UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Deep Learning To Optimize Viral Vector Production For Human Gene Therapy

João Lucas Figueiredo Ferraz

Mestrado em Engenharia Informática

Dissertação orientada por:
Profª. Doutora Cátia Luísa Santana Calisto Pesquita
Doutora Ana Filipa Rodrigues

2025

Agradecimentos

Em primeiro lugar agradeço à minha orientadora Cátia Pesquita pela sua dedicação, paciência, compreensão e suporte ao longo de todo o trabalho desta tese, foi um apoio que me permitiu não só uma grande evolução a nível profissional como também a nível pessoal, foi um privilégio para mim ter passado este ano sob a sua orientação.

Agradeço à minha co-orientadora Ana Filipa Rodrigues, pelo suporte contínuo, pelas discussões esclarecedoras e pela sua atenção ao detalhe, que enriqueceram imenso este projeto e a minha evolução ao longo do ano.

Aos meus colegas do LISEDA, pelo espírito de companheirismo, partilha e apoio mútuo. Em especial, agradeço ao Pedro Cotovio, pela sua disponibilidade, sugestões e pela sua amizade, estando sempre lá nos momentos mais desafiantes e contribuindo imenso para o meu desenvolvimento profissional.

Ao LASIGE, agradeço por proporcionar os recursos, espaços e ambiente de investigação que tornaram possível a realização deste trabalho.

Agradeço também a todos professores da FCUL que me acompanharam ao longo dos anos.

Por fim, agradeço de forma especial à minha família pelo amor, compreensão e apoio incondicional ao longo de toda a minha vida e jornada académica.

Este trabalho é o resultado de um esforço conjunto, e estou profundamente grato a todos que dele fizeram parte.

Dedico esta tese aos meus pais, João e Miriam, e à minha irmã Clara (aka Bino), que sempre estiveram ao meu lado, expresso-vos a minha mais sincera gratidão. Que esta dedicatória seja um reflexo do apreço que tenho por vocês e por tudo o que sempre fizeram por mim. Cada conquista minha é também vossa. Muito obrigado.

Resumo

A crescente aplicação de modelos de Deep Learning na área da Bioinformática tem mostrado resultados promissores na previsão funcional e design de sequências proteicas. A utilização de Protein Language Models (PLMs) pode permitir a exploração de um espaço maior de possibilidades nestas tarefas, capturando padrões que podem não ser identificados por métodos tradicionais. O uso de PLMs pode transformar o campo da engenharia de proteínas, permitindo previsões mais rápidas e precisas sobre a funcionalidade de novas sequências. Esta abordagem pode também facilitar a identificação de novas variantes de sequências de proteínas com propriedades otimizadas para aplicações específicas. Esta tese investiga o potencial dos PLMs para otimizar o design de novas sequências de Vírus Adeno-Associado 2 (AAV2), concentrando-se em dois objetivos principais: a classificação de sequências e o design generativo das mesmas.

Para a tarefa de classificação, um modelo de linguagem pré-treinado (ProtBERT) foi fine-tuned para diferenciar com precisão entre sequências viáveis e não viáveis de AAV2. Os resultados demonstraram um alto desempenho de classificação em todos os modelos treinados, validando a hipótese de que o fine-tuning de um PLM num domínio específico permite aos PLMs capturar efetivamente características importantes das sequências de AAV2. No que diz respeito à geração de sequências, um modelo generativo condicional (ProGen) foi treinado para desenhar sequências viáveis da proteína do cápside do AAV2. Embora o modelo tenha gerado sequências estruturalmente diversas, avaliações detalhadas indicaram a necessidade de refinamentos adicionais para alinhar consistentemente as sequências geradas com os critérios de viabilidade.

O modelo de classificação destaca o potencial dos PLMs na previsão da viabilidade de sequências, oferecendo uma abordagem elaborada que pode reduzir custos experimentais e simplificar o processo de design de novas sequências AAV2, já que a capacidade de um modelo distinguir sequências viáveis de não viáveis com uma alta fidelidade pode ter implicações significativas para o design de vetores virais, reduzindo a necessidade de testes laboratoriais extensos e dispendiosos, guiando a pesquisa de forma mais informada. O fato de os PLMs conseguirem captar padrões complexos dentro das sequências AAV2 reforça a sua utilidade como ferramenta complementar para investigadores na área. Além disso, a abordagem generativa, embora necessite de otimizações, representa uma nova possibilidade para o design de variantes funcionais do AAV2. Melhorar o desempenho do modelo generativo permitirá explorar novas alterações de sequências e abrirá caminho para a descoberta de variantes inovadoras que podem ser imprescindíveis para a evolução da área de investigação de vetores virais.

Quanto a resultados específicos, antes do fine tuning, o modelo de classificação apresentou um desempenho insatisfatório (o que é expectável, face à inexistência de dados sobre AAV2 nos seus dados de treino), evidenciado por valores baixos de F1-score e ROC AUC, em torno de 0.5 para todos os modelos treinados. Esses resultados indicam a incapacidade inicial do modelo de diferenciar sequências viáveis de não viáveis, que pode também ser atribuído às possíveis diferenças subtis entre sequências viáveis e não viáveis. Após o fine tuning com dados específicos do AAV2, todas as métricas de avaliação melhoraram significativamente, com valores de F1-score e ROC AUC variando entre 0.85 e 0.92. Esta melhoria confirma a eficácia do fine-tuning e é corroborada pela visualização t-SNE das representações das sequências antes e depois do fine tuning, que mostraram uma clara separação entre sequências viáveis e não viáveis.

No que se refere ao modelo generativo, a primeira abordagem de avaliação envolveu a geração de sequências e a sua posterior análise pelos modelos de classificação para prever a sua viabilidade. Como nenhuma das sequências geradas foi classificada como viável pelos modelos de classificação elaborados nesta tese, existirá uma insuficiência na capacidade do modelo generativo de produzir sequências que se alinhem aos padrões aprendidos pelos modelos de classificação, os quais demonstraram alto desempenho na tarefa de previsão de viabilidade. Para contornar um possível enviesamento que possa ter ocorrido no treino dos modelos de classificação, realizou-se uma avaliação estrutural utilizando modelos de inverse folding para gerar representações estruturais e analisar a distância entre essas representações. Além disso, foram visualizadas as representações estruturais das sequências geradas e comparadas com subconjuntos de sequências viáveis. A análise revelou que as representações estruturais das sequências geradas estavam, de uma forma geral, mais próximas de sequências projetadas por ML (Machine Learning) do que das sequências não projetadas por ML. Este resultado pode estar relacionado à natureza adjacente dos métodos utilizados para gerar as sequências e aqueles empregados na criação do subconjunto projetado por ML (modelos baseados em transformers vs CNNs, RNNs e LR, todos métodos computacionais baseados em ML). A visualização também evidenciou que as representações estruturais das sequências viáveis e não viáveis dos subconjuntos significativos apresentaram agrupamentos mais coesos dentro dos seus respectivos grupos, enquanto que as sequências geradas demonstraram uma dispersão muito maior. Estas observações corroboram os resultados da avaliação dos modelos de classificação, sugerindo que as sequências geradas não se alinham bem ao espaço das sequências viáveis (e, neste caso, também das não viáveis).

De uma forma resumida os resultados deste trabalho indicam que o ProtBERT pode ser eficazmente adaptado para a classificação de sequências de AAV2 por meio de fine tuning, resultando num modelo que aprende representações diferenciáveis e ricas em informação ligada à viabilidade das sequências do vírus. Isso é evidenciado pelos altos valores das métricas de desempenho nos testes realizados. O modelo de classificação melhorou significativamente sua capacidade de distinguir entre sequências viáveis e não viáveis após o treino específico para o domínio do AAV2, validando a hipótese de que PLMs pré-treinados, quando ajustados com dados específicos, podem alcançar alto desempenho na tarefa de classificação de sequências do AAV2. A elaboração de

um modelo de geração de sequências viáveis de AAV2 apresentou resultados mais complexos. Embora o modelo tenha conseguido gerar sequências novas e diversas, as avaliações estruturais e de viabilidade indicam que melhorias adicionais são necessárias para garantir que as sequências geradas correspondam, de forma consistente, às sequências viáveis conhecidas, tanto em termos de viabilidade como de estrutura.

Consideramos que este trabalho demonstra o potencial do uso de PLMs no design de vetores virais. Embora o método de geração de sequências ainda exija aperfeiçoamentos para que as suas capacidades generativas sejam fiéis às expectativas, o desempenho dos modelos de classificação representa uma evidência significativa do potencial dos PLMs na área. A aplicação desses modelos pode auxiliar investigadores na avaliação de sequências virais. Apesar de apresentarem um risco de imprecisão em torno de 10%, estes constituem uma estratégia de design que permite uma procura informada de novas sequências.

Quanto a trabalho futuro, os resultados sugerem que futuras investigações devem focar-se no melhoramento do processo generativo. Uma abordagem possível seria ajustar a toplogoia mutacional das sequências geradas por meio da modulação da temperatura de geração e dos valores de top-k sampling. Outra estratégia complementar seria o treino do modelo generativo com labels explícitos de viabilidade, permitindo uma aprendizagem mais direcionada dos padrões de geração de sequências viáveis e não viáveis de AAV2. Além disso, uma proposta inovadora seria incorporar o feedback do classificador durante o processo de geração, estabelecendo um sistema integrado em que os modelos generativos e classificadores atuem em conjunto para maximizar a qualidade e a coerência das sequências geradas. Finalmente, pretendemos expandir a abordagem generativa para considerar outras propriedades do AAV2, ampliando assim a aplicabilidade e a utilidade do método.

Este trabalho estabelece uma base para o uso de Modelos de Linguagem de Proteínas na engenharia de sequências de AAV2, oferecendo uma perspeciva promissora para o uso dessas ferramentas no design de vetores virais.

Palavras-chave: Aprendizagem Profunda, Modelos de Linguagem de Proteínas, Aprendizagem por Transferência, Representações, Investigação de Sequências Proteicas

Abstract

This work explores the potential of Protein Language Models (PLMs) to advance the design of novel Adeno-Associated Virus 2 (AAV2) sequences, while focusing on two primary objectives: sequence classification and generative design. For the classification task, we fine-tuned a PLM (ProtBERT) to accurately differentiate between viable and non-viable AAV2 sequences. Results demonstrated high classification performance across multiple trained models, validating the hypothesis that domain-specific fine-tuning enables PLMs to effectively capture important AAV2 sequence features. For sequence generation, we fine-tuned a conditional generative PLM (ProGen) to design viable AAV2 capsid protein sequences. While the model generated structurally diverse sequences, extensive evaluations indicated that additional refinements are necessary to consistently align with viability criteria. The classification model highlights the potential of PLMs in predicting sequence viability, offering a reliable approach that could help reduce experimental costs. We consider that the generative approach, though requiring further optimization, introduces a novel avenue for designing diverse AAV2 variants. Future efforts will focus on refining the generative framework by incorporating explicit viability tags, classifier feedback, and more extensive generation hyperparameter testing, as well as expanding its application to additional AAV2 properties. This work lays a foundation for leveraging PLMs in AAV2 sequence engineering, offering promising prospects for the use of Language Models for viral vector design.

Keywords: Deep Learning, Protein Language Models, Transfer Learning, Embeddings, Protein Sequence Research

Contents

List of Figures

List of Tables

Acronyms

AAV Adeno-associated virus. 2–4, 12, 23–25, 29, 30, 32, 33, 42

BERT Bidirectional Encoder Representations from Transformers. 6–9

CNN Convolutional Neural Network. 12, 17

CTRL Conditional Transformer Language Model. 9

LM Language Model. xi, 1, 5–7, 9, 22

LR Logistic Regression. 12, 17

ML Machine Learning. 1–3, 7, 21, 22

MLM Masked Language Modelling. 8, 11, 32

MSA Multiple Sequence Alignment. 12, 36, 37

NCBI National Center for Biotechnology Information. 29

NLP Natural Language Processing. 1, 24

NSP Next Sentence Prediction. 7

OHE One Hot Encoding. 12, 24

PDB Protein Data Bank. 25

PLM Protein Language Model. 1–3, 8, 11, 29

RNN Recurrent Neural Network. 12, 17

t-SNE t-Distributed Stochastic Neighbor Embedding. 25, 38, 39, 42

VAE Variational AutoEncoder. 12

Chapter 1

Introduction

1.1 Motivation

Machine Learning (ML) has proven extremely valuable in tackling problems requiring a high level of abstraction, from detecting complex patterns to extracting direct knowledge from data elusive to the human brain. In addition, ML uses machine processing power to solve computationally intensive problems at unprecedented rates.

The applications of ML in biomedical research are numerous [21]. In particular, Language Models (LMs) [52] have revolutionized ML applied to gene and protein sequences. The significance of applying LMs to the study of genes and proteins lies in their ability to model complex biological data that conventional techniques may struggle to handle. Protein sequences are not merely linear chains of amino acids but contain patterns and structures that determine their function. Even slight variations in these sequences can have profound implications for biological activity [46]. LMs, adapted for gene or protein data, leverage large-scale training datasets to understand and predict these intricate relationships. Namely, these models allow the representation of sequences based on the individual position and presence of each element as well as the presence and position of that element regarding all other elements in the sentence. This, in turn, can enable a more contextualized representation of the sentence in question, capturing complex dependencies and semantic nuances within biological sequences, acting as essential blocks of information in a wide variety of biocomputational tasks [30].

1.1.1 Natural Language vs Protein Language

LMs were originally developed in the context of Natural Language Processing (NLP) [52], yet the creation of LMs specifically tailored for protein sequences - Protein Language Models (PLMs) soon followed. This is in part because protein sequences can be represented as strings of amino acids and contain modular components in their sequences that can be hierarchically rearranged and assembled, similar to words, phrases and sentences in human natural language. While there is a good analogy to be made between protein sequences and natural language, there are, however, some aspects of natural language that are not present in the context of protein sequences, such as the clear separations between sentences using punctuation.

Traditionally, LMs break down sentences into smaller and more manageable units, known as tokens. In regular LMs specialized in NLP, these normally consist of either characters, words or subwords [54, 56], while in PLMs this tokenization is frequently done at an amino acid level (although some motif and structure-level tokenization strategies have also been developed [27]). In addition, protein language exhibits particular behaviors, such as the possibility of overlapping functional units, which have no analog in human natural language.

Nonetheless, like human natural language, the fact that PLMs can account for context is essential for protein sequence research, since both the presence and position of each element (amino acid or amino acid motifs) in the sequence is relevant, as is each the position of each element relative to other elements of the sequence [38].

1.1.2 ML in Viral Vector Design

An important and relatively recent application of ML to biological sequences is the design of viral vectors. Viral vectors are a key tool in gene therapy, a clinical approach that consists of transferring genetic material into patient cells to treat or prevent disease. Depending on the condition, gene therapy may seek to correct dysfunctional genes, inhibit their expression or add functional copies of malfunctioning genes. The vehicle used for gene transfer is generally called a vector and vectors derived from viruses (viral vectors) are currently the most used due to their high efficiency of gene delivery and ability to define tissue-targeting approaches. Among viral vectors, those derived from adeno-associated viruses (AAVs) are currently the most used for in vivo gene transfer protocol [4]. The design and bioengineering of viral vectors, i.e., generating viral vectors with specific desired characteristics or properties, is an area of high interest and active research. However, this has been mainly based on trial-and-error approaches or cumbersome evolution-based protocols, which lack efficiency and result in substantial costs. Commonly desirable characteristics for these vectors include increased variability, reduced toxicity or immunogenicity of the vectors, and improved manufacturability [49]. Regardless of the desired properties, the viability of these vectors is a basic functional requirement. Viability is defined here as the ability of an amino acid sequence to enable the protein(s) it encodes to assemble and sustain the core function of delivering genetic material into target cells. In the scope of this project, viability is the only desirable function to target.

