UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Machine learning-based viral design for bioengineering

Ana Filipa de Albuquerque Ferreira Rodrigues

Mestrado em Ciência de Dados

Dissertação orientada por:
Cátia Luísa Santana Calisto Pesquita

2025

It's never too late

Agradecimentos

Gostaria de expressar os meus mais sinceros agradecimentos a todos aqueles que, directa ou indirectamente, contribuíram para a realização deste trabalho de tese.

À Professora Cátia Pesquita, por me ter acolhido no seu grupo de investigação num momento de mudança e incerteza, sempre oferecendo palavras de incentivo e encorajamento. Pela orientação técnica e científica mas também pelo apoio pessoal, ambos de valor inestimável. Pela abertura e disponibilidade para explorar a área do *ML-based protein design*, o que permitiu alavancar o meu *background* em Biotecnologia e Ciências da Vida, enquanto me aventurava no fascinante mundo da Ciência de Dados e Inteligência Artificial, uma combinação que me ajudou a evoluir mais rapidamente e suavizou a curva de aprendizagem.

Ao Professor Tiago Guerreiro, coordenador do Mestrado em Ciência de Dados no ano lectivo de 2022/2023, pela forma positiva e motivadora com que recebeu o meu interesse em realizar este Mestrado e por me direccionar, de forma atenta e cuidadosa, para o grupo LiSeDa, onde as minhas competências puderam ser potenciadas facilitando a transição para esta nova área de investigação que tanto me entusiasma actualmente.

A todos os colegas do grupo LiSeDa, actuais e passados, pelo acolhimento amistoso e pela boa disposição. Por partilharem os seus conhecimentos, ajudando-me a crescer, e por me mostrarem, com a sua experiência e apoio, que não estava sozinha nos desafios que enfrentava, o que me permitiu sentir-me mais integrada e confiante.

Entre os colegas do grupo, uma palavra especial para o Lucas. A tua generosidade e paciência em lidar com as minhas dificuldades no ‘computês’ foram, e continuam a ser, essenciais. O teu entusiasmo é também uma fonte de inspiração.

Finalmente, um agradecimento a todos aqueles que trabalham para manter e fortalecer as instituições associadas a este trabalho, nomeadamente do Departamento de Informática, da sua unidade de investigação LASIGE e a todos os que fazem da Faculdade de Ciências da Universidade de Lisboa um local magnífico para se aprender e crescer.

Resumo

O design de proteínas tem sido, desde há muito tempo, uma área de grande interesse e investigação ativa no campo das ciências biológicas e da bioengenharia. O desenvolvimento de métodos computacionais para orientar o design e a bioengenharia de proteínas, em particular os baseados em aprendizagem automática (machine learning), constituiu uma das alavancas mais significativas nesta área. Estes métodos ajudam a reduzir o vasto espaço de procura, permitindo identificar sequências com maior probabilidade de cumprir funções desejadas ou de possuir propriedades específicas. Com os avanços mais recentes neste campo, deram-se também os primeiros passos para aplicar o design de proteínas orientado por máquinas na bioengenharia de partículas virais terapêuticas, nomeadamente, vectores virais para terapia génica. O design e a bioengenharia de partículas virais têm um enorme potencial no desenvolvimento de terapias baseadas em vírus, além de impulsionarem novas descobertas nas áreas da virologia e da vectorologia, sendo por isso uma área de elevado interesse.

O presente trabalho explorou métodos de ponta no design de proteínas orientado por máquinas, com uma atenção particular nas técnicas de aprendizagem automática, contribuindo para a evolução da bioengenharia de vectores virais. Como estudo de caso, o foco recaiu sobre os vectores baseados em vírus adeno-associados (adeno-associated virus, AAV), que são atualmente um dos veículos mais utilizadas para a transferência de material genético em terapias génica. Especificamente, propôs-se o desenvolvimento de classificadores binários capazes de distinguir sequências viáveis de não viáveis para auxiliar no design de novos AAVs altamente diversos, ou seja, substancialmente diferentes dos atualmente em uso. Esta diversidade é relevante para aspectos como redução da imunogenecidade do tratamento em terapias que requeiram administrações múltiplas.

Embora o design de proteínas baseado em aprendizagem automática tenha vindo a ser utilizado com sucesso na bioengenharia de novos vectores para terapia génica, em especial os AAVs, existem ainda algumas limitações nas abordagens que foram executadas, e que este trabalho se propôs explorar. Essas limitações estão principalmente relacionadas com os formatos usados para a representação de sequências de proteínas, que até agora têm recorrido predominantemente a métodos de codificação posicional simples, como a codificação *one-hot*, e que são pobres em informação biológica. Outra limitação reside na escolha de modelos para a aprendizagem, que tem sido predominantemente baseada em modelos individuais, sem explorar a complementaridade de diferentes abordagens.

O trabalho iniciou-se com a aplicação de métodos de análise exploratória de dados, como a construção do *landscape* de mutações de conjuntos de sequências de interesse (por exemplo, treino-validação, teste, viáveis, não-viáveis, etc.), e a avaliação das suas estatísticas de mutação. Esse processo permitiu a identificação de assinaturas e padrões característicos para cada grupo de sequências, que foram posteriormente utilizados para interpretar e compreender os resultados ao longo do estudo. Como parte da análise exploratória inicial, foram também aplicados métodos de aprendizagem não supervisionada, como o agrupamento *K-means* e t-SNE (*t-Distributed Stochastic Neighbor Embedding*), que ajudaram a confirmar a necessidade de métodos mais sofisticados para distinguir sequências viáveis de não-viáveis. Estas análises forneceram, adicionalmente, informações importantes sobre as diferenças nos formatos de representação e ajudaram a perceber melhor o desempenho subsequente dos modelos de classificação.

No que diz respeito à representação das proteínas, foram explorados formatos mais sofisticados do que os baseados em codificação posicional simples dos aminoácidos, com o intuito de capturar informações biológicas mais complexas, tais como dependências posicionais e relações entre aminoácidos nas sequências de proteínas. Para isso, foram usados modelos de linguagem de proteínas para gerar representações

vectoriais significativas (*embeddings*), capazes de capturar informações biológicas detalhadas, como o contexto dos aminoácidos, interações de curta e longa distância, padrões e motivos nas sequências, capturando eficazmente as dependências e relações dos aminoácidos dentro das sequências. A hipótese subjacente era que os *embeddings* de modelos de linguagem de proteínas, sendo ricos em informação biológica, poderiam fornecer um suporte mais eficaz para a aprendizagem e melhorar o desempenho na previsão de funções das proteínas. Os resultados demonstraram que a riqueza de informações biológicas capturada por este formato oferece vantagem sobre os métodos tradicionais de codificação posicional, com melhorias substanciais no desempenho durante a fase de teste e uma maior capacidade de generalização para novas sequências ainda não vistas. Dentro dos modelos de linguagem de proteínas, selecionou-se o ProtBERT, um dos codificadores mais avançados na área, para uma avaliação aprofundada das suas variantes de embedding. Esta análise forneceu insights valiosos, orientando a escolha das variantes mais adequadas para aplicações futuras.

No que diz respeito à seleção de modelos, após a análise do desempenho de três modelos distintos (florestas aleatórias, XGBoost e regressão logística), cada um treinado e validado com cinco tipos diferentes de representações, foi explorado o uso de conjuntos heterogéneos (ensemble) compostos por várias combinações de pares modelo-representação. A hipótese subjacente era que, dado que diferentes modelos capturam diferentes limites de decisão e padrões, e diferentes representações capturam diferentes estruturas e propriedades dos dados, um modelo que integrasse várias combinações bem-sucedidas poderia superar o desempenho dos melhores pares individuais. O ensemble contou ainda com diferentes estratégias de votação majoritária, incluindo a atribuição de pesos diferenciados (linear ou exponencial) aos pares modelo-representação que mostraram um desempenho particularmente bom, além de incorporar a possibilidade de remoção de pares que apresentaram um desempenho significativamente inferior. Os resultados mostraram que esta abordagem proporcionou um bom desempenho na classificação e pode ser útil para priorizar sequências para validação experimental, especialmente ao permitir o cálculo de *scores* adicionais de confiança, como o *score* de concordância entre votos. No entanto, não conseguiu superar o desempenho do melhor par individual modelo-representação. Vários factores podem ter contribuído para este resultado, incluindo o sobreajuste (*overfitting*) e a falta de sequências desafiadoras nas fases de treino e validação, a existência de limitações inerentes aos métodos de votação baseados em maioria, e alguma falta de diversidade dos modelos na composição do *ensemble*.

Além da construção do *ensemble*, a análise comparativa do desempenho dos diferentes pares modelo-representação permitiu identificar padrões interessantes nos dados. Em particular, foi possível distinguir um conjunto de sequências, denominadas fáceis, que foram corretamente classificadas por todos os pares, incluindo os de baixo desempenho, e um segundo conjunto, as difíceis, que foram consistentemente mal classificadas, mesmo pelos pares de melhor desempenho. A análise destas sequências revelou que todas as sequências fáceis eram negativas, e a sua facilidade de classificação parece estar relacionada com a violação óbvia de uma assinatura de viabilidade. Esta assinatura indica que sequências viáveis tendem a evitar mutações na região dos aminoácidos 567-576, sendo que, quando ocorrem mutações nesta área, estas são em número moderado e são principalmente substituições. Por outro lado, as sequências difíceis eram todas positivas, e a dificuldade em classificá-las parece residir na complexidade de reconhecer a viabilidade de todas as possíveis substituições. Embora estas sequências respeitassem a assinatura de viabilidade de evitar inseções e delecções na região dos aminoácidos 567-576 e apresentar apenas algumas substituições, o desafio parece estar nas substituições específicas realizadas, já que as diferenças nas propriedades físico-químicas dos aminoácidos significam que nem todas as substituições têm o mesmo impacto.

Em resumo, este trabalho explorou métodos de ponta no design de proteínas orientado por máquinas, com foco nas técnicas de aprendizagem automática, com o objectivo de contribuir para desenvolvimento e bioengenharia de vectores virais. Além de avançar o potencial dos AAVs como plataforma para terapia génica, abre também o caminho para a aplicação mais ampla das tecnologias de aprendizagem automática na bioengenharia de outras partículas virais quer seja para génica quer para vacinação.

Palavras-chave: Aprendizagem Automática; Modelos de Linguagem de Proteínas, Classificação, Vectores Virais, Terapia Génica

Abstract

This work explores advanced machine learning methods for protein design, specifically aimed at enhancing the bioengineering of viral vectors for human gene therapy, with a focus on adeno-associated viruses (AAVs). Despite the success of ML-based protein design in developing new AAV vectors, some challenges remain that were tackled herein, namely in the representation of protein sequences and the selection of optimal models for sequence-to-function prediction.

For protein representation, embeddings generated by protein language models (PLMs) were investigated as a representation format potentially superior to traditional methods like one-hot encoding, since they capture richer biological insights such as amino acid context, patterns, short- and long-distance interactions, and sequence motifs. ProtBERT, a state-of-the art PLM encoder, was selected for an in-depth evaluation of its embedding variants. The results showed ProtBERT embeddings improved performance and generalization on unseen sequences. Furthermore, embeddings capturing individual amino acid information were found to deliver the best results, showcasing the value of biological information for model performance.

For model selection, an ensemble approach combining multiple models trained on different sequence representations was investigated. While the ensemble approach demonstrated strong classification performance, it did not outperform the best individual model. This limitation was attributed to overfitting, as the training data lacked sufficiently challenging sequences, and potentially to the lack of models' diversity in the ensemble composition.

Finally, newly designed sequences were evaluated using the models developed in this study, simulating future steps of this project where generative PLMs will be employed to design new sequences for laboratory production and testing. This work not only advances AAVs as a platform for gene therapy but also contributes to the broader application of machine learning technologies in bioengineering other viral particles for gene delivery and vaccination.

Keywords: Machine Learning; Protein Language Models, Classification, Viral Vectors, Gene Therapy

List of figures

List of tables

List of abbreviations

AdV	Adenoviral vector
ANN	Artificial neural network
ASR	Ancestral sequence reconstruction
BERT	Bidirectional encoder representation from transformers
CNN	Convolutional neural network
DCA	Direct coupling analysis
GAN	Generative adversarial networks
GNN	Graph neural network
GPU	Graphics processing unit
IQR	Interquartile range
L-BFGS	Limited-memory Broyden–Fletcher–Goldfarb–Shanno
MCMC	Markov chain Monte Carlo
ML	Machine learning
NLP	Natural language processing
PAM	Point accepted mutations
PCA	Principal component analysis
PLM	Protein language model
PSSM	Position-specific scoring matrices
RNN	Recurrent neural network
SVD	Singular value decomposition
SVM	Support vector machines
TPU	Tensor processing unit
t-SNE	t-Distributed Stochastic Neighbor Embedding
VAE	Variational auto-encoder

Contents

1. Introduction

1.1. Motivation

Proteins are essential players in biological systems, serving as building blocks of cellular components, catalysts of biochemical reactions, regulators of cellular processes, transporters of molecules, and signal transmitters, among other roles. Given their centrality in health and disease and their value in biotechnology applications, protein design has long been an area of significant interest and active research. However, the nearly infinite design space for even a short-length protein, combined with our limited understanding of the factors determining protein function, have made this a significant challenge (reviewed by Woolfson (2021) [1]). In this context, the introduction of computational methods revealed a real game-changer. Initially, biophysics-based approaches provided significant advancements to the field, and more recently, machine learning (ML) has pushed the boundaries of protein design and engineering to unprecedented levels (reviewed by Ferruz *et al.*, (2023) [2]). In ML-based protein design, protein sequences are used to train models that learn the sequence-to-function fitness landscape, i.e., how different amino acid sequences relate to changes in protein function or properties [3]. These models can then assess the fitness of novel, unseen sequences and guide the engineering of new proteins with user-defined properties. Riding on the latest developments in this arena, the first steps were taken to apply machine-guided protein design to bioengineering therapeutic viral particles, namely, viral vectors for gene therapy [4–11].

Gene therapy is a medical procedure consisting of transferring genetic material into a patient's cells to treat or prevent a disease (https://www.fda.gov/vaccines-blood-biologics/cellular-gene-therapy-products/what-gene-therapy). It has been considered a revolution in 21st-century medicine since it provides treatment for previously unmet medical needs, from genetic disorders to several types of cancer [12]. The vehicles used as genetic material transfer shuttles are generally called vectors, from which those based on recombinant viral particles, called viral vectors, are currently the most used due to the high gene delivery efficiency [12]. Among viral vectors, those based on adeno-associated viruses (AAV) are one of the most popular due to their ability to deliver genetic material with high precision and low risk of immune response, particularly during the first treatment [13]. Additionally, AAV vectors are non-pathogenic, making them safer for clinical applications, can target a broad range of tissues, making them highly versatile for treating several conditions, and can be manufactured with high-yield large-scale processes, facilitating broader clinical application and commercial distribution. Considerable efforts have been made to bioengineer AAV particles with user-desirable properties, such as improved manufacturability, reduced in vivo immunogenicity, targeted tissue tropism, and enhanced gene delivery efficiency (reviewed by Becker *et al.*, (2022) [14]). Many of the latest advancements in this area have been driven by machine-guided protein design [4–8,10,11], and the field is now extending into other viral vectors beyond AAVs [9].

This work explores state-of-the-art methods in machine-guided protein design, specifically focusing on those using ML, to contribute to the evolving landscape of viral vector bioengineering. While advancing the potential of AAVs as a platform for gene therapy, it also paves the way for broader applications of these transformative technologies, including their use to bioengineer other viral particles for gene delivery or vaccination.

1.2. Objectives and research questions

ML has been successfully used to assist in designing and engineering new gene therapy vectors, especially AAVs [4–8,11]. However, notable limitations remain in current approaches, particularly regarding

the representation formats used for protein sequences and the selection of ML models for protein function prediction. The present study proposes to address these challenges.

1.2.1. Objective 1

The nature and format of protein sequence data present unique challenges due to the complexity of the information involved. Each amino acid in a protein sequence brings distinct physicochemical properties, and the function and overall characteristics of the protein result from the interplay between these properties. Furthermore, this interplay is not merely additive: the properties of amino acids can vary depending on their neighbors, leading to emergent characteristics based on the local environment. Additionally, as the protein chain folds into a three-dimensional structure, amino acids interactions can span from short-range to long-range within the molecule. Finally, amino acids form functional units known as motifs, crucial for the protein's final properties and functions. However, most representation formats used in the context of ML-based AAV design have been limited to either indicating the presence of amino acids at specific sequence positions (e.g., one-hot encoding) or, at best, incorporating the biophysical properties associated with those residues.

Inspired by the previous limitation, the **first objective (O1)** of this project was to explore more sophisticated representation formats beyond simple positional encoding of amino acids that capture biologically rich information, including positional dependencies and relationships between amino acids within protein sequences, and evaluate the impact on the learning performance for protein function prediction. Protein language models (PLMs) generate meaningful vectorial representations (embeddings) that encapsulate rich biological information, such as amino acid context, patterns, short- and long-distance interactions, and sequence motifs, effectively capturing amino acid dependencies and relationships within sequences, providing an approach that has not yet been applied in the context of ML-based AAV design and bioengineering. This exploration was guided by the **first research question (RQ1)** on 'How does learning performance differ when using PLM embeddings compared to one-hot encoding representations?'

1.2.2. Objective 2

Most studies on ML-based viral vector design have used a single or a limited set of models for protein function prediction. However, there is no clear evidence that any specific ML model consistently outperforms others, as performance tends to vary depending on factors such as the protein type, the localization, and extent of targeted sequence variations, the specific function(s) or property(ies) being engineered, or other aspects and characteristics of the data. Overall, relying on a single model or a small set of models undermines confidence in the final predictions, which is critical given the significant time and resources required for the experimental validation of newly designed sequences.

Motivated by the limitations outlined above, the second objective (O2) of this project was to develop an approach to enhance prediction confidence and aid in prioritizing sequences for laboratory evaluation. To that end, we propose to employ an ensemble of models, leveraging the strengths and operation mode diversity of several algorithms while compensating for the weaknesses of individual models. This approach, which has not yet been applied in ML-based AAV design, was explored in this study. Moreover, building on the work developed in O1, the ensemble incorporates models trained with OHE and PLM embeddings. This way, the information provided by both representation types can be combined, avoiding choosing one over the others. This second objective was guided by the **second research question (RQ2)** on 'How can an ensemble approach, which combines multiple models trained on different data representations, improve the prioritization of sequences for experimental validation?'.

1.3. Contributions

The work of this thesis introduces two innovative contributions. First, it pioneers the application of PLM embeddings for machine-guided protein design in the context of viral vector bioengineering. While this approach had not been utilized before this thesis commencement, it has since then been employed and described by Lyu *et al.* (2024) [9] for adenoviral vectors (AdVs). However, applying this method to AAV vectors remains a novel contribution to this work. Additionally, this work provides a comprehensive and systematic analysis of the performance of different ProtBERT embedding variants, offering a structured evaluation of one of the most widely used PLMs for protein embedding generation and that can guide the choice of variants for future applications. Second, it proposes a new approach of building an ensemble of different ML models, each trained on different data representation methods, to integrate both classical positional encoding and advanced embedding techniques to improve classification performance and confidence.

1.4. Thesis structure

This thesis is organized into four sections. Following this introductory chapter, which outlines the context and motivation behind the research, the second section, "Related Work," reviews pertinent prior studies and explains how they relate to the current work. Next, the "Methodology" section details the primary methods and experimental approaches used in this research, followed by the "Results and Discussion" section, where the main findings are presented and analyzed. Finally, the "Conclusion and Perspectives" section summarizes the key outcomes of this thesis and provides insights into potential directions for future research.

2. Related work

2.1. Types of machine learning and ensembles

A traditional distinction in ML is the division between unsupervised and supervised learning. In supervised learning, a model is trained on a labeled dataset, where each input is associated with a corresponding output [15]. The algorithm learns to map input features to their correct targets by minimizing error through iterative adjustments, enabling it to make predictions on new, unseen data. Within supervised learning, a further distinction is often made between regression and classification. In regression continuous-valued targets are predicted, while classification is about predicting discrete labels or categories. This work primarily focuses on supervised learning, specifically binary classification, since the models developed here classify AAV sequences as viable or non-viable.

In unsupervised learning, the model is trained on data without predefined labels or target variables. Instead of making predictions based on known outcomes, these algorithms are used to uncover the underlying organization of the data, cluster similar data points, or reduce data dimensionality [15]. Visualizing the outcomes of these processes is particularly useful during the initial stages of data exploration, which is how it was used in this work, since it helps uncover hidden patterns, relationships, and trends that can provide valuable insights for further analysis and guide subsequent data-driven decisions.

A third category, self-supervised learning, is also considered, where models are trained on data with labels automatically derived from the data itself, eliminating the need for manual annotation [16]. This approach leverages inherent structures or patterns in the data, using tasks such as predicting missing parts of an input or reconstructing corrupted data. This category has gained significant attention recently due to its ability to harness large volumes of unlabeled data. None of the models in this work used self-supervised learning, although it is a common approach, especially in language models.

