to minimise and this can be observed as the log likelihood of a sampled sequence given the protein backbone structure. Equation 6.12 is optimised using gradient descent for computing gradients of the network outputs with respect to its inputs using backpropagation.

$$\mathcal{L} = -\log p(structure|sequence) \tag{6.12}$$

MSA Masking Objective: For an input sequence, AlphaFold2 first generates a multiple sequence alignment by performing a search against genetic databases of protein sequences to create an evolutionary profile of sequences. A subset of sequences N_{clust} are selected randomly as the cluster centres for the MSA, with the first cluster centre being set to the queried amino acid sequence. The rest of the sequences are assigned with their closest match cluster based on the hamming distance. Masking is performed over the MSA with 15% probability such that amino acids are replaced with an amino acid a sampled uniformly from the distribution $a \sim \mathcal{U}(\mu, \sigma^2))$ with a probability of 1.5%, or the amino acids are replaced with an amino acid a sampled from the generated MSA profile for the given position with a probability of 1.5%, or with a 10.5% probability the amino acids are replaced with a special token $< mask >$, and with a probability of 1.5%, the amino acids are not replaced at all and remain the same. Equation 6.13 depicts the masking procedure where AA stands for amino acids $\in \mathbb{R}^20$ and MSA represents the MSA profile $\in \mathbb{N}_{seq} \times \mathbb{N}_{res}$. The masked positions are the prediction targets, and masking is used during both the training as well as during the inference time.

$$f_{mask}\left(s_i\right) = \begin{cases} < \text{mask} > & 70\% \quad \text{of cases} \\ s_i & 10\% \quad \text{of cases} \\ Random(AA) \sim MSA & 10\% \quad \text{of cases} \\ \text{Random}(AA) \sim \mathcal{U}(\mu, \sigma^2) & 10\% \quad \text{of cases} \end{cases} \tag{6.13}$$

Optimisation: MuFormer uses gradient descent to optimise over the amino acid space for curating a sequence at each step. The sequences are mutated with the optimiser by exploring the sequence space. For the task of protein sequence design, gradient descent converges quicker in about 100 iterations, which is fewer than the iterations required for monte carlo based optimisations [158]. The space of amino acids is a discrete space, and optimising over such a hard space of one hot encoded sequence can be challenging for complex topologies. To ameliorate this, a phased sequence design process adapted from [143] has been applied here

(Equation 6.14) which first optimises the logits for 50 gradient steps to treat it as a continuous space, and then another 50 gradient steps over the softmax of logits ($softmax(\frac{logits}{temp} \cdot inputs)$) for ensuring that the probability of all amino acids at each position sums up to one, where the temperature is set to $temp := 1$, to add randomness. This is followed by 10 steps of argmax discretization in the discrete space of amino acids ($one_hot(logits) \cdot input$). The *argmax* function is used for selecting the amino acid having the highest probability for each position.

$$\begin{aligned} \hat{Y} &= softmax(\tfrac{logits}{temp}) \\ X &= onehot(argmax(\hat{Y})) \end{aligned} \tag{6.14}$$

The output generated from a single forward pass of AlphaFold includes all atom coordinates, two reliability indexes including predicted aligned error (PAE) and predicted local-distance difference test (pLDDT), along with a distogram representing contact maps amongst residue-residue interactions. These outputs are used to define the loss function, which is backpropagated for the computation of the gradients at each step for designing the protein sequence, and then updated and predicted in a loop for optimisation until convergence. Throughout the optimisation, the protein changes its structure. The local and global confidence metrics change as well. In general, any function for the protein sequence, structure or the prediction confidence metrics may be used for the optimising MuFormer. Below, two distinct optimisation methods consisting of an accumulation of several metrics have been described. FixBB optimisation leverages pLDDT, Distogram CCE, RMSD and PAE losses, where pLDDT is maximised and others are minimised. Whereas, the backbone atom optimisation uses all these metrics used by FixBB along with the addition of a newly introduced loss metric based on the radius of gyration [162] loss function to optimise over the three critical atoms of the protein backbone geometry, namely: N, C_α, C.

6.3.1 FixBB Optimisation

The fixed backbone design optimisation procedure, termed as fixbb, for fixed backbone design uses an inverted AlphaFold2 mode. During optimisation with fixbb, it observes the entire protein conformational space, and seeks to maximise the amino acids at each position in the designed protein sequence towards the protein structure backbone by minimising the loss function. The fixbb optimisation method is an amalgamation of multiple loss functions added together to handle distinct aspects of protein design, from residue-residue contacts to fitting the backbone structure conformation well and folding metrics. These loss functions include $plDDT$, PAE, $RMSD$, and $Distogram$ categorical cross entropy (cce) losses. Instead of the introduced $FAPE$ loss in AlphaFold2 [4] which involves computation of the distances between the ground truth and the predicted backbone atoms in the local reference frame for each residue, for the fixbb optimisation $distogram\ cce$ loss has been used here as it is

considered to be more stable for the task of protein sequence design [143].

AlphaFolds confidence is expressed as an accumulation of predicted local distance difference test (pLDDT), along with the predicted aligned error (PAE). The predicted local distance difference is a per-residue confidence metric that measures the quality of the local model and colours the residues of the prediction. Whereas, the predicted aligned error provides a measure of confidence for all possible amino acid pairs in the sequence, and this is crucial for assessing the confidence in the large scale topology and domain packing for a given protein. Based on these, sequences can be optimised for maximising or minimising any target function for the outputs returned by AlphaFold2, which could include minimising the room mean square distance (RMSD) for a given target structure and along with that maximising AlphaFolds prediction confidence such as pLDDT. AlphaFolds suitability to perform complex structure prediction, allows us to constrain conformational space with an inductive bias hypothesis by adding the encoded embedding for a sequence using a protein language model.

6.3.1.1 Masked MSA Prediction Loss

Similar to BERTs masked language modelling objective [78], the multiple sequence alignment generated internally by AlphaFold2 follows the masked procedure of corrupting a random sample of amino acids and tries to reconstruct them to deduce the evolutionary point mutations in an MSA. For the masking procedure, AlphaFold considers 23 distinct classes including 20 amino acids, an unknown type token $< unk >$, a gap token $< pad >$, and a mask token $< mask >$. The resulting MSA representations $\{m_{si}\}$ are projected linearly into the output classes $\in \mathbb{R}^c$, where $c := 23$, and these logits are passed through a softmax function to generate probabilities, and scored with a categorical cross entropy loss such that:

$$\mathcal{L}_{msa} = -\frac{1}{N_{mask}} \sum_{s,i \in mask} \sum_{c=1}^{23} y_{si}^c \log p_{si}^c \tag{6.15}$$

where y_{si}^c represent the one-hot encoded ground truth values, p_{si}^c are the predicted class probabilities, with the mean taken for the masked tokens to predict a token for that position.

6.3.1.2 Distogram CCE Prediction Loss

Distogram helps in providing a discrete representation of a protein structure's three-dimensional coordinates view for each atom [144]. Distogram represents a protein structure as a $[N, N, D]$ tensor, consisting of D-dimensional one-hot encodings for a discretized distance measure amongst all possible pairs of amino acids in the protein where *N* is the length. Distogram prediction loss seeks to minimise the categorical cross entropy amongst residue pairs within

the selected distance cutoff for MuFormer i.e $C_\beta - C_\beta < 14.0\mathring{A}$. To compute the distogram prediction, the symmetrised pair representations of residues $(z_{ij} + z_{ji})$ are linearly projected into 64 distance bins and the probabilities for each bin p_{ij}^b are obtained by applying softmax. The categorical cross entropy loss that we seek to minimise for all residue pairs can be represented as:

$$\mathcal{L}_{dist} = \frac{1}{N_{res}^2} \sum_{i,j} \sum_{b=1}^{64} y_{ij}^b \log p_{ij}^b \tag{6.16}$$

where y_{ij}^b are the prediction targets (one-hot encoded binned residue distances).

6.3.1.3 pLDDT Prediction Loss

The predicted local distance difference test (pLDDT) is a per-residue estimate of the model's predicted confidence ranging from 0-100, which corresponds to the predicted score on the $lDDT - C\alpha$ metric [163]. pLDDT provides a measure for each amino acids local confidence. It is computed by projecting the final single representation generated by the structure module into 50 bins, with each bin covering 2 $lDDT - C\alpha$ range. The final predicted structure (Equation 6.17) is scored with the per-residue $lDDT - C\alpha$ and the ground truth protein structure, which is discretized into 50 bins and utilised as the categorical cross entropy loss target value $\mathcal{L}_{conf}$ averages over all the residues. The pLDDT metric is maximised, the higher the score, the better the sequence folds into the predicted protein backbone conformation.

$$\begin{aligned} p_i^{pLDDT} &= softmax(Linear(a_i)), p_i^{pLDDT} \in \mathbb{R}^{|v_{bins}|} \\ p_i^{true\ LDDT} &= one_hot(r_t^{true\ LDDT}, v_{bins}) \\ \mathcal{L}_{conf} &= mean_i(p_i^{true\ LDDT^{\mathbb{T}}} \log p_i^{pLDDT}) \end{aligned} \tag{6.17}$$

$$where\ a_i = relu(Linear(relu(Linear(LayerNorm(s_i)))))$$

6.3.1.4 PAE Prediction Loss

The predicted aligned error provides the predicted error for every amino acid position within a local coordinate frame of each other amino acid and to compute this AlphaFold predicts a binned distribution of values. This error is independent of the three dimensional structure. It represents if two residues x and y are aligned in the predicted and true structures. A low PAE value for residue pairs x and y indicates that a well-defined relative position and orientations have been predicted for them. Whereas, a high PAE value indicates the relative positions and orientations for a pair of residues to be uncertain.

$$PAE_{ij}(x, y) = |T_{x,i}x_i - T_{y,j}y_i| \tag{6.18}$$

where $T_{x,i}$ represents the coordinate transformation that moves a vector to the local coordinate system for the backbone atom x_i.