AAVs (specifically AAV2) were the first focus for applying ML in vector design and bioengineering [39, 11, 34, 35, 45, 23]. More recently, adenoviral vectors (AdVs) have also been targeted [31]. AAV vectors were particularly compelling as a starting point for several reasons. Firstly, they are widely used in gene therapy, making advancements in their design highly relevant. Secondly, there is significant potential for their improvement in areas such as diversification, tissue-targeting capabilities, and resistance to immune system elimination. Lastly, and most crucially, AAVs consist of a single type of protein (the capsid), unlike most viral particles. This simplifies the application of ML, making the transition from single protein design to viral vector design more straightforward. Most works of ML-based AAV design have represented protein sequences as en-

codings [11, 34, 35, 39, 45]. These encodings are unique numerical values or arrays attributed to each amino acid. These representations fail to consider the protein sequence as a whole, only considering the presence of each amino acid, which may reduce the predictive abilities of the model. In addition to this sequence representation limitation, most works of ML-based vector design done so far focused on sequence classification using classical ML models, with generative approaches being clearly underexplored [39, 45, 31].

The current landscape of ML-guided vector design presents a remarkable opportunity to explore transformer-based PLMs for viral vector design and bioengineering, since there is value to be gained in exploring the potentialities of PLMs both to improve the classification of AAV capsid protein sequences as well as to generate new AAV capsid sequences.

1.2 Objectives and Research Questions

This thesis aims to evaluate the potential of PLMs in enhancing the design and classification of AAV2 sequences, focusing on two main objectives: improving the accuracy of sequence viability classification and advancing the generation of novel, functional AAV2 variants.

O1: Explore PLMs to improve AAV2 viability classification. The first objective is based on the hypothesis that the use of PLM representations can yield better classification results due to their self-attention-based capabilities. To accomplish this objective, we propose to fine-tune a PLM model on AAV2 data for AAV2 sequence classification.

RQ1: Can AAV2 viability be used to fine-tune a general-purpose PLM into a high-performance classifier? This research question arises from O1 and is based on the hypothesis that AAV2 sequence properties can be accurately predicted using a fine-tuned PLM model, which remains to be validated in the context of AAV2 sequence viability. Namely, we wish to assess whether high-performance classification models can be developed for classification of sequences obtained from different methodologies (such as ML-designed sequences).

O2: Use PLMs to generate viable AAV2 vectors. To achieve the second objective, we propose a novel approach for conditional fine-tuning of a PLM that considers protein function and taxonomy to guide the generation of AAV2 protein sequences. Our hypothesis is that this type of conditional generation, along with the fact the PLM attention-based interpretations line up particularly well with protein sequence syntax, should be able to generate high-quality sequences, comparatively to the traditional protein design methods and representations.

RQ2: Can viable AAV2 protein sequences be designed by a fine-tuned autoregressive conditional PLM? Building on O2 and recognizing the innovative potential of this approach, this research question seeks to evaluate whether the parametric knowledge present in a conditional generative pre-trained PLM can effectively aid in generating viable sequences, specifically for the AAV2 capsid, also through fine-tuning of a pre-trained model, particularly considering that AAV2 data is not included in the model's pre-training scope.

1.3 Contributions

The primary goal is to develop a system capable of addressing two key challenges in AAV2 engineering: the classification of sequence properties (specifically on viability), and the generation of novel, viable AAV sequences. We aim to provide the following contributions:

- The development of a sequence classification model for predicting AAV2 viability.
- The development of a sequence generation model to design novel, viable AAV2 variants.
- The introduction of a scalable workflow for AAV2 sequence design that can incorporate new datasets and reduce laboratory costs, guiding development based on previously unreachable contextual information.

1.4 Funding

This work was supported by the LASIGE Research Unit, ref. UIDB/00408/2020 (https://doi.org/10.54499/UIDB/00408/2020) and ref. UIDP/00408/2020 (https://doi.org/10.54499/UIDP/00408/2020). It was also partially supported by the KATY project which has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 10101745.

Chapter 2

Background

2.1 Language Models

LMs are deep learning algorithms most famously based on the transformer architecture [52]. These models are trained to understand and generate sequences by predicting the next element in a sequence or filling in masked elements, allowing them to capture patterns and dependencies within data. The core aspect of language models is the ability to create rich, context-sensitive representations of input data and/or use those meaningful representations to perform context-sensitive tasks. The generation of these representations is achieved through mechanisms such as self-attention, which enables models to account for the importance of different parts of a sequence. The use of the transformer architecture introduced by Vaswani et al. [52] has become state-of-art for a large number of modern LMs by allowing parallelization and capture of long-range dependencies, which are not considered by the earlier Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN) approaches [37]. Based on the modules of the transformer architecture used in their own architecture (and consequently the task the models are trained for), LMs can fall into three different categories: encoder models (for representation learning), decoder models (for sequence generation) and encoder-decoder hybrids. Encoder models specialize in creating high-dimensional context-rich representations of the input, making them specially suited for data classification or tasks that benefit from context-rich representations in general. Decoder models are designed for autoregressive generation tasks. These models predict the next token in a sequence based on previously generated tokens, making them well-suited for tasks like text generation, summarization, or machine translation. Encoder-decoder models combine both architectures, enabling sequence-to-sequence tasks such as translation, summarization, and sequence transformation. These models encode input data into dense representations from the encoder module and decode them into new (complete) sequences.

2.2 The Self-Attention Mechanism

Given an input sequence of tokens, the self-attention modules present in transformer modules convert each token into three individual vectors: a query (Q), a key (K), and a value (V). These

vectors are generated by multiplying the input embedding X (obtained as output from the previous preprocessing layers in the transformer model) by learned weight matrices W^Q, W^K, and W^V.

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

The model then uses these vectors to calculate attention scores. The model calculates an attention score for every pairing of tokens in the input. The attention score between any two tokens is computed by taking the dot product of the query and key vectors (and also scaling it by d_k (the dimensionality of the vectors)).

$$A = \frac{QK^T}{\sqrt{d_k}}$$

The attention scores are then normalized using a softmax function.

$$\alpha_{ij} = \text{softmax}(A_{ij}) = \frac{\exp(A_{ij})}{\sum_k \exp(A_{ik})}$$

Using the normalized attention scores, the model then calculates a weighted sum of the value vectors.

$$O_i = \sum_j \alpha_{ij} V_j$$

To allow this process to be more efficient (through operation parallelization), the computation of the attention is distributed between a set of heads in a Multi-head Self-Attention mechanism, where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ and W^O is the output projection matrix (the learned weight matrix built from each individual head's calculations).

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

2.3 LM pre-training and fine-tuning

Fine-tuning consists of a transfer learning approach in which parameters of a pre-trained model (usually pre-trained on large general datasets) are leveraged to adapt the model to a downstream task (typically a task adjacent to the data and task the model was pre-trained on). This procedure allows models to achieve high performances without training an untrained model and with less data (regarding the downstream task).

2.4 Encoder LMs and LM embeddings

LMs based on the encoder portion of the transformer architecture make use of a multi-head self-attention mechanism to generate context-rich representations of the input, enabling this type of model to represent both the importance of the individual elements of the input and the effect of each of those elements on the input as a whole. This is the case with the BERT (Bidirectional

Encoder Representations from Transformers) models [15], a family of encoder LMs capable of generating contextual representations of varied textual inputs. These representations are a type of embedding.

Embeddings are high-dimensional vectorial representations learned by the LM throughout its training process and at a more abstract level, embeddings are mappings between inputs and locations in a continuous vector space. Embeddings generally represent concepts in a format processable by neural networks or other similar ML frameworks. Embeddings take into account the syntactic and semantic relationships between the elements they are representing, in this way capturing nuanced relationships present in the data.

As an example, below we explain the generation of embeddings by the general BERT architecture:

1. Tokenization

The first step involves preprocessing the text sequence to ensure compatibility with the neural network input layer. This step divides the input into smaller fragments and replaces them with tokens, which are a fixed numerical and unique representation of those fragments.

2. Generation of "component" embeddings

After tokenization, BERT's embedding layer processes the tokens to produce three distinct "component" embeddings: a token embedding, a segment embedding and a position embedding. The parameters for generating these embeddings are learned during its training, allowing the mapping of each input to a point in a vector space.

- Token embedding
  This embedding represents the learned vectorial representations of each token present in the input, which is computed by one or more Token Layers (TLs).

- Segment Embedding
  Since BERT was pre-trained on the Next Sentence Prediction (NSP) task (in which the model must determine whether any two given sentences are sequential), this embedding allows the model to differentiate between sequences appearing first and second. Segment Layers (SLs) are responsible for generating this embedding.

- Position Embedding
  This embedding contains the learned vectorial representation of each position of the input. This embedding is crucial to account for the semantic relevance of every position within the input. It allows the model to distinguish sentences with the same tokens in different positions and assign appropriate weights based on their semantic relevance and considering their position in the sequence. The generation of this embedding is done by Positional Layers (PLs).

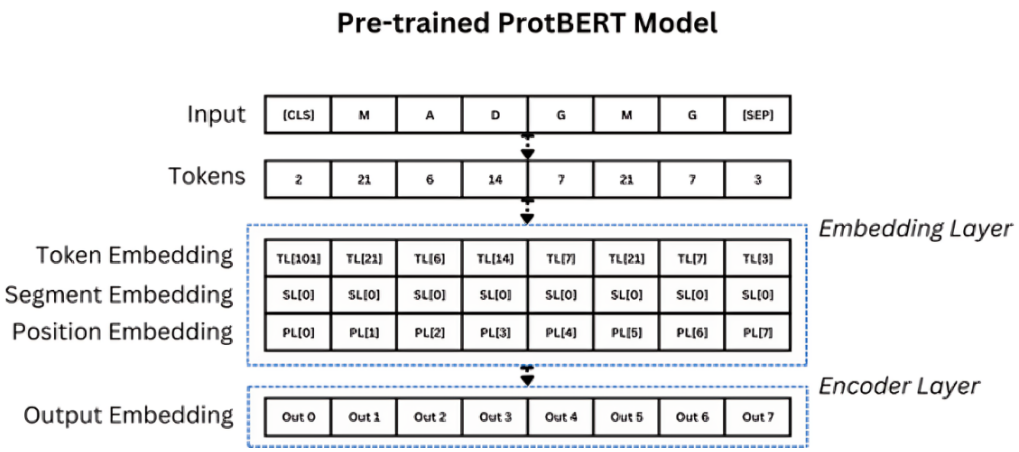


Figure 2.1: BERT embedding generation

3. Generation of the output embedding

The model combines the three ”component” embeddings to generate an output embedding, by performing an element-wise sum of the three. This output embedding accounts for the identity, type and position of every token within the sequence and the relative position of every other token in regard to it. This output is then processed by the remaining dense layers of the model, generating a final output embedding (Figure 4.1).

ProtBERT

One of the PLM models used in this work is the ProtBERT model. ProtBERT is an autoencoder PLM, specifically designed for protein sequence data. It was first proposed by Elnaggar et al. (2021) [18] and was trained in the task of Masked Language Modelling (MLM) of sequences of uppercase amino acids using millions of sequences from UniRef100 [12]). MLM consists in the task of randomly replacing tokens in sequences with a special ’[MASK]’ token and having the ML model predict the original masked tokens based on the surrounding context. Since MLM is a self-supervised task, no data labeling was required, and the training alone enabled the model to learn the data features without human intervention. Similarly to the BERT model, ProtBERT outputs are embeddings representative of the inputs passed, in the context of the task it was trained in. Therefore, the resulting embedding of passing a protein sequence through ProtBERT is the vectorial representation of that protein sequence.

After training, ProtBERT effectively captured essential biophysical amino acid features. ProtBERT was also able to successfully differentiate between protein sequences belonging to different domains of life, including viral proteins.

The attention mechanism, a core component of LMs, is particularly well-suited for capturing essential biological properties of protein sequences, namely existing relationships between amino acids close to each other within a sequence and between amino acids far away from each other sequence-wise but spatially close in the protein three-dimensional structure, as discussed in Vig

et al (2020) [53]. This study was conducted on three different LM architectures (including BERT [15]) and two different protein datasets [17, 47] providing consolidated evidence of the attention mechanism's ability to model complex protein interactions.

2.5 Decoder LMs and autoregressive generation

Decoder LMs use autoregression to iteratively generate tokens, one token at a time. They consist of an example of text generation based on probabilistic modeling, as they internally output probabilistic distributions of the tokens in their vocabularies and sample from those distributions based on a set of criteria. Given an input sequence $X = x_1, ..., x_t$, the model predicts the next token x_{t+1} by conditioning the generation based on the tokens that exist at that step. The mathematical formula of the generation is then:

$$P(x_{t+1}|x_1, x_2, \ldots, x_t) = \text{softmax}(W^O O_t)$$

Decoder LMs are trained with masked (obscured) sequences starting from a given position (future token masking), ensuring the model is trained to predict a number of tokens immediately succeeding the last token in the input, at a given step.
It is important to mention that one of the main limitations of autoregressive models is their innate tendency for error propagation: as more tokens are generated, the inadequacy of a token for a specific position will probabilistically negatively impact the rest of the output, as the problematic token will be used as context for their generation (the same will be true for the newly generated tokens, propagating and accumulating the error).

2.6 Conditional Transformer Language Models

The generative approach described in this thesis is based on fine-tuning a generative model which is a variation of the Conditional Transformer Language Model (CTRL) architecture. The CTRL architecture was first introduced by Keskar et al. (2019) [28]. As mentioned before, given a sentence $x = (x_1, ..., x_n)$, where x is a sentence and $x_1 ... x_n$ are the elements within that sentence, the goal of language modeling is to predict $p(x)$ i.e. the probability distribution of the occurrence of x. Through the application of the chain rule of probability, this task is decomposed into the task of next-token prediction [40]:

$$p(x) = \prod_{i=1}^{n} p(x_i|x_{<i})$$

Currently, state-of-the-art LMs use neural networks, each with a set of parameters θ to minimize the negative log-likelihood of each element in a sequence, given the sequence thus far and the training data D it was trained on. With this negative log-likelihood, a model loss function can be defined as:

$$L(D) = -\sum_{k=1}^{|D|} log p_\theta(x_i^k | x_{<i}^k)$$

The goal of the CTRL model architecture is to enhance the ability to control specific characteristics (such as style, content and task-specific details) of sequences generated by these language models. This is achieved by conditioning the generated sequences using the proposed CTRL control codes or CTRL IDs - a set of numerical values uniquely mapped to all tokens in the input vocabulary (including style, content or task-specific identifiers), consisting in the main interpretable unit by the model.

Assuming c (a given CTRL ID), the CTRL model will learn the distribution $p_\theta(x|c)$, where x is the sentence constrained by the condition that c represents. As such, through decomposition using the chain law of probability:

$$p(x|c) = \prod_{i=1}^{n} p(x_i | x_{<i}, c)$$

At a stepwise level, considering the task to be the prediction of the next element of the sentence, the CTRL model will learn the probability distribution of $p_\theta(x_i|x_{<i}, c)$. Now considering the previously mentioned negative log-likelihood calculation, we have that:

$$L(D) = -\sum_{k=1}^{|D|} log p_\theta(x_i^k | x_{<i}^k, c^k)$$

Regarding the CTRL model's handling of the input data, two different cases are considered:

- During training, each sequence in the training data and its reverse is passed to the model as input (in the form of CTRL IDs). Each input is converted by the model into a set of n embeddings in vector space $\mathbb{R}^d$, where n is the length of that input and d is the model's dimensions. Each of these embeddings is calculated through a sum of a learned token embedding and a learned sinusoidal positional embedding, as proposed in Vaswani et al. (2017) [52]. The set of embeddings for each sentence in the training data is then stacked in a matrix and passed through the model's attention layers, outputting a score for each possible candidate element.