One powerful technique often employed within supervised learning is the use of ensembles. An ensemble combines the predictions of multiple models to produce a more robust and accurate result. By leveraging the strengths of different models and reducing overfitting or bias, ensembles can significantly improve performance [17]. Although ensembles are typically constructed using models of the same type, the concept can also be applied to mixes of different models, an approach known as heterogeneous ensemble learning [17]. Different models operate in distinct ways, using different algorithms and strategies to learn from data, which means each can capture different aspects of the problem. For example, decision trees capture hierarchical relationships, while neural networks can identify complex, non-linear patterns. By aggregating predictions from models with different operational modes, heterogeneous ensembles harness this diversity to better address the complexities of a given task. This is especially valuable when it is unclear which model will perform best for a particular problem or dataset, as is often the case in protein classification tasks. Therefore, this work explores the implementation of heterogeneous ensembles incorporating models of random forests, XGBoost, and logistic regression classifiers to raise confidence in classification predictions. Each model used in this work is briefly described in the following subsections.

2.1.1. Random forests

Random Forests are ensembles of decision trees, a popular ML algorithm that recursively splits the data into subsets based on feature values [18]. At each split, the decision tree algorithm selects the feature and threshold that best separates the data according to a criterion like Gini impurity (for classification) or mean squared error (for regression). The process continues, with the data being divided into smaller, more homogenous groups, until a stopping condition is met, such as reaching a maximum tree depth or having a subset that cannot be split further. Predictions are made by following the path from the root to a leaf node,

where the final prediction is assigned based on the majority class (classification) or the average target value (regression) within that leaf. Random forests were a first choice for our model ensemble due to their ability to capture hierarchical relationships between features. This is a desirable aspect in protein sequence classification because proteins inherently have hierarchical structures, such as motifs, domains, and functional sites, that contribute to their biological roles.

2.1.2. XG-BOOST

Extreme Gradient Boosting (XGBoost) is an ML algorithm developed by Chen and Guestrin (2016) [19] based on boosting. Like random forests, boosting is also an ensemble learning technique, i.e., it builds a predictive model by combining multiple weak learners (typically, decision trees). However, boosting builds trees sequentially instead of training all trees independently (bagging) as in random forests. Each new tree is trained to correct the errors made by the previous, creating a model that improves over time. In XGBoost, this process, known as Gradient tree boosting [20], is enhanced by using gradient descent to optimize the model. XGBoost introduces additional key features such as regularization, which prevents overfitting by penalizing overly complex trees, and, in later implementations, histogram-based splitting, which reduces computational complexity with an efficient method of constructing trees.

Adding XGBoost to an ensemble that already includes random forest can be beneficial. Random forests reduce variance by training independent trees on random subsets of data, making it robust to noise and overfitting. However, it may miss more complex patterns. XGBoost, on the other hand, builds trees sequentially to correct errors, improving bias and variance, and its iterative process enables the capture of more complex, subtle patterns in the data.

2.1.3. Logistic regression

Logistic regression is an ML algorithm widely used for binary classification tasks, where the goal is to model the probability that a given input belongs to one of two classes [21]. It can also be extended to multi-class classification problems using techniques such as one-vs-rest. The algorithm fits a linear equation to the input features, transforming the output using the logistic (sigmoid) function, which maps values to a range between 0 and 1. The resulting value represents the estimated probability of the positive class. Logistic regression optimizes the coefficients of the input features by minimizing a loss function, typically log-loss or cross-entropy, using gradient descent methods. Logistic regression was a third model choice since it explicitly models linear and additive relationships between features and the log-odds of the outcome, which tree-based models do not. While random forests and XGBoost capture complex interactions, they may struggle when the relationship is globally linear or monotonic. Logistic regression is also more stable with collinear features, whereas tree models may produce unstable splits. Thus, it is a good complement to tree-based models to add to the ensemble.

2.2. Machine-guided protein

Machine-guided approaches for protein design and engineering can be divided into four main categories based on the type of information used: i) sequence-based tools that perform multiple sequence alignment of homologs of the query protein to predict the effects of amino acid changes, ii) structure (only)-based tools, that use the 3D structure of the protein to predict the effect of amino acid changes in its biophysical properties including interatomic contacts, surface features, and substrate interactions, iii) molecular dynamics simulations, that use the 3D structure of the protein and physics laws to predict the spatial position of each atom as a function of time within a biological system and iv) data-driven ML methods, which leverage large datasets of protein sequences, structural information, experimental measurements, and simulation results to identify patterns and relationships that can predict protein behavior and properties. Hybrid strategies that combine multiple approaches are also becoming popular, such as informed ML

discussed in detail by von Rueden *et al.* (2021) [22]. These approaches may prove especially useful when data is scarce, allowing for the creation of better models despite limited training resources. Among these, physics-informed ML has been increasingly used in protein design [23], leveraging known physical properties like protein structure or molecular dynamics and interactions to guide the learning process.

In the context of this work, we focus on data-driven ML methods. A range of ML techniques have been employed to guide protein design and engineering, with this thesis specifically concentrating on supervised learning, namely discriminative and generative models. Although unsupervised learning has also been widely applied in the field, it has been excluded from this background revision for the sake of simplicity.

2.2.1. Discriminative models

Discriminative models learn the relationship between input features (X) and the output (Y) by estimating the conditional probability P(Y|X) [24]. In other words, based on the input X, they are trained to predict the output Y, whether a label or category (for classification) or a continuous value (for regression). In the case of protein design, the input consists of amino acid sequences, which may be supplemented with additional data such as evolutionary relationships, physicochemical properties, or structural information. The output is the protein's function or property, which can either be the assignment to a given label/category (e.g., "active" vs. "inactive") or the prediction of a continuous value for that protein (e.g., binding affinity or enzyme activity). Early machine learning applications in protein design and engineering primarily used discriminative models. Some examples include the use of random forests to predict protein solubility [25], support vector machines [26,27] and decision trees to assess changes in enzyme stability following mutations [28], and K-nearest-neighbor classifiers to identify enzyme functions [29] and mechanisms [30].

Compared to generative approaches, discriminative models: i) are often more straightforward and require less computational resources, which can be particularly advantageous when working with large-scale protein sequence datasets, ii) focus on learning discriminative features that distinguish between classes, for example, structural characteristics or functional domains that are specific to particular class, which is helpful to add biological interpretability and explainability, iii) are often more robust to noise in the data by learning discriminative features, which can help mitigate the impact of protein sequencing errors or other sources of noise in the data. Furthermore, a good discriminative model can function as a filter to evaluate sequences designed by generative AI, saving time and resources in laboratory validation.

Within discriminative models, classifiers are particularly relevant to protein design. Protein classification enables the categorization of a protein sequence into a set of classes that represent key functions or properties, ranging from binary classification (e.g., active' vs. 'inactive') to multiclass classification (e.g., 'healthy', 'pre-malignancy diseased', and 'malignancy'). From a practical standpoint, classification is often preferred over regression because functional labels implicitly encapsulate optimal continuous-valued outputs of interest. For example, a functional antibody is characterized by specific thresholds for properties such as affinity, avidity, stability, and solubility. Thus, instead of individually targeting these properties with regression approaches, mainly when optimal values are unknown, classification simplifies the process by learning categories that inherently capture the necessary continuous-valued outputs for the desired functionality. Given these advantages, this work focused on classification. Specifically, the learning models developed herein are binary classifiers designed to categorize protein sequences as 'viable' or 'non-viable' based on their ability to produce functional AAV vectors.

2.2.2 Generative models

Generative models learn the joint probability distribution P(X, Y), capturing both the relationship between input features (X) and output (Y) and the underlying data generation process [24]. They learn the

conditional probability P(X|Y), modeling how inputs (X) are generated given specific outputs (Y). This enables generative models to propose plausible inputs (X) tailored to particular outputs (Y). In protein design, generative models can create novel amino acid sequences optimized for a desired function or property (e.g., high binding affinity or enhanced stability).

Generative models emerged more recently in protein design compared to discriminative models. First, learning the joint distribution P(X|Y) is computationally more challenging than the conditional distribution P(Y|X) used in discriminative models. This required advancements in both the models and computational infrastructure. Sophisticated deep learning architectures, such as generative adversarial networks (GANs), variational autoencoders (VAEs), and transformers, have only recently been developed or reached the level of sophistication required for protein design. On the computational infrastructure side, high-performance graphics processing units (GPUs) and tensor processing units (TPUs) have only become widely available in the past decade, further enabling these models. Moreover, discriminative models benefitted from the availability of mid-size labeled datasets in protein science, such as enzyme activity or binding affinity data, which have been around for years. In contrast, generative models require large volumes of high-quality large-scale data to learn patterns and generate meaningful outputs effectively. The collection, curation, and expansion of protein sequence-function datasets have only seen significant progress in recent years with advances in sequencing technologies and high-throughput functional assays [31].

Many generative models have been used for protein design, including models based on deep neural networks, such as GANs [32–34], VAEs [35–40], recurrent neural networks (RNNs) [41,42], language models [43–48], graph neural networks (GNNs) [49], other types of neural networks [50,51], normalizing flows [52], diffusion models [53–59], energy-based models (EBMs), as well as statistical methods such as ancestral sequence reconstruction (ASR) [60,61] and direct coupling analysis (DCA) [62–64]. For a review of the subject refer to Strokach and Kim (2022) [65] and Winnifrith *et al.* (2024) [66].

Generative models are now considered state-of-the-art for protein design due to their ability to generate novel protein sequences. However, their implementation can be challenging, requiring significantly more computational resources and training time than discriminative models. Additionally, generative models can struggle with ensuring the biological validity of the sequences they generate since they focus on learning the underlying distribution of data rather than directly optimizing for specific functional properties. This can result in generating sequences that, while plausible given the statistics learned, may not exhibit the desired functionality. In this study, generative models were not implemented for AAV design. However, this area has been actively explored within our research group and was investigated in parallel by Ferraz (2025) [67]. In that work, ProGen [43] was used. ProGen is a decoder-based PLM that employs autoregression to generate amino acid sequences, one token at a time, conditioning each token on the previous ones. It was trained on millions of protein sequences from several organisms across all domains of life using a masked language modeling approach, incorporating both taxonomical and functional metadata in the form of tags preceding the sequences. These functional tags make ProGen sensitive to specific prompts, enabling it to generate protein sequences with desired characteristics. In Ferraz (2025) [67], ProGen was fine-tuned on the same dataset used here, with additional functional tags corresponding to the AAV2 taxonomy. This fine-tuning aimed to enable the model to generate AAV sequences. A total of 100 sequences generated by the AAV-specific ProGen model were then evaluated in the final stage of this study.

2.3. Markov chain Monte Carlo processes for protein sequence generation

Several approaches can be used to generate new protein sequences. In its simplest form, even a random mutation strategy of introducing a specified number of mutations of any type (insertion,

deletion, or substitution) at any position can be algorithmically defined and serve as a sequence generator. What distinguishes these simpler approaches from those described in the previous section is the existence of a model that guides the generation process. This model is learned from data, thereby biasing the generation process towards sequences that incorporate complex patterns and dependencies features while also adhering to the structural and evolutionary constraints present in the data. However, these models are more likely to generate sequences that, although innovative and biologically plausible, may fall outside the desired functional fitness depending on the model's training and parameters. Therefore, in this work, we sought to implement a simpler generator to design new sequences to complement those generated by the PLM-based approach implemented by Ferraz (2025) [67].

A mid-term approach between complete randomness and being guided by a probability distribution are simpler stochastic methods, which balance some structure with a degree of randomness. These methods still involve probabilistic generation but offer a more straightforward implementation than complex generative AI models, requiring fewer resources and less computational complexity. One example of such a method is Markov chain Monte Carlo (MCMC), used in this work.

An MCMC process to generate new protein sequences begins by analyzing a pool of sequences that meet a specific desired property (e.g., all are viable). From this pool, the underlying distribution of sequence features is derived, which serves as the basis for a likelihood function. In the context of protein sequences, this likelihood function could be based on simple features such as amino acid frequencies or more sophisticated models such as substitution matrices [68], hidden Markov models [69], or other probabilistic frameworks that capture sequence conservation, evolutionary relationships, or functional constraints. The more effectively the selected features capture patterns associated with functionality, the greater the chance that the generated sequences will mirror the properties of the original set and exhibit functional relevance. Next, new variants are generated by applying mutations to an initial sequence, typically the wild type or a functional starting point. The resulting sequence is then analyzed to determine whether it falls within the distribution of the original pool and to decide if it will be the starting point for the next round of mutations. Sequences that fall within the distribution are more likely to be accepted but can still be rejected, while those outside the distribution are generally rejected, although acceptance is possible. Rejecting potentially functional sequences is not problematic since the process reverts to a previous state, i.e., closer to the functional starting point. Accepting sequences that deviate from the distribution increases diversity, aiding exploration of broader sequence space and helping avoid being stuck in local optima. While an underlying distribution guides the process, there is randomness in mutation parameters and acceptance/rejection decisions. This blends exploration and exploitation and drives the process toward sequences that balance novelty and functionality. The MCMC process also allows the control of sequence diversity by adjusting parameters such as the number of mutations per step and the total number of steps in the chain.

2.4. Representation of protein sequences for machine learning

Most ML methods operate exclusively on numerical data. This implies that non-numeric data, namely protein sequences, needs to be transformed into a numerical representation. The type of representation used directly impacts the performance of ML due to several aspects, from computational efficiency to the ability of that representation to capture information relevant to the tasks being learned. Protein sequences can be represented using different formats, each capturing various types of information, as recently reviewed by Harding-Larsen *et al.* (2024) [70]. A brief summary is provided in Table 2.1 and discussed next.

Table 2.1. Overview of the main formats used to represent protein data in machine learning

Format	Key information	Level of biological informativeness
Basic representations • One hot encoding • Integer encoding • K-mer encoding	Symbols only Positional referencing of amino acids or amino acids motifs No biological relationships	Low
Biophysical representations • Physicochemical properties • PSSM • PAM substitution matrices	Bio-physiochemical and evolutionary information	Moderate
Distributed representations • Word embeddings • PLM embeddings • Other types of embeddings	Contextual, evolutionary, and functional information	High
Structural representations • Secondary structure prediction • 3D structure prediction • Contact maps	Structure information	Very high
Graph-based representations	Structure, sequence, and relations	Extremely high

PAM: Point accepted mutations; PLM: Protein language model; PSSM: Position-specific scoring matrices

Basic representations like one-hot encoding (OHE), integer encoding, and K-mers are commonly used to convert protein sequences into numerical formats. However, these methods are biologically limited as they treat amino acids or subsequences as independent symbols, ignoring evolutionary, biochemical, and structural relationships between them. For instance, OHE encoding fails to capture similarities between amino acids, such as hydrophobicity or size, while integer encoding assigns unique integers to each, missing contextual information. K-mers identify short motifs but lack broader context, overlooking long-range dependencies. Despite these limitations, these representations are computationally efficient, easy to implement, and provide straightforward transformations of protein sequences.

Biophysical representations are more informative than basic representations. The encoding of physicochemical properties incorporates characteristics of amino acids, like size, hydrophobicity, and charge, which offer valuable insights into protein folding, stability, and interactions. Position matrices are biologically relevant since they capture evolutionary constraints, highlighting which amino acid substitutions are more likely at each position based on sequence conservation, offering deeper insights into protein function and evolution. However, these representation types focus on local properties, limiting their ability to capture the broader context of protein structure and amino acid interactions.

Distributed representation formats, such as word embeddings and PLM embeddings, are highly informative because they are learned from large-scale protein databases using deep learning [71]. PLM embeddings, in particular, capture contextual relationships, evolutionary information, and residue interactions. This enables them to implicitly encode sequence-level properties, such as function, structure, and conservation, without explicit feature engineering. As main limitations, we find the need for substantial computational resources especially for embedding generation and model training. Additionally, they may still struggle with rare or novel protein sequences that fall outside the scope of the training data, potentially leading to less accurate predictions in these cases.

Structural representations, such as secondary structure descriptions, 3D coordinates, and contact maps, are highly biologically informative because protein structure is a key determinant of function. These representations are directly linked to functional properties, including binding sites, enzymatic activity, and

stability, making them essential for structure-based tasks like protein-ligand docking, interaction prediction, and structure classification. However, these representations are often challenging to obtain, as they require experimental data (such as X-ray crystallography or cryo-EM) or complex computational methods, which can be time-consuming and resource-intensive. Additionally, structural data may not be available for all proteins, particularly those that are difficult to crystallize, or it may lack sufficient resolution. There is an added layer of uncertainty for structures modeled using computational methods. For example, despite significant advancements like AlphaFold2, which predicts protein structures from sequences [72], the generated structures are not always accurate, especially for challenging targets such as membrane proteins.

Finally, graph-based representations model proteins as graphs, where nodes represent amino acids or protein atoms, and edges capture spatial, sequential, or chemical relationships. These are extremely biologically informative since they can combine sequence and structural information and can incorporate evolutionary or functional annotation. They are useful in tasks like protein-protein interaction prediction, structure-based drug design, and functional site identification. However, graph-based representations can be computationally demanding, especially when dealing with large proteins or high-resolution structural data. Additionally, constructing accurate graphs requires detailed structural information, which may not always be readily available, particularly for proteins that lack experimentally determined structures.

Among the formats described above, embeddings offer an optimal balance between capturing rich biological information and maintaining acceptable computational efficiency. Their use also brings innovation to this work since PLM embeddings have never been used in ML-based AAV design. OHE format served as a comparison, providing a simple, computationally efficient representation.

2.5. PLM embeddings and ProtBERT

An embedding is a continuous vector-based representation of an entity generated by deep learning models. Each vector element captures different aspects, concepts, or properties of the entity it represents, learned automatically during model training [73]. This allows embeddings to encode rich, high-level information beyond raw data without explicit user intervention. Also, each dimension of the embedding corresponds to the same concept across all entities [73], for example, if one dimension represents ‘hydrophobicity’ in proteins, it will represent hydrophobicity consistently across all protein sequences, with that specific value varying by protein. These values serve as quantitative scores for the property/aspect they capture, making embeddings useful for calculating similarity between entities.

Embeddings generated with transformer-based models have the added value of incorporating attention mechanisms. Attention allows the model to dynamically focus on different parts of the input sequence when generating the embedding, depending on the context. In the case of protein sequences, for example, attention mechanisms enable the model to focus on specific amino acids or regions of the sequence that are critical for understanding the protein's structure or function. This allows the model to capture complex, long-range dependencies within the sequence. As a result, the protein embeddings generated by models with attention (e.g., ProtBERT [74], ESM [75] or TAPE [76]) are more context-sensitive and capture relevant relationships between different parts of the amino acid sequence.

Among transformer-based protein language models, ProtBERT has gained popularity due to… chosen due to its strong record of success on protein-related tasks, proven performance across benchmarks, and ability to generate context-sensitive embeddings that capture long-range dependencies without the need for evolutionary data.

ProtBERT is based on Google’s BERT [77] which stands for Bidirectional Encoder Representation from Transformers. BERT is an encoder model, that is, a deep neural network designed to generate vector

representations of input tokens based on the encoder architecture of the Transformers introduced by Vaswani *et al.* (2017) [78]. It was trained on large-scale text corpora using a masked language modeling objective and, because it leverages bidirectional self-attention to capture contextual dependencies in both left-to-right and right-to-left directions of the input sequence, it produces contextualized representations that enhance the understanding of token relationships within the sequence. ProtBERT extends the capabilities of BERT by adapting it to model protein sequences instead of natural language. ProtBERT is pre-trained on massive datasets of proteins, allowing it to learn biologically meaningful representations of amino acids and their contextual relationships. Consequently, while BERT captures linguistic semantics and syntactic structures, making it highly effective for tasks such as text classification, named entity recognition and question answering, ProtBERT captures biochemical and structural properties of proteins, making it well-suited for tasks like protein classification, function prediction, and secondary structure identification.

Three types of embeddings can be directly generated with ProtBERT (Figure 2.1): i) cls_raw_EMB, ii) cls_t_EMB, and iii) aa_EMB.

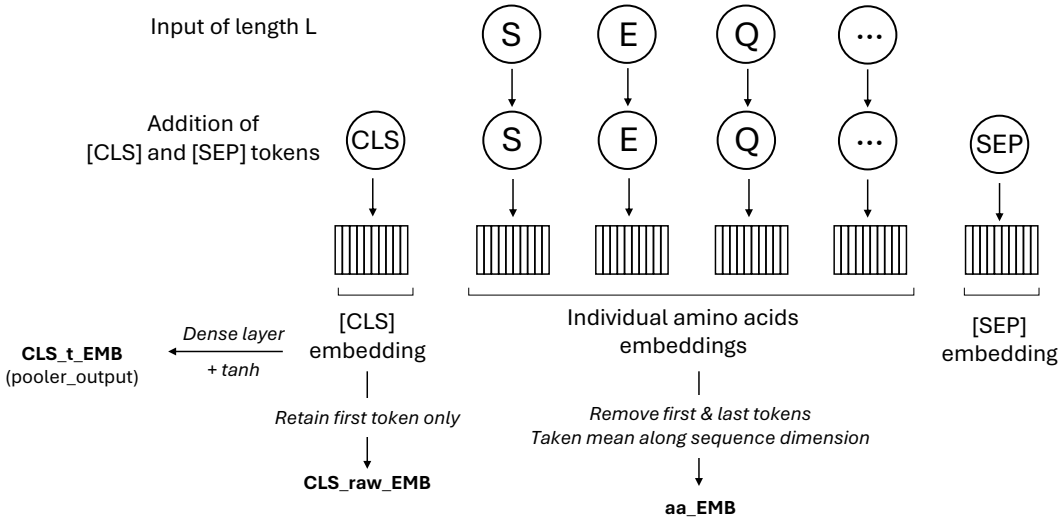

Figure 2.1. Embedding generation with ProtBERT. A protein sequence, such as 'SEQ', is tokenized, i.e., each amino acid is converted into a token. Two additional tokens, CLS and SEP, are incorporated. Each token is then passed through the model to generate 1024-dimensional context-aware embeddings extracted from the last hidden layer of the model. These embeddings can be of three types: i) cls_raw_EMB, which represents global sequence information captured by the CLS token; ii) cls_t_EMB, which is the CLS token after processing through a dense layer and a tanh activation function; and iii) aa_EMB, which is the global average of individual amino acids embeddings, excluding the CLS and the SEP tokens.