6.3.1.5 Root mean square distance loss

Root mean square distance (RMSD) is a common metric used to express similarity amongst true protein structures, which involves superimposing the structure and then calculation of the root mean square distance between the C_α atoms of the equivalent residues. It is possible to lessen the impact of flexible tails and crystal-packing artefacts by taking the median RMSD-Cα on the best predicted 95

$$RMSD(x, y) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}||x_i - y_i||^2} \tag{6.19}$$

6.3.2 Backbone Atom Optimisation

This additional newly introduced backbone atom (BBA) optimisation procedure establishes a radius of gyration [162] loss function to optimise over the three critical atoms of the protein backbone geometry, namely: N, C_α, C to guide the protein sequence design process with a bias towards the sequence embedding representation. This new loss function is used in addition to the ones described for FixBB optimisation to optimise the sequence design process. Here, the focus is over the backbone structure, without taking into account the side chains of the protein. Each of the 20 amino acids has a unique side chain, yet they all have a common collection of atoms that constitute the whole amino acid backbone, and these include *N*, C_α (alpha carbon) and C atoms. The sequence design process with backbone atom optimisation seeks to minimise the generalisation error between these selected atoms and their true values, to optimise the designed sequence backbone towards AlphaFolds generated structure of the MSA representation. This helps in injecting evolutionary context, whilst maintaining the stability of the protein sequence. The loss function for backbone atom optimisation is represented in Equation 6.20 where a is the set of backbone atoms N, C_α, C. The initial input sequence (sampled) used for the fixed backbone design process with backbone atom optimisation is initialised with gumbel distribution such that $S \sim \mathcal{G}(\mu, \sigma^2)$

$$\mathcal{L}_{bba} = \sum_{a \in \{N, C_\alpha, C\}} [\frac{1}{n}\sum_{i=1}^{n}(y_i^a - \hat{y}_i^a)^2] \tag{6.20}$$

7
Experimental Results

In this section, MuFormer suite of models have been assessed for their ability to learn an evolutionary mutational landscape specifically for SARS-CoV-2 virus. Several metrics for structure based energy computation and sequence based pathogenicity predictions metrics have been leveraged to measure the designed mutation sequences and predict their fitness. The experiments' thoroughness and coverage offer a look at the mutational fitness landscape of specific proteins and give quantitative connections between protein sequence and function. MuFormer is adapted to anticipate the quantitative effect of mutations.

For the experimental results, four variants of MuFormer model along with vanilla AlphaFold2 [4] as the baseline model have been used for the task of *de novo* sequence generation. The MuFormer model variants include: MuFormer2-fixbb, MuFormer2-bba, MuFormer1-fixbb, and MuFormer1-bba. MuFormer2 leverages the ESM-2 protein language model [116] as an encoder, whereas MuFormer1 uses the ESM-1b protein language model [15] as its encoder to generate a hidden representation of the input sequence which is augmented as an inductive bias to the *de novo* protein sequence design optimisation loop. The *-fixbb* model variants utilise the *FixBB* optimisation and *-bba* model variants use both *FixBB* and *BBA* optimisation functions using gradient descent. The gradient descent based optimisation is performed for about 110 design iterations, including 50 epochs of optimising over the logits in the continuous space, followed by another 50 epochs of softmax over the $\frac{logits}{temp}$ with *temp = 1.0*, culminating with the final 10 iterations of argmax discretisation. Each of these gradient-based design iterations generates the most likely mutation sequence by maximising the probability of amino acids appearing at each position in the sequence *S*. NVIDIA Tesla T4 Cloud GPU was used to optimise these models, with each model taking roughly 2 hours for 110 iterations.

To assess the applicability of MuFormer for the purpose of generating mutational landscape, all MuFormer models were given the protein sequence for Alpha variant as an inductive bias and the protein structure as input (PDB: 7LWV, chain A [155]). The inverse sequence design process over the protein sequence was performed for 110 gradient steps, by maximising the likelihood of amino acid appearing at each position. The baseline model AlphaFold2 does not take any sequence input, but just the protein structure from which it samples a sequence, as there is no protein language model used in the baseline.

Figure 7.1 showcases the sequence design process for every 10th sequence in the design loop generated by MuFormer2-fixbb along with the conservation gaps amongst the amino acid positions for residues in the range 493-634. A higher conservation gap reflects the stability in amino acid at that position, whereas a lower conservation gap represents that particular amino acid position's mutational variability. The positions with lower conservation gap are known as the co-evolved positions.

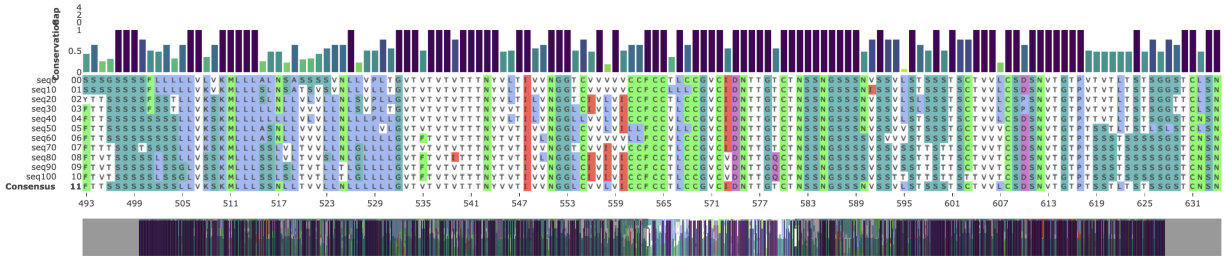


Figure 7.1: Sequence Alignment Chart for MuFormer2-fixbb designed protein sequences for SARS-CoV-2 Alpha Vairant (B.1.1.7), PDB: 7LWV

7.1 Fitness Prediction

Evaluating the fitness of the designed protein sequences allows us to assess the effects of mutations over the ability of the mutated protein to execute its function. The mutations could be in the form of amino acid substitutions, or indels i.e insertions or deletions. From all of the random perturbations in a sequence that are possible, phylogenetic evolution amongst protein families is biased towards selected those that possess a consistent fitness [164]. Here, the intra-protein variant effect has been used for predicting the effect of unobserved mutations with in vitro mutagenesis of the protein using MuFormer, which is highly applicable in protein engineering [165]. As the MuFormer and AlphaFold2 models have been trained with the masked language modelling objective setting, the models try to predict a masked amino acid position in a given protein sequence S, by maximising the probability of the amino

acid appearing at each position given the surround and MSA profile context. In the masked language modelling objective, the unobserved amino acid tokens that influence a protein's fitness, such as structure, function, and stability, leave a trail in the distribution of the observed ones.

A widely used approach to predict the fitness F_x of a designed protein sequence is to quantify the pseudo log likelihood amongst the mutated sequence x^{mut} and the naturally occurring wildtype reference sequence x^{wt} for that protein family [21]. This log likelihood has been found to predict the impact of mutations [76] [21]. Meier et al. [22] discovered that the log-likelihoods of large protein language models are capable of predicting mutational effects. Mathematically, the pseudo log likelihood of mutated sequences is represented as:

$$PLL(x) = \sum_i \log p(x_i = x_i^{mut} | x_{\neg i}^{mut}) - \log p(x_i = x_i^{wt} | x_{\neg i}^{wt}) \tag{7.1}$$

Figure 7.2 (a) represents the relative difference in sequence conditional log likelihoods taken as a predictor for the stability measure of protein sequence, assuming that more stable sequences score a higher log likelihood. From the log likelihood results, it can be observed that the sequences designed by MuFormer2-fixbb and MuFormer1-fixbb outperform the sequences generated by the baseline model AlphaFold2. However, MuFormer2-bba and MuFormer1-bba model variants have a lower log likelihood value with the average log likelihood of $-7e-2$ and $-8e-2$ respectively. Whereas, the mean log likelihood of MuFormer2-fixbb and MuFormer1-fixbb is $8e-2$, which surpasses AlphaFold2's log likelihood of $2e-2$. This suggests that the backbone optimisation in addition to the FixBB optimisation, degrades the stability of the protein. This also sheds light over the ability of MuFormer FixBB models to design mutated protein sequences of greater fitness and stability, which could help in the prediction of strains capable of viral escape.

Further, the pseudo perplexity of the protein sequences has been evaluated where a sequence with lower perplexity has a higher likelihood ratio and fitness. The inverse likelihood of native sequences in the anticipated sequence distribution is measured by perplexity. One of the most used measures for evaluating language models is perplexity (PPL). The exponentiated average negative log-likelihood of a sequence is defined as perplexity. Considering a tokenized sequence $X = (x_0, x_1, \ldots, x_t)$, then the perplexity of X is:

$$\mathrm{PPL}(X) = \exp\left\{-\frac{1}{t}\sum_i^t \log p_\theta\left(x_i \mid x_{<i}\right)\right\} \tag{7.2}$$

where $\log p_\theta(x_i|x_{<i})$ is the log-likelihood of the ith token conditioned over the preceding tokens $x_{<i}$. The results for perplexity shown in Figure 7.2 (b) corroborate the findings from

pseudo log likelihood, with MuFormer2-fixbb and MuFormer1-fixbb having the lowest mean perplexity score of 1.75 and 1.76 respectively, indicating these models have learned to better generate protein sequences from the target protein family. MuFormer2-bba and MuFormer1-bba again perform poorly on this metric with the highest perplexity compared to the other model variants and the baseline of 1.99 and 2.01 (average) respectively.