- For generation, in each generation step, the CTRL model generates an embedding of size equal to the number of possible characters. The values in this array correspond to the selection probabilities for each character to be the next character generated. The model then selects one of the candidate characters through top-k sampling.

It was demonstrated by Keskar et al. (2019) [28] that using CTRL IDs paired with task-specific data the model was able to perform well in tasks such as question answering and machine translation without compromising its generalization ability.

2.7 The ProGen model

The CTRL generative model used in this work is ProGen, introduced by Madani et al. (2023) [32]. ProGen is a 1.2 billion-parameter sized language model based on the previously described CTRL architecture and pre-trained on a dataset of 280 million protein sequences of different families from Uniparc (Leinonen et al. (2004) [29]), UniprotKB (Bairoch et al. (2005) [5]), SWISS-PROT (Bairoch et al. (2004) [6]), TrEMBL (Boeckmann et al. (2003) [10]) and Pfam (Bate- man et al. (2004) [7]). It was pre-trained trained on the MLM task to generate all 25 amino acids designated in IUPAC [42].

ProGen is a generative PLM designed to produce prompt-sensitive protein sequences, therefore capable of generating protein sequences with a set of desired characteristics on input (Figure 2.2). This capability is achieved by leveraging additional information, including taxonomical and functional data that was available for each protein sequence during model pre-training. This information is passed in the form of unique numerical tags for taxonomical and keyword attributes. The taxonomical attributes (or the taxonomical lineage) determine the taxonomical sets the organism in question belongs to and correspond to unique NCBI taxonomy IDs. This allows the model to differentiate between proteins from different organisms, while still maintaining cohesion between proteins of organisms of the similar taxonomical families. On the other hand, the functional attributes (or keyword attributes) correspond to unique UniProt functional IDs and determine the functions pertaining to that protein, allowing the model to discern between protein functions based on their keyword attributes, which should be closely tied to protein sequence, and therefore are accounted for as input. As for the keyword attributes, their presence is justified by the fact that protein function is closely tied to the existence of amino acid motifs within specific zones of the sequence, therefore, by passing keyword tags in the input the model will be biased to produce a sequence that contains the amino acid motifs responsible for the function those tags are encoding. Overall, by incorporating all this additional information, the model learns to generate outputs that adhere to the constraints it has learned during the training process. At the time of its publication, ProGen reached unprecedented high-quality level model perplexity (a performance metric used to measure a probabilistic model's uncertainty when making predictions), and was also able to generalize well to sequences from randomly sampled unseen protein families.

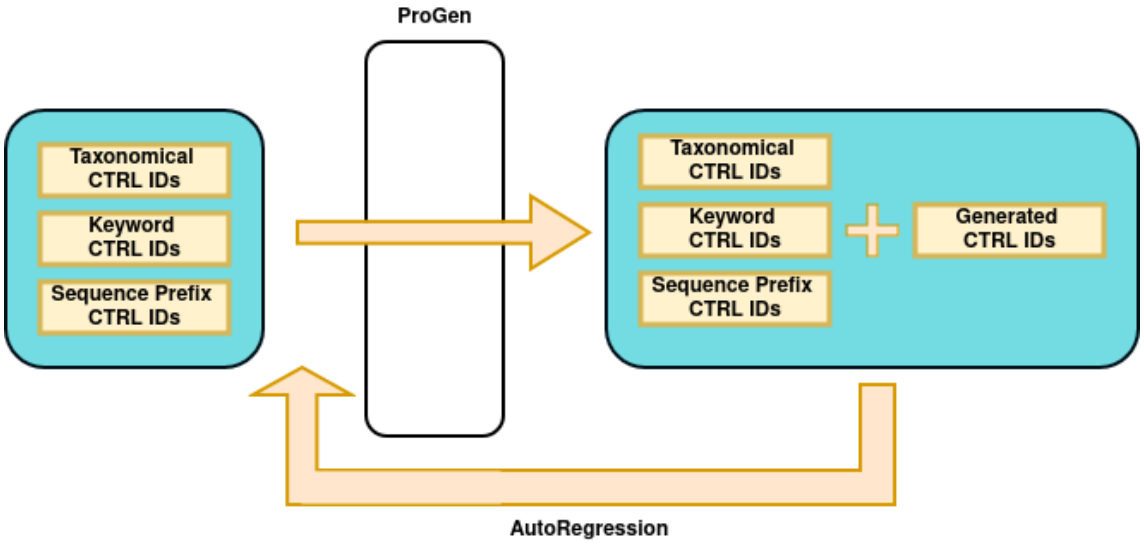


Figure 2.2: ProGen's forward pass, autoregressive loop and outputs

2.8 Related Work on ML-based viral vector design

As a reference point, in 2019, Ogden et al. [39] utilized a Position-Specific Scoring Matrix (PSSM) sampling to construct an AAV fitness landscape, predicting multiple different sequence properties, using One-Hot Encoded (OHE) protein representations. Although this method does not employ generative ML techniques, it provided an early example of algorithm-driven AAV design. By 2021, Marques et al. [34] employed Artificial Neural Networks (ANNs) and Support Vector Machines (SVMs) to classify AAV2 viability by detecting amino acid properties essential for capsid assembly. This study used both one-hot encoding and amino acid properties for protein representation. In the same year, Bryant et al. [11] explored sequence diversification, thermal stability, and hepatotropism using Logistic Regression (LR), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs). Additionally, Mikos et al. (2021) [35] utilized RF models for AAV2 sequence classification, relying on amino acid microenvironment encodings. This work exemplifies the use of structure-based encodings in AAV2 sequence classification. Regarding generative approaches, Sinai et al. [45] employed a Variational Autoencoder (VAE) trained on Multiple-Sequence Alignments (MSAs) of dependo-parvoviruses — the *genus* AAV2 belongs to — to generate novel AAV2 capsid sequences. This marked one of the first successful applications of generative models for AAV capsid design. In 2023, Han et al. [23] used embedding matrices coupled with CNNs, integrating transformer-based models to classify AAV2 vectors on transduction efficiency. Lastly, in 2024, Lyu et al. [31] combined a VAE with a PLM as the encoder, generating capsid sequences with increased functional stability and reduced immunogenicity. This approach represents the first major integration of PLM representations in viral vector design. The overall differences in protein representations, models used and properties addressed in all of the mentioned works are presented in Table 2.1.

This thesis wishes to investigate the performance of the clearly underutilized PLM-based ap-

proaches in the design of novel viable AAV2 sequences. PLMs afford some features that may prove valuable for viable sequence generation: first, their potential to generate sequences with high contextual coherence may prove better suited to preserve patterns that are crucial for protein function and structure compared to other approaches; second, they are better at modelling long-range dependencies which improves their ability to understand and generate sequences where distant parts of the sequence are biologically interdependent; finally, PLMs can generate more diverse sequences, while still maintaining their plausibility, due to the flexibility of sampling strategies, while other methods, such as VAEs produce more constrained outputs due to the limitations of the latent space.

Reference	Property	Protein Representation	Computational Models	Task
Ogden et al. (2019) [39]	Viability Thermal stability Neutralization resistance Several tissues tropism	One hot encoding	PSSM-based sampling	Experimental fitness landscape Sequence generation
Bryant et al. (2021) [11]	Viability Sequence diversification Thermal stability Hepatotropism	One hot encoding	Logistic regression CNN RNN	Sequence classification
Marques et al. (2021) [34]	Viability	One hot encoding Amino acid properties encoding	ANN SVM	Sequence classification
Mikos et al. (2021) [35]	Manufacturability	Amino acid microenvironment encoding	Random forest	Sequence classification
Sinai et al. (2021) [45]	Viability Sequence diversification	One hot encoding	Variational autoencoder	Sequence generation
Han et al. (2023) [23]	Transduction efficiency	Embedding matrix	CNN Transformer-based model	Sequence classification
Lyu et al. (2024) [31]	Sequence diversification	Embedding matrix	PLM Variational autoencoder	Sequence generation

Table 2.1: Overview of related work regarding AAV (and adjacent AdV [31]) vector design aligned with ML-guided strategies

Chapter 3

Methodology

3.1 Overview

The body of thesis comprises two main distinct yet interconnected chapters: Chapter 4 focuses on the development of the classification model, while Chapter 5 is dedicated to the development of the generative model. Both tasks involve data from a single dataset, which is further explored in the next subsection of this chapter. The goal of this chapter is to provide a clear overview of the work conducted in this thesis and the respective experimental design behind it.

The overall scope of this work is presented in Figure 3.1. The development of the classifier models was conducted by fine-tuning the pre-trained ProtBERT models with a classification head on non-ML-designed sequences from the Bryant et al. (2021) [11] dataset. The models were then evaluated using ML-designed sequences from that same dataset, yielding a set of metrics on classification performance. For the development of the generative model, we used viable sequences from the dataset to further train a pre-trained ProGen model. After training, we evaluated the model by having it generate 100 new sequences (with varying sizes reflecting the size distribution of the data) and evaluating them using two different approaches: by using the previously mentioned trained classifier models to classify those sequences and by using AlphaFold to predict the structure of the sequences, obtaining structural embeddings using an inverse folding model and analyzing the similarity between those embeddings and structural embeddings from sequences in the dataset. Additionally, we used data from [43] for two additional evaluations: an evaluation using AAV2-trained non-transformer-based classification models and an evaluation through analysis of the generated sequence mutational topology (both of which are not represented in Figure 3.1). Even though both pre-trained models (ProtBERT and ProGen) presented a good generalization performance on unseen proteins, they were not trained on viral protein data and since viral proteins have unique structural and functional properties and may have distinct sequence patterns or structural constraints that are underrepresented in the pre-training data of the models. By performing fine-tuning on these models we expect to be able to adapt the models to these viral domain-specific characteristics.

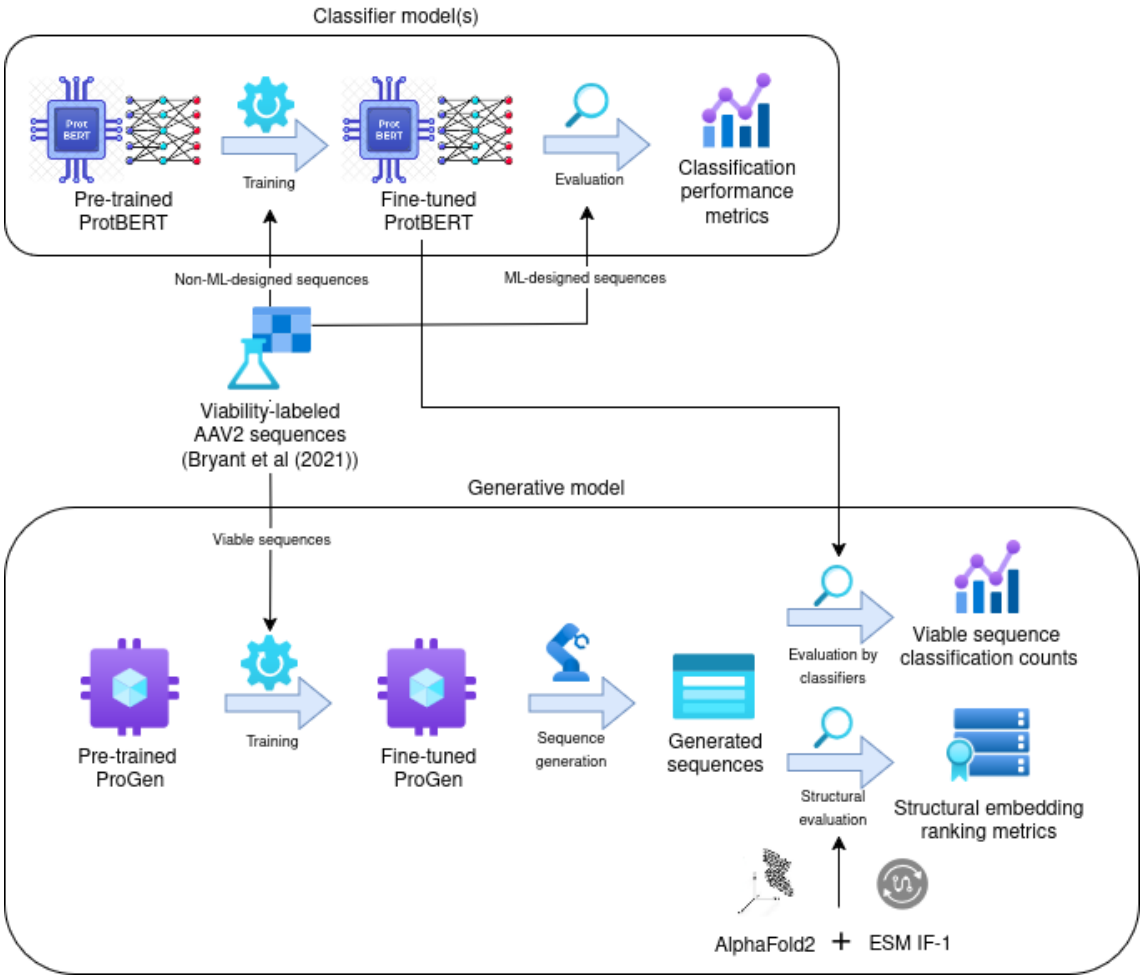


Figure 3.1: Overall thesis model training / evaluation overview

3.2 Data

In this thesis a high-quality dataset featuring AAV2 sequences and their viability was used, both for the classification and generative models, provided by Bryant et al. (2021) [11]. Bryant et al. (2021) [11] tested the capability of ML methods coupled with high-throughput DNA-synthesis and sequencing technologies to design functional diverse variants of an AAV2 wild-type sequence. Diversity is of the essence when it comes to the design of AAV sequences due to one of the limitations of the use of AAVs in gene therapy being reduced treatment efficiency after prior exposure to that same AAV. The authors conducted this work based on the hypothesis that data obtained from high-throughput experiments could be useful to train ML models on, in order to allow them to generate viable protein sequences substantially different from their natural counterparts. The resulting dataset consists in the most comprehensive effort to date in AAV2 capsid diversification. The sequences present in these dataset are each either positively or negatively labeled, in regard to their viability. All of these sequences were produced in laboratory and their viability tested. The dataset is comprised of 296970 variants of a wild-type AAV2 sequence, generated through substitutions, deletions and insertions. Different data subsets are present in this dataset, since the mutations were applied using multiple different methodologies, such as stochastic sampling, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and Logistic Regression (LR). A description of these subsets and the respective mutation methodologies used is given in Table 3.2. This data was remarkably useful for this thesis since it consists of sets of AAV sequence labelled data with multiple characteristics that we were able to use to specifically train and test models on, allowing us to evaluate our models in sets of data with varying natures.