The cls_raw_EMB represents the embedding of the CLS (classification) token, a special token added to the input sequence to capture the overall context and serve as a summary representation of that sequence. Because it aggregates information from the entire sequence, the CLS embedding is traditionally used in natural language processing (NLP) in tasks such as classification and fine-tuning. A similar use can be considered for proteins. However, its use in tasks requiring more detailed, sequence-specific information, such as structural prediction or sequence similarity analysis, may have limitations. The cls_t_EMB is the CLS token embedding transformed after processing through a dense layer and a tanh activation function, which in NLP provides the advantage over the raw CLS token of capturing more complex, nonlinear relationships, refining the sequence representation, and making it better suited for some applications like sentiment analysis. However, its effectiveness for protein sequences is uncertain since the utility of this

transformation may not translate directly from sentences to protein sequences. Finally, the aa_EMB represents the global average of individual amino acid embeddings, excluding the CLS and SEP tokens. Its equivalent is not typically used in NLP because it would be equivalent to averaging word embeddings to represent a sentence, disregarding crucial contextual information and word order. However, in the context of proteins, it is biologically meaningful because it captures the properties of all amino acids, providing a representation of the sequence very rich in biological information. It is also the suggested format for supervised learning in protein-level classification in the original ProtBERT publication [74]. Removing [CLS] and [SEP] ensures that only essential sequence-level information is preserved.

The suitability of each embedding type as a representation format for protein sequences described above is largely empirical and depends on the parallels that can be drawn between natural language and proteins, but no systematic comparison of these three formats or their potential combinations has yet been conducted. This work provides such a comparative analysis.

2.6. Machine-guided design of viral vectors

Machine-guided protein design has been used to bioengineer recombinant viral particles, particularly viral vectors used in gene therapy. Table 2.2 summarizes key publications on machine-guided design of viral vectors for gene therapy, which is next discussed in terms of: i) targeted vector, ii) properties to bioengineer, iii) computational approaches employed, and iv) method to represent the protein sequence.

Table 2.2. Previous studies on machine-guided protein design for viral vector bioengineering

Reference	Vector	Protein	Property[a]	Computational models	ML task	Protein representation
Ogden *et al.* (2019) [4]	AAV 2	Capsid	Viability Thermal stability Neutralization resistance Several tissues tropism	PSSM-based sampling	-	-
Bryant *et al.* (2021) [6]	AAV 2	Capsid	Viability Thermal stability Hepatotropism	Logistic regression CNN RNN	Sequence classification	One hot encoding
Marques *et al.* (2021) [5]	AAV 2	Capsid	Viability	s-ANN SVM	Sequence classification	One hot encoding aa properties encoding[b]
Mikos *et al.* (2021) [8]	AAV 2	Capsid	Manufacturability	Random forest	Sequence classification	aa microenvironment encoding
Sinai *et al.* (2021) [79]	AAV 2	Capsid	Viability	Variational auto-encoder	Sequence generation	One hot encoding
Han *et al.* (2023) [10]	AAV 2	Capsid	Transduction efficiency	CNN Transformer-based model	Sequence classification	Embedding matrix
Lyu *et al.* (2024) [9]	AdV 5	Hexon	Viability	Variational auto-encoder	Sequence generation	ProtBERT embeddings
Vu Hong *et al.* (2024) [11]	AAV 9	Exogenous peptide	Myotropism	AlphaFold Rosetta	-	-

a. Desirable property/function to be bioengineered. b. Encoding of amino acid (aa) physicochemical properties. AAV2/9: adeno-associated vector type 2 or type 9, CNN: convolutional neural network; PSSM: position-specific scoring matrices; RNN: recurrent neural network; s-ANN: artificial neural network; SVM: support vector machine.

As targeted vectors, AAV vectors have been the preferred choice in most studies conducted so far, except that by Lyu *et al.* (2024) [9]. This preference relates to the widespread interest of AAVs in current gene

therapy and their relatively simple composition since they consist of a single protein type (the capsid). This makes the bioengineering challenge more manageable and benefits from the extensive knowledge gained from numerous studies on single-protein design using machine-guided approaches. Even in the case of the Lyu *et al.* (2024) [9] study, which targets adenoviral vectors (AdVs) composed of multiple protein types, the focus was limited to a single protein within the vector, the capsid hexon. Overall, the design and bioengineering of multi-protein complexes, which include most viral particles and all gene therapy vectors beyond AAVs, remain largely unexplored.

Considering properties for bioengineering, viability stands out as the most targeted attribute. Viability refers to the ability of the protein to enable the assembly of a viral particle capable of delivering its genomic content to target cells. Therefore, viability is not only desirable but essential. Even in cases where other properties are the primary focus, viability remains an implicit property to target. Other properties actively pursued include tissue specificity (tropism), enhanced manufacturability, improved delivery efficiency, and a range of beneficial characteristics, such as thermal resistance and resistance to antibody neutralization.

Based on the computational approaches used, the studies listed in Table 1 can be divided into two groups: those that do not employ ML ([4] and [11]) and those that do (all the others). The study by Hong *et al.* (2024) [11] used molecular simulation and docking studies. This study also differs from the others in that the bioengineering target was not a protein from the vector but an exogenous peptide added to the vector to enhance its affinity toward muscle tissue. On the other hand, the study conducted by Ogden *et al.* (2019) [4] also did not use ML but, like the other studies, targeted a protein from the vector. This study focused on experimentally mapping the first-order fitness landscape of the AAV2 capsid protein by evaluating every possible amino acid substitution at each of the 735 positions while keeping the other 734 positions unchanged. The resulting single-mutant dataset for viability was used to generate a position-specific scoring matrix (PSSM) for the AAV2 capsid, which was then used to guide the design of multi-mutants by a (non-ML) additive sampling model. Regarding the ML-based studies from Table 1, these can be divided into sequence classification using discriminative models [5,6,8,10] or sequence generation using generative models [7,9]. Within sequence classification, a wide range of models has been employed, including logistic regression, random forests, support vector machines, and various network-based models such as shallow artificial neural networks (s-ANN), convolutional neural networks (CNN), recurrent neural networks (RNN), and transformers. While the model landscape is diverse, most studies use individual models. This presents as an opportunity to enhance prediction confidence by exploring an ensemble-based strategy that leverages a broader range of models, an innovation brought by this work. In addition, the dominance of classification models highlights the potential for innovation by exploring more advanced generative techniques, such as autoregressive generative PLMs. Our group is pursuing this line of research for AAV design, although not in the scope of this thesis work.

Finally, we find that OHE has been prevalent concerning the method used for protein sequence representation, which is relevant in the case of ML-based approaches. Additional techniques, most notably encoding amino acid properties, have also been used. This highlights the opportunity to explore novel representation formats in ML-based AAV design, namely, state-of-the-art PLM embeddings, which is another innovation introduced by this work.

3. Methodology

This work employed a comprehensive methodological approach to explore ML-based design of the AAV2 capsid protein, aiming to bioengineer new variants with user-defined properties, herein, viability (Figure 3.1).

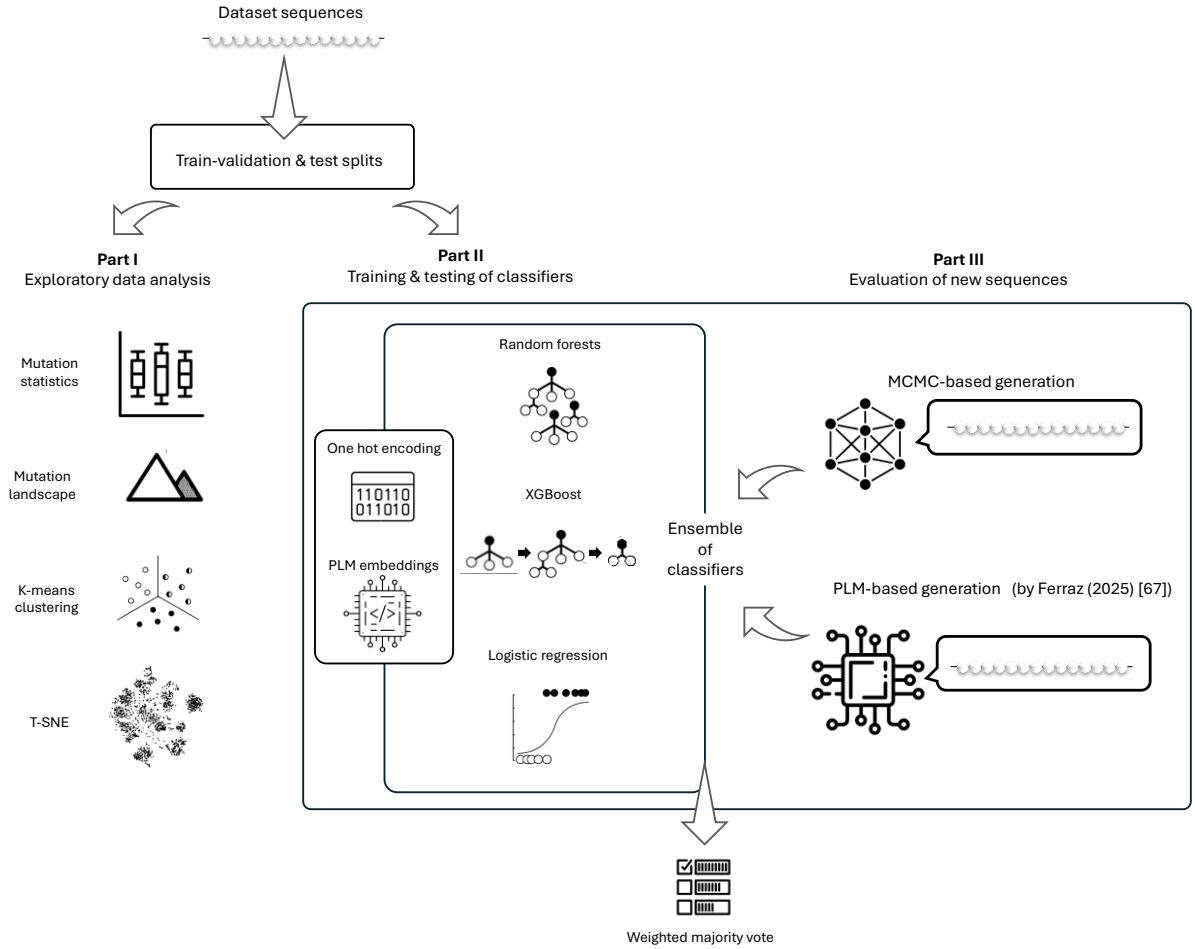

Figure 3.1. Overview of workflow and methodological approaches used in this work. The dataset was first analyzed to define a splitting strategy into a training-validation set and independent test sets. Each set was assessed for mutation statistics and mutation landscape characteristics. The training-validation set underwent further exploratory analysis using K-means clustering and t-SNE to uncover patterns within the data. In the next phase, three models were trained for the binary classification of AAV2 capsid viability using the training-validation set and evaluated on the test sets. Each model was tested with five different representation formats: OHE or one of four PLM embeddings generated with ProtBERT, allowing for a systematic comparison of representation performance. From the best-performing pairs, a heterogenous ensemble model was built where the final output is determined by a weighted majority voting, accompanied by an agreement score reflecting the consensus strength across the pairs. In the final stage, new sequences were generated using either an MCMC approach or a generative PLM fine-tuned with viable AAV sequences (developed by Ferraz (2025) [67]) and analyzed by the best-performing models.

3.1. Dataset and data pre-processing

The data used in this work is part of a dataset published by Bryant *et al.* (2021) [6], which reports a comprehensive study for machine-guided AAV2 capsid diversification. The set consists of 296968 AAV2 capsid protein variants, both viable (153691) and non-viable (143278), all of which have been produced and

experimentally validated in the laboratory. The original dataset, provided as Supplementary Data 1 with the publication, is available for download at https://www.nature.com/articles/s41587-020-00793-4. This file contains details of the targeted fragment and the modifications introduced in each variant. For this project, the data in this file was processed to reconstruct the complete capsid protein sequences, incorporating the respective mutated fragments. This processing included: i) joining of the fragments with the pre- and post-fragment sequence to reconstruct the entire protein and capitalizing all amino acid letters, since in the original file, amino acids that resulted from insertion or substitutions were represented in lowercase, and this was found to complicate further processing, namely the encoding phase, and ii) adding a unique self-incremental identifier to each sequence based on its subset.

After pre-processing for whole-sequence reconstitution, the dataset was examined for duplicates, which were then removed. The removal process involved discarding three complete subsets: 'previous_chip_viable' and 'previous_chip_nonviable,' since most of their sequences were present in other subsets, and the 'singles' subset, since all its sequences were duplicated in the 'single' subset (more details on subset classifications and design strategies in section 4.1). Additionally, some sequences from the 'random_doubles' subset were eliminated because they were duplicated in the 'designed' or 'rand' subsets. In total, 3135 sequences were removed, resulting in a final dataset of 293835 sequences.

3.2. Mutation statistics summary and mutation landscape analysis

Each sequence in the dataset was analyzed against the reference sequence (https://www.ncbi.nlm.nih.gov/protein/P03135.2) for substitutions, deletions, and insertions using the *pairwise2* method of Biopython. This method performs pairwise sequence alignment, allowing for customization of penalties for matches, mismatches, and gaps. The default scoring parameters were used (match: +1 point, mismatch: 0 points, gap: 0 points). While these can be adjusted if the alignment score is used for downstream analysis, they were irrelevant here since the focus was solely on assessing the number and nature of mutations per sequence.

The number and nature of mutations of chosen data subsets were analyzed using custom-made functions to: (i) perform mutation landscape analysis by identifying the frequency of deletions, substitutions, and insertions at each amino acid position; and (ii) calculate the average number and types of mutations per sequence, which were visualized using box plots.

3.3. Sequence encoding

The primary data for this project consists of amino acid sequences represented as letter strings. Since this format is incompatible with most ML methods, the sequences were converted into numerical representations (encodings) that can be processed computationally. This work employed two types of encoding: OHE and ProtBERT encodings, from which four variants were used. Depending on the mutations' type and combinations, some sequences were shorter or longer than the original sequence, requiring padding for OHE data.

3.3.1. One hot encoding

For OHE a custom function was developed and applied to each sequence in the datasets. This function executes the following steps: (i) converts the sequences into lists of individual amino acid characters, (ii) pads the sequences with the token 'X' (a symbol not representing any amino acid) up to a maximum length, which in this case was the longest sequence found across the entire dataset (750 amino acids), (iii) defines the order of amino acids for the binary vector (ACDEFGHIKLMNPQRSTVWY, along with 'X'), and (iv)

iterates over each amino acid in each sequence to generate its corresponding binary vector, appending these vectors sequentially.

3.3.2. ProtBERT encoding

For ProtBERT encoding, the sequences were pre-processed to align with the method's requirements (https://huggingface.co/Rostlab/prot_bert) as follows: (i) all characters were converted to uppercase, and non-standard or ambiguous amino acids were replaced with an 'X' (although such amino acids were not present in the datasets, this step was included to adhere to best practices), (ii) the [CLS] token was added at the beginning and the [SEP] token at the end of each sequence, and (iii) a space was inserted between each letter, as well as after the [CLS] token and before the [SEP] token.

After the sequence pre-processing, embeddings were generated using the BertModel and BertTokenizer from the *transformers* library to leverage the BERT architecture. The *transformers* library was developed by Hugging Face [80], and provides a broad suite of pre-trained models and tokenizers to use in NLP-related tasks, and operates on top of PyTorch [81], a deep learning framework known for its flexibility and efficient tensor computations. Three types of embeddings were directly extracted from the model output:

i) cls_raw_EMB, extracted with the instruction 'output.last_hidden_state[:, 0, :]', that is, the embedding corresponding to the [CLS] token, which captures the overall context of the input sequence.
ii) cls_t_EMB, extracted with the instruction 'output.pooler_output', that is, the [CLS] token embedding transformed using a nonlinear function (tanh), which refines the sequence-level representation, at least in the context of NLP tasks.
iii) aa_EMB, extracted with the instruction 'output.last_hidden_state[:, 1:-1, :].mean(dim=1)', that is, the global average of the individual amino acid embeddings, which captures the overall information of the protein sequence, excluding the special tokens [CLS] and [SEP].

An additional embedding, referred to as 'cls_aa_EMB_con', was generated by concatenating the cls_raw_EMB and aa_EMB.

3.4. Dimensionality reduction

Dimensionality reduction techniques aim to reduce the number of input variables in a dataset while retaining as much relevant information as possible. This process makes it easier to analyze and visualize the data, while helping to mitigate overfitting and enhance computational inefficiency. No dimensionality reduction was applied for datasets represented by ProtBERT embeddings, since the number of features (1024 or 2048) was much lower than the number of observations. Additionally, PLM embeddings capture distilled, non-redundant features, making additional dimensionality reduction unnecessary. However, for OHE data, the number of features was 15750. While this is lower than the number of observations, it is still considerably high. Additionally, the OHE representation leads to very sparse matrices, making computations more time-consuming and challenging. Therefore, for OHE representation, dimensionality reduction was applied.

Common dimensionality reduction methods include principal component analysis (PCA). However, for large and sparse datasets such as those created by one-hot encoding (OHE) PCA can be computationally expensive and inefficient because it relies on calculating a dense covariance matrix, which is not well-suited to sparse data. Alternative techniques like truncated singular value decomposition (SVD) are often more effective, in such cases. SVD handles sparse matrices more efficiently, preserving the underlying structure of the data while reducing its dimensionality. SVD decomposes a matrix into three components: $A=U\Sigma V^T$, where the singular values in Σ represent the importance or variance captured by the corresponding singular vectors. The higher the singular values, the more they capture the most significant features or variance in the

original data. Therefore, the first few singular values often contain the most critical information about the structure of the data. Truncated SVD is a simplified version of standard SVD that takes advantage of this and only computes and retains the top k singular values and their corresponding singular vectors, rather than the full decomposition [82]. This reduces the computational cost by discarding smaller singular values that contribute less to the data's variance and makes it more efficient for large datasets or when an approximation of the full decomposition is sufficient. Herein, the *TruncatedSVD* method of SciKit-Learn was used (https://scikit-learn.org/dev/modules/generated/sklearn.decomposition. Truncated SVD.html) with k=1024, i.e., the top 1024 singular values were retained, ensuring that the number of features was consistent between classifiers using ProtBERT embeddings and OHE representation, thereby making their comparison more balanced.

3.5. Data scaling

Data scaling is an important preprocessing step in ML since many algorithms perform better when features are on a similar scale. However, the need for scaling largely depends on the specific data characteristics and the algorithms being used.

Word embeddings are typically constrained within a fixed range, often between -1 and 1, or 0 and 1, depending on the normalization method used during the embedding generation process. In the case of ProtBERT, the embeddings generally range between -1 and 1, with values centered around 0, although they may extend beyond this range based on the specific distributions learned during training. This central value and boundaries were confirmed for the embeddings generated in this work (Annexes, Figure A1).

OHE results in binary features (0 or 1), i.e., each feature is already in a fixed range. This representation type does not benefit from scaling, and it may even introduce distortions. However, when OHE data is transformed through dimensionality reduction using SVD, the range of the values can vary significantly, as was the case here. Therefore, standardization was applied to the SVD-transformed OHE data to ensure the features were appropriately scaled. Standardization is a scaling technique that transforms data by subtracting the mean and dividing by the standard deviation. This helps that all features contribute equally to subsequent analysis or model training. In this study, the *StandardScaler* method of SciKit-Learn was used (https://scikit-learn.org/1.5/modules/generated/sklearn.preprocessing.StandardScaler.html#sklearn.preprocessing.StandardScaler).

3.6. K-means clustering

K-means clustering is a widely used unsupervised learning algorithm that partitions a dataset into a specified number (k) of clusters. It was independently proposed and developed by several researchers in the 1950s and 1960s, as reviewed by Ikotun *et al.* (2023) [83], and in its basic form consists of four core steps. The process begins by randomly selecting k initial centroids, which serve as the central points of the clusters. Then, each data point is assigned to the nearest centroid based on Euclidean distance or other distance metric of choice. Once all points are assigned, the algorithm recalculates the centroids by computing the average of all data points within each cluster. This assignment and update process is repeated iteratively until the centroids stabilize and do not change significantly. The algorithm's simplicity, ease of implementation, and low computational complexity make it a popular choice for initial data exploration and one of the top clustering algorithms in data mining [83]. Here, it was employed as part of an initial exploratory analysis and the *KMeans* method of SciKit-Learn was used (https://scikit-learn.org/1.5/modules/generated/

sklearn.cluster.KMeans.html) setting k=2 clusters to assess whether the data would naturally segregate according to the known viability labels (viable or non-viable sequences).