Along with perplexity and log likelihood, another metric known as sequence recovery is commonly used for the task of fixed backbone protein design [33]. Sequence recovery gauges over the frequency of sampled sequences matching the native wildtype reference sequence at each position. This metric helps us understand how far the generated sequence is from the wildtype sequence. Since with MuFormer we are generating protein sequences with mutations, sequence recovery can be expected to be low for models that end up exploring beyond the evolutionary space biased by the protein language model and internal MSA profile. From the sequence recovery metric analysis in Figure 7.3 (a), it can be observed that MuFormer2-fixbb has the highest mean sequence recovery of 0.21, followed by MuFormer1-fixbb with 0.19 mean sequence recovery. MuFormer2-bba has a mean sequence recovery of 0.167, and for MuFormer1-bba it is 0.13, with AlphaFold2 having the lowest sequence recovery with a mean of 0.05. The fixbb optimisation model variants have a better sequence recovery score than the backbone atom (bba) optimisation variants, and MuFormer2 models perform better than MuFormer1 which reflects over the improved ESM-2 [116] encoder model which has been trained over a greater number of protein sequence than ESM-1b [15], and employs RoPE positional embeddings [115].

Further, the sequence pairwise alignment in Figure 7.3 (b) showcases the alignment of the generated sequences by the models with the original alpha variant sequence. The pairwise alignment results suggest that backbone atom (bba) optimised models- MuFormer2-bba (434.55 mean) and MuFormer1-bba (420.10 mean) have a higher pairwise alignment such that the regions of similarity are maximised. AlphaFold2 has the lowest pairwise alignment score of 354.47 (mean), out of all models.

The mutational fitness landscape provides a deep insight about the properties of protein sequences and their stability. Amalgamation of next generation *de novo* protein sequence generation along with a mutagenesis screen allows the possibility of predicting unseen mutations that could cause viral escape.

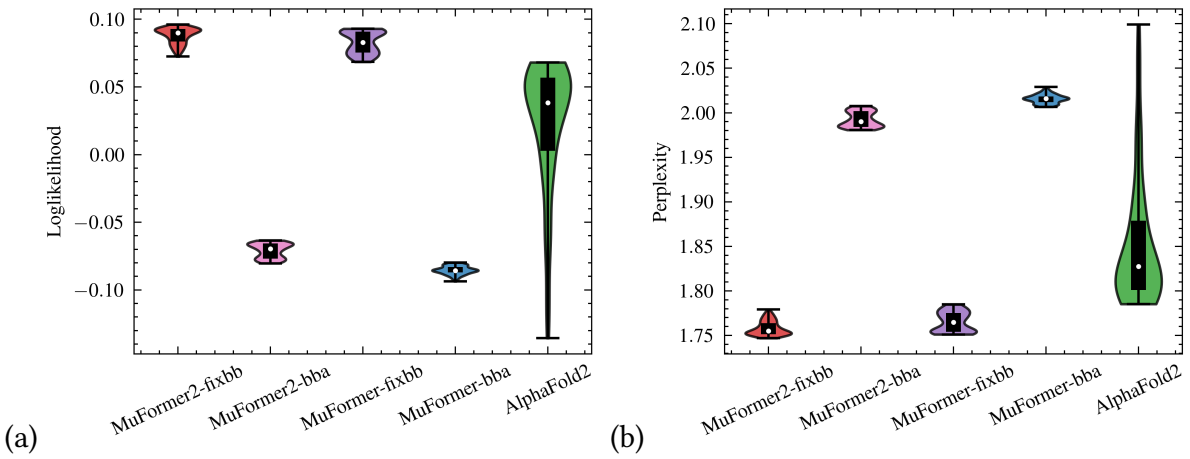


Figure 7.2: Log Likelihood and Perplexity for MuFormer models and AlphaFold2 Baseline

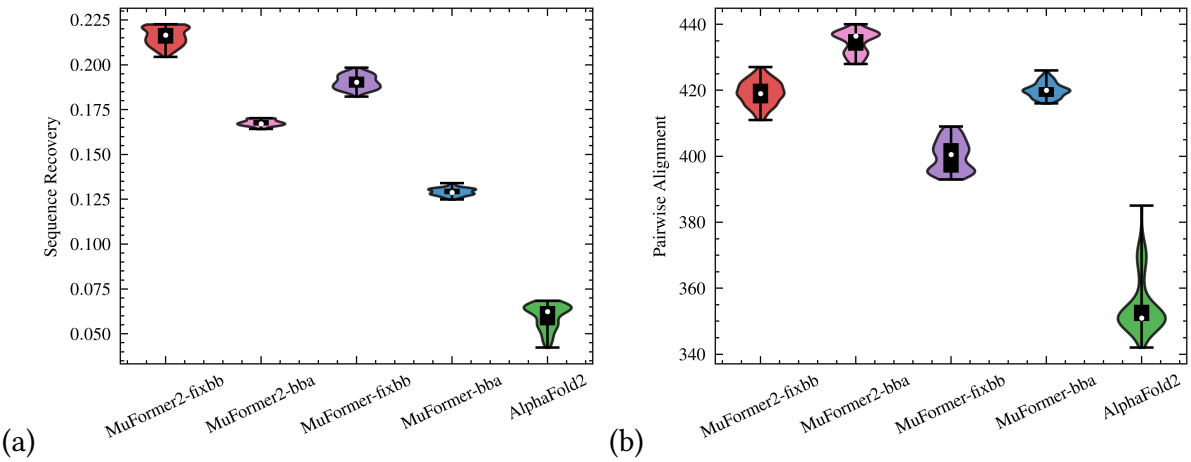


Figure 7.3: Sequence Recovery and Pairwise Alignment Score for MuFormer models and AlphaFold2 Baseline

7.2 Structure likelihood Prediction

In this section, the predicted structure likelihood for the designed mutational sequences has been analysed in terms of several protein structure energy computation and likelihood metrics. Out of these metrics, pLDDT, RMSD, PAE and distogram losses are optimised by both fixbb and backbone atom (bba) optimisation strategies.

The predicted local distance difference test (pLDDT) represents the per-residue estimate for a model's predicted local confidence which is the difference between the predicted final protein structure and the ground truth protein structure. Since the pLDDT metric is maximised, higher scores represent better protein structure confidence. A cut-off of >60 usually represents a well folded protein structure with a correct backbone prediction. The predicted aligned error (PAE) represents if two residues a and b are aligned in the predicted and the true structures,

such that a low PAE score for residues a, b indicates that a well-defined relative position has been predicted for them.

Root mean square distance loss expresses similarity between the ground truth protein structure and the predicted protein structure consisting of calculation of the root mean square distance between the C_α atoms of the equivalent residues, where lower distance represents a better match. The distogram loss seeks to minimise the categorical cross entropy loss between all the possible residue pairs within the selected distance cutoff of $C_\beta - C_\beta < 14.0\mathring{A}$.

Figure 7.5 and 7.4 (a) - (d) showcases the values for these selected structural loss metrics (pLDDT, PAE, RMSD, Distogram) computed for all five models, progressing with gradient steps. It can be observed that pLDDT score increases with further gradient steps, and the other metrics (PAE, RMSD, Distogram) are minimised. For pLDDT, MuFormer2-bba has the highest mean pLDDT score of 0.59, followed by MuFormer1-bba with a mean score of 0.56, 0.45 for MuFormer2-fixbb and 0.43 for MuFormer1-fixbb. Whereas, AlphaFold2 has a mean pLDDT score of only 0.32. pLDDT scores < 50 tend to appear in disordered regions, and thus AlphaFold2 and fixbb variants of MuFormer might be unable to generate proper protein structures.

Predicted aligned error has the lowest mean for AlphaFold2 of 0.60, whereas MuFormer2-fixbb and MuFormer2-bba stand at 0.65 and 0.67 respectively, and MuFormer1-fixbb and MuFormer1-bba with 0.63 and 0.68 respectively. For root mean squared distance error, MuFormer2-bba and MuFormer1-bba have the lowest mean RMSD, 24.14 and 24.83 respectively, followed by MuFormer1-fixbb with 27.52, and then MuFormer2-fixbb with 29.24, and finally AlphaFold2 with a mean RMSD of 29.55. Lastly, for the distogram cross-entropy loss, MuFormer2-bba and MuFormer1-bba again have the lowest mean scores of 2.22 and 2.24 respectively, followed by MuFormer1-fixbb with 2.37, MuFormer2-fixbb with 2.40, and AlphaFold2 with the highest distogram score of 2.48.

For all these structural metrics apart from PAE, MuFormer models are outperforming the structural predictions generated by the baseline AlphaFold2 model. This reflects over the structural properties and information transferred from the pre-trained large protein language model as an encoder.