Table 3.1: Size and description of each data subset from Bryant et al. (2021)

Subset name	**Size**	**Description**
'wild_type'	1	Set containing the single base (unchanged) sequence
'stop'	57	Set containing the sequences regarding stop codon mutations
'single'	1112	Set of sequences obtained through single site mutation of the wild-type sequence.
'singles'	1085	Set mostly equivalent to the 'single' subset, however missing 27 sequences (the nature of the missing sequences is not explicit in the paper)
'random_doubles'	25040	Set of sequences obtained from mutations on 2 sites
'rand'	9885	The set of sequences obtained from mutating the wild-type sequence from 2 up to 10 times
'designed'	56372	Set of sequences obtained from stochastically sampling from additive models
'cnn_standard_seed'	1924	Set of sequences selected through a CNN from 'single' and 'rand'
'cnn_standard_walked'	20395	Set of sequences generated through small mutations of the sequences from the above subset
'cnn_rand_doubles_plus_single_seed'	2022	Set of sequences selected through a CNN from 'single' and 'doubles'
'cnn_rand_doubles_plus_single_walked'	20454	Set of sequences generated through small mutations of the sequences from the above subset
'cnn_designed_plus_rand_train_seed'	1898	Set of sequences selected through a CNN from 'rand' and 'designed'
'cnn_designed_plus_rand_train_walked'	20759	Set of sequences generated through small mutations of the sequences from the above subset
'rnn_standard_seed'	1916	Set of sequences selected through a RNN from 'single' and 'rand'
'rnn_standard_walked'	20838	Set of sequences generated through small mutations of the sequences from the above subset
'rnn_rand_doubles_plus_singles_seed'	2045	Set of sequences selected through a RNN from 'single' and 'doubles'
'rnn_rand_doubles_plus_singles_walked'	20154	Set of sequences generated through small mutations of the sequences from the above subset
'rnn_designed_plus_rand_train_seed'	2065	Set of sequences selected through a RNN from 'rand' and 'designed'

Continued on next page

Subset name	Size	Description
'rnn_designed_plus_rand_train_walked'	20731	Set of sequences generated through small mutations of the sequences from the above subset
'lr_standard_seed'	1989	Set of sequences selected through LR from 'single' and 'rand'
'lr_standard_walked'	20456	Set of sequences generated through small mutations of the sequences from the above subset
'lr_rand_doubles_plus_single_seed'	2071	Set of sequences selected through LR from 'single' and 'doubles'
'lr_rand_doubles_plus_single_walked'	19999	Set of sequences generated through small mutations of the sequences from the above subset
'lr_designed_plus_rand_train_seed'	2030	Set of sequences selected through LR from 'rand' and 'designed'
'lr_designed_plus_rand_train_walked'	19680	Set of sequences generated through small mutations of the sequences from the above subset
'previous_chip_viable'	994	Set of sequences considered as viable, obtained with the same methodology as the ones from the "designed" subset, generated in a previous experiment by the authors of this paper.
'previous_chip_nonviable'	998	Set of sequences considered as nonviable, obtained with the same methodology as the ones from the "designed" subset, generated in a previous experiment by the authors of this paper.

Chapter 4

PLM-based classification of AAV2 capsid viability

4.1 Context

The first part of this work focused on investigating the potential of PLMs to enhance viability classification, particularly by enabling the model to undergo fine-tuning and specialize in specific protein types, herein, variants of the AAV2 capsid. The reasoning behind the use of the ProtBERT model (a PLM encoder) is that ProtBERT's AAV2 sequence representations should include biologically relevant features that influence the viability of AAV2 capsid variants, allowing the model to represent implicit knowledge of sequence patterns, structural motifs, and functional constraints (in this case viability-based functional constraints) in a rich dense vector representation that can then be classified in accordance. This chapter describes the process of developing the ProtBERT classification models, starting with an overview of the data preprocessing, followed by a description of the fine-tuning methodology and concluding with the model evaluation and a brief discussion of the results.

4.2 Dataset and data preprocessing

The dataset used in this part of the work was published by Bryant et al. (2021) [11] and is available as supplementary material in [1]. The dataset is separated into different subsets, consisting of sequences designed using different methodologies. The nature of this data, as well as the size of each subset is presented in Table 3.2. Briefly, we further separated the dataset into two distinct sets: ML-designed sequences and non-ML-designed sequences (Table 3.2). The first consists of the set of sequences that were designed by the ML models developed by Bryant et al. (2021) [11], while the latter are sequences that were either algorithmically designed (through additive sampling from known viable mutations) or by random mutations, both synthesized and evaluated in laboratory to serve as training set for the ML models in the same study. In this work, we used the data from these datasets to assess the classification model's training and performance on data obtained from targeted mutation methods (additive sampling or random mutation) or data sourced from ML models, respectively. For preprocessing, the sequences were modified to meet

Table 4.1: Bryant et al (2021) dataset viability class distribution

Dataset	No. sequences	Viable sequences (%)
Non-ML-designed	95486	56.78
ML-designed	199437	51.73

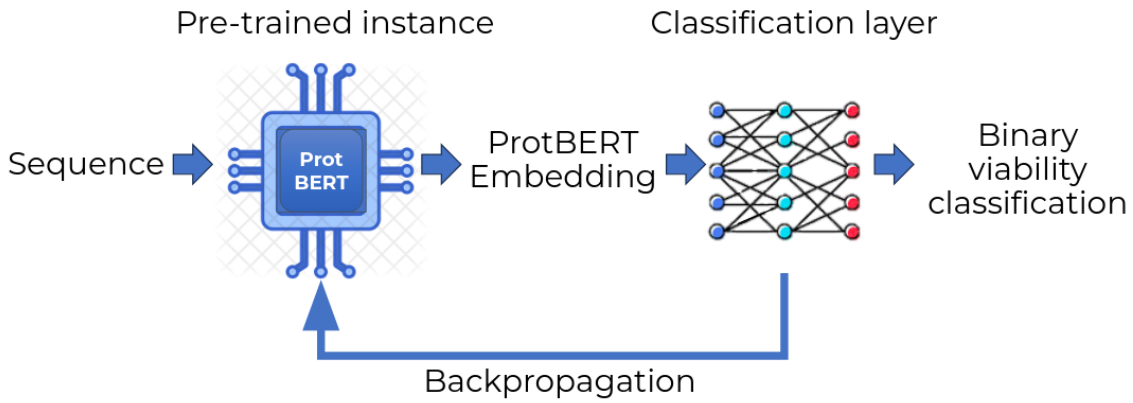


Figure 4.1: Overall architecture of the classifier model

ProtBERT model format requirements [18], namely, rare amino acid symbols were replaced with "X", all amino acid symbols were uppercased and every amino acid symbol in the sequences was intercalated with empty spaces.

4.3 Methodology

To achieve our task, we built an architecture that consisted of a pre-trained encoder model (ProtBERT) that converts each protein sequence into its embedding representation and a neural network layer tailored for binary classification (Figure 4.1).

The pre-trained ProtBERT component, composed by a BERT tokenizer and a pre-trained ProtBERT model, was sourced from the HuggingFace library [2], an open source library known for its extensive collection of LMs and tokenizers. The classification layer was implemented using the PyTorch library. Given that the HuggingFace's transformers architecture is also built on PyTorch, compatibility is ensured between the two components. This integrated architecture provided the model with the necessary functionalities, enabling loss backpropagation from the classification layer to the ProtBERT instance, allowing for parameter updates within ProtBERT, thereby refining the model performance.

For the development of each model instance we decided to train the model on 'non-ML-designed' sequences and to test it on 'ML-designed' sequences.

This experimental design should allow us to evaluate how well our models are able to generalize from training data regarding sequences obtained through simple position score-based methodologies ('non-ML-designed', without any ML-related biases) to sequences with a mutational landscape determined by ML mutational approaches ('ML-designed').

4.3.1 Model fine-tuning

At every step of the training of this model, loss was calculated based on the 'pooled output' of the model. The 'pooled output' corresponds to the hidden representation of the '[CLS]' token of each sequence in the batch, pooled into a single embedding (through a linear layer followed by a Tanh activation function). This methodology is traditionally used in BERT-based classification as the 'pooled output' is considered to capture enough context to be used as the semantic representation of the sequence by the model. For each of the cases previously mentioned, stratified K-fold cross-validation was applied to maintain the balance of the class between these subsets in K different fractions (although the subdatasets were already nearly perfectly balanced), with K = 10, a common value for K in K-fold cross-validation. Models were trained using batch sizes from size 1 up to 5. The different batch sizes were selected to verify whether an increase in batch size would cause a significant impact performance-wise, as smaller batch sizes tend to achieve better results at the cost of slower convergence speed. The loss function of every model was Inverse Cosine Similarity (ICS). This loss function was selected due to cosine similarity being a distance metric particularly well-suited for dense continuous vector comparison, which aligns with our task-oriented goal of allowing the ProtBERT instance to learn representations for viable and non-viable AAV sequences as geometrically (and semantically) distant from each other as possible. A gradually decreasing learning rate strategy was also implemented, allowing the model to adjust its parameters in decreasing orders of magnitude, helping to prevent overshooting and enabling faster convergence and increased model stability. The set starting learning rate was 10^{-6}. Every model was subjected to up to 3 training epochs. To evaluate each model instance, accuracy, precision, recall, F1 score and ROC AUC were considered. These metrics were calculated using the 'ML-designed' subset.

4.4 Results

Table 4.2: Performance of the pre-trained ProtBERT model (before fine-tuning)

Test set	Accuracy	Precision	Recall	F1 Score	ROC AUC
ML-designed	0.4827	0.2414	0.5	0.3256	0.5

As previously mentioned, two partitions were considered for the development of the classification models: training on 'non-ML designed' data and testing on 'ML-designed' data. The evaluation results (performance metrics before any fine-tuning) are displayed in Table 4.2, while the evaluation results (after fine-tuning on AAV2 data) are present in Table 4.3. Before fine-tuning (Table 4.2), the low F1 score values indicate the model was unable to consistently predict sequence viability at that point (without any training on AAV2 sequence classification). The value of ROC AUC being 0.5 also indicates the model was unable to differentiate between the classes present in

Table 4.3: Performance comparison of ProtBERT, Random Forest, and Logistic Regression classification models on 'ML-designed' AAV2 sequence viability classification

Model	Batch Size	Representation	Accuracy	Precision	Recall	F1 Score	ROC AUC
ProtBERT*	1	-	0.8801	0.8828	0.8857	0.8800	0.8858
ProtBERT*	2	-	0.8618	0.8689	0.8539	0.8579	0.8539
ProtBERT*	3	-	0.8701	0.8687	0.8685	0.8687	0.8685
ProtBERT*	4	-	0.8836	0.8821	0.8856	0.8830	0.8855
ProtBERT*	5	-	0.9274	**0.9260**	**0.9277**	**0.9268**	0.9278
Random Forest	1	OHE	0.9525	0.6743	0.4287	0.5912	0.8971
Random Forest	1	AA embedding	0.8412	0.7504	0.6728	0.7476	0.8165
Logistic Regression	1	OHE	**0.9727**	0.8740	0.7930	0.8737	**0.9761**
Logistic Regression	1	AA embedding	0.8235	0.7678	0.7350	0.7767	0.8126

* Models developed for this thesis

the data, as expected, due to the large scale of data used to train the model, the model likely learned the features of the sequences of the most prominent protein organisms and families in the dataset, which is not the case for AAV2. This lack of AAV2 representation, coupled with a highly likely lack of easily discriminative features between viable and nonviable AAV2 sequences, should be the main factors behind this difficulty in discerning AAV2 sequences based on viability (without AAV2 training). However, after fine-tuning the models (Table 4.3), all metrics showed significant improvement, raising most values to a range around 0.85 and 0.92, across all batch sizes tested. The increase in these metrics, particularly F1 score and ROC AUC, implies a significant improvement in the models' capabilities of sequence classification and class differentiation, respectively.
We also compared the results from the fine-tuned ProtBERT model to four baseline approaches that combine two traditional ML classification models (two Random Forest models and two Logistic Regression models) and two approaches for sequence representation (OHE and amino-acid embeddings) established in Rodrigues (2025) [43]. These models were trained on the same subset of data as the fine-tuned ProtBERT model ('non-ML-designed'). The OHE method represents protein or nucleotide sequences as binary matrices, where each position in the sequence is mapped to a one-hot vector corresponding to its identity (amino acid or nucleotide), while the amino acid embedding representation is a numerical encoding of each amino acid in a dense vector space, similarly to word embeddings in NLP. The classification performance results of these models (Random Forest and Logistic Regression (first introduced in Ho et al. (1995) [24] and Cox et al. (1958) [13], respectively) in the ML-designed subset are present in Table 4.3. We can observe that the developed ProtBERT models outperform all baselines in terms of precision, recall, and F1-score. It is important to note that while Logistic Regression with OHE achieves the highest overall accuracy (0.9727) and ROC AUC (0.9761), it does so at the cost of recall and consequently generalization power.

To better elucidate the representational power of the fine-tuned PLM we produced visualizations using t-distributed stochastic neighbor embedding (t-SNE) (first introduced in Van et al (2008) [51]). t-SNE is a nonlinear dimensionality reduction technique that allows the visualiza-

tion of high-dimensional data, such as protein embeddings, by mapping it into a lower-dimensional space (typically 2D or 3D) while preserving local structure and relationships between data points. Specifically, our goal was to discern whether the fine-tuning process had the expected effect on the sequence embeddings: before any training, we would expect the model to generate embeddings for viable and nonviable sequences in a non-discerning manner, without clear general differences between the two groups of embeddings. Additionally, it would be expected that after the fine-tuning process the model would generate viable sequence embeddings clearly distinguishable from nonviable sequence embeddings. The analysis consisted of applying the t-SNE algorithm to the embeddings generated by ProtBERT from the sequences in our dataset. As control, the embeddings from similar-sized sequences randomly sampled from PDB [9] were used. This enabled graphical visualization of the similarity between embeddings of the different protein sequences: viable (from our dataset [11]), non-viable (from our dataset [11]) and randomly obtained sequences from PDB (acting as control sequences). The resulting plots (Figure 4.2) display the embeddings generated by ProtBERT before and after fine-tuning (left side plot column and right side plot column, respectively). In the plots regarding the pre-fine tuning embeddings, we can clearly observe that there is no clear clustering of embeddings of the same nature, as expected, since the pre-trained ProtBERT model (the model directly generating these embeddings) was unlikely to have captured differences between viable and nonviable examples of AAV2 sequences during its pre-training, as we do not expect it to have been trained with nonviable AAV2 sequences at all. After the fine-tuning, however, we can observe the abundance of two types of clusters: clusters of nonviable and random (control) sequences and clusters of viable sequences. The clearest and most explicit example of the formation of these clusters is in the post-fine-tuning plot of case A, where a cluster of nonviable and random (control) sequence embeddings is present across the left side of the plot, followed by a cluster comprised mostly by viable sequences on the right side of the plot. These clusters are also easily identifiable in the post-fine-tuning plots of the remaining cases (B, C and D). Interestingly, in the post-fine-tuning plots of cases A and C there is also a small cluster of nonviable sequences between these two larger clusters, which should correspond to embeddings of sequences generated by the model that were neither implicitly defined as viable nor nonviable through the model's representation.