3.7. t-Distributed stochastic neighbor embedding

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a dimensionality reduction technique first introduced by van der Maaten and Geoffrey (2008) [84] particularly suited for visualizing high-dimensional data. It maps high-dimensional data into a lower-dimensional space (commonly 2D or 3D) while preserving the local structure and relationships between data points. t-SNE operates by first computing pairwise similarities between data points in the high-dimensional space using a Gaussian probability distribution. It then initializes points randomly in the lower-dimensional space and iteratively adjusts their positions to match the high-dimensional similarities. To model the similarities in the lower-dimensional space, t-SNE uses a Student's t-distribution, which helps to maintain the local structure and reduces the crowding of the data. Here, it was employed as part of an initial exploratory analysis, and the *TSNE* method of SciKit-Learn (https://scikit-learn.org/1.5/modules/generated/sklearn. manifold.TSNE.html) was used with the following modifications: (i) the number of iterations was increased from 1000 to 10000 to ensure better convergence and capture more complex patterns in the data, and (ii) the perplexity was raised from 30 to 50 to account for the large dataset size, enabling the algorithm to consider more neighbors when modeling the local structure of the data.

3.8. Binary classification

3.8.1. Grid search for models' hyperparameter tuning

Most ML models have a set of parameters that are not learned from the training data but are set before the training process, generally called hyperparameters. These hyperparameters control several aspects of the learning process, such as the learning rate, regularization strength, or the number of layers in a neural network. Proper tuning of these hyperparameters results in optimized model performance and better results overall. Hyperparameter tuning is commonly associated with supervised learning but also applies to unsupervised learning. However, since labels are typically unavailable in unsupervised tasks, the methods for evaluating model performance differ, often relying on alternative metrics to assess the quality of the fit. In this work, hyperparameter tuning was employed only for supervised learning models. To that end, a grid search was performed, systematically evaluating a predefined set of hyperparameter values by combining every screened value. The description of each parameter choice is detailed in the respective algorithm description in the next sections. This approach identifies the best configuration within the tested values for each model by comparing their performance across the specified hyperparameter grid. In this work, the *GridSearchCV* method of SciKit-Learn was used (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html), which enables performing grid search with a 10-fold cross-validation strategy (detailed below). The performance during grid search was evaluated based on F1-score. In theory, precision would be the target metric for this study, since minimizing false positives is crucial due to the high cost associated with experimental validation of the newly designed sequences. However, optimizing based on precision can increase false negatives, limiting the discovery of novel and more challenging sequences. In addition, maximizing precision may negatively impact recall, which is necessary to identify as many true positives as possible. Therefore, the F1 score was chosen as the evaluation metric in a grid search since it balances both precision and recall.

3.8.1. Data partition for train-validation and evaluation metrics

Model training and the grid search process described above were done using cross-validation as a data partitioning method. This approach divides the dataset into multiple subsets (folds). The model is trained (or screened) on a portion of the data, typically consisting of all but one fold, while its performance is assessed on the remaining fold. This process is repeated across all folds, providing a more robust estimate of the model's performance and helping to mitigate overfitting. In this project, 10-fold cross-validation was employed. The *StratifiedKFold* function from SciKit-Learn (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html) was used to ensure a balanced representation of each class in every fold. The *cross_validate* function (https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html) was applied to perform cross-validation on the chosen models, which in this case were the random forest, XG-Boost, and logistic regression.

The final classifiers (models with the best parameters) were evaluated considering a set of metrics commonly used in classification, namely, precision, accuracy, recall, F1 score, and receiver operating characteristic area under the curve (ROC AUC). The reported metrics represent the median values across 10 evaluation rounds. Statistical significance was evaluated using the Wilcoxon rank-sum test, with a significance threshold set at 0.05.

3.8.2. Models

For random forests, the *RandomForestClassifier* class from SciKit-Learn (sklearn.ensemble.RandomForestClassifier) was used (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier. html). The hyperparameters tuned here were the number of trees in the forest (n_estimators) and the maximum depth of each tree (max_depth), while all other parameters remained at their default settings. The number of trees in the forest influences the model's stability and accuracy. More trees generally improve performance and reduce variance, although it increases computational cost. The maximum depth of each tree controls the complexity of each tree. The deeper the trees, the more patterns are captured although the risk of overfitting also increases.

For logistic regression, the *LogisticRegression* class from SciKit-Learn (sklearn.linear_model.LogisticRegression) was used (https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html). Hyperparameter tuning here focused on the type and strength of regularization. An initial screening with various solvers was also conducted. However, only the L-BFGS solver (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) consistently achieved convergence across all representation types within a reasonable time frame. Using this solver, the hyperparameters tuned were the regularization type, which helps control overfitting by constraining the magnitude of the coefficients (with L2 or None being compatible with L-BFGS), and the regularization strength (C parameter), which adjusts the degree of regularization either by shrinking the model's coefficients or allowing it to fit more closely to the training data. The C parameter is the inverse of regularization strength; thus, the smaller the C, the larger the regularization strength.

For XG-Boost, the *XGBClassifier* class from SciKit-Learn application programming interface (API) for XGBoost classification (xgboost.XGBClassifier) used (https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.sklearn). Two hyperparameters were chosen here for tuning: the maximum depth of each tree (max_depth) and the learning rate, while all other parameters remained at their default settings. The maximum depth controls tree complexity, with deeper trees capturing more patterns (although increasing overfitting risk). The learning rate determines how quickly the model updates, balancing convergence speed and accuracy.

3.9. Heterogeneous ensemble

The heterogeneous ensembles developed in this work integrate the classification outputs of three distinct models (random forests, XGBoost, and logistic regression) across up to five different data representations (OHE, cls_raw_EMB, cls_t_EMB, aa_EMB, and cls_aa_EMB_con). This approach aims to enhance predictive performance by leveraging the complementary strengths of each classifier, which capture diverse decision boundaries and patterns, and each representation, which encodes different structural, sequential, and contextual properties of the data. By integrating these complementary perspectives, the ensemble improves robustness, reduces uncertainty, and increases confidence in the classification outcome. Up to 15 classification predictions are obtained for each sequence by applying three models across the five representation types (first baseline), although in some cases representations providing very low performance were excluded. The final classification is determined through majority voting among the remaining pairs. The majority voting can be either simple (second baseline), where the votes of all pairs have equal weight, or weighted, where each vote's influence is adjusted based on the performance of the corresponding pairs. In the weighted approach, the weight assigned to the vote of each pair reflects the pair performance relative to all other pairs, based on one or more evaluation metrics among precision, accuracy, recall, F1 score, and ROC-AUC. Two methods for weighting were tested: linear weighting (Eq. 3.1) and exponential weighting (Eq. 3.2). In linear weighting, each pair contribution is directly proportional to its performance. In exponential weighting, higher-performing pairs are assigned exponentially larger weights, increasing their influence in the ensemble.

Equation 3.1:

$$Linear\ w_i^{(j)} = \frac{\left(M_{ij} - M_j^{min}\right) + \alpha}{(M_{max} - M_{min})}$$

where M_{ij} denotes the performance score of the i^{th} pair for the j^{th} evaluation metric, M_j^{min} and M_j^{max} are, respectively, the minimum and the maximum value of the j^{th} metric across all pairs. A small constant α=0.01 is added to prevent the lowest-performing pair from receiving a zero weight.

Equation 3.2:

$$Exponential\ w_i^{(j)} = e^{M_{ij} - M_j^{min}}$$

where M_{ij} denotes the performance score of the i^{th} pair for the j^{th} evaluation metric and M_j^{min} is the minimum value of the j^{th} metric across all pairs. When more than one metric was used for weighting, the combined weight for each pair was calculated by aggregating the individual weights derived from each metric (Eq. 3.3):

Equation 3.3:

$$w_i = \frac{1}{N} \sum_{j=1}^{N} w_i^{(j)}$$

where N is the total number of metrics used and w_i^j is the scaled weight for the pair p_i based on metric j. Finally, the weighted vote is calculated for each possible class c as follows (Eq. 3.4):

Equation 3.4:

$$V(c) = \sum_{i \mid P_i = c} w_i$$

where P_i is the prediction made by the pair p_i for the given instance and V(c) is the accumulated weight for class c. The final prediction is the class with the highest weighted vote given by (Eq. 3.5):

Equation 3.5:

$$\hat{y} = \arg\max_c V(c)$$

In addition to the final prediction, an agreement score was calculated to quantify the reliability of the classification. The agreement score is the proportion of votes received by the majority class relative to the total number of votes. This score reflects the certainty of the ensemble's decision, ranging from 0.5 (indicating complete disagreement or a tie between the pairs) to 1 (indicating total agreement among the pairs). The agreement score can also be used to rank sequences classified as positive, enabling prioritization of those with the highest certainty for further laboratory evaluation.

3.10. Generation of new sequences with MCMC

The MCMC process implemented here utilizes the frequency of amino acids found in the pool of all positive sequences from the train-validation set as the likelihood function, that is, the likelihood is based on how well the amino acid composition of a sequence matches the composition of the positive sequences. The algorithm follows the Metropolis-Hastings framework [85], where a new sequence state (the proposal) is generated from the current state. A probabilistic acceptance criterion determines whether the proposal is accepted or rejected, ensuring the chain converges to the target distribution, which is the distribution of amino acid frequencies observed in the positive pool.

The algorithm begins with an initial state, which in this case was the wild-type sequence. It then generates a proposal sequence by introducing a single mutation of any type (insertion, substitution, or deletion). The likelihood of a sequence (current or proposal) is computed based on its amino acid composition (Eq. 3.6):

Equation 3.6:

$$\text{log\_likelihood}\,(sequence) = \sum_{i=1}^{N} f_i \log\,(p_i)$$

where f_i represents the frequency of amino acid i in the sequence, and p_i is the frequency of amino acid i in the positive pool, and N the total number of amino acids.

To discourage excessive length changes, a length penalty (Eq. 3.7) is applied based on the absolute difference in lengths between the current and proposed sequences.

Equation 3.7:

$$length\_penalty = \lambda . \left| len_{proposal\_sequence} - len_{current\_sequence} \right|$$

where λ is a constant that controls the magnitude of the penalty, and here it was set to 0.2. This penalty is subtracted from the proposed sequence's likelihood to calculate the adjusted likelihood for that sequence (Eq 3.8).

Equation 3.8:

$$adj\_likelihood_{proposal\_sequence} = likelihood_{proposal\_sequence} - length\_penalty_{proposal\_sequence}$$

Next, it calculates an acceptance ratio computed as the probability of accepting the proposed sequence, given the likelihoods of the current and proposed sequences (Eq. 3.9). This ensures that the probability of accepting a proposal increases with the likelihood of the proposed sequence, but the presence of the length penalty adjusts this in favor of sequences with lengths closer to the current sequence.

Equation 3.9:

$$acceptance\_ratio = \min\left(1, e^{(adj_{likelihood_{proposal_{sequence}}} - likelihood_{current_{sequence}})}\right)$$

The acceptance ratio is used to the acceptance/rejection step, where a random value (r) is drawn from a uniform distribution between 0 and 1 (Eq. 3.10) . If this value is less than or equal to the adjusted acceptance ratio, the proposed sequence is accepted as the new current state. Otherwise, the algorithm rejects the proposed sequence, and the current state remains unchanged (Eq. 3.11). This probabilistic acceptance mechanism enables occasional acceptance of less favorable proposals, reducing the risk of getting trapped in local optima.

Eq. 3.10

$$r \sim U(0,1)$$

Eq. 3.11

$$\begin{cases} r \leq acceptance_{ratio}, & accept\ the\ proposal \\ r \geq acceptance_{ratio}, & reject\ the\ proposal \end{cases}$$

If the proposed sequence is accepted, it becomes the new current state, and the chain proceeds to the next iteration, repeating the process. If the proposal is rejected, the current sequence remains unchanged, and the algorithm continues from that point. The above steps are repeated for a predefined number of iterations *N* where, for each iteration:

- Generate a proposed mutation;
- Compute the likelihoods and length penalty;
- Calculate the acceptance ratio;
- Accept or reject the proposal based on the random draw.

The MCMC chain terminates after *N* iterations, with the final sequence being the one accepted in the most recent iteration. In this work, 5 independent chains were generated of 20 steps each (N=20), and any sequence matching those in the original dataset was discarded. As a result, a maximum of 100 sequences could be generated, with each sequence having between 1 and 20 accumulated mutations. Only the targeted fragment (region 561-588) was considered for both the amino acid frequency calculation and sequence generation. The pre- and post-fragment sequences were added to the generated fragments to reconstitute the complete sequences.

4. Results and discussion

4.1. Protein target and dataset partition strategy

The protein subject to bioengineering in this work was the capsid protein of the AAV2, which is the central structural component of the AAV. The particle adopts an icosahedral structure, comprising 60 capsid protein monomers arranged as 20 triangular facets, each triangle consisting of three capsid protein monomers that assemble into a trimeric unit (Figure 4.1). While the AAV capsid protein is 735 amino acids in length, this study focuses on the region between amino acids 561 and 588. This region is frequently targeted in AAV engineering because it plays a key role in determining the vector's tropism, immune response, and overall stability [86], making it a prime candidate for modifications aimed at optimizing therapeutic potential. Therefore, the dataset used in this project contains sequence variants of the AAV2 capsid protein, with changes in the region between amino acids 561 and 588 (Figure 4.1), consisting of single and multiple mutations, including substitutions, deletions, and insertions, alone or in combination.

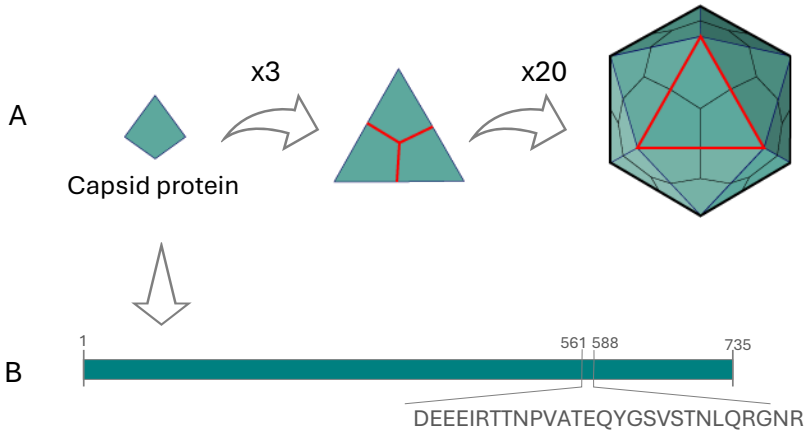

Figure 4.1. AAV2 capsid assembly and mapping of the region targeted for variation in the dataset. (A) Assembly of the AAV particle from the capsid protein monomers. (B) Highlight of the targeted region within the 735 amino acid sequence of the AAV2 capsid protein.

To determine an effective strategy for splitting the dataset into training-validation and test sets, the approach used to design each sequence in the dataset was considered, which broadly falls into two categories: 'non-ML-designed' and 'ML-designed' sequences (Table 4.1).

Non-ML-designed sequences are variants designed during the first stage of the Bryant *et al.* (2021) [6] study that were used to train ML models in the second stage of that work. The term 'non-ML-designed' is a general designation to distinguish these initial sequences from those designed by ML models in the second stage of that study. These sequences were created using algorithmic approaches such as random mutations or additive sampling (Table 4.1). To facilitate duplicate elimination, some subsets of non-ML-designed sequences were excluded, including 'previous_chip_viable', 'previous_chip_nonviable,' and 'singles,' along with 319 sequences from the 'random_doubles' subset. Additionally, the 'wild_type' subset was removed, since it contains the sequence which, being the reference, should not be used for training. The 'stop' subset was also excluded, since these sequences introduce significant complexity in pre-processing due to uncertainties in post-fragment reconstruction while contributing only 57 records to the dataset.

ML-designed sequences are modifications generated by the ML models trained on 'non-ML-designed' sequences during the second stage of the Bryant *et al.* (2021) [6] study. They represent an additional diversification effort (guided by the ML models) compared to the 'non-ML designed' sequences which makes them a good independent test set for assessing the generalization ability of the classification models implemented in this work.

Table 4.1. Dataset sequences and split strategy for train-validation and test sets

Design approach and subset designation in the original study		Nr of mutations per sequence		Number of sequences		Datasets in this thesis		
		Min	Max	Viable	Non-viable	Set	Nr of sequences	Viability
Non ML-designed*	designed	2	42	35217	21155	Train-validation set	92090	44.8%
	single	1	1	646	466			
	rand	2	10	964	8921			
	random_doubles	2	2	4395	20326			
	stop	1	1	0	57	Not used	-	-
ML-designed	cnn_designed_plus_rand_train_seed	5	25	529	1369	Test set	201426	55.0%
	cnn_designed_plus_rand_train_walked	5	29	14968	5791			
	cnn_rand_doubles_plus_single_seed	5	25	381	1641			
	cnn_rand_doubles_plus_single_walked	5	29	11229	9225			
	cnn_standard_seed	5	25	476	1448			
	cnn_standard_walked	5	29	13086	7309			
	lr_designed_plus_rand_train_seed	5	25	340	1690			
	lr_designed_plus_rand_train_walked	5	29	6134	13546			
	lr_rand_doubles_plus_single_seed	5	25	114	1957			
	lr_rand_doubles_plus_single_walked	5	29	1483	18516			
	lr_standard_seed	5	25	486	1503			
	lr_standard_walked	5	29	19211	1245			
	rnn_designed_plus_rand_train_seed	5	25	711	1354			
	rnn_designed_plus_rand_train_walked	5	29	11973	8758			
	rnn_rand_doubles_plus_singles_seed	5	25	575	1470			
	rnn_rand_doubles_plus_singles_walked	5	29	13056	7098			
	rnn_standard_seed	5	25	412	1504			
	rnn_standard_walked	5	29	15525	5313			

* During the step of duplicates elimination, subsets 'previous_chip_viable', 'previous_chip_nonviable', 'singles' as well as 319 sequences from the 'random_doubles' subset were removed from the original dataset.

4.2. Exploratory data analysis

Following the definition of the split strategy, the dataset underwent a preliminary evaluation of mutation statistics and mutational landscape analysis for the two subsets (train-validation set and test set, Figure 4.2).

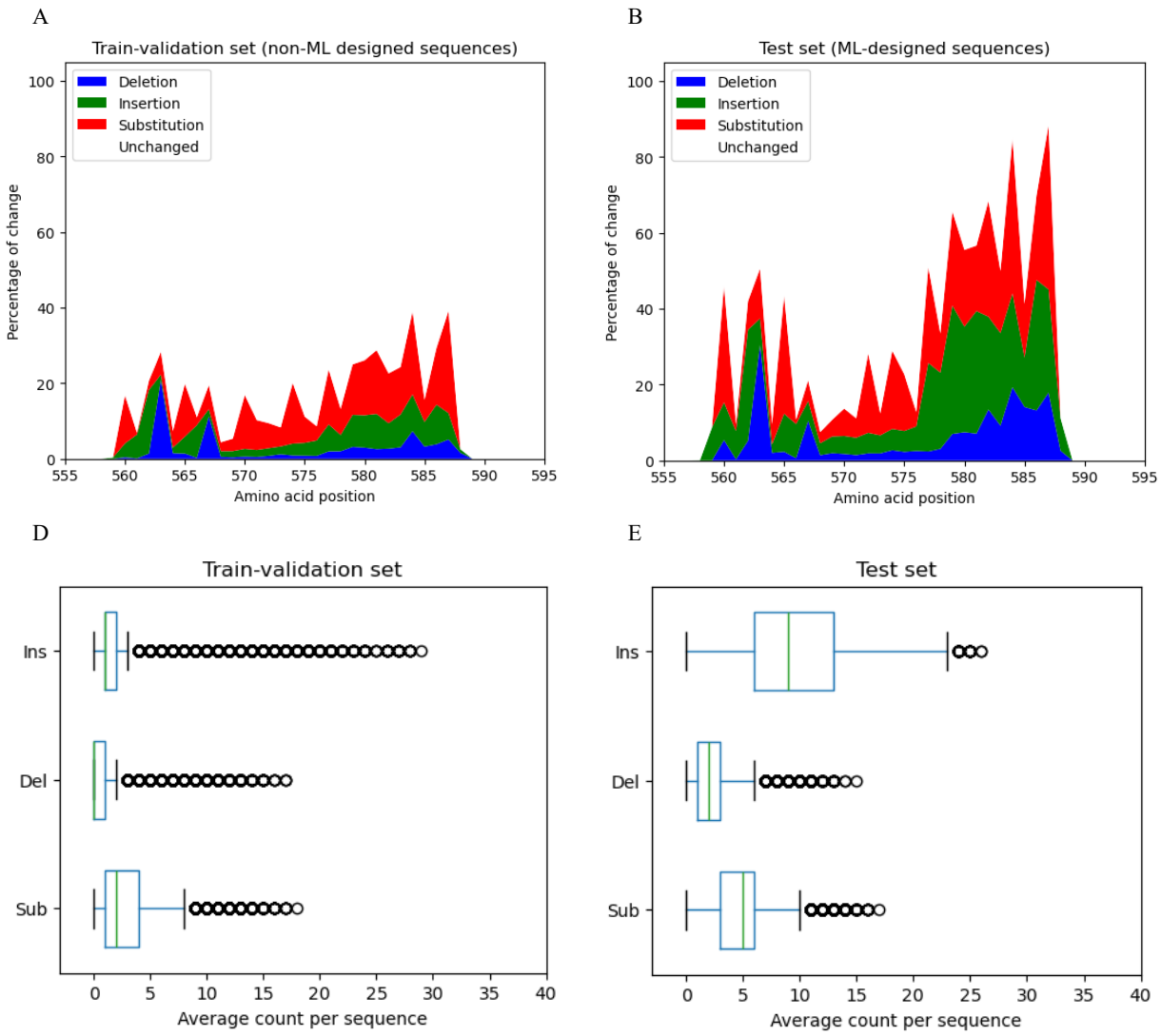

Figure 4.2. Mutation landscape and statistics sequences from the train-validation and test sets. Mutational landscape (A and B) and boxplot mutation statistics (C and D) of the AAV sequences used in this thesis grouped by set: train-validation set (non-ML designed sequences, A and C) and test set (ML-designed sequences, B and D). Mutations are indicated as insertions (Ins), deletions (Del), or substitutions (Sub).