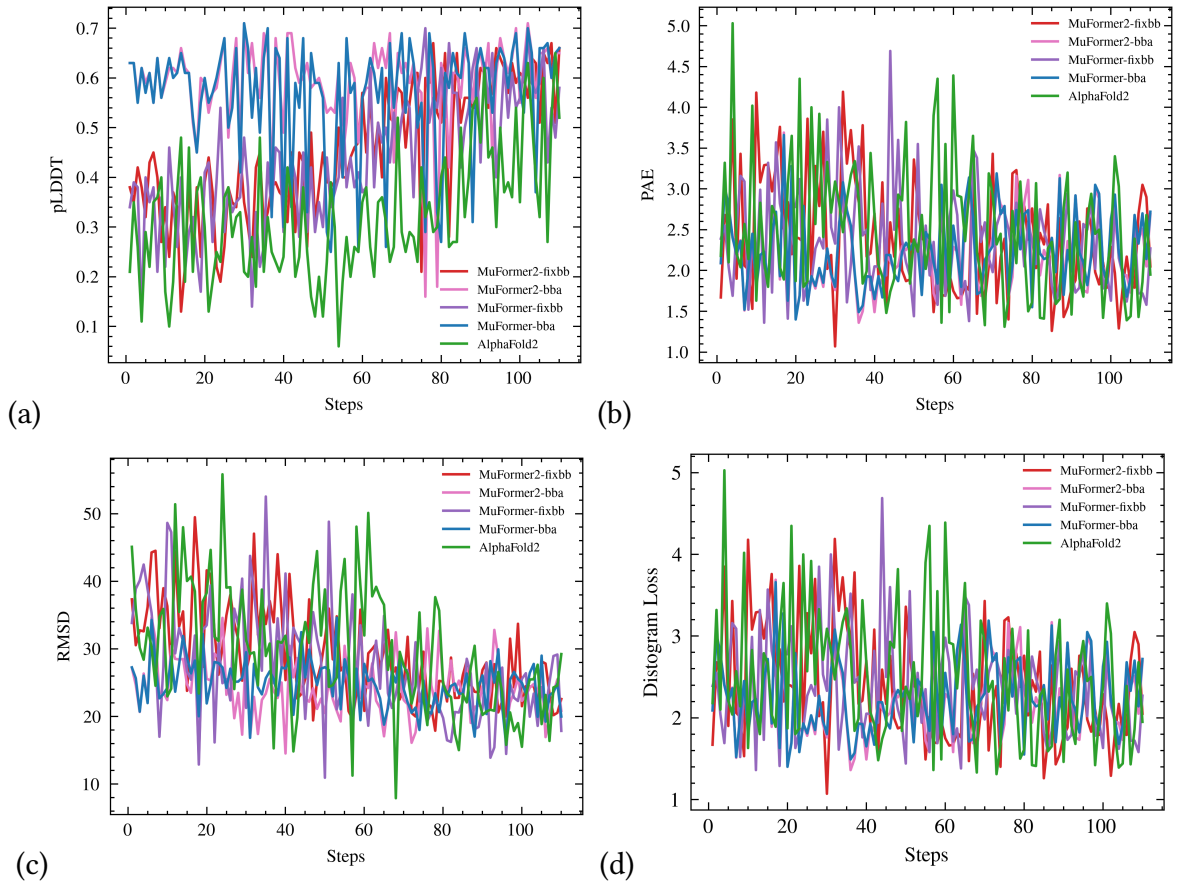


Figure 7.4: Structural likelihood metrics vs epochs- (a) plDDT, (b) PAE, (c) RMSD, and (d) Distogram loss for MuFormer models and AlphaFold2 Baseline

7.3 In vitro Mutagenesis Generation

The final generated mutations from all five models were searched on the Global Initiative for Sharing All Influenza database (GISAID [6]) Audacity Blast tool to identify the most sequentially similar SARS-CoV-2 genomic sequence and its lineage. The mutations were generated by all five models, MuFormer suite of models and AlphaFold2 given the protein sequence for SARS-CoV-2 Alpha variant (B.1.1.7 lineage) as an inductive bias (encoded by a protein language model) and the protein structure as input (PDB: 7LWV, chain A [155]). Each iteration of gradient descent, mutated all positions to embed the most likely amino acid, whilst minimising RMSD, distogram loss and PAE metrics, along with maximising the pLDDT loss.

The final sequence generated by MuFormer2-fixbb was classified as BA.2 lineage which comes under Omicron variant, with a genomic distance of zero i.e identical match. Given just Alpha variants sequences encoded representation along with gradient based fixed backbone

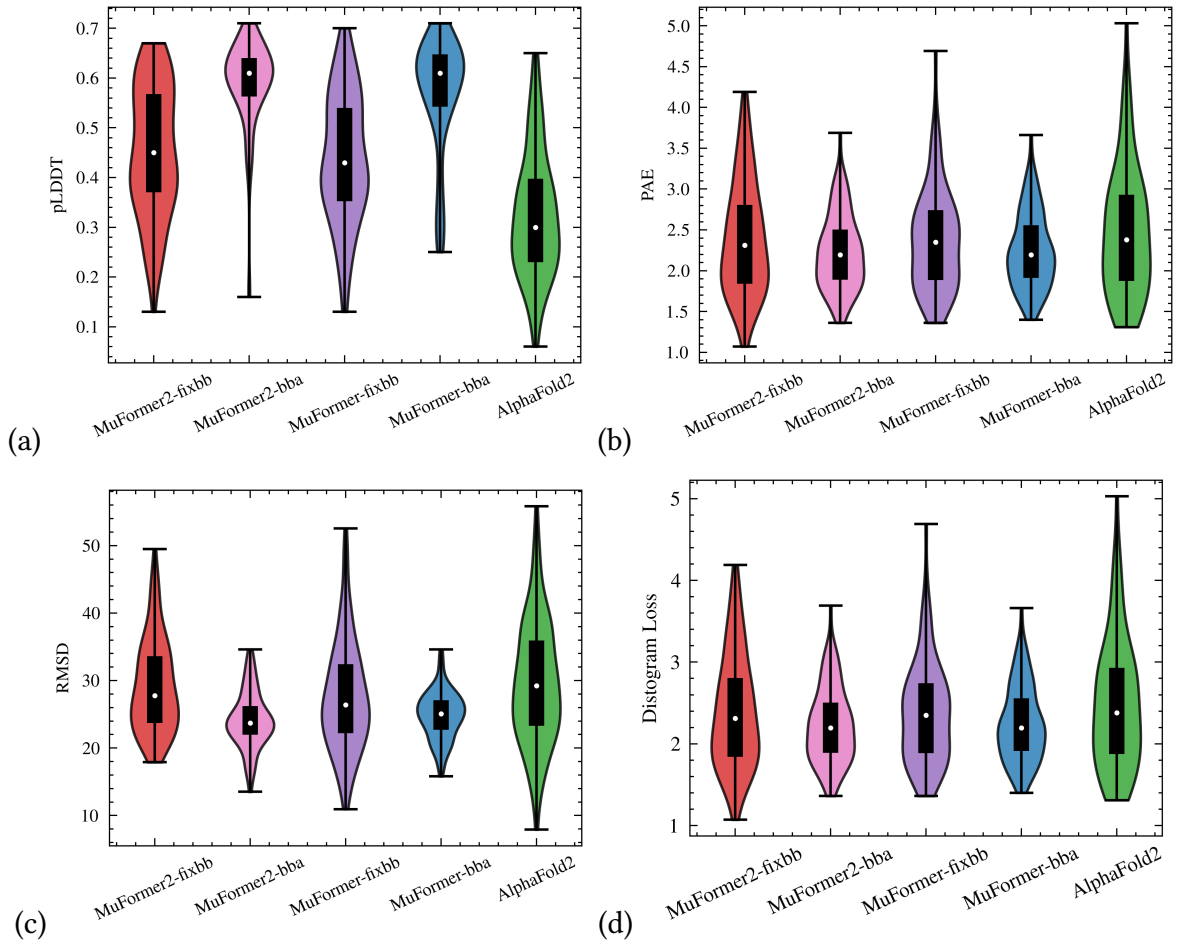


Figure 7.5: Structural likelihood metrics- (a) plDDT, (b) PAE, (c) RMSD, and (d) Distogram loss for MuFormer models and AlphaFold2 Baseline

design using AlphaFold2 which curates an internal MSA profile, the model was able to encapture mutational behaviours in the evolutionary protein sequence search space. Figure 7.7 (a) shows the GISAID search results showing the most frequent lineages that the designed sequence matches with. The distance colouring signifies the genomic distance between the searched sequence and the closest match genome sequence. Moreover, all the predicted closest match lineages belong to Omicron variant, which sheds light over the capability of MuFormer to mutate Alpha variant sequence into Omicron variant.

The MuFormer2-bba model is able to detect its final generated sequence as AY.25 lineage (Figure 7.8 (b)) being the most similar to it with a genomic distance of zero, which represents an exact match. AY.25 lineage comes under Delta variant, and the other frequent lineages detected (B.1.617.2 and AY.25.1) also come under Delta variant. Similarly, the MuFormer1-bba model detects its final generated sequence as AY.122 (Delta variant) with a genomic distance

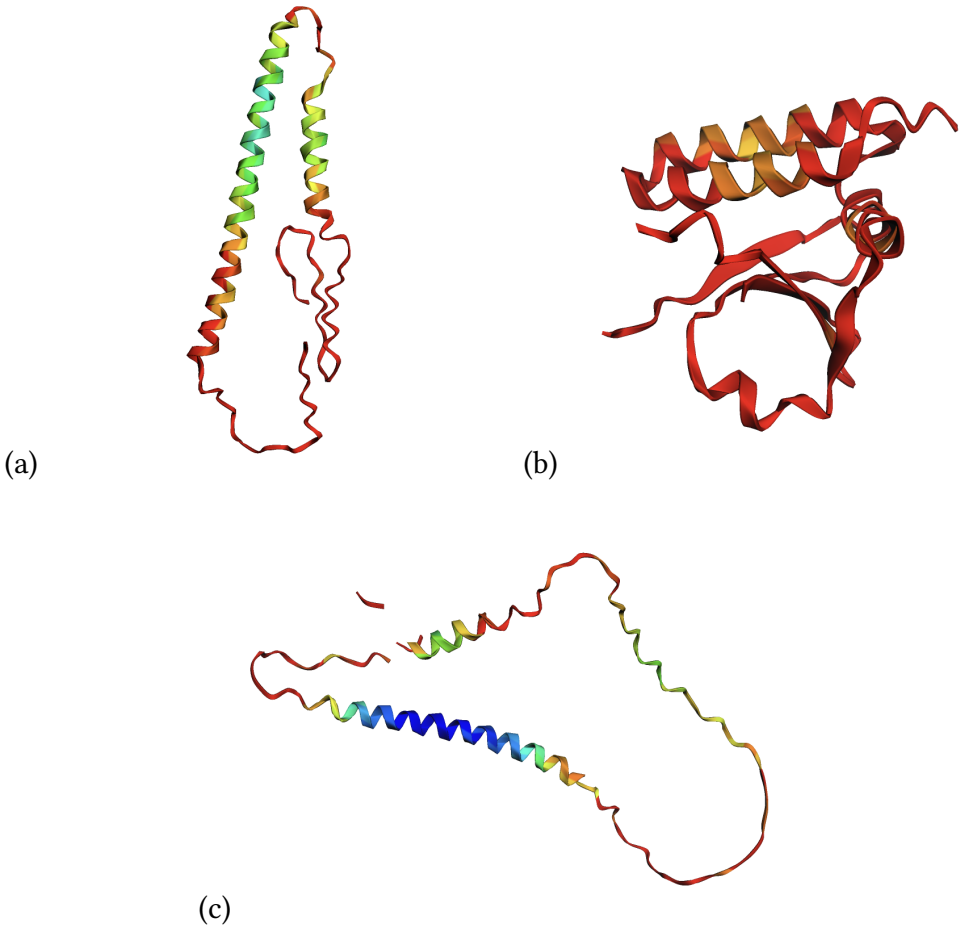


Figure 7.6: Alpha Variant Mutational Protein with (a) MuFormer's Fixed Backbone design (b) MuFormer's Backbone atom design (c) AlphaFold2