In summary, before fine-tuning, the classification models exhibited lacking performance, indicated by low F1 score and ROC AUC values around 0.5 (for all models trained with varying batch sizes). These results showcase the model's initial inability to distinctly represent viable from nonviable AAV2 sequences, which is expected and is likely attributed to the lack of AAV2 representation in the model's pre-training data and the potentially subtle differences between viable and nonviable AAV2 sequences. After the fine-tuning process on AAV2 sequences, all evaluation metrics displayed a significant improvement, namely, the F1 score and ROC AUC values increased to a range between 0.85 and 0.92. This improvement is indicative of the success of the fine-tuning procedure and is further corroborated by the t-SNE visualization of the pre-fine-tuning and post-fine-tuning ProtBERT embeddings, which displayed an evolution of the ProtBERT

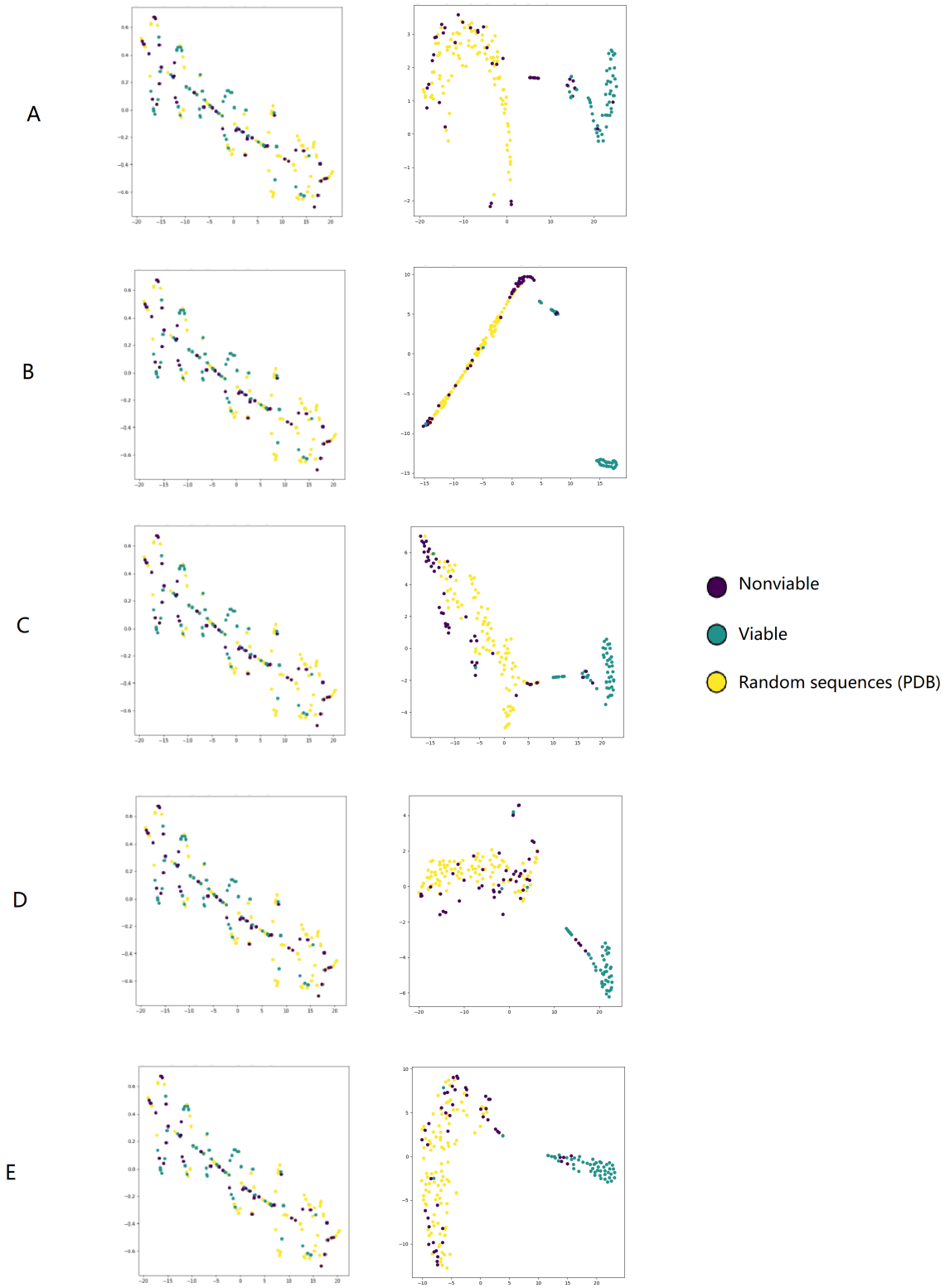


Figure 4.2: t-SNE analysis of ProtBERT embeddings before (left) and after (right) fine-tuning (A through E are results from the models trained using batch sizes 1 through 5, respectively)

AAV2 sequence representations, showing a clear differentiation of viable and nonviable sequence embeddings after the fine-tuning process.

Chapter 5

PLM-based generation of viable AAV2 sequences

5.1 Context

Although PLM representations hold significant theoretical promise to aid ML-guided AAV2 vector design, they have not been applied to new sequence generation. In this thesis, this gap was addressed by developing a PLM model fine-tuned in viral vector protein data, specifically, by fine-tuning the ProGen model [33] in AAV2 data. This chapter describes the development of the ProGen generative model, starting with an explanation of the data preprocessing steps and the details of the input, followed by additional information of the fine-tuning approach and generation of sequences for evaluation, and finally presenting two main model evaluations (through viability classification and structural evaluation) along with the respective discussions of the results.

ProGen designs proteins with desired characteristics by employing a conditional transformer model whereby protein sequences are complemented by tags that describe them. ProGen accepts two different types of tags as input: taxonomical tags and keyword tags. The taxonomical tags (the taxonomical lineage) correspond to the set of National Center for Biotechnology Information (NCBI) taxonomy IDs associated with the organism from which the target protein originates. The keyword tags (the keyword lineage) correspond to the set of UniProt functional IDs that describe the desired protein's characteristics. Overall, the taxonomical lineage guides the protein generation to be tailored for a specific organism, while the keyword lineage biases the model to generate a protein with specific functional characteristics. To achieve this, we obtained the taxonomical tags and keyword tags of the AAV2 capsid from NCBI and UniProt, respectively, as shown in Figure 5.3 and Figure 5.4 (with additional AAV2 keyword information present in Table 6). When the model receives this input, it converts each element (taxonomical tags, keyword tags and (optional) skeleton amino acid sequence) to its CTRL ID (Figure 5.1). These CTRL IDs are part of the model CTRL vocabulary, mapping each interpretable element (taxonomical tags, keyword tags and amino acids) to their unique CTRL ID.

Since ProGen is an autoregressive PLM, it generates the output sequence in a sequential man-

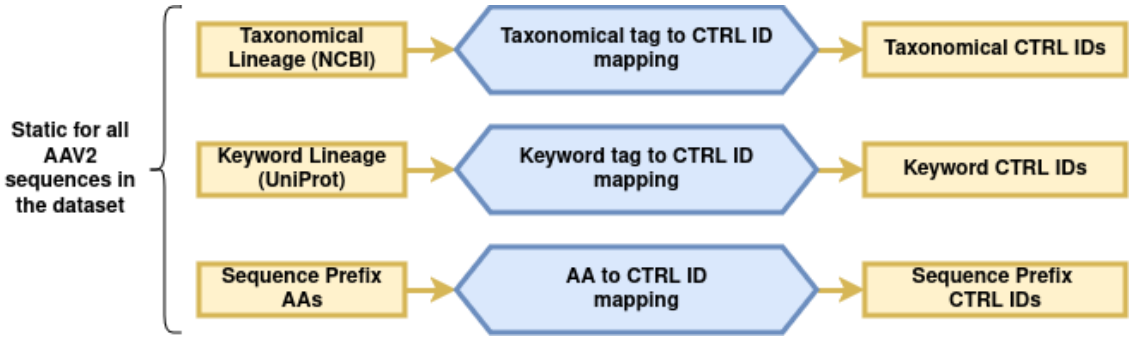


Figure 5.1: Processing of ProGen's input from NCBI IDs, UniProt IDs and amino acids to CTRL IDs

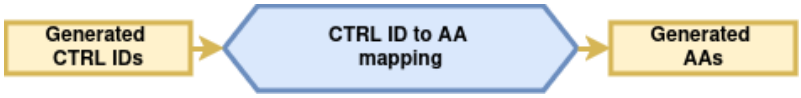


Figure 5.2: Conversion of ProGen CTRL ID outputs to amino acids

ner, using the content generated so far as input for the next step, until the sequence is completely generated. At every step, ProGen interprets the taxonomical CTRL IDs, keyword CTRL IDs and amino acid CTRL IDs of the amino acids generated up to that point. ProGen then returns a one-dimensional array of size N, where N is the number of different amino acids that the model can generate (N = 25 in the case of ProGen). This array contains the probabilities of each amino acid being selected as the next element to add to the sequence. ProGen then selects an amino acid through a top-k probability sampling (sampling from the distribution of the k amino acids with highest associated probability) and returns the input with the selected amino acid's CTRL ID appended to it (Figure 2.2). In a final step, the amino acid CTRL IDs are converted into the symbols of those amino acids, allowing us to obtain the generated sequence using the conventional amino acid vocabulary (Figure 5.2).

The pre-trained ProGen model was not originally trained on AAV2 data, implying that there were taxonomical tags and keywords specific to AAV2s that were not present in the ProGen vocabulary. To address this, mappings between the AAV2 taxonomies and keywords to new CTRL IDs were added to the vocabulary file. However, adding these new taxonomical tags and keywords to the ProGen vocabulary was not sufficient to ensure compatibility with the AAV2 training data. The weight matrix of the pre-trained ProGen's embedding layer also had to be extended. To accommodate the CTRL IDs of AAV2's taxonomical tags and keywords, 14 additional rows were appended to the weight matrix to account for the novel tags (7 taxonomy tags + 7 keyword tags), and randomly initialized.

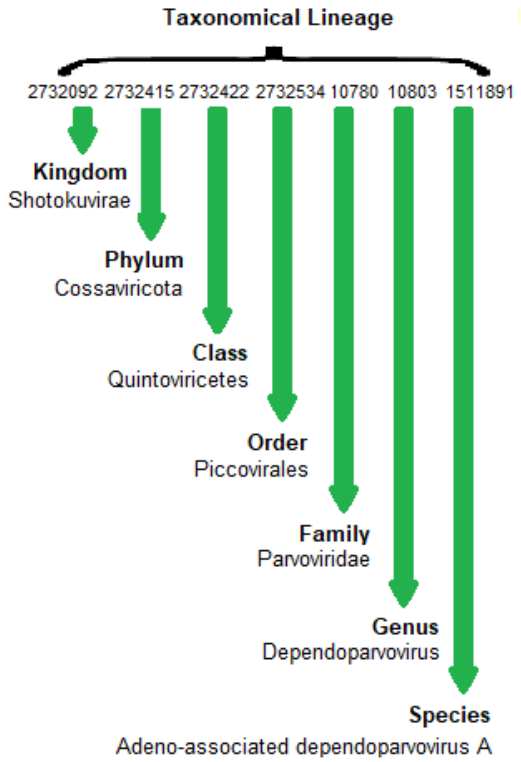


Figure 5.3: AAV2 Taxonomical Lineage

		Keyword ID
Biological process	#Clathrin-mediated endocytosis of virus by host	KW-1165
	#Host-virus interaction	KW-0945
	#Viral attachment to host cell	KW-1161
	#Viral penetration into host cytoplasm	KW-1162
	#Viral penetration via permeabilization of host membrane	KW-1173
	#Virus endocytosis by host	KW-1164
	#Virus entry into host cell	KW-1160

Figure 5.4: AAV2 Keyword Lineage

5.2 Dataset and data preprocessing

In the second part of this project, viable sequences from the same dataset employed in the classification model development were used to fine-tune the model. For every sequence, the AAV2 taxonomy and keyword tags were prepended to the uppercase version of each input sequence.

5.3 Methodology

5.3.1 Model fine-tuning

The methodology for training the generative model on viable non ML-designed AAV2 data was similar to the one used in the pre-training phase [32]. However, this fine-tuning involved training the model on the task of Masked Language Modeling (MLM) and using AAV2 taxonomical and functional sequence data (both tags and sequences). The generative model was trained over 3 epochs, with a dynamically decreasing learning rate of 10^{-3} and batch size of 1, following the recommendations for fine-tuning given in ProGen's repository. We evaluated the generated sequences using two independent methods: using the classifier model presented in the previous chapter to discern generated sequence viability (and also using the set of traditional ML classifiers from Rodrigues (2025) [43]); and assessing the structural similarity between the generated sequences and empirically viable sequences from the dataset.

5.3.2 Generation of sequences for evaluation

For both approaches, the generative model generated a set of 100 sequences of varying lengths. Our goal is to evaluate the model in the task of generating a specific subsequence of AAV2. This target region is also employed bt Bryant et al. (2021) [11] and corresponds to positions 561-588 of the sequence, a region that encompasses buried, surface and interface regions, and overlaps known heparin-binding as well as antibody binding sites (Figure 5.5).

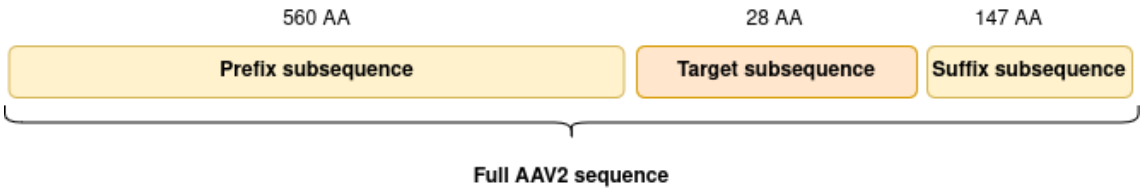


Figure 5.5: Sizes of the subsequences of the AAV2 wild-type sequence considered in the scope of this work

These sequences were generated by passing the fixed AAV2-specific skeleton subsequence to the AAV2-trained ProGen model (the prefix portion of unmutated amino acids, which remains consistent for all sequences in the dataset, the same subsequence as the 'Prefix subsequence' in 5.5) with a length of 560 amino acids. The generation was achieved by having the fine-tuned model generate a number of characters equal to sampled values from the mathematical distribution of sizes of the target subsection of each of the viable sequences in the dataset and adding the fixed AAV2-specific amino acid subsequence (the same subsequence as the 'Suffix subsequence' in 5.5) to its end with a length of 147 amino acids (the size distribution of the generated target subsequences are presented and the size distribution of the target subsequences in the training data are presented in Figures 5.7 and 5.6, respectively).

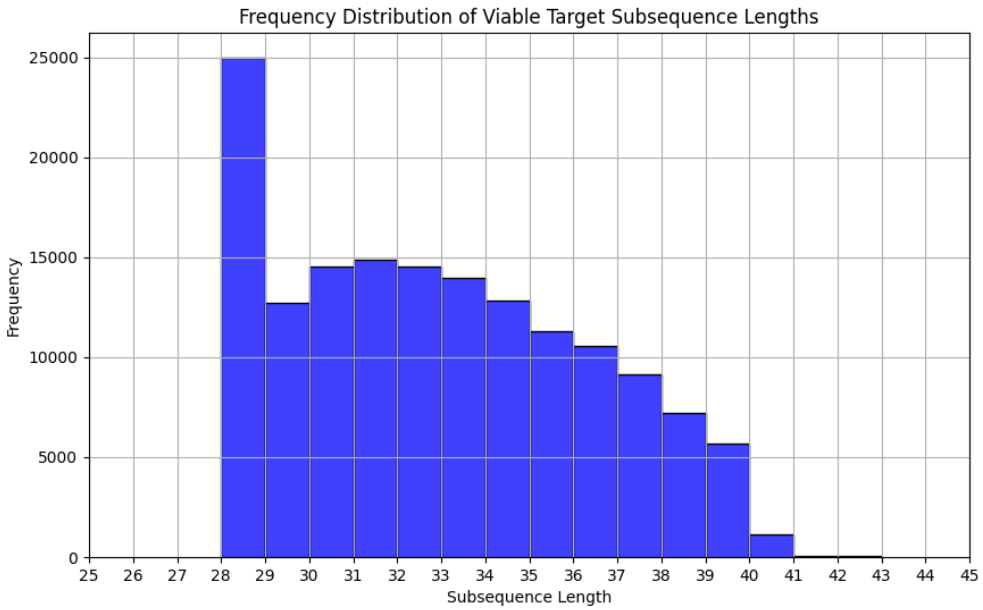


Figure 5.6: Size distribution of the target subsequence of the viable sequences in the dataset

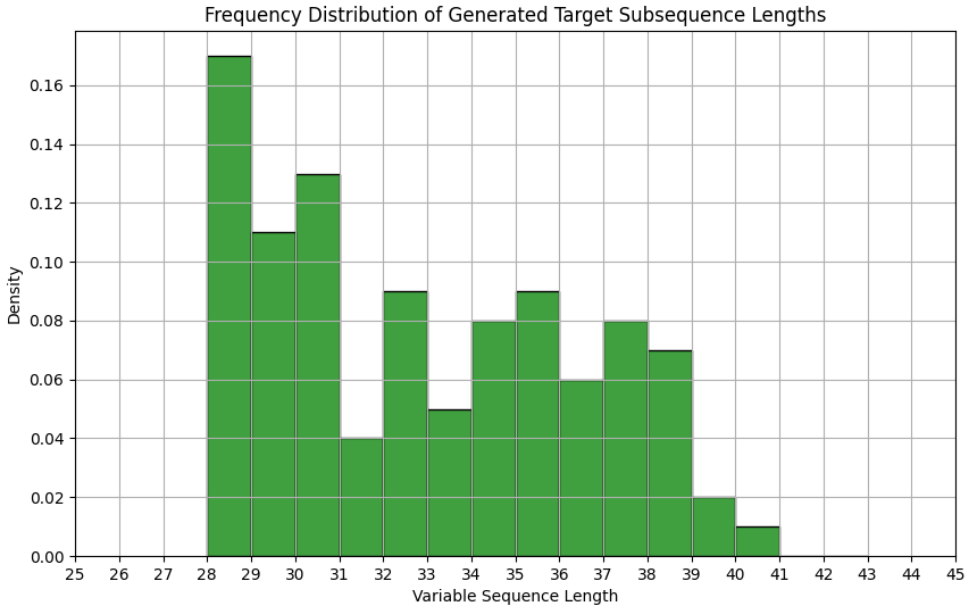


Figure 5.7: Size distribution of the target subsequence of the generated sequences

5.4 Results

5.4.1 Generated sequence evaluation using classifier models

This evaluation strategy relies in employing classification models trained to distinguish between viable and non-viable sequences. We employed both the classifier developed in the previous chapter, and a set of classifiers trained on the same dataset using classical ML classification models, including Random Forests and Logistic Regression from Rodrigues (2025) [43] which performance results are present in Table 4.3.