The mutation landscape analysis (Figure 4.2 A and B) revealed that both the train-validation set (non-ML-designed sequences) and the test set (ML-designed sequences) exhibited similar mutation profiles in terms of positional targeting, that is, the specific positions targeted for mutation, and the choice of mutation types at given positions. However, the test set showed a higher average number of mutations per sequence and an increased incidence of insertions. The box-plot analysis further supports this finding (Figure 4.2 C and D). This aligns with the fact that ML-designed sequences underwent a more extensive diversification process than non-ML-designed sequences, leading to more mutations per sequence relative to the wild type. As detailed in the original study by Bryant *et al.* (2021) [6], the ML models introduce a broader range of modifications, driving greater sequence variation. The strategy learnt by those models appears to focus on increasing the number of mutations per sequence in positions that can accommodate them, with a higher prevalence of insertions, which may be better tolerated due to their occurrence in more flexible, less

structurally constrained regions of the protein. These findings support the proposed strategy in this thesis, where non-ML designed sequences serve as the train-validation set, and ML-designed sequences act as an independent and different test set to assess the generalization capabilities of the developed models.

Following the preliminary mutation analysis, K-means clustering was employed as an initial approach to explore the dataset and identify potential patterns within the data. Specifically, we investigated whether setting K=2 (corresponding to the two classes in the dataset) would result in each cluster being enriched in one of the classes. The results revealed that neither cluster showed enrichment for a particular class (Figure 4.3), indicating that K-means was unable to distinguish between the viable and non-viable sequences. Part of this limitation could be due to K-means relying on simple Euclidean distance-based similarity, which captures only linear relationships and simple patterns in the data. To address this, we applied t-SNE, a technique capable of capturing both linear and non-linear relationships (Figure 4.4).

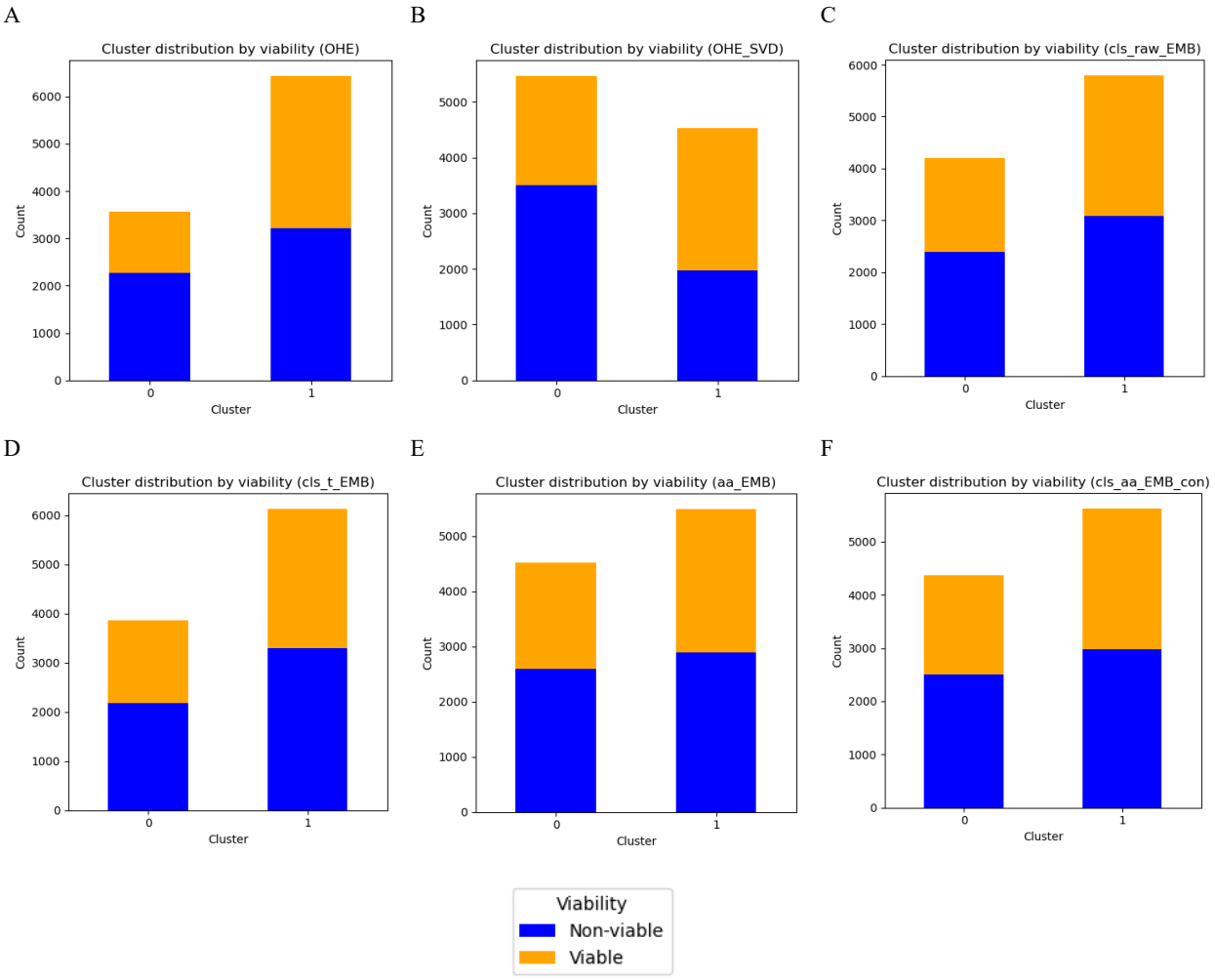

Figure 4.3. K-means analysis of train-validation set sequences. K-means clustering was performed with K=2 on sequences from the train-validation set, using each of the five representation types (OHE, cls_raw_EMB, cls_t_EMB, aa_EMB and cls_aa_EMB_con) evaluated in this work; within OHE representation, the SVD-reduced dataset was also analyzed. Due to the large dataset size, the analysis was conducted using a random sample of 10000 sequences.

By mapping high-dimensional data into a lower-dimensional space while preserving local similarities, t-SNE can reveal more complex structures, clusters, and hidden patterns that simpler clustering algorithms might miss. Additionally, t-SNE provides a powerful visualization tool. Using this approach, we observed emerging aggregation patterns between the classes, although complete separation between viable and non-viable sequences was still not achieved (Figure 4.4).

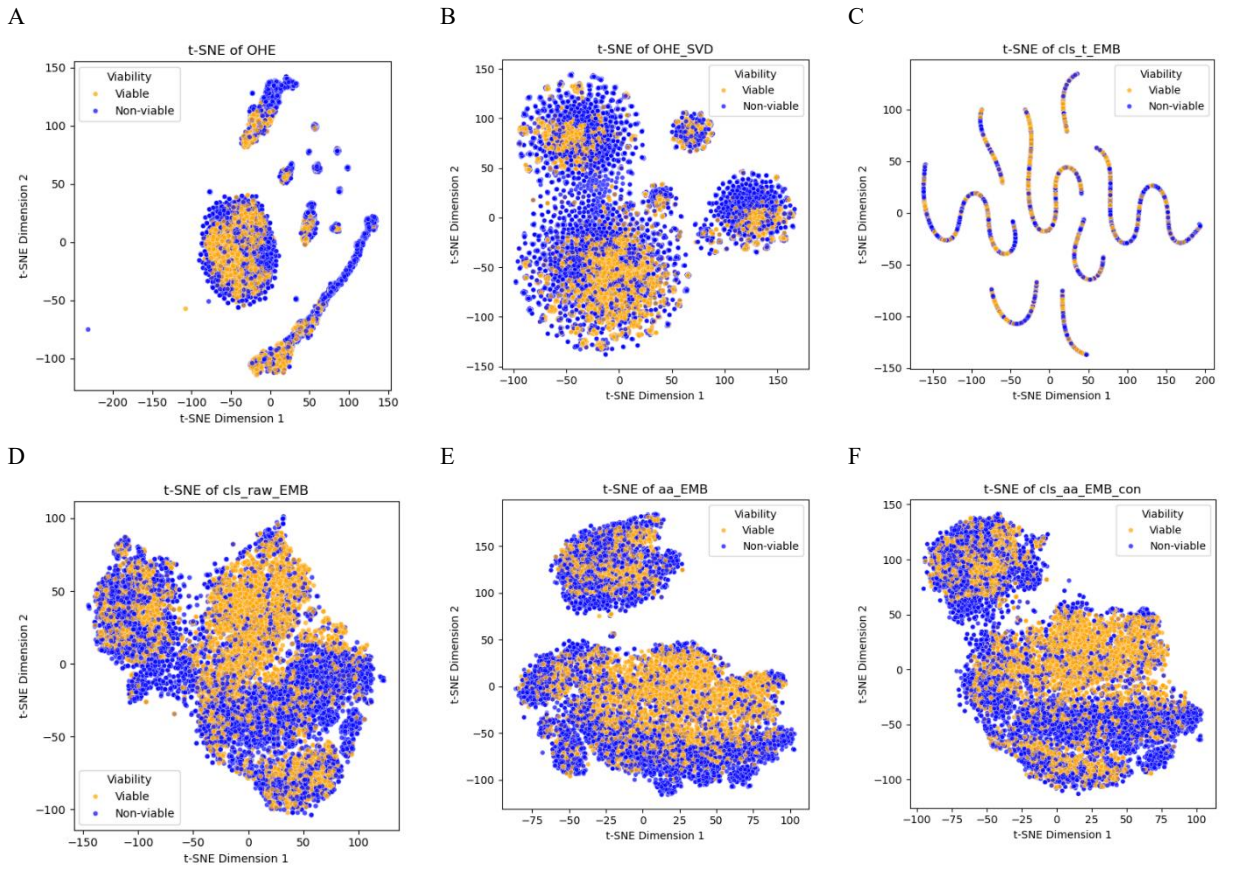

Figure 4.4. t-SNE analysis of train-validation set sequences. t-SNE analysis performed with perplexity=50 on sequences from the train-validation set, using each of the five representation types (OHE, cls_raw_EMB, cls_t_EMB, aa_EMB, and cls_aa_EMB_con) evaluated in this work; within OHE representation, the SVD-reduced dataset was also analyzed. Due to the large size of the dataset, the analysis was conducted using a random sample of 10000 sequences (same as for K-means analysis).

This lack of clear separation, observed in both K-means and t-SNE analyses, is unsurprising since unsupervised methods rely solely on inherent structures in the data. The complexity of the underlying relationships, possibly involving highly complex non-linear interactions and subtle patterns, can dictate that without the guidance of labeled data, these methods are not sufficient for uncovering the distinguishing factors between viable and non-viable sequences.

Regardless of the inability to separate viable from non-viable sequences, it was interesting to notice that different sequence representations produced distinct patterns in the t-SNE analysis. cls_raw_EMB, aa_EMB, and cls_aa_EMB_con displayed a largely globular distribution, suggesting that they capture similar structural or functional features of the sequences. Since cls_aa_EMB_con is a concatenation of cls_raw_EMB and aa_EMB, the shape of this data is similar to that of the individual representation that composes it. OHE data, on the other hand, although also roughly globular, appeared more compact and

presented better-defined shapes. Interestingly, although clear separation was still lacking, the SVD-transformed OHE data showed improved class distinction, showing that dimensionality reduction helped to eliminate noise, as expected. Meanwhile, the cls_t_EMB representation, derived from cls_raw_EMB through a dense layer with a tanh activation function, showed a marked distinctly different pattern, forming curved-line structures with low dispersion. This suggests that the transformation reduced variability, making all sequences more similar. Overall, these variations show that each representation encodes relationships in the data differently. This influences their ability to capture structural patterns, functional similarities, and sequence variability, and affects their utility in downstream tasks.

4.3. Classifiers training and testing

Following the exploratory data analysis, the work shifted to the core of this thesis: training a set of classifiers for AAV2 capsid protein classification using diverse ML models, namely, random forests, XGBoost, and logistic regression. Table 4.2 presents the results from the grid search performed for hyperparameter optimization of each model, while Tables 4.3 through 4.8 summarize the train-validation and test phase outcomes. To facilitate an overall analysis of the performance across the three classifiers and five representation types, the results from Tables 4.3 through 4.8 were also consolidated into a plot (Figure 4.5).

Table 4.2. Summary of best-performing parameters from grid search for each model

Model	Parameter	Values evaluated	Best performing parameter				
			OHE-SVD	EMB			
				cls_raw	cls_t	aa	cls_aa_con
Random Forests	Number of trees in the forest	{10, 20, 50, 100, 200}	200	200	10	200	200
	Maximum depth of each tree	{10, 20, 30, 50}	50	20	30	50	20
Logistic Regression	Regularization type	{L2, None}	L2	L2	L2	L2	L2
	Regularization strength (C)	{0.01, 0.1, 1, 10}	10	10	0.01	10	10
XG-Boost	Maximum depth of each tree	{10, 20, 30, 50}	10	10	10	10	10
	Learning rate	{0.01, 0.05, 0.1}	0.1	0.1	0.05	0.1	0.1

The grid search revealed that the optimal parameter values vary with the representation type. This outcome was expected since different feature spaces can influence how a model learns and generalizes patterns, requiring tailored hyperparameters to achieve the best results. Optimizing each representation separately, rather than applying a single set of parameters across all, ensures a fair comparison by allowing each model to perform at its full potential rather than being constrained by suboptimal settings. Grid search was successfully conducted for all representations except cls_t_EMB. In the case of cls_t_EMB, all records were classified as negatives.

Table 4.3. Results of train-validation stage with random forest classifiers

Representation		Precision	Accuracy	Recall	F1 Score	ROC AUC
OHE-SVD	Median	**0.8680**	**0.8926**	**0.8946**	**0.8819**	**0.9633**
	IQR	0.0033	0.0046	0.0110	0.0052	0.0008
cls_t_EMB	Median	0.5050	0.5568	0.4931	0.5002	0.5747
	IQR	0.0083	0.0073	0.0179	0.0127	0.0065
cls_raw_EMB	Median	0.8329	0.8409	0.8070	0.8193	0.9221
	IQR	0.0045	0.0038	0.0029	0.0040	0.0021
aa_EMB	Median	0.8514	0.8456	0.7933	0.8215	0.9255
	IQR	0.0020	0.0034	0.0101	0.0051	0.0015
cls_aa_EMB_con	Median	0.8383	0.8461	0.8132	0.8256	0.9259
	IQR	0.0043	0.0043	0.0086	0.0055	0.0020

Statistical significance was assessed using the Wilcoxon rank-sum test for the comparisons of each ProtBERT embedding format versus OHE-SVD. Bold values highlight the highest performance metrics, while significantly higher values are further distinguished by underlining. IQR: interquartile range.

Table 4.4. Results of test stage with random forests classifiers

Representation	Precision	Accuracy	Recall	F1 Score	ROC-AUC
OHE-SVD	**0.9922**	0.5108	0.1107	0.1991	**0.8725**
cls_t_EMB	0.5841	0.5101	0.3767	0.4580	0.5433
cls_raw_EMB	0.9348	0.6653	0.4202	0.5798	0.8039
aa_EMB	0.9294	0.6809	0.4538	0.6098	0.8026
cls_aa_EMB_con	0.9091	**0.6954**	**0.4953**	**0.6413**	0.8025

Bold values highlight the highest performance metrics.

Table 4.5. Results of train-validation stage with XGBoost classifiers

Representation		Precision	Accuracy	Recall	F1 Score	ROC AUC
OHE-SVD	Median	**0.8884**	**0.9139**	**0.9203**	**0.9054**	**0.9753**
	IQR	0.0042	0.0027	0.0064	0.0032	0.0016
cls_t_EMB	Median	0.5200	0.5682	0.4682	0.4934	0.5927
	IQR	0.0043	0.0039	0.0120	0.0083	0.0037
cls_raw_EMB	Median	0.8533	0.8667	0.8510	0.8506	0.9446
	IQR	0.0083	0.0052	0.0088	0.0049	0.0023
aa_EMB	Median	0.8537	0.8698	0.8530	0.8546	0.9458
	IQR	0.0071	0.0056	0.0112	0.0068	0.0012
cls_aa_EMB_con	Median	0.8619	0.8752	0.8599	0.8601	0.9500
	IQR	0.0063	0.0039	0.0075	0.0042	0.0010

Statistical significance was assessed using the Wilcoxon rank-sum test for the comparisons of each ProtBERT embedding format versus OHE-SVD. Bold values highlight the highest performance metrics, while significantly higher values are further distinguished by underlining. IQR: interquartile range

Table 4.6. Results of test stage with XGBoost classifiers

Representation	Precision	Accuracy	Recall	F1 Score	ROC-AUC
OHE-SVD	**0.9279**	0.6366	0.3673	0.5263	0.7708
cls_t_EMB	0.6047	0.5094	0.3095	0.4095	0.5565
cls_raw_EMB	0.9273	0.7171	0.5264	0.6716	**0.8775**
aa_EMB	0.9074	0.7217	**0.5496**	**0.6846**	0.8627
cls_aa_EMB_con	0.9232	**0.7219**	0.5388	0.6805	0.8759

Bold values highlight the highest performance metrics.

Table 4.7. Results of train-validation stage with logistic regression classifiers

Representation		Precision	Accuracy	Recall	F1 Score	ROC AUC
OHE-SVD	Median	**0.8822**	**0.9097**	**0.9209**	**0.9014**	**0.9698**
	IQR	0.0016	0.0016	0.0030	0.0017	0.0012
cls_t_EMB	Median	0.0000	0.5524	0.0000	0.0000	0.5013
	IQR	0.0000	0.0000	0.0000	0.0000	0.0080
cls_raw_EMB	Median	0.7760	0.7864	0.7340	0.7537	0.8650
	IQR	0.0091	0.0052	0.0096	0.0066	0.0061
aa_EMB	Median	0.7691	0.7801	0.7286	0.7475	0.8579
	IQR	0.0081	0.0062	0.0083	0.0070	0.0059
cls_aa_EMB_con	Median	0.7865	0.8014	0.7627	0.7747	0.8807
	IQR	0.0130	0.0145	0.0201	0.0171	0.0171

Statistical significance was assessed using the Wilcoxon rank-sum test for the comparisons of each ProtBERT embedding format versus OHE-SVD. Bold values highlight the highest performance metrics, while significantly higher values are further distinguished by underlining. IQR: interquartile range

Table 4.8. Results of test stage with logistic regression classifiers

Representation	Precision	Accuracy	Recall	F1 Score	ROC-AUC
OHE-SVD	0.6374	0.5403	0.3794	0.4756	0.6001
cls_t_EMB	0.0000	0.4505	0.0000	0.0000	0.6283
cls_raw_EMB	0.8583	0.7622	0.6794	0.7584	0.8667
aa_EMB	0.8061	0.7389	**0.6910**	0.7441	0.8310
cls_aa_EMB_con	**0.8609**	**0.7658**	0.6843	**0.7625**	**0.8714**

Bold values highlight the highest performance metrics.