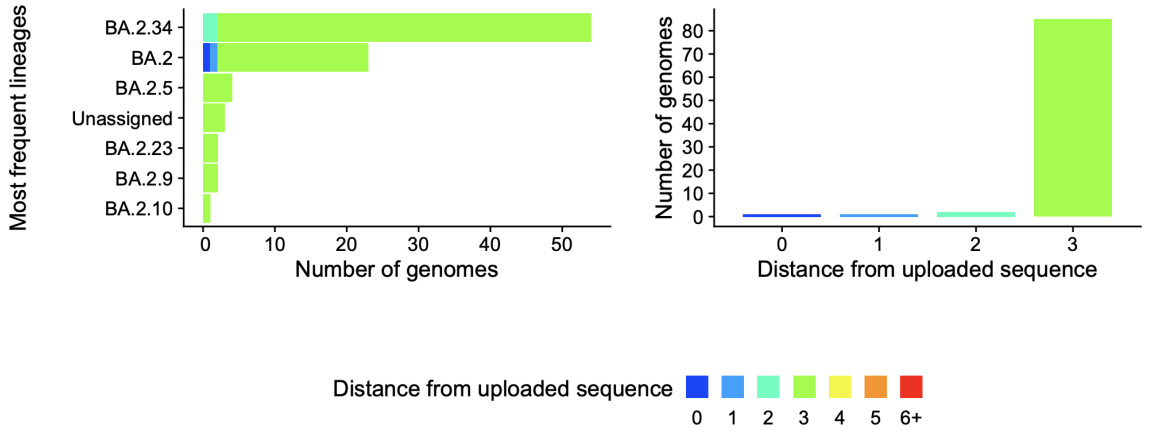
of zero (Figure 7.8 (d)). Although most of the genomes here are at a genomic distance of 4.

The MuFormer1-fixbb model matched its final sequence design with BA.1.1 lineage (Figure 7.8 (c)) which comes under Omicron variant with a genomic distance of zero. Moreover, all the closest match genomes detected are within the genomic distance of at most one, with almost 30 closest match genome sequences being in the genomic distance of zero. The other predicted lineage BA.1 comes under Omicron variant as well.

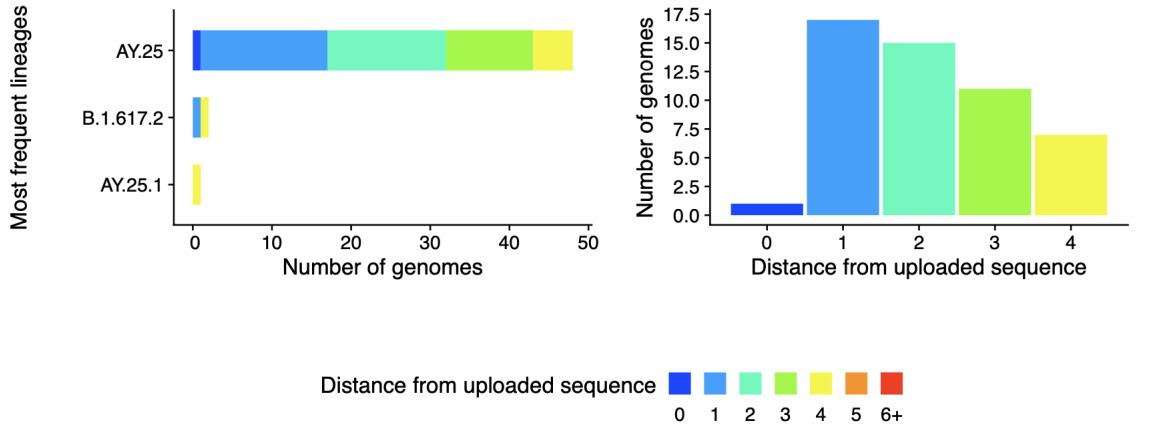
For the baseline AlphaFold2 model, GISAID or even NCBI blast tools [166] were unable to find any similar sequence, which depicts the inability of AlphaFold2 to mutate within a constrained protein space. The addition of sequence representation in the form of an inductive bias enabled the MuFormer suite of models to perform in vitro mutagenesis in the constrained virus space, whilst learning mutational relations from the embeddings and MSA profile.

Further, as an experiment, CoVBERT protein language model was used to generate the input encoded embeddings for the input sequence, and these embeddings were fed into AlphaFold2 as an inductive bias for the same fixed backbone design process to mutate sequences. The final designed sequence after 110 gradient steps, was identified as B.1.1.7 variant which is Alpha variant, as the closest match on GISAID with a genomic distance of zero (Figure 7.9). This suggests that although CoVBERT was able to remain in the constrained protein space with its learned language of proteins, it was unable to capture mutational relationships. It is hypothesised that the reason for this could be that CoVBERT was pre-trained over just the SARS-CoV-2 spike glycoprotein data, whereas here we are trying to design a sequence for the entire genome.

All in all, the ability of MuFormer to mutate the sequence tokens with mutational information sheds light over the capability of large pre-trained language models to decipher protein properties, functions along with the mutational landscape.



(a)



(b)

Figure 7.7: SARS-CoV-2 closest genomic lineage detected for sequences designed by (a) MuFormer2-fixbb and (b) MuFormer2-bba

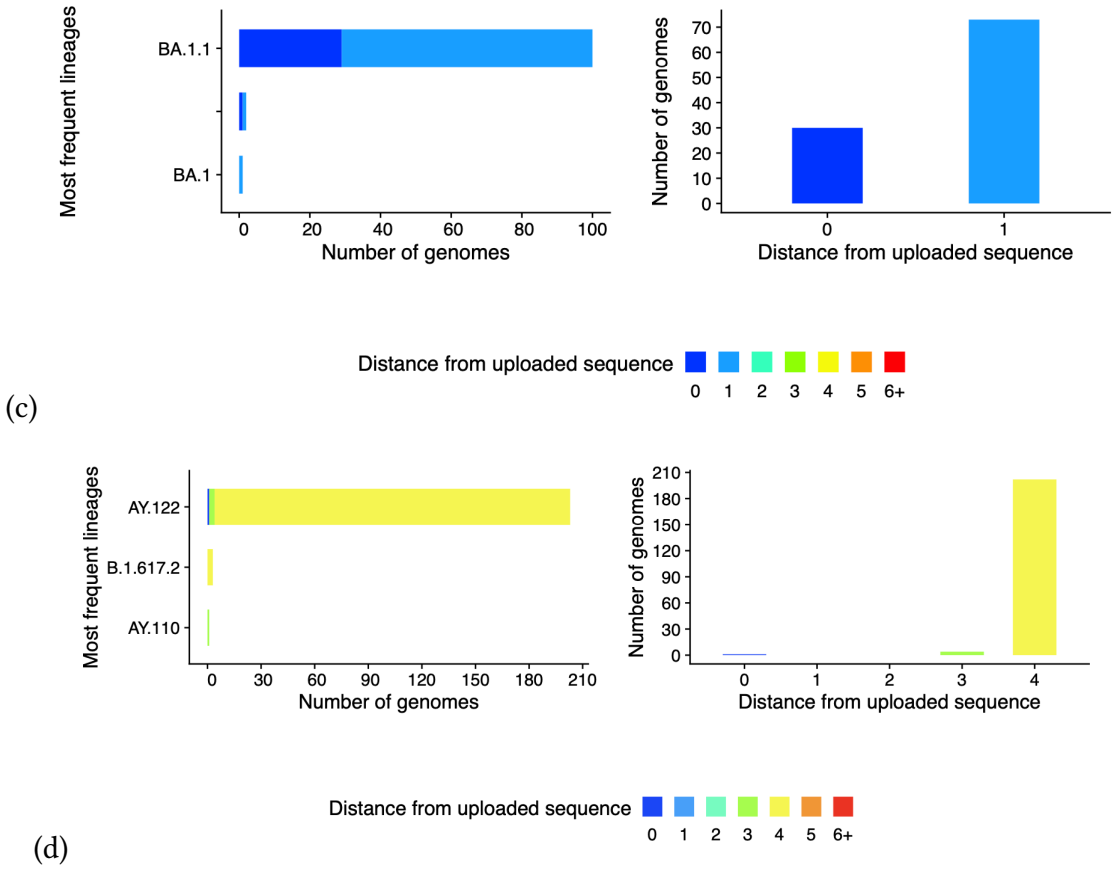


Figure 7.8: SARS-CoV-2 closest genomic lineage detected for sequences designed by (a) MuFormer1-fixbb and (b) MuFormer1-bba

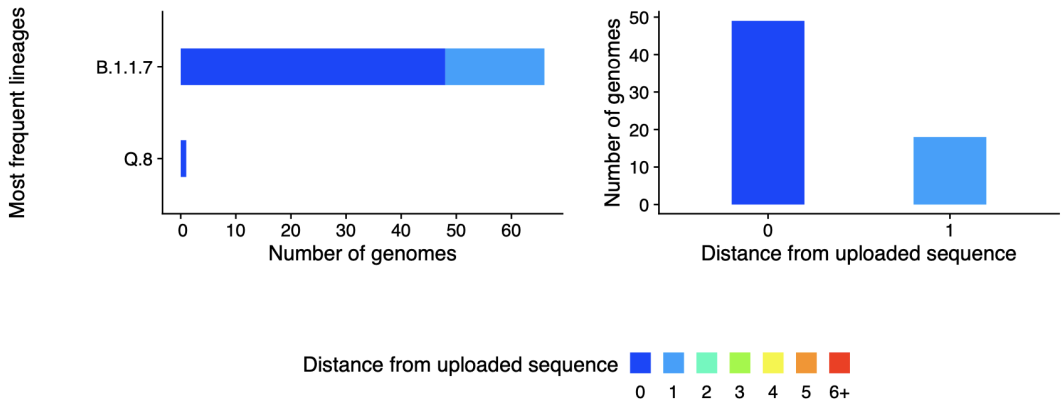


Figure 7.9: SARS-CoV-2 closest genomic lineage detected for the sequence designed by CoVBERT-MuFormer (with fixbb optimisation)