Each of 100 generated sequences were evaluated using each of the trained classifier models, with no generated sequence being considered viable by any of the ProtBERT classifiers (Table 5.1). In addition, as can be discerned from the viability probability predictions of the Random Forest and Logistic Regression models (Figures 5.8 and 5.9, respectively), neither of the models predicted any generated sequence to be viable using the OHE representations (left plot of both figures). However, some sequences, although a reduced number, were considered viable when using the amino acid embedding representation (right side of both figures). While these results clearly imply feature-level dissimilarities between the generated sequences and the viable-labeled sequences, some sequences were still considered viable when using amino acid embedding representations (although these positive classifications may also in reality consist of false positives).

Table 5.1: Results of classification of sequences generated by the generative model (as classified by the ProtBERT viability classification models)

Batch size	No. of sequences classified as viable	No. of sequences classified as nonviable
1	0	100
2	0	100
3	0	100
4	0	100
5	0	100

5.4.2 Generated sequence structural evaluation

This evaluation approach consisted of two sequential steps: the generation of a prediction of the three-dimensional structure of each generated sequence; and the generation of structural embeddings of the sequences, based on the predicted structures. These steps were applied to the other sequences from the dataset that we wished to compare to the newly generated sequences.

To predict the three-dimensional structures of the generated sequences, the AlphaFold2 model (developed by Jumper et al. (2021) [27]) was employed. AlphaFold2 was introduced as the first computational method capable of atomic-level protein structure prediction even in cases with no similar structure is known. In the paper it was introduced (Jumper et al. (2021) [27]), the authors demonstrate the capability of AlphaFold2 to make predictions accurate enough to rival experimentally achieved protein structural verification results. As such, we considered AlphaFold2

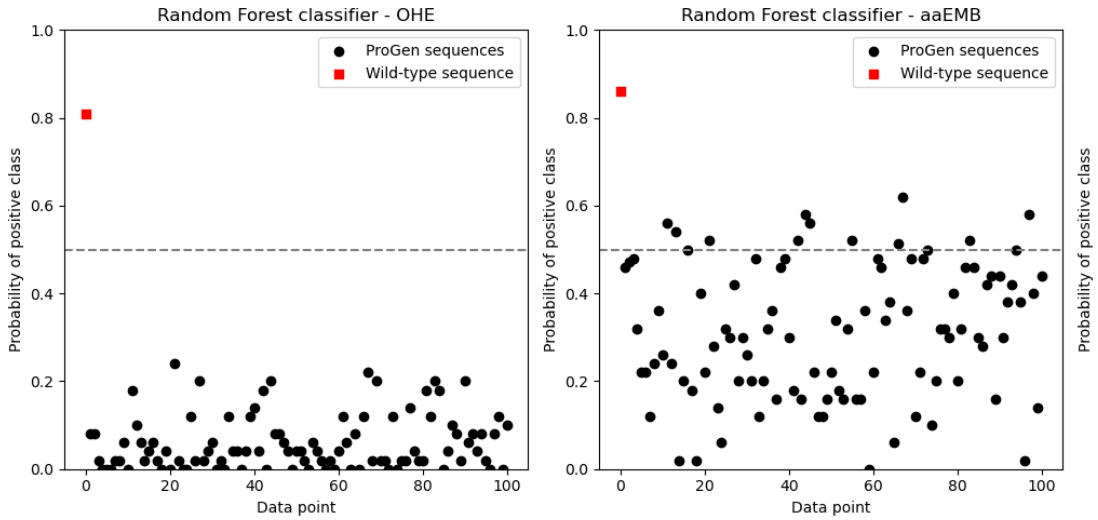


Figure 5.8: Generated sequence viability prediction results from Random Forest classifiers

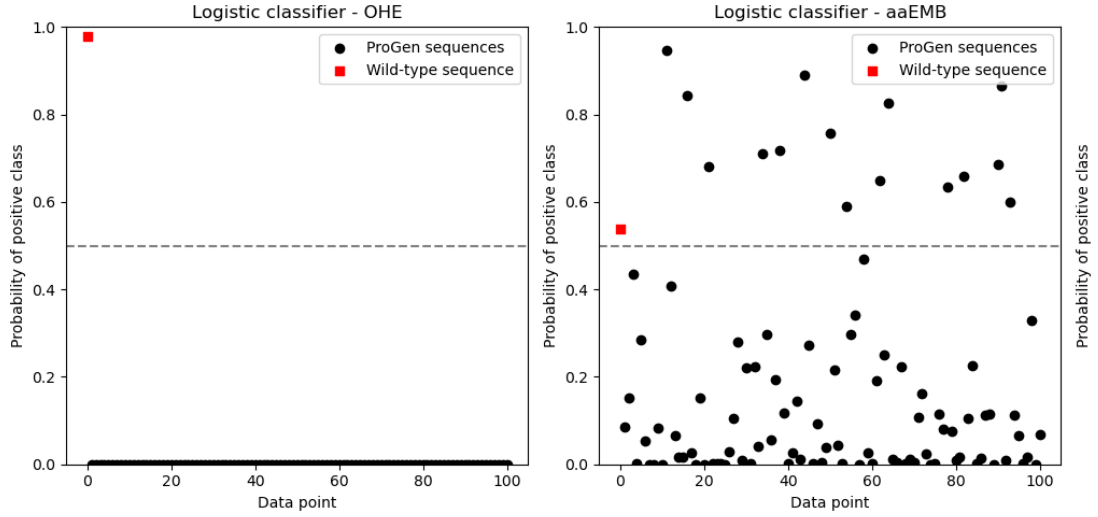


Figure 5.9: Generated sequence viability prediction results from Logistic Regression classifiers

predictions to be fitting as a replacement for experimentally obtained results (e.g. crystallography) as they are much easier and less expensive to obtain.

The AlphaFold2 prediction takes as input a FASTA file (a file containing a brief description of a biological structure and its corresponding nucleotide or amino acid sequence). AlphaFold2 then computes two separate components: a Multi-Sequence Alignment (MSA) and a pairwise distance plot. The MSA consists of a comparison matrix between the protein sequence encoded within the FASTA file and protein sequences of related organisms to the one tied to the sequence in the FASTA file. Alongside it, the pairwise distance plot consists of a matrix of predicted distances between residues within the sequence. These two component serve as the main features of AlphaFold: the MSA provides vital evolutionary context, highlighting conserved residues and co-evolutionary relationships (e.g., residues that mutate together), while the pairwise distance plot

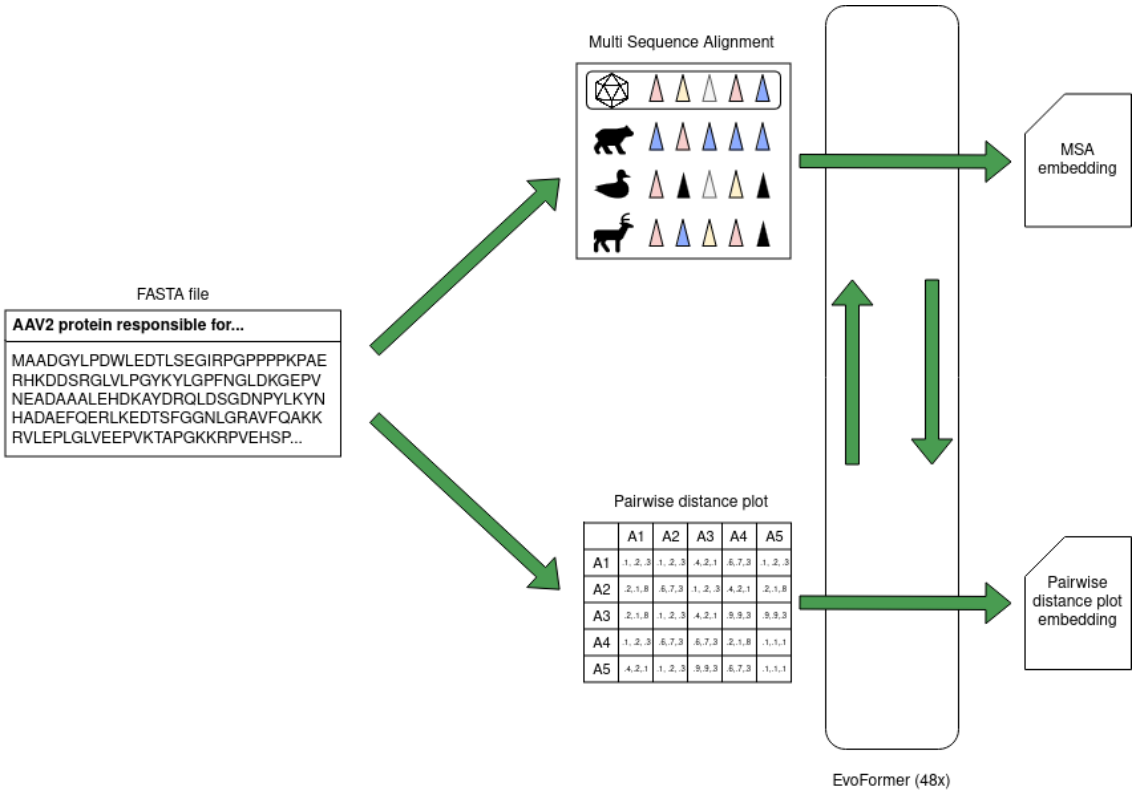


Figure 5.10: AlphaFold2 input features and EvoFormer module

contributes with spatial context. In the next step, using an EvoFormer layer (a transformer-based module), AlphaFold2 computes a representation for the MSA and a representation for the pairwise distance plot. The EvoFormer generates these representations simultaneously, taking the MSA and the pairwise distance plot as inputs. These representations are then passed to a structure module (a module responsible for translating the MSA and pairwise distance plot representations into a three-dimensional structure), which outputs a PDB file, containing a matrix with shape 3xN, with N being the number of atoms in the predicted structure, where each line is the predicted three-dimensional position of that atom. In order to potentially improve prediction quality, an optional final step can be performed (also known as the relaxation step), where the resulting representations are passed through the model again as input.
For this work, we deployed an instance of AlphaFold2 in a Google Cloud Virtual Machine and conducted sequence structure predictions in 'monomer' mode (since all sequences being analyzed are monomers), with the complete set of AlphaFold2 databases loaded on disk (for faster querying) and without relaxation (due to time constraints).

The following step consisted of obtaining structural embeddings based on the generated structural predictions of the generated sequences using AlphaFold. To achieve this, we used the Evolutionary Scale Modeling Inverse Folding 1 model (ESM-IF1) (published in Hsu et al. (2022) [25]). This model was developed for protein sequence prediction, given a protein's backbone atom coordinates. ESM-IF1 was trained on 12 million protein structures predicted by AlphaFold2, along-

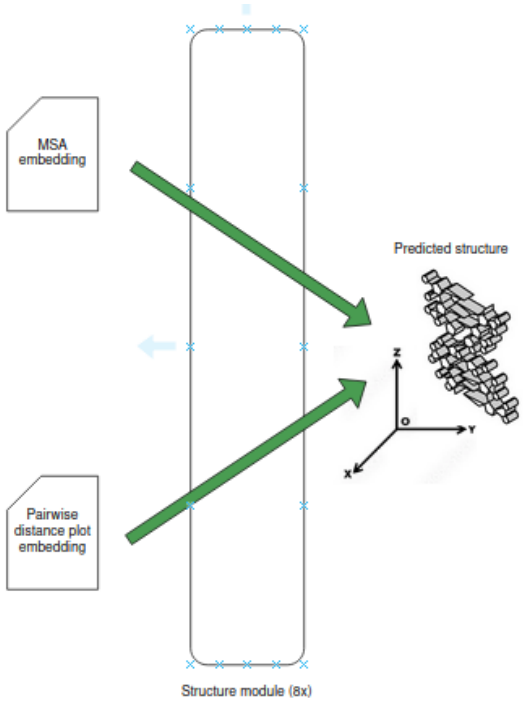


Figure 5.11: AlphaFold2 structure module and output

side approximately 16000 experimentally validated structures from the CATH database (Pearl et al. (2003) [41]). This dual supervision ensures that the model learns from both ML-predicted and experimentally determined structures. The model's architecture consists of a set of Geometric Vector Perceptron (GVP) layers (first introduced by Jing et al. (2020) [26] and designed for extracting features regarding spatial properties of protein structures), followed by an autoregressive encoder-decoder model, a generic implementation of the transformer architecture (Vaswani et al. (2017) [52]). As such, we were able to extract protein structural embeddings from the output of the encoder module.

Through dimensionality reduction, we were able to calculate distances between proteins relevant to this work, namely, we calculated the distance between our ProGen-generated sequences and samples of sequences in our dataset (both ML-designed and non-ML designed). Specifically, we sampled 250 viable and 250 non-viable sequences both from ML-designed subsets and non-ML-designed subsets. This sampling was done through a stratified extraction of a number of sequences of each type from each of the smaller subsets of each of the subsets comprising the ML-desinged and non-ML-designed sets of sequences, in order to preserve class distribution in the sampled data. For each generated sequence embedding, we then ranked the other non-generated sequence embeddings (either ML-designed or non-ML-designed) based on their cosine similarity to that embedding and calculated meaningful ranking metrics (present in Table 5.2).

In Figures 5.12 and 5.13 the t-SNEs of the generated sequence embeddings and embeddings of sequences from multiple data subsets are displayed (the first compares generated sequence embeddings to non-ML-designed sequence embeddings and the second compares generated sequence

embeddings to ML-designed sequence embeddings (based on the type of ML model used to obtain them in the dataset)). In these plots each point corresponds to an individual sequence embedding. For non-generated sequence emebeddings, points that are merely outlined indicate that that sequence was marked as nonviable in the dataset, while a color-filled point indicates the sequence was defined as viable.