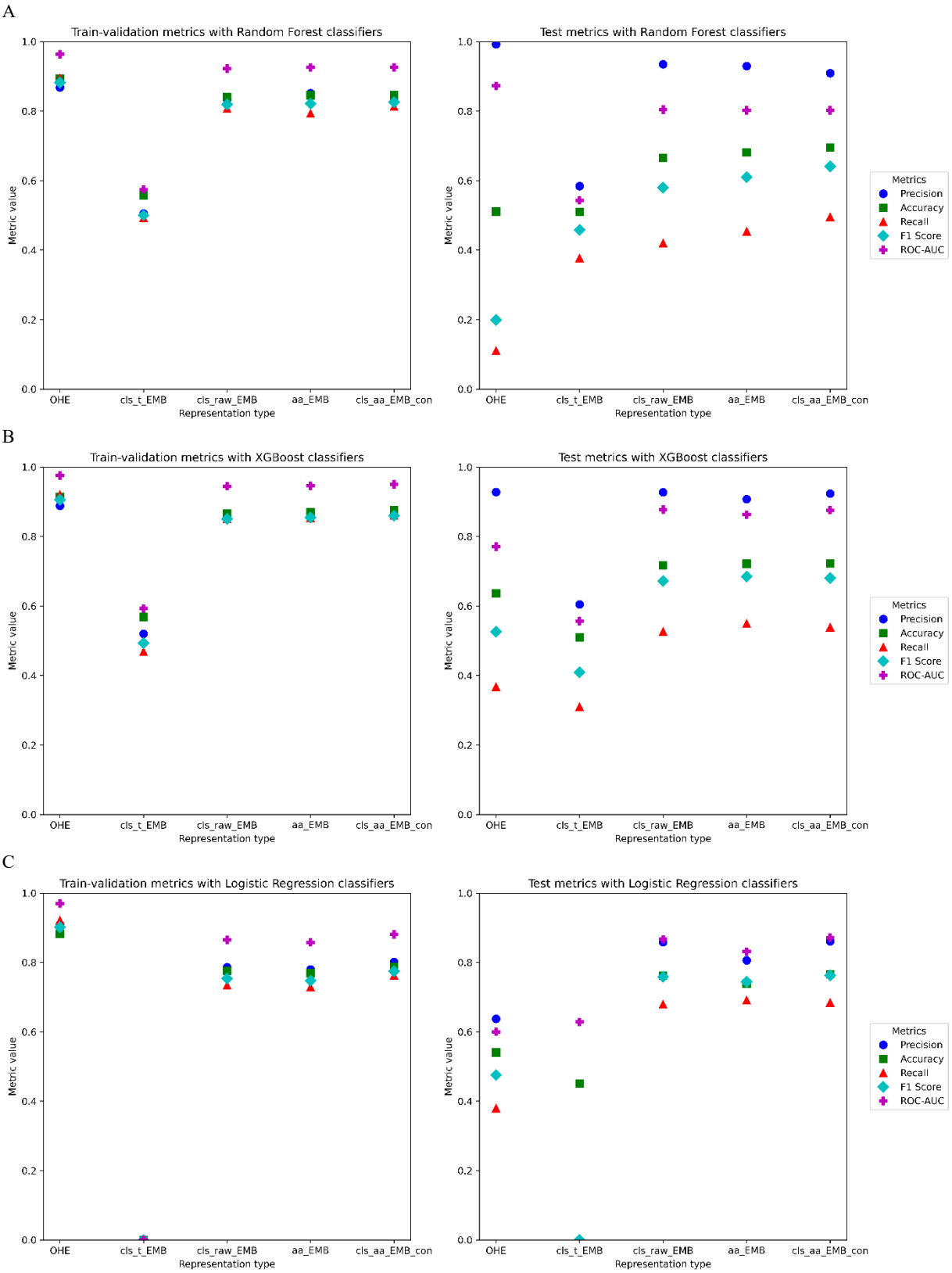

Figure 4.5. Summary of classification metrics. Global visualization of classification metrics of random forests (A), XGBoost (B) or logistic regression (C) classifiers for the train-validation (left panels) and the test (right panels) stages.

The results suggest that the models exhibited significant overfitting, evidenced by a sharp decline in performance metrics from the train-validation stage to the test stage, particularly evident for the tree-based methods. The only exception to this trend was observed in precision, where performance improved in the test stage in random forests and XGBoost maintained relatively stable performance. Overfitting is a common issue, especially when using grid search, since it exhaustively explores a fixed set of hyperparameters on the training data. This approach can lead to the selection of hyperparameters that optimize performance on the training data but fail to generalize well to new, unseen data, especially if it is different from the train-validation data. To improve model generalization, stronger regularization strategies may be needed. In the case of tree-based models, the maximum tree depth of 50 (selected through grid search) is likely a key factor in overfitting, since such a deep tree is prone to capturing noise. In addition, the test set data differs from the train-validation set, since it includes more diverse sequences. Therefore, the generalization ability of the models may be compromised if the current strategy of using simpler, non-ML-designed sequences in the train-validation set, and more complex, ML-designed sequences in the test set, is maintained. An alternative approach could be to reserve part of the ML-designed sequences for validation during training, thus forcing the models to cope with a more challenging generalization. Overall, this resulted in better performance for the logistic regression models at the test stage, which might initially suggest that simpler, linear relationships in the data are sufficient to distinguish between classes. However, this is likely not the case. In fact, the overall metrics were not particularly strong. Instead, the superior performance likely derives from the logistic regression model being less prone to overfitting compared to the tree-based models, allowing it to generalize better.

The second observation was that, across all models, the cls_t_EMB representation consistently delivered poor performance. This result suggests that, as speculated, the transformation of the cls_raw_EMB representation through a dense layer followed by a tanh activation function reduces its value for classification tasks. This was interesting to notice because the cls_t_EMB is actually very informative and useful in NLP. However, in NLP, the transformation of the CLS token helps capture complex semantic relationships, disambiguate word meanings, and adapt the representation for different tasks. This works well because language has hierarchical structures and contextual dependencies based on syntactic and semantic rules. In contrast, while protein sequences also have hierarchical structures and contextual dependencies, these are governed by biochemical and structural principles rather than linguistic ones. As a result, the raw embeddings may already encode the most relevant functional information, and applying a nonlinear transformation could distort rather than refine this representation, making it less effective for sequence-to-function learning and classification.

The third observation was the consistently high performance of OHE representation over all the remaining at train-validation. However, in the test stage, this was no longer the case where, except for the cls_t_EMB case, models using PLM embeddings generally outperformed those using OHE across all metrics in most cases. Additionally, although the differences are small, the aa_EMB and cls_aa_EMB_con representations appear to be the most effective globally. This highlights the value of the biological information captured by PLM embeddings in identifying meaningful patterns and enhancing generalization across different datasets. Moreover, the improved performance of cls_aa_EMB_con compared to aa_EMB alone demonstrates the benefit of incorporating the additional information from cls_raw_EMB. Among the individual models, logistic regression achieved the best performance across the board. Therefore, if a single classifier and representation format were to be chosen, logistic regression with the cls_aa_EMB_con representation would be the optimal choice.

In addition to evaluating the performance of each model-representation pair using the classification metrics discussed above, we also conducted a side-by-side comparison of the prediction correctness for each sequence across all pairs (Figure 4.6). This analysis highlighted the generally superior performance of models utilizing PLM embeddings (except for cls_t_EMB) and uncovered an interesting pattern: some sequences were correctly classified by all pairs, including the low-performing ones, while others were consistently misclassified, even by the top-performing pairs.

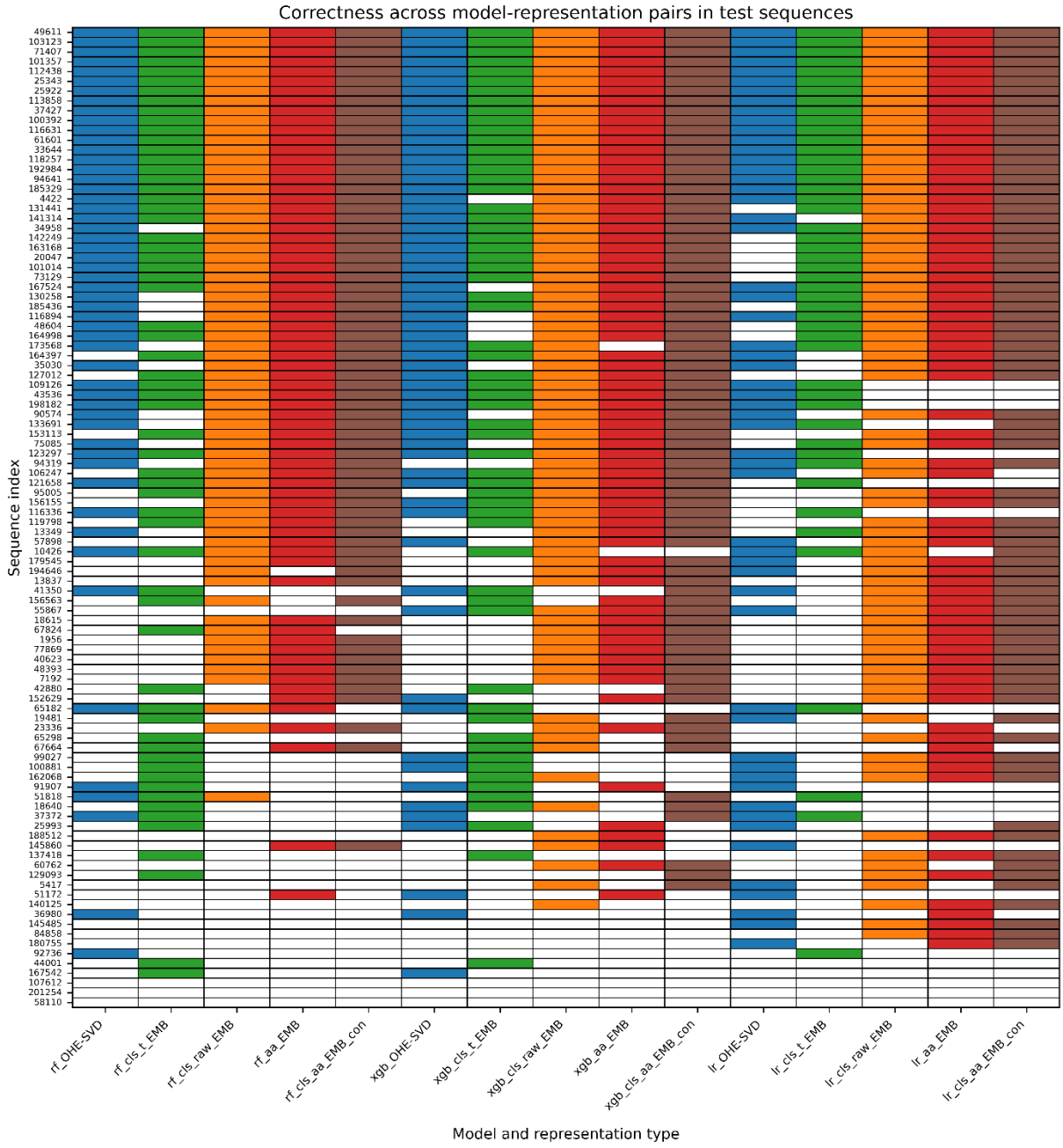

Figure 4.6. Classification correctness of each representation type across the three classifiers. Heatmap of classification output of each representation type across the three classifiers. A random sample of 100 sequences from the test set was analyzed. Model-representation pairs are colored by the representation type in each pair: OHE-SVD (blue), cls_t_EMB (green), cls_raw_EMB (orange), aa_EMB (red) and cls_aa_EMB_con (brown). Colored indicates correct and white indicates incorrect predictions.

To further investigate these patterns, the composition of easy and difficult sequences was analyzed with respect to their ground truth class and the type of ML model used in their design (Figure 4.7). Although no specific ML model was disproportionately represented in either easy or difficult sequences, the analysis clearly showed that all easy sequences belong to the non-viable class, whereas all difficult sequences belong to the viable class. This suggests that the inherent properties of viable sequences pose a greater challenge for accurate classification, possibly due to higher complexity or similarity to non-viable sequences.

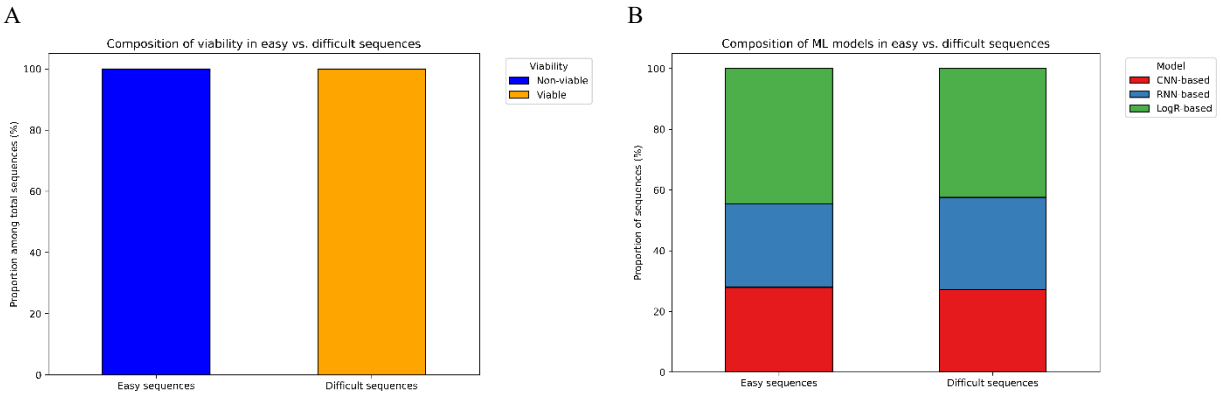

Figure 4.7. Composition of easy and difficult sequences based on ground truth class and ML model design. The distribution of 32086 easy sequences (correctly classified by all pairs) and 6336 difficult sequences (misclassified by all pairs) is shown, categorized by their ground truth class (A) and the type of ML model used in their design (B).

To better understand the source of this challenge, the mutation landscape of both viable and non-viable sequences, as well as that of easy and difficult sequences, was further examined (Figure 4.8). This revealed a distinct signature for viable sequences: they tend to avoid mutations in the region of amino acids 567-576, and when mutations do occur in this region, they are mostly substitutions. This suggests that this region is sensitive to changes in sequence length (insertions or deletions) and only tolerates substitutions at a moderate rate. This explains why easy sequences (all negative) are consistently classified correctly, given that they heavily target this forbidden region with all types of mutations. On the other hand, difficult sequences closely resemble the positive class pattern, suggesting potential viability. However, the challenge likely lies not in the mutation type, since they also favor moderate substitutions, but in the specific substitutions made, since differences in amino acid physicochemical properties mean that not all substitutions have the same impact. Indeed, models using representation formats that based on amino acid information outperformed the others (Figure 4.6), highlighting the importance of capturing these biological differences beyond simple positional targeting. The added difficulty of these 6336 sequences could be due to other factors such as subtle physicochemical changes affecting protein folding, local structural constraints, or epistatic interactions that complicate accurate classification.

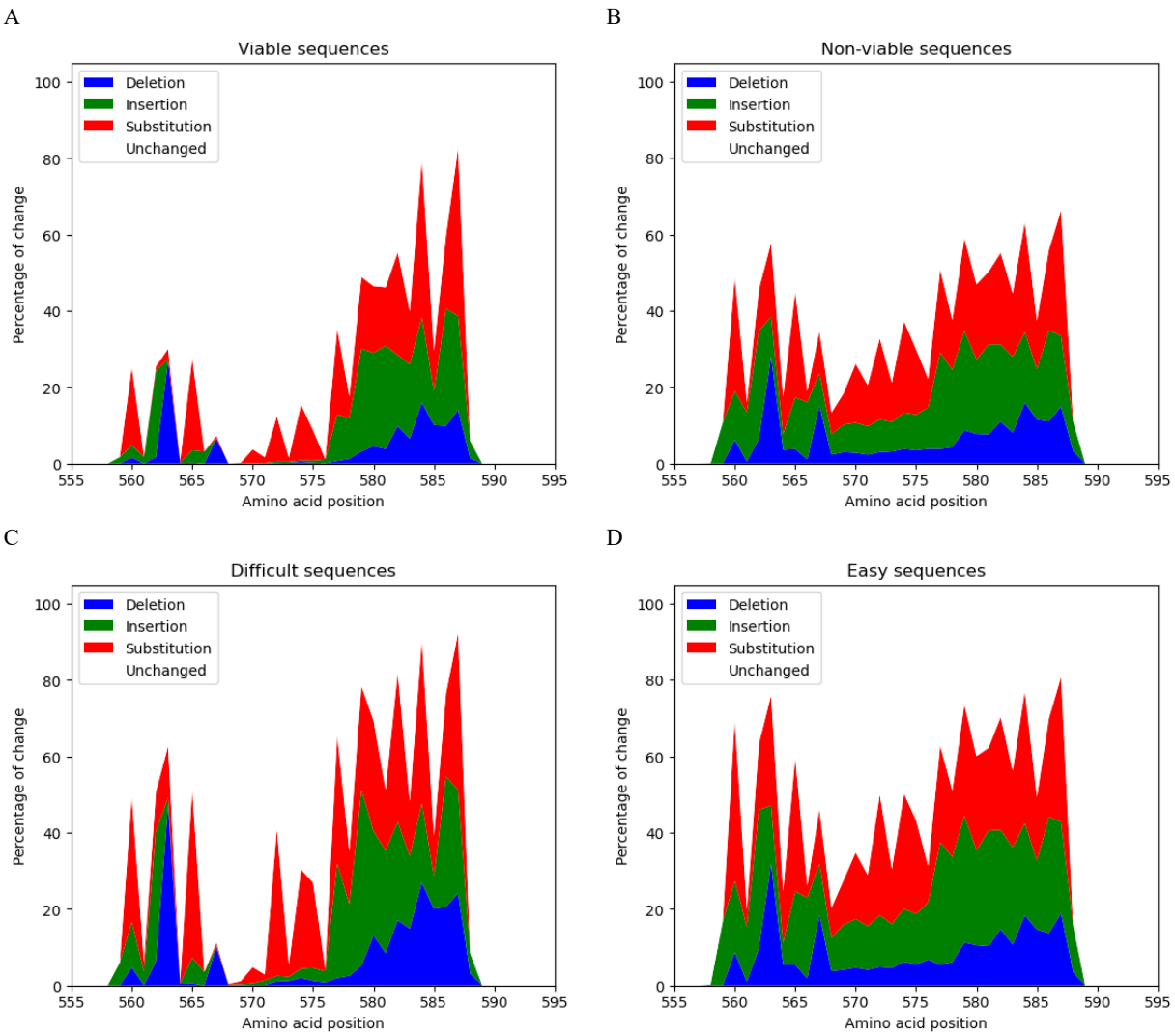

Figure 4.8. Mutation landscape of positive, negative, easy and difficult sequences. The mutational landscape is shown for viable sequences (A), non-viable sequences (B), difficult sequences (C), and easy sequences (D). For viable and non-viable categories, all sequences from both the train-validation set (non-ML-designed) and the test set (ML-designed) were included. For the easy and difficult categories, all 32086 sequences correctly classified by all pairs (easy sequences) and all 6336 sequences misclassified by all pairs (difficult sequences) were analyzed.

4.4. Ensemble of classifiers

In the final stage of this work, the three classifiers implemented were combined to produce a unified classification output. This approach follows an ensemble paradigm, where multiple learners collaborate to generate a more robust result. Two types of sequences were analyzed with this ensemble. First, ML-designed sequences from the test set, for which the ground truth labels were known. This allowed for a detailed comparison of each model-representation pair for the same sequence, while also providing an opportunity to assess the overall performance of the ensemble. Next, newly generated sequences, either produced with an MCMC-based process or a generative PLM [67], were analyzed. These sequences represent a first set of candidates that, based on the findings of this thesis, should be evaluated in the laboratory soon.

4.4.1. Test set sequences: ensemble evaluation

The ensemble approach consisted of a majority voting from the classification outputs from each representation type across the three classifiers, and the respective classification metrics were computed. Several voting strategies were explored, differing in the metrics used to determine the weights and the weighting method (linear or exponential). In the linear approach, the weight assigned to each vote is directly proportional to the model-pair performance relative to all others. The exponential approach assigns exponentially greater weight to higher-performing pairs, amplifying their influence in the ensemble.

The evaluation began with a simple majority vote using all pairs, followed by the exclusion of pairs involving cls_t_EMB due to their poor performance. Since this exclusion improved the results, the weighting strategies were then assessed using only the remaining, better-performing pairs. The results are summarized in Table 4.9. The results from the best-performing ensemble (weight based on F1 exponentially weighted) were also plotted together with the metrics from individual pairs for easier comparison (Figure 4.9).

Lastly, in addition to the ensemble classification metrics, agreement scores were calculated for each sequence. The purpose of this score is to help prioritize sequences classified as positive when analyzing new candidates. The agreement score represents the consensus strength across all predictions where unanimous agreement among all model-representation pairs scores 1. The lowest possible confidence score is 0.5 in the case of a tie between votes. In such cases, the classification is defaulted to 0 (non-viable) because, even if the actual outcome is positive, it carries the lowest possible agreement. As a result, it will never be prioritized for laboratory evaluation. To investigate the relationship between agreement, score and prediction correctness and assess the score's usefulness, the percentage of correct classifications was calculated for different levels of this score; the agreement score of the wild-type sequence was also calculated for reference (Table 4.10).

Table 4.9. Results of test set classification with the heterogeneous ensemble

Model-representation pairs	**Model weight strategy**	**Precision**	**Accuracy**	**Recall**	**F1 Score**	**ROC-AUC**
All	No weights	0.9514	0.6955	0.4700	0.6291	0.7203
Without cls_t_EMB	No weights	0.9515	0.7079	0.4936	0.6500	0.7314
	Based on F1 (linear weight)	**0.9595**	0.6961	0.4667	0.6279	0.7213
	Based on F1 (exponential weight)	0.9497	**0.7086**	**0.4961**	**0.6517**	**0.7320**
	Based on F1+ ROC-AUC (linear weight)	0.9593	0.6961	0.4668	0.6280	0.7213
	Based on F1+ ROC-AUC (exponential weight)	0.9496	0.7086	0.4961	0.6517	0.7320
	Based on F1+ ROC-AUC+ Precision (linear weight)	0.9574	0.6985	0.4725	0.6327	0.7234
	Based on F1+ROC-AUC+Precision (exponential weight)	0.9497	0.7085	0.4959	0.6516	0.7319
	Based on F1+ ROC-AUC + Precision + Recall + Accuracy (exponential weight)	0.9593	0.6963	0.4671	0.6283	0.7215
	Based on F1 + ROC-AUC + Precision + Recall + Accuracy (linear weight)	0.9506	0.7082	0.4948	0.6508	0.7317

Bold values highlight the highest performance metrics.