8 Conclusion

Protein representation learning is a fascinating field that has a lot of potential for growth, innovation, and influence. Self-supervised learning will continue to play a key role in the future of computational protein modelling due to the constantly rising gap between labelled and unlabeled protein data. Deep language models are indeed an exciting breakthrough in protein sequence modelling, enabling us to find structural and functional characteristics of a corpus of sequences based solely on evolutionary links. If neural networks can use information learnt from protein sequences to build functional proteins, they might be combined with prediction models to develop and optimise sequences for specific activities. The current size of sequence data and its projected growth reflect the possibility of a general-purpose generative model to condense the totality of sequence statistics whilst integrating fundamental chemical and biological concepts such as structure, function, activity, localization, binding, and dynamics to generate new unseen sequences. The possibility of generating phylogenetic future mutations of viruses possesses massive potential to help eradicate pandemics at an early stage with the development of robust drugs and relevant policy undertakings. Moreover, by learning the language of proteins, there is huge potential for drug development and antibiotics resistance studies.

8.1 Discussion and Future Work

In this work, two novel protein language models have been introduced: CoVBERT and MuFormer wherein, CoVBERT relies on just protein sequence data, and MuFormer leverages both sequence and structural data for fixed backbone sequence design.
CoVBERT, which is trained with a masked language modelling objective is able to capture

infectious point mutations for the spike glycoprotein. However, in a virus genome sequence the mutational positions could occur at contiguous positions as well. In order to pre-train the model with contiguous positions of co-occurring mutations, it might be better to use Span Masking instead of the vanilla masking from BERT [78]. Further, use of rotary positional embeddings (RoPE) [115] or Attention with Linear Biases [167] as the positional embeddings for CoVBERT could improve its architecture by enabling it to extrapolate beyond the context window of 2048 and encapture longer range dependencies since the whole SARS-CoV-2 genome consists of about 29k nucleotides.

Although, transformers are a state-of-the-art architecture, it scales quadratically with respect to the length of the sequence. As we're dealing with protein sequences which could go upto 29k characters, the sparsity of mutations with regular transformers could lead to overfitting of similar parts and ignoring the mutations. Transformer variants like Performer, which performs an unbiased fast attention approximation with a lower variant [168] can help in bringing down the time complexity to linear time and space. Apart from this, training CoVBERT for greater than 5,000 steps could potentially assist in improving the point mutation prediction further, which would require longer training duration and GPUs. CoVBERT could further also be fine-tuned to downstream tasks for predicting the proteins structural and functional properties.

With just sequence embeddings from a protein language model as an inductive bias, MuFormer FixBB and BBA models are able to perform better than AlphaFold2. However, having a full end-to-end loop with gradients backpropagating to the language model as well could potentially improve the results further, instead of using just the frozen protein language model weights.

All in all, language models ability to decipher mutations in proteins evolutionary context reflects over its usefulness to predict future virus mutations for SARS-CoV-2 as well as other viruses, to help inform therapeutic decisions early on.

Appendices

.1 AdamW Optimisation Algorithm

The pre-training for CoVBERT has been performed using AdamW optimiser [120] (Algorithm TODO) with a learning rate of $\eta = 5e - 5$ and a decoupled weight decay of 1e-2. AdamW which is a stochastic optimisation method, adapts gradient step sizes for each individual weight and consists of a decoupled weight decay (decoupled from the gradient update) which is performed after controlling the parameter-wise step size, unlike the regular Adam optimiser [146].

The L_2 regularisation for Adam optimiser is usually implemented with:

$$g_t = \nabla f(\theta_t) + w_t \theta_t \tag{1}$$

where w_t is the weight decay rate at time t.

On the contrary, in the AdamW optimiser, the weight decay term is adjusted to appear in the gradient update such that:

$$\theta_{t+1,i} = \theta_{t,i} - \eta(\frac{1}{\sqrt{\hat{v}_t + \epsilon}} \cdot \hat{m}_t + w_{t,i}\theta_{t,i}), \forall t \tag{2}$$

Algorithm 7 AdamW algorithm with decoupled weight decay

Input: γ (lr), β_1, β_2 (betas), θ_0 (params), $f(\theta)$(objective), ϵ (epsilon), λ (weight decay), maximise
Initialise: $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$, second moment, $\hat{v_0}^{max} \leftarrow 0$

for $t = 1$ to $\ldots$ **do**
  if maximise: **then**
    $g_t \leftarrow -\nabla\theta f_t(\theta_t - 1)$
  else
    $g_t \leftarrow \nabla\theta f_t(\theta_t - 1)$
  end if
  $\theta_t \leftarrow \theta_{t-1} - \gamma\lambda\theta t - 1$
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  $\hat{m}_t \leftarrow \frac{m_t}{(1-\beta_1^t)}$
  $\hat{v}_t \leftarrow \frac{v_t}{(1-\beta_2^t)}$
  $\theta_t \leftarrow \theta_t - \frac{\gamma \hat{m}_t}{\sqrt{\hat{v}_t}} + \epsilon$
end for
return θ_t

References

[1] Áine O'Toole et al. "Assignment of epidemiological lineages in an emerging pandemic using the pangolin tool". In: *Virus Evolution* 7.2 (2021), veab064.

[2] Fabian Sievers et al. "Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega". In: *Molecular systems biology* 7.1 (2011), p. 539.

[3] Martin Steinegger et al. "HH-suite3 for fast remote homology detection and deep protein annotation". In: *BMC bioinformatics* 20.1 (2019), pp. 1–15.

[4] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589.

[5] Karim Beguir et al. "Early computational detection of potential high risk SARS-CoV-2 variants". In: *bioRxiv* (2021).

[6] Yuelong Shu and John McCauley. "GISAID: Global initiative on sharing all influenza data–from vision to reality". In: *Eurosurveillance* 22.13 (2017), p. 30494.

[7] Ali Madani et al. "Progen: Language modeling for protein generation". In: *arXiv preprint arXiv:2004.03497* (2020).

[8] Jung-Eun Shin et al. "Protein design and variant prediction using autoregressive generative models". In: *Nature communications* 12.1 (2021), pp. 1–11.

[9] Noelia Ferruz, Steffen Schmidt, and Birte Höcker. "ProtGPT2 is a deep unsupervised language model for protein design". In: *Nature communications* 13.1 (2022), pp. 1–10.

[10] Daniel Hesslow et al. "RITA: a Study on Scaling Up Generative Protein Sequence Models". In: *arXiv preprint arXiv:2205.05789* (2022).

[11] Ethan C Alley et al. "Unified rational protein engineering with sequence-based deep representation learning". In: *Nature methods* 16.12 (2019), pp. 1315–1322.

[12] Michael Heinzinger et al. "Modeling the language of life–deep learning protein sequences". In: *Biorxiv* (2019), p. 614313.

[13] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[14] Baris E Suzek et al. "UniRef: comprehensive and non-redundant UniProt reference clusters". In: *Bioinformatics* 23.10 (2007), pp. 1282–1288.

[15] Alexander Rives et al. "Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences". In: *Proceedings of the National Academy of Sciences* 118.15 (2021).

[16] Roshan Rao et al. "Evaluating protein transfer learning with TAPE". In: *Advances in neural information processing systems* 32 (2019).

[17] Ahmed Elnaggar et al. “ProtTrans: towards cracking the language of Life’s code through self-supervised deep learning and high performance computing”. In: *arXiv preprint arXiv:2007.06225* (2020).

[18] Nadav Brandes et al. “ProteinBERT: A universal deep-learning model of protein sequence and function”. In: *Bioinformatics* 38.8 (2022), pp. 2102–2110.

[19] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[20] Roshan M Rao et al. “MSA transformer”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8844–8856.

[21] Adam J Riesselman, John B Ingraham, and Debora S Marks. “Deep generative models of genetic variation capture the effects of mutations”. In: *Nature methods* 15.10 (2018), pp. 816–822.

[22] Joshua Meier et al. “Language models enable zero-shot prediction of the effects of mutations on protein function”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[23] Ratul Chowdhury et al. “Single-sequence protein structure prediction using language models from deep learning”. In: *bioRxiv* (2021).

[24] Brian L Hie, Kevin K Yang, and Peter S Kim. “Evolutionary velocity with protein language models predicts evolutionary dynamics of diverse proteins”. In: *Cell Systems* 13.4 (2022), pp. 274–285.

[25] Jeffrey A Ruffolo, Jeffrey J Gray, and Jeremias Sulam. “Deciphering antibody affinity maturation with language models and weakly supervised learning”. In: *arXiv preprint arXiv:2112.07782* (2021).

[26] Ali Madani et al. “Deep neural language modeling enables functional protein generation across families”. In: *bioRxiv* (2021).

[27] Brian Hie et al. “Learning the language of viral evolution and escape”. In: *Science* 371.6526 (2021), pp. 284–288.

[28] Jesse Vig et al. “Bertology meets biology: Interpreting attention in protein language models”. In: *arXiv preprint arXiv:2006.15222* (2020).

[29] Philippe Schwaller et al. “Extraction of organic chemistry grammar from unsupervised learning of chemical reactions”. In: *Science Advances* 7.15 (2021), eabe4166.

[30] David Weininger. “SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules”. In: *Journal of chemical information and computer sciences* 28.1 (1988), pp. 31–36.