In the non-ML-designed embedding t-SNE (Figure 5.12), several observations can be made regarding the positioning and clustering of the sequences: for each set of non-generated sequences, two types of clusters were formed - one with viable sequences from that subset and one with nonviable sequences from that subset (this can be clearly observed in the 'Single' subset, where a cluster of nonviable sequence points is present at the top center of the plot, with a cluster of viable sequence points directly below it). Another observation to be made is that these pairs of viable and nonviable sequence clusters are mostly positioned close to their counterparts in their subset (which is also to be expected due to the nature of the subsets: since each subset was obtained through a single and mostly unique mutational methodology, the structural sequence embeddings should also be similar within those subsets, as they should reflect sequence similarity, due to the link between sequence and structure). However, we can also clearly discern that the generated sequence points display significant dispersion across the reduced dimension, indicating high variability between them and a lack of clustering around a central region or around any other formed cluster. This implies that the generated sequences not only did not display any clear level of cohesion between them, but also did not display direct set similarity (in regards to structure) to any of the sequences in the subsets.

In the ML-designed embedding t-SNE (Figure 5.13), there is an overall clear separation between the non-generated sequence embeddings and the generated sequence embeddings, as multiple clusters of ML-designed sequence embeddings are observable (bottom left, top left, top, center right and top right, with the center right-positioned cluster being comprised mostly by nonviable sequence embeddings) while generated sequence embeddings are mostly confined to the bottom right area of the plot. While this indicates an overall dissimilarity between the generated sequence embeddings and the ML-designed sequence embeddings, in the area between the bottom right region of the plot and the rest of the plot there exists a belt of four clusters comprised by generated sequence embeddings and some ML-designed embeddings, indicating some similarity (although reduced, considering the rest of the plot) between these generated sequence embeddings and ML-designed embeddings.

For a stricter and more conclusive evaluation, we calculated the distances between each generated sequence structural embedding and each of 1000 structural embeddings from ML-designed sequences and non-ML-designed sequences (250 viable and 250 nonviable sequence embeddings from each). With those distances, for each generated sequence embedding, we ranked the non-generated embeddings from closest to farthest to that embedding. Using those rankings for each generated sequence, we calculated the ranking of the closest viable sequence embedding and obtained meaningful statistics, such as Hits@K (K $\in$ [1, 5, 10]) and the Mean Reciprocal Rank

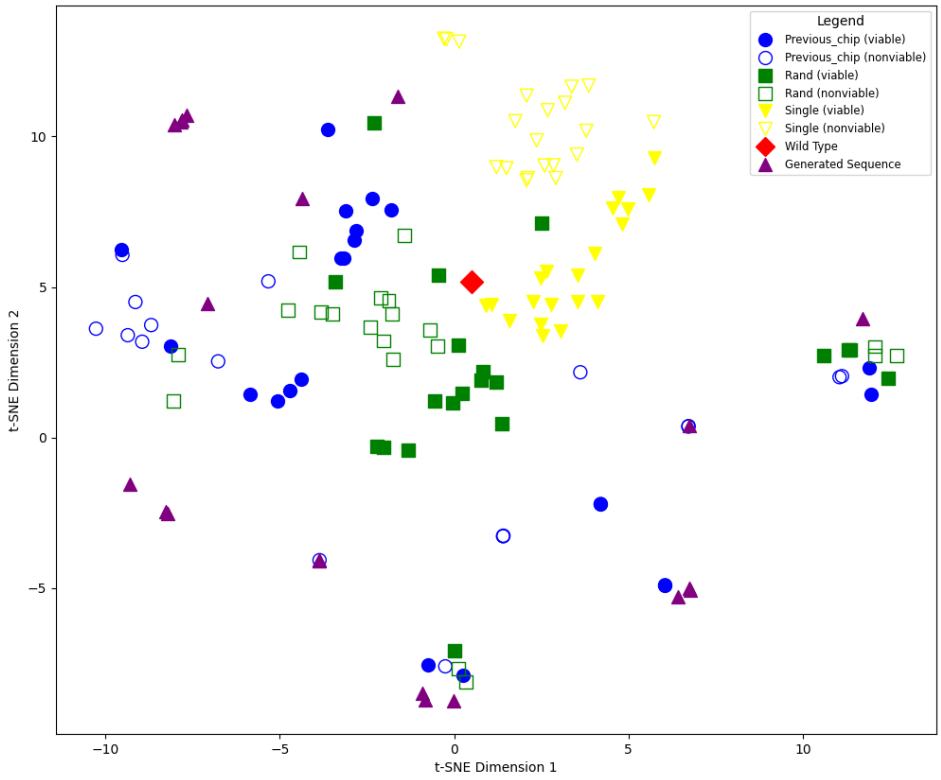


Figure 5.12: t-SNE of sequences from notable non-ML designed data subsets and generated sequences

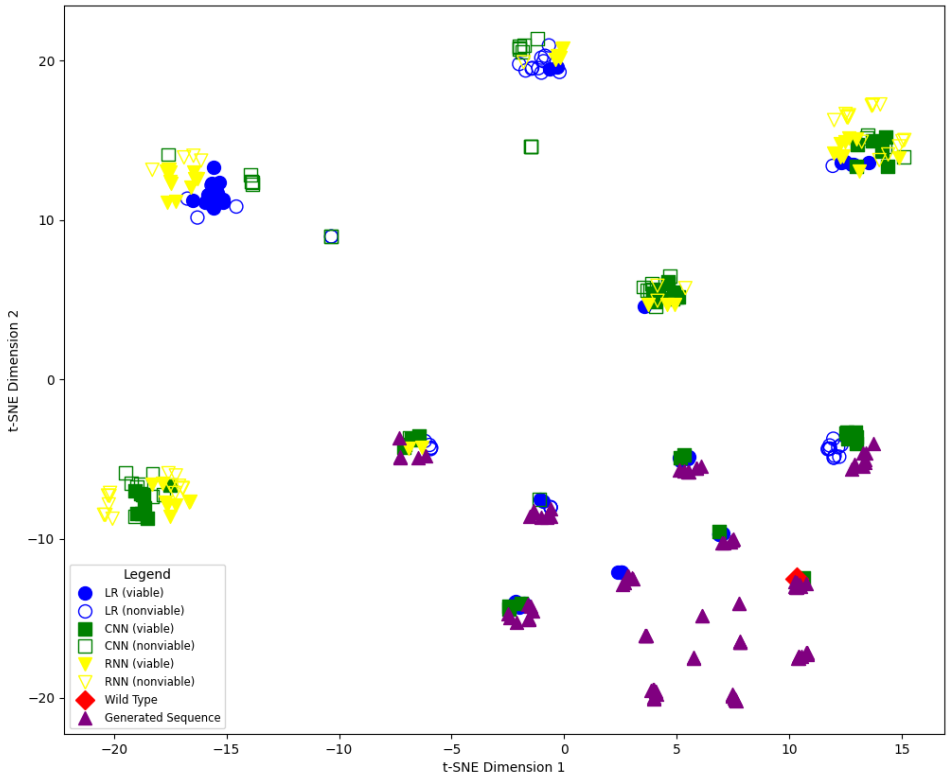


Figure 5.13: t-SNE of ML-designed sequences (based on ML model used to design them) and generated sequences

(MRR). These statistics are displayed in Table 5.2. The results appear to indicate an overall higher similarity between the generated sequence embeddings and the ML-designed sequence embeddings than the non-ML-designed sequence embeddings across all considered metrics, which supports the findings from the previously discussed t-SNE plots (Figures 5.12 and 5.13). This suggests that the generative model captures and reproduces structural, functional, or evolutionary constraints in a way that aligns with the ones captured and reproduced by the ML methods used to design the 'ML-designed' data, suggesting a sequence design convergence between the ProGen model and these ML techniques.

Table 5.2: Distance ranking metrics of each generated sequence structural embedding (of the set of 100 generated sequences) regarding the closest viable sequence structural embedding

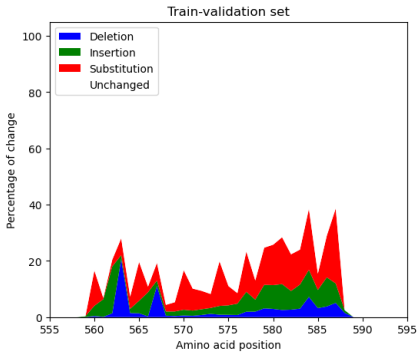
Model	Hits@1	Hits@5	Hits@10	MRR
ML-designed	79	91	99	0.8455
Non-ML-designed	28	58	61	0.3987

5.4.3 Generated sequence mutational topology analysis

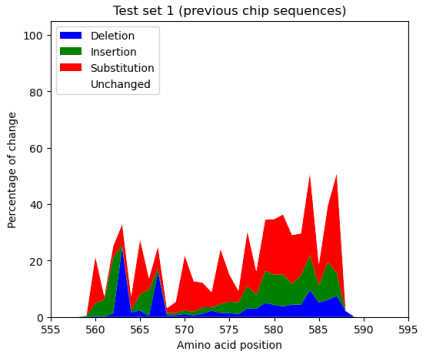
This additional evaluation was based on information present in Rodrigues (2025) [43] and aimed to assess whether the distribution of the amino acid mutations of the generated sequences displayed similarities to the mutational landscapes of the viable sequences in the dataset (and to other meaningful subsets in the dataset). This evaluation consisted in counting the number of insertions, deletions and substitutions in each target position of each sequence in the data and subsequently plotting the percentage of change in that position comparatively to the original sequence. This evaluation encompassed all 100 generated sequences and all sequences from the respective subsets. The plot regarding the mutational topology of the generated sequences (Figure 5.14f) displayed significant differences in regard to every other mutational topology plot (Figures 5.14a to 5.14e). Specifically, the generated sequences displayed higher mutation density than the sequences in the subsets, especially in specific regions such as 558-568 and 568-575, which are considered meaningful for sequence viability (as evidenced by Figure 5.14d). This divergence suggests that the generated sequences lack the mutational constraints observed in naturally occurring or experimentally viable sequences, likely contributing to their reduced viability. Additionally, the lack of similarity between the mutational topologies of the generated sequences and the nonviable sequences (Figures 5.14f and 5.14e, respectively) indicates that the generated sequences also do not display characteristics similar to nonviable sequences either, implying these sequences significantly differ from native AAV2 sequences (which is also substantiated by the dissimilarity between the mutational topology of the generated sequences and the mutational topologies of the sequences from the previous chip and ML-designed subsets (Figures 5.14b and 5.14c, respectively)).

As a closing note on the chapter, from the generative approach evaluation we can assess that the generative model failed to produce viable sequences, as none were classified as such by the ProtBERT classification models and very few were classified as viable by the models from Ro-

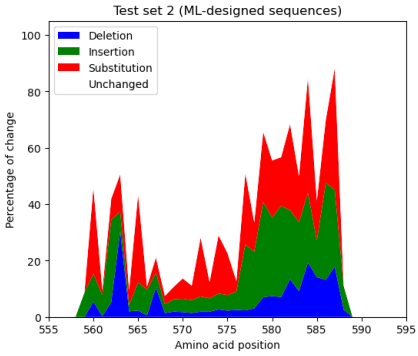
drigues (2025) [43]. To rule out classification bias, structural analysis using inverse folding models and embedding distance comparisons were conducted. Results showed hat generated sequences were structurally closer to ML-designed sequences than non-ML-designed ones, likely due to similarities in the natures of transformer-based models and the ML-based approaches employed in the generation of the data. The t-SNE visualization revealed high dispersion among generated sequences, without a clear clustering of generated sequences with viable and nonviable sequences in meaningful subsets, implying the presence of an increased generated sequence diversity but also dissimilarity between the generated sequences and the sequences in the subsets.



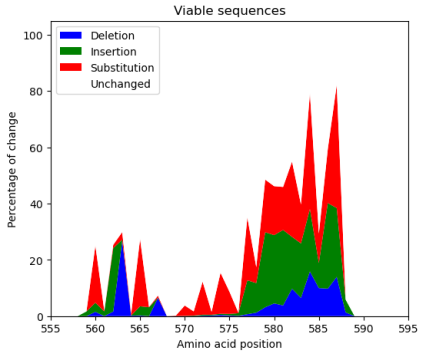
(a) Mutation topology of the train test sequences



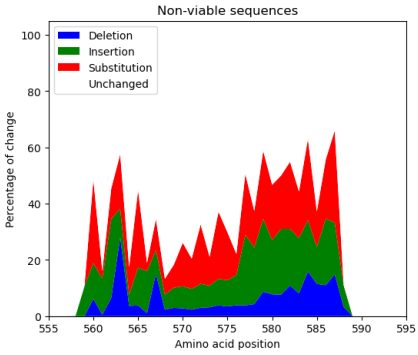
(b) Mutation topology of the previous chip sequences



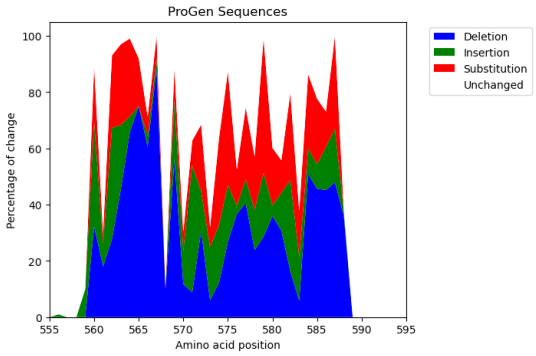
(c) Mutation topology of the ML-designed sequences



(d) Mutation topology of the viable sequences in the dataset



(e) Mutation topology of the nonviable sequences in the dataset



(f) Mutation topology of the generated sequences

Figure 5.14: Comparison of mutation topology across different sequence categories.

Chapter 6

Conclusion

The results of this work suggest that ProtBERT can be successfully adapted to the classification of AAV sequences through fine-tuning, resulting in a model that effectively learns meaningful AAV2 sequence representations (addressing **O1**). This enhancement demonstrates the effectiveness of the fine-tuning process, further supported by t-SNE visualizations of ProtBERT embeddings, which revealed a clear differentiation between viable and nonviable sequences post-fine-tuning. The classification models' ability to distinguish between viable and nonviable sequences improved significantly after AAV2 domain-specific training, answering **RQ1** validating the hypothesis that pre-trained PLMs, when fine-tuned on AAV2 data, can achieve high performance in the task of AAV2 sequence classification.

As for the development of the generative model (**O2**), the key observation is the fact that none of the passed sequences were classified as viable when using the ProtBERT classifiers. This fact clearly indicates an insufficiency in the generative model's capability of generating sequences with patterns that align with the viability-adjacent patterns the classification models learned throughout their training process (which we consider to be reliable due to their good performance on the viability classification task). In order to circumvent the possibility of a bias in the classification of the generated sequences, we performed a structural evaluation of the generated sequences through the use of inverse folding models (for the generation of structural embeddings and subsequent embedding distance analysis). Additionally, we visualized the generated sequence structural embeddings (along with other viable sequence structural embeddings from meaningful data subsets) and performed a distance-based ranking of the dataset sequence embeddings for each generated sequence. Through these methods, we were able to inquire that the generated sequence structural embeddings were overall closer to the ML-designed structural embeddings than non-ML-designed structural embeddings, which could be attributed to the adjacent nature of the methods used to generate the sequences and methods used to generate the ML-designed subset (transformer-based models vs CNNs, RNNs and LR, all ML-based computational methods). This result may imply the existence of a shared bias between the generative model and these ML-based methods, potentially influencing the 'true' novelty of the generated sequences when comparing the transformer-based approach to the already existing approaches. Through t-SNE visualization, we were also able to observe that the structural embeddings of the meaningful subsets' viable and nonviable sequences clus-

tered more closely (within their respective groups). In contrast, generated sequences displayed a much greater dispersion. This further supports the classifier models' evaluation results, implying that the generated sequences did not align well with the established space of viable (and, in this case, nonviable) sequences. While the model successfully generated diverse sequences, all the previously mentioned evaluations indicate that further refinement is needed to reliably generate sequences that consistently align with known viable AAV2 sequences (both in viability and in structure) (**RQ2**).