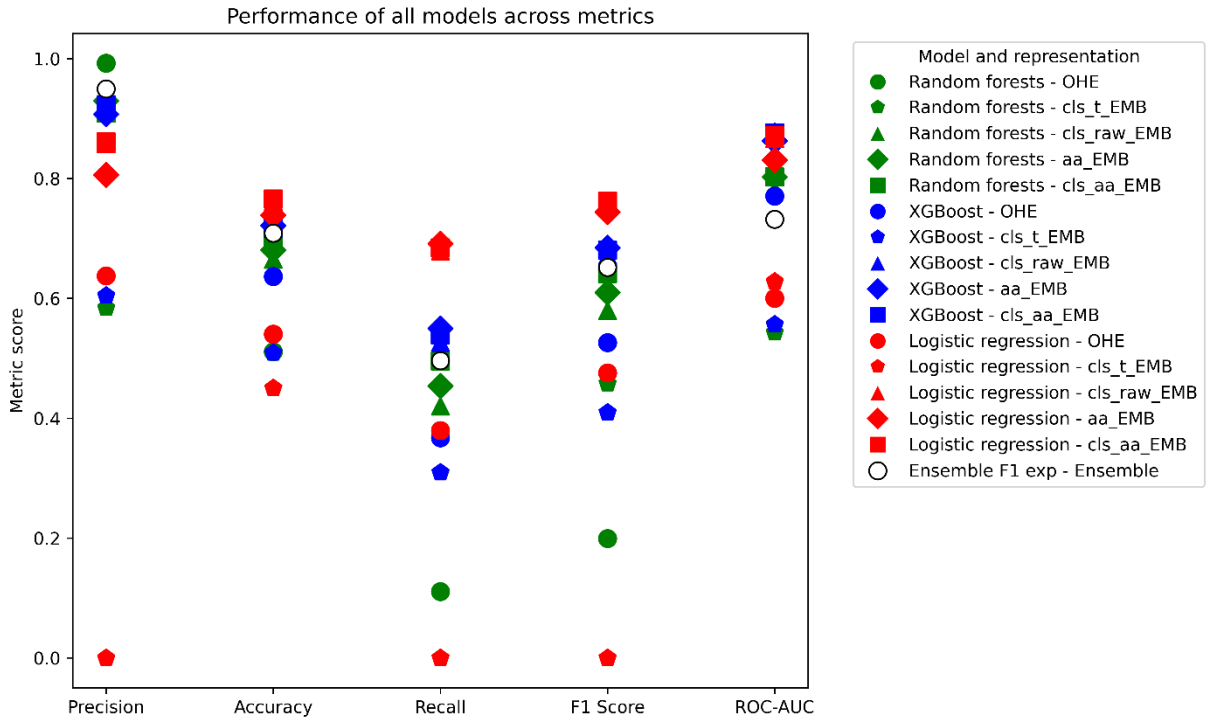

Figure 4.9. Comparison of ensemble performance with individual model-representation pairs at test stage.

The ensemble model delivered good performance across various metrics, although it never outperformed the best individual models. One possible explanation is that the individual models within the ensemble may have disagreed significantly. This is further supported by the analysis of classification correctness across models and representation types (see Figure 4.6), revealing frequent disagreement among models using OHE representation and those using protBERT embeddings. Additionally, the weights for the majority vote were based on train-validation metrics, and model performance varied significantly between the train-validation and test stages. This can lead to suboptimal weighting in the ensemble, as the models that performed well during training did not necessarily generalize as effectively to the test data. Another potential reason is the lack of diversity among the models. Since two of the three ensemble models consisted of tree-based methods, they may have learned similar decision boundaries, leading to highly correlated predictions. This redundancy limits the potential benefit of ensembling, since combining models that make similar errors does not significantly enhance performance. Additionally, the choice of ensemble method could have impacted the results. Majority voting or vote weighting may not be the most effective way to combine the outputs of different models. More sophisticated techniques, such as stacking, might better leverage the strengths of each model, leading to improved overall performance.

Regarding the agreement score, the highest accuracy (above 77%) was observed for agreement scores above 0.9. This threshold is particularly relevant, as the wild-type sequence, known to be positive, has an agreement score of 0.91. Therefore, to prioritize sequences for laboratory evaluation, only those with an agreement score above 0.9 should be considered.

Table 4.10. Correctness of classification based on agreement score levels

Agreement score	Correctness (%)
0.9 – 1.0	77.3
0.8 – 0.9	68.0
0.7 – 0.8	57.2
0.6 – 0.7	39.8
0.5 – 0.6	73.3
0.91	Wild-type sequence

4.4.2. Classification of newly generated sequences

The new sequences generated by the PLM model [67] or by the MCMC-based generator implemented in this study were evaluated using the best-performing classifier identified in this work (Figure 4.10). Since the ensemble did not outperform the top individual model-representation pair (logistic regression with the cls_aa_EMB_con representation) both the ensemble and this combination were used and compared. For the ensemble, the agreement score served as the confidence threshold, while for the logistic regression model, the probability associated with the positive class prediction was used.

The MCMC-based generator produced two sequences classified as viable by the ensemble, with only one having an agreement score above 0.9. It also generated five sequences classified as viable by the logistic regression model, all of which exceeded the confidence threshold, since this is a requirement for being considered positive. However, it might be preferable to set higher thresholds in this case, although the only reference available here is the probability of the wild-type sequence, which was 0.61. There is a strong alignment between the ensemble and the logistic regression model in classifying MCMC-generated sequences. Even in cases where the logistic regression predicted a sequence to be positive, but the ensemble did not, the agreement score for those sequences within the ensemble was still relatively high. This increases the confidence in these predictions. The second observation was that only sequences from the early steps of the chains (Figure 4.10 A and B) were classified as positive. This was expected since the number of accumulated mutations increases as the chain progresses. Consequently, the initial proposals, closer to the starting wild-type sequence, are more likely to remain positive. Additionally, since the likelihood function used (amino acid frequency) is relatively naïve and unlikely to capture patterns associated with functionality, the accumulation of mutations primarily drives the generated sequences toward the amino acid frequencies found in the pool of positive sequences rather than refining the sequences based on functional relevance. Therefore, while this approach may be able to generate a decent number of positive candidates, it is unlikely to produce sequences with high diversity, which was an important goal in this work. This limitation could be addressed by using more advanced likelihood functions, such as substitution matrices or hidden Markov models. However, this would also increase both implementation complexity and computational resource demands, making a stronger case for switching to generative models. In that context, we found that none of PLM-generated sequences were classified as positive by the ensemble, although four sequences were classified as positive by the logistic regression model, two of them with probabilities in line with that of the wild type sequence and one considerably higher (0.92). However, the lack of positive classifications by the ensemble for any of the PLM-generated sequences warrants caution, since some degree of alignment between the two approaches would be expected. On the other hand, PLM-generated sequence are likely to have many more mutations than those generated by the MCMC processes, therefore, expecting the same level of alignment might not be realistic.

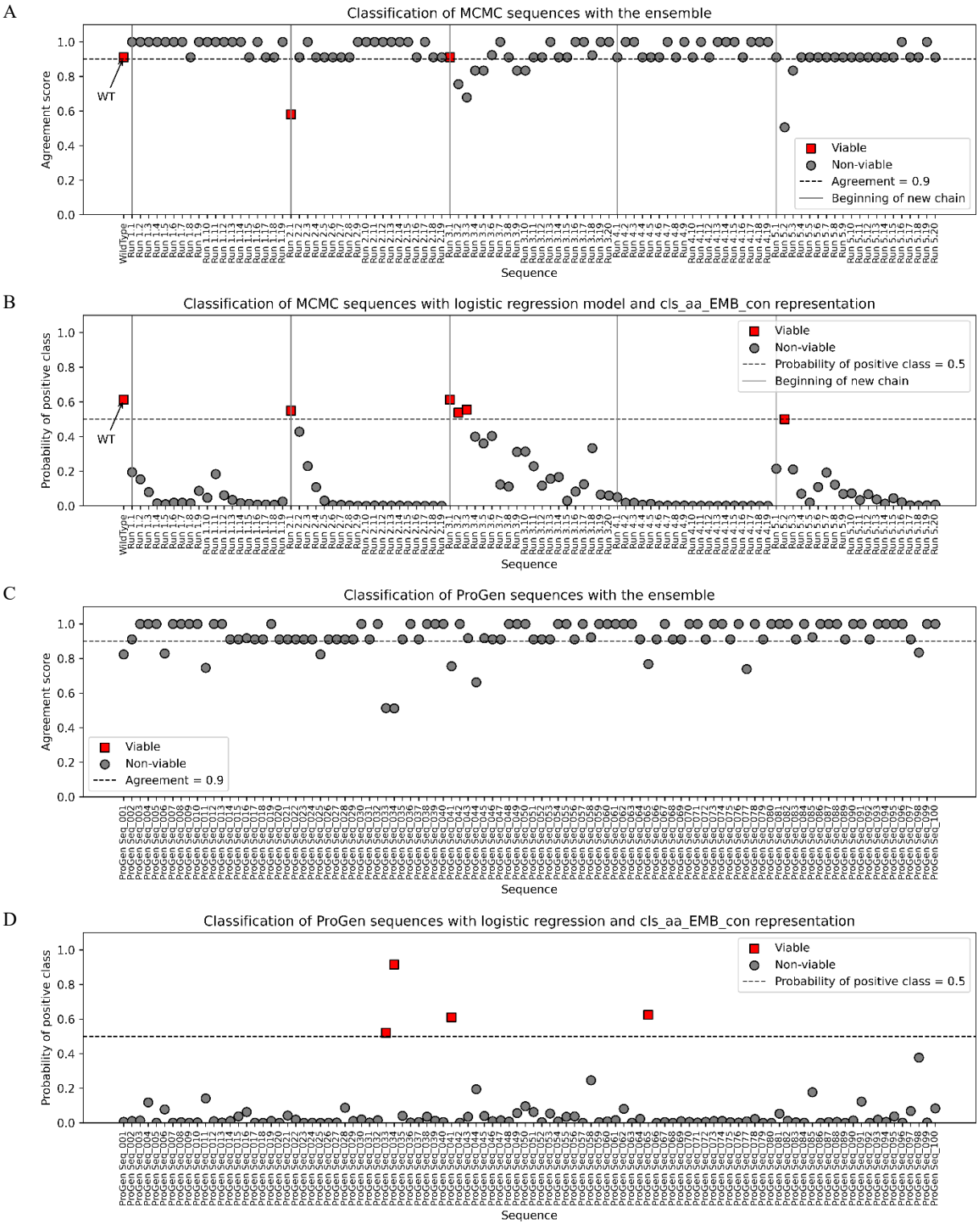

Figure 4.10. Classification of newly designed sequences. MCMC-generated sequences (A and B) and PLM-generated sequences (C and D) were evaluated with the ensemble (A and C) for class prediction and agreement score or with the logistic regression model using cls_aa_EMB_con as representation format (B and D). In the case of MCMC sequences list, the first sequence is the wild-type that was included as a control.

To gain deeper insight into these results, a final analysis of the mutation landscape of the generated sequences (whether produced by the MCMC generator or the generative PLM model) was conducted and compared to the mutation landscapes of both viable and non-viable sequences (Figure 4.11).

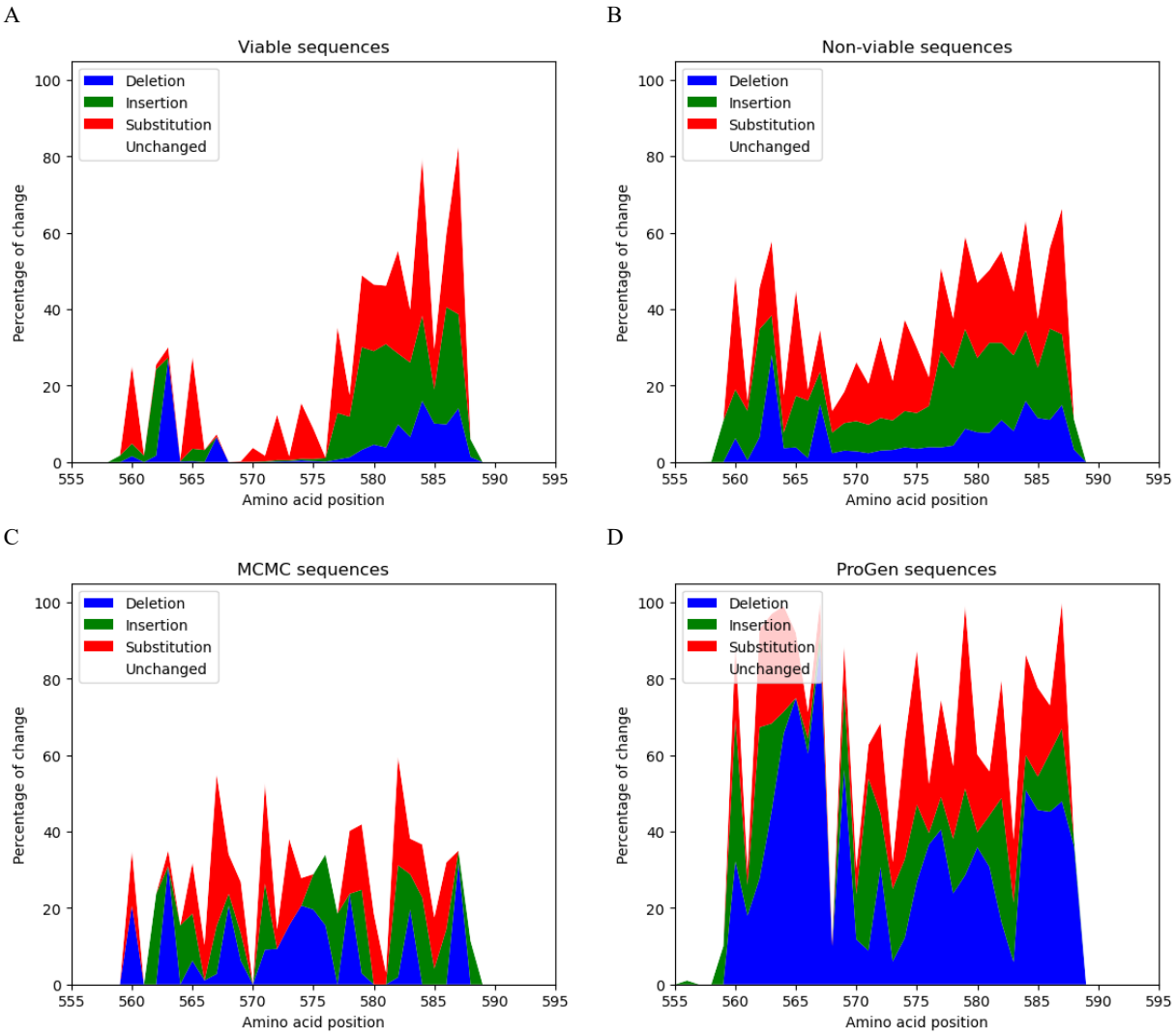

Figure 4.11. Mutation landscape of viable, non-viable, MCMC-generated and ProGen-generated sequences. Mutational landscape of viable (A), non-viable (B), MCMC-generated (C) and ProGen-generated sequences. Although the mutational landscapes of viable and non-viable sequences were previously presented in Figure 4.8, they are repeated here to facilitate direct comparison.

The analysis revealed that neither generation method could replicate the positional targeting signatures of positive sequences. Specifically, both methods failed to avoid mutations in the 567-576 region, a signature of viable sequences, and neither could mimic the nature of acceptable mutations in this area, where, when happening, they should be almost exclusively substitutions. This justifies why most generated sequences were classified as negative. In the case of MCMC-generated sequences, this outcome was expected, as discussed earlier, due to the simplicity of the likelihood function. Regarding the PLM-based generated pool, the low number of sequences classified as positive could be due to the fine-tuning process employed to build this mode, where only positive sequences were used. As a result, the model may not have

learnt to differentiate between positives and negatives and, while it may produce biologically plausible sequences within the 'protein grammar rules', these sequences may fail to exhibit the functional signatures found in the positive sequences. To improve performance, the PLM generative model could benefit from a supervised fine-tuning approach that includes both positive and negative sequences, with explicit functional labels on viability, that can be used as prompts for sequence generation. In addition, the intensity of targeting in PLM-generated sequences was found to be extremely high, in fact, higher than anything seen in the training dataset. Although this high diversity is advantageous for exploring a broad sequence space, the current setting may be excessively diverse, potentially straying beyond the boundaries of the viable design space. Therefore, this aspect should also be revisited in the follow-up of this work.

5. Conclusion and perspectives

This work aimed to enhance ML-based protein design to support the bioengineering of novel and diverse AAVs for human gene therapy by addressing challenges related to protein sequence representation formats and ML model selection for protein function prediction.

For representation format, we explored the use of PLM embeddings and concluded that the richness in biological information captured by this representation format provides advantages over traditional positional encoding methods, such as OHE. This superiority showed in improved performance at the test stage and better generalization capability for new unseen sequences. Within PLMs we selected ProtBERT, a state-of-the-art PLM encoder, for an in-depth evaluation of its embedding variants. This analysis offered valuable insights to guide the choice of suitable variants for future applications. The results showed that the cls_t_EMB representation consistently underperformed, limiting its effectiveness for classification tasks. In contrast, amino acid-based embeddings (aa_EMB and cls_aa_EMB_con) yielded the best overall performance, emphasizing the importance of capturing biological information for protein function prediction. The superior performance of cls_aa_EMB_con suggests that combining the contextual information from a global classification token (cls_raw_EMB) with the biological details from amino acid-level tokens (aa_EMB) enhances sequence-to-function learning. Therefore, for future applications requiring a single representation format, this concatenated approach is recommended as the preferred choice.

For ML model selection, we explored an ensemble approach that combined multiple models trained on different sequence representations. We found that this method provided strong classification performance and helped prioritize sequences for experimental validation, although it did not outperform the best individual model-representation pair. The performance of individual models, particularly those more susceptible to overfitting, may have been limited by the lack of more challenging ML-designed sequences during the train-validation stages, reducing the potential benefits of ensembling. Therefore, in the follow-up of this work, it will be valuable to incorporate more diverse and challenging sequences into the train-validation stage to assess the ensemble approach's true potential. Further strengthening regularization will also help mitigate overfitting, enhancing the overall performance and robustness of both individual models and ensemble strategies. Finally, expanding the ensemble to include other model types, such as multi-layer perceptrons or CNN-based classifiers, could further strengthen its predictive power.

The next phase of this work will involve evaluating newly designed sequences using the models developed here (or their future improved derivatives) to select candidates for laboratory testing. As the current state-of-the-art, we expect these sequences to be generated by generative PLMs, such as the one currently being developed in our group by Ferraz (2025) [67]. The first version of this model has already been used to generate the first pool of candidates that were evaluated in this study.

6. References

[1] D.N. Woolfson, A Brief History of De Novo Protein Design: Minimal, Rational, and Computational, Journal of Molecular Biology 433 (2021) 167160. https://doi.org/10.1016/j.jmb.2021.167160.

[2] N. Ferruz, M. Heinzinger, M. Akdel, A. Goncearenco, L. Naef, C. Dallago, From sequence to function through structure: Deep learning for protein design, Computational and Structural Biotechnology Journal 21 (2023) 238–250. https://doi.org/10.1016/j.csbj.2022.11.014.

[3] K.K. Yang, Z. Wu, F.H. Arnold, Machine-learning-guided directed evolution for protein engineering, Nat Methods 16 (2019) 687–694. https://doi.org/10.1038/s41592-019-0496-6.

[4] P.J. Ogden, E.D. Kelsic, S. Sinai, G.M. Church, Comprehensive AAV capsid fitness landscape reveals a viral gene and enables machine-guided design, Science 366 (2019) 1139–1143. https://doi.org/10.1126/science.aaw2900.

[5] A.D. Marques, M. Kummer, O. Kondratov, A. Banerjee, O. Moskalenko, S. Zolotukhin, Applying machine learning to predict viral assembly for adeno-associated virus capsid libraries, Molecular Therapy - Methods & Clinical Development 20 (2021) 276–286. https://doi.org/10.1016/j.omtm.2020.11.017.

[6] D.H. Bryant, A. Bashir, S. Sinai, N.K. Jain, P.J. Ogden, P.F. Riley, G.M. Church, L.J. Colwell, E.D. Kelsic, Deep diversification of an AAV capsid protein by machine learning, Nat Biotechnol 39 (2021) 691–696. https://doi.org/10.1038/s41587-020-00793-4.

[7] S. Sinai, E. Kelsic, G.M. Church, M.A. Nowak, Variational auto-encoding of protein sequences, (2017). https://doi.org/10.48550/ARXIV.1712.03346.

[8] G. Mikos, W. Chen, J. Suh, Machine Learning Identification of Capsid Mutations to Improve AAV Production Fitness, Bioinformatics, 2021. https://doi.org/10.1101/2021.06.15.447941.

[9] S. Lyu, S. Sowlati-Hashjin, M. Garton, Variational autoencoder for design of synthetic viral vector serotypes, Nat Mach Intell (2024). https://doi.org/10.1038/s42256-023-00787-2.

[10] Z. Han, N. Luo, F. Wang, Y. Cai, X. Yang, W. Feng, Z. Zhu, J. Wang, Y. Wu, C. Ye, K. Lin, F. Xu, Computer-Aided Directed Evolution Generates Novel AAV Variants with High Transduction Efficiency, Viruses 15 (2023) 848. https://doi.org/10.3390/v15040848.