[31] John Ingraham et al. “Generative models for graph-based protein design”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[32] Andrew Leaver-Fay et al. “ROSETTA3: an object-oriented software suite for the simulation and design of macromolecules”. In: *Methods in enzymology*. Vol. 487. Elsevier, 2011, pp. 545–574.

[33] Chloe Hsu et al. “Learning inverse folding from millions of predicted structures”. In: *bioRxiv* (2022).

[34] Justas Dauparas et al. “Robust deep learning based protein sequence design using ProteinMPNN”. In: *bioRxiv* (2022).

[35] Basile IM Wicky et al. “Hallucinating protein assemblies”. In: *bioRxiv* (2022).

[36] Jeanne Trinquier et al. “Efficient generative modeling of protein sequences using simple autoregressive models”. In: *Nature communications* 12.1 (2021), pp. 1–11.

[37] Jinwoo Leem et al. “Deciphering the language of antibodies using self-supervised learning”. In: *bioRxiv* (2021).

[38] Brian Hie et al. “Learning mutational semantics”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9109–9121.

[39] Pascal Notin et al. “Tranception: protein fitness prediction with autoregressive transformers and inference-time retrieval”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 16990–17017.

[40] Hakime Öztürk et al. “Exploring chemical space using natural language processing methodologies for drug discovery”. In: *Drug Discovery Today* 25.4 (2020), pp. 689–705.

[41] Douglas M Fowler and Stanley Fields. “Deep mutational scanning: a new style of protein science”. In: *Nature methods* 11.8 (2014), pp. 801–807.

[42] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. “A neural probabilistic language model”. In: *Advances in neural information processing systems* 13 (2000).

[43] Claude Elwood Shannon. “A mathematical theory of communication”. In: *ACM SIGMOBILE mobile computing and communications review* 5.1 (2001), pp. 3–55.

[44] Jack Kiefer and Jacob Wolfowitz. “Stochastic estimation of the maximum of a regression function”. In: *The Annals of Mathematical Statistics* (1952), pp. 462–466.

[45] Yoav Goldberg. “A primer on neural network models for natural language processing”. In: *Journal of Artificial Intelligence Research* 57 (2016), pp. 345–420.

[46] Zellig S Harris. “Distributional structure”. In: *Word* 10.2-3 (1954), pp. 146–162.

[47] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[48] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[49] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[50] Mike Schuster and Kuldip K Paliwal. “Bidirectional recurrent neural networks”. In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[51] Francois Chollet. *Deep learning mit python und keras: das praxis-handbuch vom entwickler der keras-bibliothek*. MITP-Verlags GmbH & Co. KG, 2018.

[52] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[53] Paul J Werbos. “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[54] Mikael Boden. “A guide to recurrent neural networks and backpropagation”. In: *the Dallas project* 2.2 (2002), pp. 1–10.

[55] Kyunghyun Cho et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014).

[56] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems* 27 (2014).

[57] Dan Ofer, Nadav Brandes, and Michal Linial. "The language of proteins: NLP, machine learning & protein sequences". In: *Computational and Structural Biotechnology Journal* 19 (2021), pp. 1750–1758.

[58] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[59] Zhilin Yang et al. "Xlnet: Generalized autoregressive pretraining for language understanding". In: *Advances in neural information processing systems* 32 (2019).

[60] Robert C Edgar and Serafim Batzoglou. "Multiple sequence alignment". In: *Current opinion in structural biology* 16.3 (2006), pp. 368–373.

[61] Adam Riesselman et al. "Accelerating protein design using autoregressive generative models". In: *BioRxiv* (2019), p. 757252.

[62] IUPAC-IUB JCBN. "Recommendations 1983 on 'Nomenclature and Symbolism for Amino Acids and Peptides". In: *Pure Appl. Chem* 56 (1984), p. 595.

[63] Zach Solan et al. "Unsupervised learning of natural languages". In: *Proceedings of the National Academy of Sciences* 102.33 (2005), pp. 11629–11634.

[64] Lijia Yu et al. "Grammar of protein domain architectures". In: *Proceedings of the National Academy of Sciences* 116.9 (2019), pp. 3636–3645.

[65] Michael Schantz Klausen et al. "NetSurfP-2.0: Improved prediction of protein structural features by integrated deep learning". In: *Proteins: Structure, Function, and Bioinformatics* 87.6 (2019), pp. 520–527.

[66] José Juan Almagro Armenteros et al. "DeepLoc: prediction of protein subcellular localization using deep learning". In: *Bioinformatics* 33.21 (2017), pp. 3387–3395.

[67] Maria Schelling, Thomas A Hopf, and Burkhard Rost. "Evolutionary couplings and sequence variation effect predict protein binding sites". In: *Proteins: Structure, Function, and Bioinformatics* 86.10 (2018), pp. 1064–1074.

[68] Brian L Hie, Kevin K Yang, and Peter S Kim. "Evolutionary velocity with protein language models". In: *bioRxiv* (2021).

[69] Surojit Biswas et al. "Low-N protein engineering with data-efficient deep learning". In: *Nature methods* 18.4 (2021), pp. 389–396.

[70] Tristan Bepler and Bonnie Berger. "Learning protein sequence embeddings using information from structure". In: *arXiv preprint arXiv:1902.08661* (2019).

[71] Tristan Bepler and Bonnie Berger. "Learning the protein language: Evolution, structure, and function". In: *Cell systems* 12.6 (2021), pp. 654–669.

[72] Brian Hie, Bryan D Bryson, and Bonnie Berger. "Leveraging uncertainty in machine learning accelerates biological discovery and design". In: *Cell systems* 11.5 (2020), pp. 461–477.

[73] Yunan Luo et al. "Evolutionary context-integrated deep sequence modeling for protein engineering". In: *International Conference on Research in Computational Molecular Biology*. Springer. 2020, pp. 261–263.

[74] Frances H Arnold. "Design by directed evolution". In: *Accounts of chemical research* 31.3 (1998), pp. 125–131.

[75] Po-Ssu Huang, Scott E Boyken, and David Baker. “The coming of age of de novo protein design”. In: *Nature* 537.7620 (2016), pp. 320–327.

[76] Thomas A Hopf et al. “Mutation effects predicted from sequence co-variation”. In: *Nature biotechnology* 35.2 (2017), pp. 128–135.

[77] Jaclyn K Mann et al. “The fitness landscape of HIV-1 gag: advanced modelling approaches and validation of model predictions by in vitro testing”. In: *PLoS computational biology* 10.8 (2014), e1003776.

[78] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[79] Zak Costello and Hector Garcia Martin. “How to hallucinate functional proteins”. In: *arXiv preprint arXiv:1903.00458* (2019).

[80] Yanbin Wang et al. “A high efficient biological language model for predicting protein–protein interactions”. In: *Cells* 8.2 (2019), p. 122.

[81] Ehsaneddin Asgari, Alice C McHardy, and Mohammad RK Mofrad. “Probabilistic variable-length segmentation of protein sequences for discriminative motif discovery (DiMotif) and sequence embedding (ProtVecX)”. In: *Scientific reports* 9.1 (2019), pp. 1–16.

[82] Asa Ben-Hur and Douglas Brutlag. “Sequence motifs: highly predictive features of protein function”. In: *Feature extraction*. Springer, 2006, pp. 625–645.

[83] Wang Liang and Zhao KaiYong. “Detecting" protein words" through unsupervised word segmentation”. In: *arXiv preprint arXiv:1404.6866* (2014).

[84] Nadav Brandes, Dan Ofer, and Michal Linial. “ASAP: a machine learning framework for local protein properties”. In: *Database* 2016 (2016).

[85] Annalisa Barla et al. “Machine learning methods for predictive proteomics”. In: *Briefings in bioinformatics* 9.2 (2008), pp. 119–128.

[86] Edward A Weathers et al. “Reduced amino acid alphabet is sufficient to accurately recognize intrinsically disordered protein”. In: *FEBS letters* 576.3 (2004), pp. 348–352.

[87] Dan Ofer and Michal Linial. “ProFET: Feature engineering captures high-level protein functions”. In: *Bioinformatics* 31.21 (2015), pp. 3429–3436.

[88] Ehsaneddin Asgari and Mohammad RK Mofrad. “Continuous distributed representation of biological sequences for deep proteomics and genomics”. In: *PloS one* 10.11 (2015), e0141287.

[89] Dan Ofer. “Machine Learning for Protein Function”. In: *arXiv preprint arXiv:1603.02021* (2016).

[90] Lynne Reed Murphy, Anders Wallqvist, and Ronald M Levy. “Simplified amino acid alphabets for protein fold recognition and implications for folding”. In: *Protein engineering* 13.3 (2000), pp. 149–152.

[91] Urvashi Khandelwal et al. “Sharp nearby, fuzzy far away: How neural language models use context”. In: *arXiv preprint arXiv:1805.04623* (2018).

[92] Sepp Hochreiter et al. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. 2001.

[93] Alan S Lapedes et al. “Correlated mutations in models of protein sequences: phylogenetic and structural effects”. In: *Lecture Notes-Monograph Series* (1999), pp. 236–256.

[94] Martin Weigt et al. “Identification of direct residue contacts in protein–protein interaction by message passing”. In: *Proceedings of the National Academy of Sciences* 106.1 (2009), pp. 67–72.

[95] Yaron Geffen, Yanay Ofran, and Ron Unger. "DistilProtBert: A distilled protein language model used to distinguish between real proteins and their randomly shuffled counterparts". In: *bioRxiv* (2022).

[96] Bette Korber et al. "Tracking changes in SARS-CoV-2 spike: evidence that D614G increases infectivity of the COVID-19 virus". In: *Cell* 182.4 (2020), pp. 812–827.

[97] Rishi Bommasani et al. "On the opportunities and risks of foundation models". In: *arXiv preprint arXiv:2108.07258* (2021).

[98] Nicholas Magazine et al. "Mutations and evolution of the SARS-CoV-2 spike protein". In: *Viruses* 14.3 (2022), p. 640.

[99] Wilson L Taylor. ""Cloze procedure": A new tool for measuring readability". In: *Journalism quarterly* 30.4 (1953), pp. 415–433.