The previous conclusions suggest that future work should focus on improving the generative process. One hypothesized approach would be to further adjust the generated sequence mutational landscape by adjusting generation temperature and top-k sampling values and extending the models' training period through additional epochs. One other approach, also compatible with the previously mentioned ones, would be to train the generative model with explicit viability tags to more directly constrain the learned patterns by learning (and discerning between) the patterns for generating viable and nonviable AAV2 sequences. Another exciting idea would be to directly incorporate classifier feedback during generation to improve the quality and coherence of the generated sequences (establishing a full framework that would contain both types of models). Additionally, we wish to extend the scope of the generative approach to address other AAV2 properties to further enhance the utility of the research and broaden its applications.

Overall, we consider that this work showcases the potential of the usage of PLM-based techniques in viral vector design. While the sequence generation method present in this work still requires further improvement to correctly adhere its generative capabilities to the expected performance, we believe that the performance of the classification models constitutes a meaningful testimony to the potential of PLMs in viral vector design, as the use of this models could potentially guide viral vector researchers in evaluating viral vector prediction, since the use of PLMs can be a computationally efficient way to explore this space, narrowing down promising candidates for further testing, decreasing research costs and improving general design efficiency, accelerating the development of the viral vector design field.

Bibliography

[1] Deep diversification supplementary materials, 2021. https://www.nature.com/articles/s41587-020-00793-4 Accessed: 2024-09-08.

[2] Huggingface library, 2024. https://huggingface.co/ Accessed: 2024-09-08.

[3] AM Alanazi and FO Alanazi. Machine learning and biomedicine. *Journal of Contemporary Scientific Research (ISSN (Online) 2209-0142)*, 4(1):7, 2020.

[4] Fatemeh Arabi, Vahid Mansouri, and Naser Ahmadbeigi. Gene therapy clinical trials, where do we go? an overview. *Biomedicine & Pharmacotherapy*, 153:113324, 2022.

[5] Amos Bairoch, Rolf Apweiler, Cathy H Wu, Winona C Barker, Brigitte Boeckmann, Serenella Ferro, Elisabeth Gasteiger, Hongzhan Huang, Rodrigo Lopez, Michele Magrane, et al. The universal protein resource (uniprot). *Nucleic acids research*, 33(suppl_1):D154–D159, 2005.

[6] Amos Bairoch, Brigitte Boeckmann, Serenella Ferro, and Elisabeth Gasteiger. Swiss-prot: juggling between evolution and stability. *Briefings in bioinformatics*, 5(1):39–55, 2004.

[7] Alex Bateman, Lachlan Coin, Richard Durbin, Robert D Finn, Volker Hollich, Sam Griffiths-Jones, Ajay Khanna, Mhairi Marshall, Simon Moxon, Erik LL Sonnhammer, et al. The pfam protein families database. *Nucleic acids research*, 32(suppl_1):D138–D141, 2004.

[8] Jonas Becker, Julia Fakhiri, and Dirk Grimm. Fantastic aav gene therapy vectors and how to find them—random diversification, rational design and machine learning. *Pathogens*, 11(7):756, 2022.

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

[10] Brigitte Boeckmann, Amos Bairoch, Rolf Apweiler, Marie-Claude Blatter, Anne Estreicher, Elisabeth Gasteiger, Maria J Martin, Karine Michoud, Claire O'Donovan, Isabelle Phan, et al. The swiss-prot protein knowledgebase and its supplement trembl in 2003. *Nucleic acids research*, 31(1):365–370, 2003.

[11] Drew H Bryant, Ali Bashir, Sam Sinai, Nina K Jain, Pierce J Ogden, Patrick F Riley, George M Church, Lucy J Colwell, and Eric D Kelsic. Deep diversification of an aav capsid protein by machine learning. *Nature Biotechnology*, 39(6):691–696, 2021.

[12] The UniProt Consortium. UniProt: the Universal Protein Knowledgebase in 2023. *Nucleic Acids Research*, 51(D1):D523–D531, 11 2022.

[13] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.

[14] Marcus Davidsson, Gang Wang, Patrick Aldrin-Kirk, Tiago Cardoso, Sara Nolbrant, Morgan Hartnor, Janitha Mudannayake, Malin Parmar, and Tomas Björklund. A systematic capsid evolution approach performed in vivo for the design of aav vectors with tailored properties and tropism. *Proceedings of the National Academy of Sciences*, 116(52):27053–27062, 2019.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[16] Xinqiang Ding, Zhengting Zou, and Charles L Brooks III. Deciphering protein evolution and fitness landscapes with latent space models. *Nature communications*, 10(1):5644, 2019.

[17] Sara El-Gebali, Jaina Mistry, Alex Bateman, Sean R Eddy, Aurélien Luciani, Simon C Potter, Matloob Qureshi, Lorna J Richardson, Gustavo A Salazar, Alfredo Smart, et al. The pfam protein families database in 2019. *Nucleic acids research*, 47(D1):D427–D432, 2019.

[18] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rehawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, et al. Prottrans: Toward understanding the language of life through self-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 44(10):7112–7127, 2021.

[19] Noelia Ferruz, Michael Heinzinger, Mehmet Akdel, Alexander Goncearenco, Luca Naef, and Christian Dallago. From sequence to function through structure: Deep learning for protein design. *Computational and Structural Biotechnology Journal*, 21:238–250, 2023.

[20] Noelia Ferruz and Birte Höcker. Controllable protein design with language models. *Nature Machine Intelligence*, 4(6):521–532, 2022.

[21] Joe G Greener, Shaun M Kandathil, Lewis Moffat, and David T Jones. A guide to machine learning for biologists. *Nature Reviews Molecular Cell Biology*, 23(1):40–55, 2022.

[22] Joe G Greener, Lewis Moffat, and David T Jones. Design of metalloproteins and novel protein folds using variational autoencoders. *Scientific reports*, 8(1):16189, 2018.

[23] Zengpeng Han, Nengsong Luo, Fei Wang, Yuxiang Cai, Xin Yang, Weiwei Feng, Zhenxiang Zhu, Jie Wang, Yang Wu, Chaohui Ye, et al. Computer-aided directed evolution generates novel aav variants with high transduction efficiency. *Viruses*, 15(4):848, 2023.

[24] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[25] Chloe Hsu, Robert Verkuil, Jason Liu, Zeming Lin, Brian Hie, Tom Sercu, Adam Lerer, and Alexander Rives. Learning inverse folding from millions of predicted structures. In *International conference on machine learning*, pages 8946–8970. PMLR, 2022.

[26] Bowen Jing, Stephan Eismann, Patricia Suriana, Raphael JL Townshend, and Ron Dror. Learning from protein structure with geometric vector perceptrons. *arXiv preprint arXiv:2009.01411*, 2020.

[27] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

[28] Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.

[29] Rasko Leinonen, Federico Garcia Diez, David Binns, Wolfgang Fleischmann, Rodrigo Lopez, and Rolf Apweiler. Uniprot archive. *Bioinformatics*, 20(17):3236–3237, 2004.

[30] Jiajia Liu, Mengyuan Yang, Yankai Yu, Haixia Xu, Kang Li, and Xiaobo Zhou. Large language models in bioinformatics: applications and perspectives. *ArXiv*, 2024.

[31] Suyue Lyu, Shahin Sowlati-Hashjin, and Michael Garton. Variational autoencoder for design of synthetic viral vector serotypes. *Nature Machine Intelligence*, 6(2):147–160, 2024.

[32] Ali Madani, Ben Krause, Eric R Greene, Subu Subramanian, Benjamin P Mohr, James M Holton, Jose Luis Olmos, Caiming Xiong, Zachary Z Sun, Richard Socher, et al. Large language models generate functional protein sequences across diverse families. *Nature Biotechnology*, 41(8):1099–1106, 2023.

[33] Ali Madani, Bryan McCann, Nikhil Naik, Nitish Shirish Keskar, Namrata Anand, Raphael R Eguchi, Po-Ssu Huang, and Richard Socher. Progen: Language modeling for protein generation. *arXiv preprint arXiv:2004.03497*, 2020.

[34] Andrew D Marques, Michael Kummer, Oleksandr Kondratov, Arunava Banerjee, Oleksandr Moskalenko, and Sergei Zolotukhin. Applying machine learning to predict viral assembly for adeno-associated virus capsid libraries. *Molecular Therapy-Methods & Clinical Development*, 20:276–286, 2021.

[35] Georgios Mikos, Weitong Chen, and Junghae Suh. Machine learning identification of capsid mutations to improve aav production fitness. *bioRxiv*, pages 2021–06, 2021.

[36] Michael Moor, Oishi Banerjee, Zahra Shakeri Hossein Abad, Harlan M Krumholz, Jure Leskovec, Eric J Topol, and Pranav Rajpurkar. Foundation models for generalist medical artificial intelligence. *Nature*, 616(7956):259–265, 2023.

[37] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*, 2023.

[38] Dan Ofer, Nadav Brandes, and Michal Linial. The language of proteins: Nlp, machine learning & protein sequences. *Computational and Structural Biotechnology Journal*, 19:1750–1758, 2021.

[39] Pierce J Ogden, Eric D Kelsic, Sam Sinai, and George M Church. Comprehensive aav capsid fitness landscape reveals a viral gene and enables machine-guided design. *Science*, 366(6469):1139–1143, 2019.

[40] Departement Operationnelle, Y. Bengio, R Ducharme, Pascal Vincent, and Centre Mathematiques. A neural probabilistic language model. 10 2001.

[41] Frances MG Pearl, CF Bennett, James E Bray, Andrew P Harrison, Nigel Martin, A Shepherd, Ian Sillitoe, J Thornton, and Christine A Orengo. The cath database: an extended protein family resource for structural and functional genomics. *Nucleic acids research*, 31(1):452–455, 2003.

[42] Leslie D Pettit and KJ Powell. The iupac stability constants database. *Chem. Int*, 56:14–15, 2006.

[43] Ana Rodrigues. Machine learning-based viral design for bioengineering (in preparation). *Faculdade de Ciências da Universidade de Lisboa*, 2025.

[44] Adrian Schwarzer, Steven R. Talbot, Anton Selich, Michael Morgan, Juliane W. Schott, Oliver Dittrich-Breiholz, Antonella L. Bastone, Bettina Weigel, Teng Cheong Ha, Violetta Dziadek, Rik Gijsbers, Adrian J. Thrasher, Frank J.T. Staal, Hubert B. Gaspar, Ute Modlich, Axel Schambach, and Michael Rothe. Predicting genotoxicity of viral vectors for stem cell gene therapy using gene expression-based machine learning. *Molecular Therapy*, 29(12):3383–3397, 2021.

[45] Sam Sinai, Nina Jain, George M Church, and Eric D Kelsic. Generative aav capsid diversification by latent interpolation. *bioRxiv*, pages 2021–04, 2021.

[46] Cristina Sotomayor-Vivas, Enrique Hernández-Lemus, and Rodrigo Dorantes-Gilardi. Linking protein structural and functional change to mutation using amino acid networks. *Plos one*, 17(1):e0261829, 2022.

[47] Baris E Suzek, Yuqi Wang, Hongzhan Huang, Peter B McGarvey, Cathy H Wu, and UniProt Consortium. Uniref clusters: a comprehensive and scalable alternative for improving sequence similarity searches. *Bioinformatics*, 31(6):926–932, 2015.

[48] Cuong T To, Christian Wirsching, Andrew D Marques, and Sergei Zolotukhin. Using machine learning to design adeno-associated virus capsids with high likelihood of viral assembly. *bioRxiv*, pages 2021–05, 2021.

[49] Rosella Tomanin and Maurizio Scarpa. Why do we need new gene therapy viral vectors? characteristics, limitations and future perspectives of viral vector transduction. *Current gene therapy*, 4(4):357–372, 2004.

[50] Giorgio Valentini, Dario Malchiodi, Jessica Gliozzo, Marco Mesiti, Mauricio Soto-Gomez, Alberto Cabri, Justin Reese, Elena Casiraghi, and Peter N Robinson. The promises of large language models for protein design and modeling. *Frontiers in Bioinformatics*, 3, 2023.

[51] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[53] Jesse Vig, Ali Madani, Lav R Varshney, Caiming Xiong, Richard Socher, and Nazneen Fatema Rajani. Bertology meets biology: Interpreting attention in protein language models. *arXiv preprint arXiv:2006.15222*, 2020.

[54] Dixuan Wang, Yanda Li, Junyuan Jiang, Zepeng Ding, Guochao Jiang, Jiaqing Liang, and Deqing Yang. Tokenization matters! degrading large language models through challenging their tokenization. *arXiv preprint arXiv:2405.17067*, 2024.

[55] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

[56] Jinbiao Yang. Rethinking tokenization: Crafting better tokenizers for large language models. *International Journal of Chinese Linguistics*, 11(1):94–109, 2024.

Appendices

Keyword ID	Function	Description
945	Host-virus interaction	Viral or cellular protein involved in a host-virus interaction. Viruses interact with many cellular pathways to achieve their replication cycle. Entry into the host cell, transport to the viral replication sites or viral exit from the host cell are all steps that require specific interactions between the virus and its host. Additionally, the evasion from the host immune response requires a lot of viral proteins to associate with and inhibit cellular proteins with antiviral functions.
1160	Virus entry into host cell	Viral protein involved in the virion entry into a host cell. Entry is a multistep process that mostly requires binding to the target cell, penetration into the host cell cytoplasm, intracellular transport of viral components and genome release to the replication site of the virus.
1161	Viral attachment to host cell	Viral surface protein implicated in the binding to specific host surface molecule(s). This binding can lead to virion entry into the host cell, it can trigger signaling pathways, or it can allow the virion to be carried by the host cell to a specific organ.
1162	Viral penetration into host cytoplasm	Viral protein involved in the entry of the entire virion, some of its contents or at least its genetic material into the host cell cytoplasm. Entry is achieved through pore formation, membrane fusion, pilus retraction, ejection, permeabilization and/or endocytosis mechanisms. Penetration reactions occur mainly in five locations: the cell membrane, early and late endosomes, caveosomes, and the ER. In prokaryotes, membrane fusion, pore formation through a puncturing device, ejection, pilus retraction or permeabilization occur at the plasma membrane, as well as at the outer membrane in prokaryotes with 2 membranes.
1164	Virus endocytosis by host	Viral protein involved in virus internalization by the host cell via endocytosis. Endocytosis can occur via: clathrin-mediated endocytosis, caveolin-mediated endocytosis or clathrin- and caveolae-independent endocytosis.

1165	Clathrin-mediated endocytosis of virus by host	Viral protein involved in virus internalization by the host cell via clathrin-mediated endocytosis. In response to an internalization signal, clathrin is assembled on the inside face of the cell membrane to form characteristic invaginations or clathrin coated pits that pinch off through the action of DNM1/Dynamin-1 or DNM2/Dynamin-2. The virus bound to its host cell receptor is internalized into clathrin- coated vesicles (CCV). Endocytic CCV deliver their viral content to early endosomes. The endosomal acidic pH and/or receptor binding usually induces structural modifications of the virus surface proteins that lead to penetration of the endosomal membrane via fusion or permeabilization mechanisms.
1173	Viral penetration via permeabilization of host membrane	Viral membrane-penetration protein, usually associated with the viral capsid or released through partial programmed disassembly of the capsid, that locally permeabilizes the bilayer of a host membrane to allow virion penetration into the cytoplasm. Viral membrane- penetration proteins often need to be activated, mostly through endosomal acidic pH or receptor binding to display their membrane penetrating activity. Non-enveloped viruses such as parvovirus, human reovirus, BDV, BTV, rotavirus, papillomavirus, Flock house virus permeabilize the host endosomal membrane to penetrate the host cytoplasm.

Table 6: Keyword lineage of the AAV2 capsid and respective descriptions