[100] ME Peters M Neumann et al. "Deep contextualized word representations". In: *arXiv preprint arXiv:1802.05365* (2018).

[101] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

[102] Pascal Vincent et al. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1096–1103.

[103] Thomas Wolf et al. "Huggingface's transformers: State-of-the-art natural language processing". In: *arXiv preprint arXiv:1910.03771* (2019).

[104] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Neural machine translation of rare words with subword units". In: *arXiv preprint arXiv:1508.07909* (2015).

[105] Nikolaos Pappas, Lesly Miculicich Werlen, and James Henderson. "Beyond weight tying: Learning joint input-output embeddings for neural machine translation". In: *arXiv preprint arXiv:1808.10681* (2018).

[106] Ofir Press and Lior Wolf. "Using the output embedding to improve language models". In: *arXiv preprint arXiv:1608.05859* (2016).

[107] Dan Hendrycks and Kevin Gimpel. "Gaussian error linear units (gelus)". In: *arXiv preprint arXiv:1606.08415* (2016).

[108] Abien Fred Agarap. "Deep learning using rectified linear units (relu)". In: *arXiv preprint arXiv:1803.08375* (2018).

[109] Ruibin Xiong et al. "On layer normalization in the transformer architecture". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 10524–10533.

[110] Mary Phuong and Marcus Hutter. "Formal Algorithms for Transformers". In: *arXiv preprint arXiv:2207.09238* (2022).

[111] Yuan Huang et al. "Structural and functional properties of SARS-CoV-2 spike protein: potential antivirus drug development for COVID-19". In: *Acta Pharmacologica Sinica* 41.9 (2020), pp. 1141–1149.

[112] Yonghui Wu et al. "Google's neural machine translation system: Bridging the gap between human and machine translation". In: *arXiv preprint arXiv:1609.08144* (2016).

[113] Linting Xue et al. "Byt5: Towards a token-free future with pre-trained byte-to-byte models". In: *Transactions of the Association for Computational Linguistics* 10 (2022), pp. 291–306.

[114] Kaj Bostrom and Greg Durrett. "Byte pair encoding is suboptimal for language model pretraining". In: *arXiv preprint arXiv:2004.03720* (2020).

[115] Jianlin Su et al. "Roformer: Enhanced transformer with rotary position embedding". In: *arXiv preprint arXiv:2104.09864* (2021).

[116] Zeming Lin et al. "Language models of protein sequences at the scale of evolution enable accurate structure prediction". In: *bioRxiv* (2022).

[117] Alex Wang and Kyunghyun Cho. "Bert has a mouth, and it must speak: Bert as a markov random field language model". In: *arXiv preprint arXiv:1902.04094* (2019).

[118] Yinhan Liu et al. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[119] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019).

[120] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017).

[121] Victor Sanh et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *arXiv preprint arXiv:1910.01108* (2019).

[122] Xiaoqi Jiao et al. "Tinybert: Distilling bert for natural language understanding". In: *arXiv preprint arXiv:1909.10351* (2019).

[123] Andrew Richards. "University Of Oxford Advanced Research Computing". In: (2015). URL: https://zenodo.org/record/22558.

[124] Karthik Gangavarapu et al. "Outbreak. info genomic reports: scalable and dynamic surveillance of SARS-CoV-2 variants and mutations". In: *Research square* (2022).

[125] Martin Steinegger, Milot Mirdita, and Johannes Söding. "Protein-level assembly increases protein sequence recovery from metagenomic samples manyfold". In: *Nature methods* 16.7 (2019), pp. 603–606.

[126] UniProt Consortium. "UniProt: a worldwide hub of protein knowledge". In: *Nucleic acids research* 47.D1 (2019), pp. D506–D515.

[127] Brigitte Boeckmann et al. "The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003". In: *Nucleic acids research* 31.1 (2003), pp. 365–370.

[128] Yang You et al. "Large batch optimization for deep learning: Training bert in 76 minutes". In: *arXiv preprint arXiv:1904.00962* (2019).

[129] Kevin Clark et al. "Electra: Pre-training text encoders as discriminators rather than generators". In: *arXiv preprint arXiv:2003.10555* (2020).

[130] Jacqui Wise. *Covid-19: The E484K mutation and the risks it poses*. 2021.

[131] Yonatan Belinkov and James Glass. "Analysis methods in neural language processing: A survey". In: *Transactions of the Association for Computational Linguistics* 7 (2019), pp. 49–72.

[132] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[133] Sarthak Jain and Byron C Wallace. "Attention is not explanation". In: *arXiv preprint arXiv:1902.10186* (2019).

[134] Philippe Schwaller et al. "Unsupervised attention-guided atom-mapping". In: (2020).

[135] Timothy Niven and Hung-Yu Kao. "Probing neural network comprehension of natural language arguments". In: *arXiv preprint arXiv:1907.07355* (2019).

[136] Yi Chern Tan and L Elisa Celis. "Assessing social and intersectional biases in contextualized word representations". In: *Advances in Neural Information Processing Systems* 32 (2019).

[137] Jun Lan et al. "Structure of the SARS-CoV-2 spike receptor-binding domain bound to the ACE2 receptor". In: *Nature* 581.7807 (2020), pp. 215–220.

[138] Roshan Rao et al. "Transformer protein language models are unsupervised structure learners". In: *International Conference on Learning Representations*. 2020.

[139] James P Roney and Sergey Ovchinnikov. "State-of-the-Art estimation of protein model accuracy using AlphaFold". In: *BioRxiv* (2022).

[140] Brian Kuhlman and David Baker. "Native protein sequences are close to optimal for their structures". In: *Proceedings of the National Academy of Sciences* 97.19 (2000), pp. 10383–10388.

[141] Rebecca F Alford et al. "The Rosetta all-atom energy function for macromolecular modeling and design". In: *Journal of chemical theory and computation* 13.6 (2017), pp. 3031–3048.

[142] Hahnbeom Park et al. "Simultaneous optimization of biomolecular energy functions on features from small molecules and macromolecules". In: *Journal of chemical theory and computation* 12.12 (2016), pp. 6201–6212.

[143] Sergey Ovchinnikov. *ColabDesign*. https://github.com/sokrypton/ColabDesign. 2022.

[144] Andrew W Senior et al. "Improved protein structure prediction using potentials from deep learning". In: *Nature* 577.7792 (2020), pp. 706–710.

[145] Joseph Turian, Lev Ratinov, and Yoshua Bengio. "Word representations: a simple and general method for semi-supervised learning". In: *Proceedings of the 48th annual meeting of the association for computational linguistics*. 2010, pp. 384–394.

[146] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[147] Andriy Kryshtafovych et al. "Critical assessment of methods of protein structure prediction (CASP)—Round XIV". In: *Proteins: Structure, Function, and Bioinformatics* 89.12 (2021), pp. 1607–1617.

[148] Jürgen Haas et al. "Continuous Automated Model EvaluatiOn (CAMEO) complementing the critical assessment of structure prediction in CASP12". In: *Proteins: Structure, Function, and Bioinformatics* 86 (2018), pp. 387–398.

[149] Jared Kaplan et al. "Scaling laws for neural language models". In: *arXiv preprint arXiv:2001.08361* (2020).

[150] Julian Besag. "Statistical analysis of non-lattice data". In: *Journal of the Royal Statistical Society: Series D (The Statistician)* 24.3 (1975), pp. 179–195.

[151] Kaiming He et al. "Identity mappings in deep residual networks". In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[152] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[153] Alec Radford et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[154] Jonathan Baxter. "A model of inductive bias learning". In: *Journal of artificial intelligence research* 12 (2000), pp. 149–198.

[155] Protein Data Bank. "Protein data bank". In: *Nature New Biol* 233 (1971), p. 223.

[156] Christopher A Voigt, D Benjamin Gordon, and Stephen L Mayo. "Trading accuracy for speed: A quantitative comparison of search algorithms in protein sequence design". In: *Journal of molecular biology* 299.3 (2000), pp. 789–803.

[157] Brian Kuhlman et al. "Design of a novel globular protein fold with atomic-level accuracy". In: *science* 302.5649 (2003), pp. 1364–1368.

[158] Christoffer Norn et al. "Protein sequence design by conformational landscape optimization". In: *Proceedings of the National Academy of Sciences* 118.11 (2021), e2017228118.

[159] Namrata Anand et al. "Protein sequence design with a learned potential". In: *Nature communications* 13.1 (2022), pp. 1–11.

[160] Martin Steinegger and Johannes Söding. "MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets". In: *Nature biotechnology* 35.11 (2017), pp. 1026–1028.

[161] Martin Steinegger and Johannes Söding. "Clustering huge protein sequence sets in linear time". In: *Nature communications* 9.1 (2018), pp. 1–8.

[162] Marshall Fixman. "Radius of gyration of polymer chains". In: *The Journal of Chemical Physics* 36.2 (1962), pp. 306–310.

[163] Valerio Mariani et al. "lDDT: a local superposition-free score for comparing protein structures and models using distance difference tests". In: *Bioinformatics* 29.21 (2013), pp. 2722–2728.

[164] Ulrike Göbel et al. "Correlated mutations and residue contacts in proteins". In: *Proteins: Structure, Function, and Bioinformatics* 18.4 (1994), pp. 309–317.

[165] Kevin K Yang, Zachary Wu, and Frances H Arnold. "Machine-learning-guided directed evolution for protein engineering". In: *Nature methods* 16.8 (2019), pp. 687–694.

[166] David L Wheeler et al. "Database resources of the national center for biotechnology information". In: *Nucleic acids research* 36.suppl_1 (2007), pp. D13–D21.

[167] Ofir Press, Noah A Smith, and Mike Lewis. "Train short, test long: Attention with linear biases enables input length extrapolation". In: *arXiv preprint arXiv:2108.12409* (2021).

[168] Krzysztof Choromanski et al. "Rethinking attention with performers". In: *arXiv preprint arXiv:2009.14794* (2020).