[11] A. Vu Hong, L. Suel, E. Petat, A. Dubois, P.-R. Le Brun, N. Guerchet, P. Veron, J. Poupiot, I. Richard, An engineered AAV targeting integrin alpha V beta 6 presents improved myotropism across species, Nat Commun 15 (2024) 7965. https://doi.org/10.1038/s41467-024-52002-4.

[12] S.L. Ginn, M. Mandwie, I.E. Alexander, M. Edelstein, M.R. Abedi, Gene therapy clinical trials worldwide to 2023—an update, The Journal of Gene Medicine 26 (2024) e3721. https://doi.org/10.1002/jgm.3721.

[13] Z. Zhao, A.C. Anselmo, S. Mitragotri, Viral vector-based gene therapies in the clinic, Bioengineering & Transla Med 7 (2022) e10258. https://doi.org/10.1002/btm2.10258.

[14] J. Becker, J. Fakhiri, D. Grimm, Fantastic AAV Gene Therapy Vectors and How to Find Them—Random Diversification, Rational Design and Machine Learning, Pathogens 11 (2022) 756. https://doi.org/10.3390/pathogens11070756.

[15] M. Alloghani, D. Al-Jumeily, J. Mustafina, A. Hussain, A.J. Aljaaf, A Systematic Review on Supervised and Unsupervised Machine Learning Algorithms for Data Science, in: M.W. Berry, A. Mohamed, B.W. Yap (Eds.), Supervised and Unsupervised Learning for Data Science, Springer International Publishing, Cham, 2020: pp. 3–21. https://doi.org/10.1007/978-3-030-22475-2_1.

[16] V. Rani, S.T. Nabi, M. Kumar, A. Mittal, K. Kumar, Self-supervised Learning: A Succinct Review, Arch Computat Methods Eng 30 (2023) 2761–2775. https://doi.org/10.1007/s11831-023-09884-2.
[17] I.D. Mienye, Y. Sun, A Survey of Ensemble Learning: Concepts, Algorithms, Applications, and Prospects, IEEE Access 10 (2022) 99129–99149. https://doi.org/10.1109/ACCESS.2022.3207287.
[18] P.A. Flach, Tree models, in: Machine Learning: The Art and Science of Algorithms That Make Sense of Data, Cambridge University Press, Cambridge ; New York, 2012: p. 396.
[19] T. Chen, C. Guestrin, XGBoost: A Scalable Tree Boosting System, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, San Francisco California USA, 2016: pp. 785–794. https://doi.org/10.1145/2939672.2939785.
[20] J.H. Friedman, Greedy function approximation: A gradient boosting machine., Ann. Statist. 29 (2001). https://doi.org/10.1214/aos/1013203451.
[21] P.A. Flach, Probabilistic models, in: Machine Learning: The Art and Science of Algorithms That Make Sense of Data, Cambridge University Press, Cambridge ; New York, 2012: p. 396.
[22] L. von Rueden, S. Mayer, K. Beckh, B. Georgiev, S. Giesselbach, R. Heese, B. Kirsch, M. Walczak, J. Pfrommer, A. Pick, R. Ramamurthy, J. Garcke, C. Bauckhage, J. Schuecker, Informed Machine Learning - A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems, IEEE Trans. Knowl. Data Eng. (2021) 1–1. https://doi.org/10.1109/TKDE.2021.3079836.
[23] S.I. Omar, C. Keasar, A.J. Ben-Sasson, E. Haber, Protein Design Using Physics Informed Neural Networks, Biomolecules 13 (2023) 457. https://doi.org/10.3390/biom13030457.
[24] E. Argouarc'h, F. Desbouvries, E. Barat, E. Kawasaki, Generative vs. Discriminative modeling under the lens of uncertainty quantification, (2024). https://doi.org/10.48550/ARXIV.2406.09172.
[25] Y. Yang, A. Niroula, B. Shen, M. Vihinen, PON-Sol: prediction of effects of amino acid substitutions on protein solubility, Bioinformatics 32 (2016) 2032–2034. https://doi.org/10.1093/bioinformatics/btw066.
[26] L. Folkman, B. Stantic, A. Sattar, Y. Zhou, EASE-MM: Sequence-Based Prediction of Mutation-Induced Stability Changes with Feature-Based Multiple Models, Journal of Molecular Biology 428 (2016) 1394–1405. https://doi.org/10.1016/j.jmb.2016.01.012.
[27] S. Teng, A.K. Srivastava, L. Wang, Sequence feature-based prediction of protein stability changes upon amino acid substitutions, BMC Genomics 11 (2010) S5. https://doi.org/10.1186/1471-2164-11-S2-S5.
[28] L.-T. Huang, M.M. Gromiha, S.-Y. Ho, iPTREE-STAB: interpretable decision tree based method for predicting protein stability changes upon mutations, Bioinformatics 23 (2007) 1292–1293. https://doi.org/10.1093/bioinformatics/btm100.
[29] P. Koskinen, P. Törönen, J. Nokso-Koivisto, L. Holm, PANNZER: high-throughput functional annotation of uncharacterized proteins in an error-prone environment, Bioinformatics 31 (2015) 1544–1552. https://doi.org/10.1093/bioinformatics/btu851.
[30] L. De Ferrari, J.B. Mitchell, From sequence to enzyme mechanism using multi-label machine learning, BMC Bioinformatics 15 (2014) 150. https://doi.org/10.1186/1471-2105-15-150.
[31] R. Vanella, G. Kovacevic, V. Doffini, J. Fernández De Santaella, M.A. Nash, High-throughput screening, next generation sequencing and machine learning: advanced methods in enzyme engineering, Chem. Commun. 58 (2022) 2455–2467. https://doi.org/10.1039/D1CC04635G.
[32] D. Repecka, V. Jauniskis, L. Karpus, E. Rembeza, I. Rokaitis, J. Zrimec, S. Poviloniene, A. Laurynenas, S. Viknander, W. Abuajwa, O. Savolainen, R. Meskys, M.K.M. Engqvist, A. Zelezniak, Expanding functional protein sequence spaces using generative adversarial networks, Nat Mach Intell 3 (2021) 324–333. https://doi.org/10.1038/s42256-021-00310-5.

[33] S.R. Maddhuri Venkata Subramaniya, G. Terashi, A. Jain, Y. Kagaya, D. Kihara, Protein contact map refinement for improving structure prediction using generative adversarial networks, Bioinformatics 37 (2021) 3168–3174. https://doi.org/10.1093/bioinformatics/btab220.

[34] M. Karimi, S. Zhu, Y. Cao, Y. Shen, De Novo Protein Design for Novel Folds Using Guided Conditional Wasserstein Generative Adversarial Networks, J. Chem. Inf. Model. 60 (2020) 5667–5681. https://doi.org/10.1021/acs.jcim.0c00593.

[35] A. Hawkins-Hooker, F. Depardieu, S. Baur, G. Couairon, A. Chen, D. Bikard, Generating functional protein variants with variational autoencoders, PLoS Comput Biol 17 (2021) e1008736. https://doi.org/10.1371/journal.pcbi.1008736.

[36] A.J. Riesselman, J.B. Ingraham, D.S. Marks, Deep generative models of genetic variation capture the effects of mutations, Nat Methods 15 (2018) 816–822. https://doi.org/10.1038/s41592-018-0138-4.

[37] X. Ding, Z. Zou, C.L. Brooks Iii, Deciphering protein evolution and fitness landscapes with latent space models, Nat Commun 10 (2019) 5644. https://doi.org/10.1038/s41467-019-13633-0.

[38] J.G. Greener, L. Moffat, D.T. Jones, Design of metalloproteins and novel protein folds using variational autoencoders, Sci Rep 8 (2018) 16189. https://doi.org/10.1038/s41598-018-34533-1.

[39] P. Das, T. Sercu, K. Wadhawan, I. Padhi, S. Gehrmann, F. Cipcigan, V. Chenthamarakshan, H. Strobelt, C. Dos Santos, P.-Y. Chen, Y.Y. Yang, J.P.K. Tan, J. Hedrick, J. Crain, A. Mojsilovic, Accelerated antimicrobial discovery via deep generative models and molecular dynamics simulations, Nat Biomed Eng 5 (2021) 613–623. https://doi.org/10.1038/s41551-021-00689-x.

[40] R.R. Eguchi, C.A. Choe, P.-S. Huang, Ig-VAE: Generative modeling of protein structure by direct 3D coordinate generation, PLoS Comput Biol 18 (2022) e1010271. https://doi.org/10.1371/journal.pcbi.1010271.

[41] E.C. Alley, G. Khimulya, S. Biswas, M. AlQuraishi, G.M. Church, Unified rational protein engineering with sequence-based deep representation learning, Nat Methods 16 (2019) 1315–1322. https://doi.org/10.1038/s41592-019-0598-1.

[42] S. Biswas, G. Khimulya, E.C. Alley, K.M. Esvelt, G.M. Church, Low-N protein engineering with data-efficient deep learning, Nat Methods 18 (2021) 389–396. https://doi.org/10.1038/s41592-021-01100-y.

[43] A. Madani, B. Krause, E.R. Greene, S. Subramanian, B.P. Mohr, J.M. Holton, J.L. Olmos, C. Xiong, Z.Z. Sun, R. Socher, J.S. Fraser, N. Naik, Large language models generate functional protein sequences across diverse families, Nat Biotechnol (2023). https://doi.org/10.1038/s41587-022-01618-2.

[44] N. Ferruz, S. Schmidt, B. Höcker, ProtGPT2 is a deep unsupervised language model for protein design, Nat Commun 13 (2022) 4348. https://doi.org/10.1038/s41467-022-32007-7.

[45] J.-E. Shin, A.J. Riesselman, A.W. Kollasch, C. McMahon, E. Simon, C. Sander, A. Manglik, A.C. Kruse, D.S. Marks, Protein design and variant prediction using autoregressive generative models, Nat Commun 12 (2021) 2403. https://doi.org/10.1038/s41467-021-22732-w.

[46] D. Sgarbossa, U. Lupo, A.-F. Bitbol, Generative power of a protein language model trained on multiple sequence alignments, eLife 12 (2023) e79854. https://doi.org/10.7554/eLife.79854.

[47] R. Verkuil, O. Kabeli, Y. Du, B.I.M. Wicky, L.F. Milles, J. Dauparas, D. Baker, S. Ovchinnikov, T. Sercu, A. Rives, Language models generalize beyond natural proteins, (2022). https://doi.org/10.1101/2022.12.21.521521.

[48] E. Nijkamp, J. Ruffolo, E.N. Weinstein, N. Naik, A. Madani, ProGen2: Exploring the Boundaries of Protein Language Models, (2022). https://doi.org/10.48550/arXiv.2206.13517.

[49] A. Strokach, D. Becerra, C. Corbi-Verge, A. Perez-Riba, P.M. Kim, Fast and Flexible Protein Design Using Deep Graph Neural Networks, Cell Systems 11 (2020) 402-411.e4. https://doi.org/10.1016/j.cels.2020.08.016.

[50] A.J. Li, M. Lu, I. Desta, V. Sundar, G. Grigoryan, A.E. Keating, Neural network-derived Potts models for structure-based protein design using backbone atomic coordinates and tertiary motifs, Protein Science 32 (2023) e4554. https://doi.org/10.1002/pro.4554.

[51] H. Lu, D.J. Diaz, N.J. Czarnecki, C. Zhu, W. Kim, R. Shroff, D.J. Acosta, B.R. Alexander, H.O. Cole, Y. Zhang, N.A. Lynd, A.D. Ellington, H.S. Alper, Machine learning-aided engineering of hydrolases for PET depolymerization, Nature 604 (2022) 662–667. https://doi.org/10.1038/s41586-022-04599-z.

[52] F. Noé, S. Olsson, J. Köhler, H. Wu, Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning, Science 365 (2019) eaaw1147. https://doi.org/10.1126/science.aaw1147.

[53] J.B. Ingraham, M. Baranov, Z. Costello, K.W. Barber, W. Wang, A. Ismail, V. Frappier, D.M. Lord, C. Ng-Thow-Hing, E.R. Van Vlack, S. Tie, V. Xue, S.C. Cowles, A. Leung, J.V. Rodrigues, C.L. Morales-Perez, A.M. Ayoub, R. Green, K. Puentes, F. Oplinger, N.V. Panwar, F. Obermeyer, A.R. Root, A.L. Beam, F.J. Poelwijk, G. Grigoryan, Illuminating protein space with a programmable generative model, Nature (2023). https://doi.org/10.1038/s41586-023-06728-8.

[54] J.L. Watson, D. Juergens, N.R. Bennett, B.L. Trippe, J. Yim, H.E. Eisenach, W. Ahern, A.J. Borst, R.J. Ragotte, L.F. Milles, B.I.M. Wicky, N. Hanikel, S.J. Pellock, A. Courbet, W. Sheffler, J. Wang, P. Venkatesh, I. Sappington, S.V. Torres, A. Lauko, V. De Bortoli, E. Mathieu, S. Ovchinnikov, R. Barzilay, T.S. Jaakkola, F. DiMaio, M. Baek, D. Baker, De novo design of protein structure and function with RFdiffusion, Nature 620 (2023) 1089–1100. https://doi.org/10.1038/s41586-023-06415-8.

[55] J. Yim, B.L. Trippe, V. De Bortoli, E. Mathieu, A. Doucet, R. Barzilay, T. Jaakkola, SE(3) diffusion model with application to protein backbone generation, (2023). https://doi.org/10.48550/ARXIV.2302.02277.

[56] Y. Liu, S. Wang, J. Dong, L. Chen, X. Wang, L. Wang, F. Li, C. Wang, J. Zhang, Y. Wang, S. Wei, Q. Chen, H. Liu, De novo protein design with a denoising diffusion network independent of pretrained structure prediction models, Nat Methods 21 (2024) 2107–2116. https://doi.org/10.1038/s41592-024-02437-w.

[57] B. Ni, D.L. Kaplan, M.J. Buehler, Generative design of de novo proteins based on secondary-structure constraints using an attention-based diffusion model, Chem 9 (2023) 1828–1849. https://doi.org/10.1016/j.chempr.2023.03.020.

[58] S. Alamdari, N. Thakkar, R. Van Den Berg, N. Tenenholtz, R. Strome, A.M. Moses, A.X. Lu, N. Fusi, A.P. Amini, K.K. Yang, Protein generation with evolutionary diffusion: sequence is all you need, (2023). https://doi.org/10.1101/2023.09.11.556673.

[59] J.S. Lee, J. Kim, P.M. Kim, Score-based generative modeling for de novo protein design, Nat Comput Sci 3 (2023) 382–392. https://doi.org/10.1038/s43588-023-00440-3.

[60] R. Furukawa, W. Toma, K. Yamazaki, S. Akanuma, Ancestral sequence reconstruction produces thermally stable enzymes with mesophilic enzyme-like catalytic properties, Sci Rep 10 (2020) 15493. https://doi.org/10.1038/s41598-020-72418-4.

[61] G. Foley, A. Mora, C.M. Ross, S. Bottoms, L. Sützl, M.L. Lamprecht, J. Zaugg, A. Essebier, B. Balderson, R. Newell, R.E.S. Thomson, B. Kobe, R.T. Barnard, L. Guddat, G. Schenk, J. Carsten, Y. Gumulya, B. Rost, D. Haltrich, V. Sieber, E.M.J. Gillam, M. Bodén, Engineering indel and substitution variants of diverse and ancient enzymes using Graphical Representation of Ancestral Sequence

Predictions (GRASP), PLoS Comput Biol 18 (2022) e1010633. https://doi.org/10.1371/journal.pcbi.1010633.
[62] W.P. Russ, M. Figliuzzi, C. Stocker, P. Barrat-Charlaix, M. Socolich, P. Kast, D. Hilvert, R. Monasson, S. Cocco, M. Weigt, R. Ranganathan, An evolution-based model for designing chorismate mutase enzymes, Science 369 (2020) 440–445. https://doi.org/10.1126/science.aba3304.
[63] J. Trinquier, G. Uguzzoni, A. Pagnani, F. Zamponi, M. Weigt, Efficient generative modeling of protein sequences using simple autoregressive models, Nat Commun 12 (2021) 5800. https://doi.org/10.1038/s41467-021-25756-4.
[64] P. Tian, J.M. Louis, J.L. Baber, A. Aniana, R.B. Best, Co-Evolutionary Fitness Landscapes for Sequence Design, Angew Chem Int Ed 57 (2018) 5674–5678. https://doi.org/10.1002/anie.201713220.
[65] A. Strokach, P.M. Kim, Deep generative modeling for protein design, Current Opinion in Structural Biology 72 (2022) 226–236. https://doi.org/10.1016/j.sbi.2021.11.008.
[66] A. Winnifrith, C. Outeiral, B.L. Hie, Generative artificial intelligence for de novo protein design, Current Opinion in Structural Biology 86 (2024) 102794. https://doi.org/10.1016/j.sbi.2024.102794.
[67] J.L. Ferraz, Deep learning to optimize viral vector production for human gene therapy, MSc Thesis, University of Lisbon, 2025.
[68] A.M. Szalkowski, M. Anisimova, Markov Models of Amino Acid Substitution to Study Proteins with Intrinsically Disordered Regions, PLoS ONE 6 (2011) e20488. https://doi.org/10.1371/journal.pone.0020488.
[69] N. Terrapon, O. Gascuel, É. Maréchal, L. Bréhélin, Fitting hidden Markov models of protein domains to a target species: application to Plasmodium falciparum, BMC Bioinformatics 13 (2012) 67. https://doi.org/10.1186/1471-2105-13-67.
[70] D. Harding-Larsen, J. Funk, N.G. Madsen, H. Gharabli, C.G. Acevedo-Rocha, S. Mazurenko, D.H. Welner, Protein representations: Encoding biological information for machine learning in biocatalysis, Biotechnology Advances 77 (2024) 108459. https://doi.org/10.1016/j.biotechadv.2024.108459.
[71] D. Ofer, N. Brandes, M. Linial, The language of proteins: NLP, machine learning & protein sequences, Computational and Structural Biotechnology Journal 19 (2021) 1750–1758. https://doi.org/10.1016/j.csbj.2021.03.022.
[72] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S.A.A. Kohl, A.J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A.W. Senior, K. Kavukcuoglu, P. Kohli, D. Hassabis, Highly accurate protein structure prediction with AlphaFold, Nature 596 (2021) 583–589. https://doi.org/10.1038/s41586-021-03819-2.
[73] Y. Bengio, A. Courville, P. Vincent, Representation Learning: A Review and New Perspectives, (2012). https://doi.org/10.48550/ARXIV.1206.5538.
[74] A. Elnaggar, M. Heinzinger, C. Dallago, G. Rihawi, Y. Wang, L. Jones, T. Gibbs, T. Feher, C. Angerer, M. Steinegger, D. Bhowmik, B. Rost, ProtTrans: Towards Cracking the Language of Life's Code Through Self-Supervised Deep Learning and High Performance Computing, (2020). https://doi.org/10.48550/ARXIV.2007.06225.
[75] A. Rives, J. Meier, T. Sercu, S. Goyal, Z. Lin, J. Liu, D. Guo, M. Ott, C.L. Zitnick, J. Ma, R. Fergus, Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences, Proc. Natl. Acad. Sci. U.S.A. 118 (2021) e2016239118. https://doi.org/10.1073/pnas.2016239118.

[76] R. Rao, N. Bhattacharya, N. Thomas, Y. Duan, X. Chen, J. Canny, P. Abbeel, Y.S. Song, Evaluating Protein Transfer Learning with TAPE, Adv Neural Inf Process Syst 32 (2019) 9689–9701.
[77] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, (2019). https://doi.org/10.48550/arXiv.1810.04805.
[78] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention Is All You Need, (2017). https://doi.org/10.48550/ARXIV.1706.03762.
[79] S. Sinai, N. Jain, G.M. Church, E.D. Kelsic, Generative AAV capsid diversification by latent interpolation, Synthetic Biology, 2021. https://doi.org/10.1101/2021.04.16.440236.
[80] The Hugging Face Cummunity, Transformers: State-of-the-Art Natural Language Processing, (2019). https://github.com/huggingface/transformers.
[81] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An Imperative Style, High-Performance Deep Learning Library, (2019). https://doi.org/10.48550/ARXIV.1912.01703.
[82] F. Anowar, S. Sadaoui, B. Selim, Conceptual and empirical comparison of dimensionality reduction algorithms (PCA, KPCA, LDA, MDS, SVD, LLE, ISOMAP, LE, ICA, t-SNE), Computer Science Review 40 (2021) 100378. https://doi.org/10.1016/j.cosrev.2021.100378.
[83] A.M. Ikotun, A.E. Ezugwu, L. Abualigah, B. Abuhaija, J. Heming, K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data, Information Sciences 622 (2023) 178–210. https://doi.org/10.1016/j.ins.2022.11.139.
[84] L. van der Maaten, H. Geoffrey, Visualizing data using t-sne, Journal of Machine Learning Research 9 (2008) 2579–2605.
[85] S. Chib, E. Greenberg, Understanding the Metropolis-Hastings Algorithm, The American Statistician 49 (1995) 327–335. https://doi.org/10.1080/00031305.1995.10476177.
[86] P. Wu, W. Xiao, T. Conlon, J. Hughes, M. Agbandje-McKenna, T. Ferkol, T. Flotte, N. Muzyczka, Mutational Analysis of the Adeno-Associated Virus Type 2 (AAV2) Capsid Gene and Construction of AAV2 Vectors with Altered Tropism, J Virol 74 (2000) 8635–8647. https://doi.org/10.1128/JVI.74.18.8635-8647.2000.

Annexes

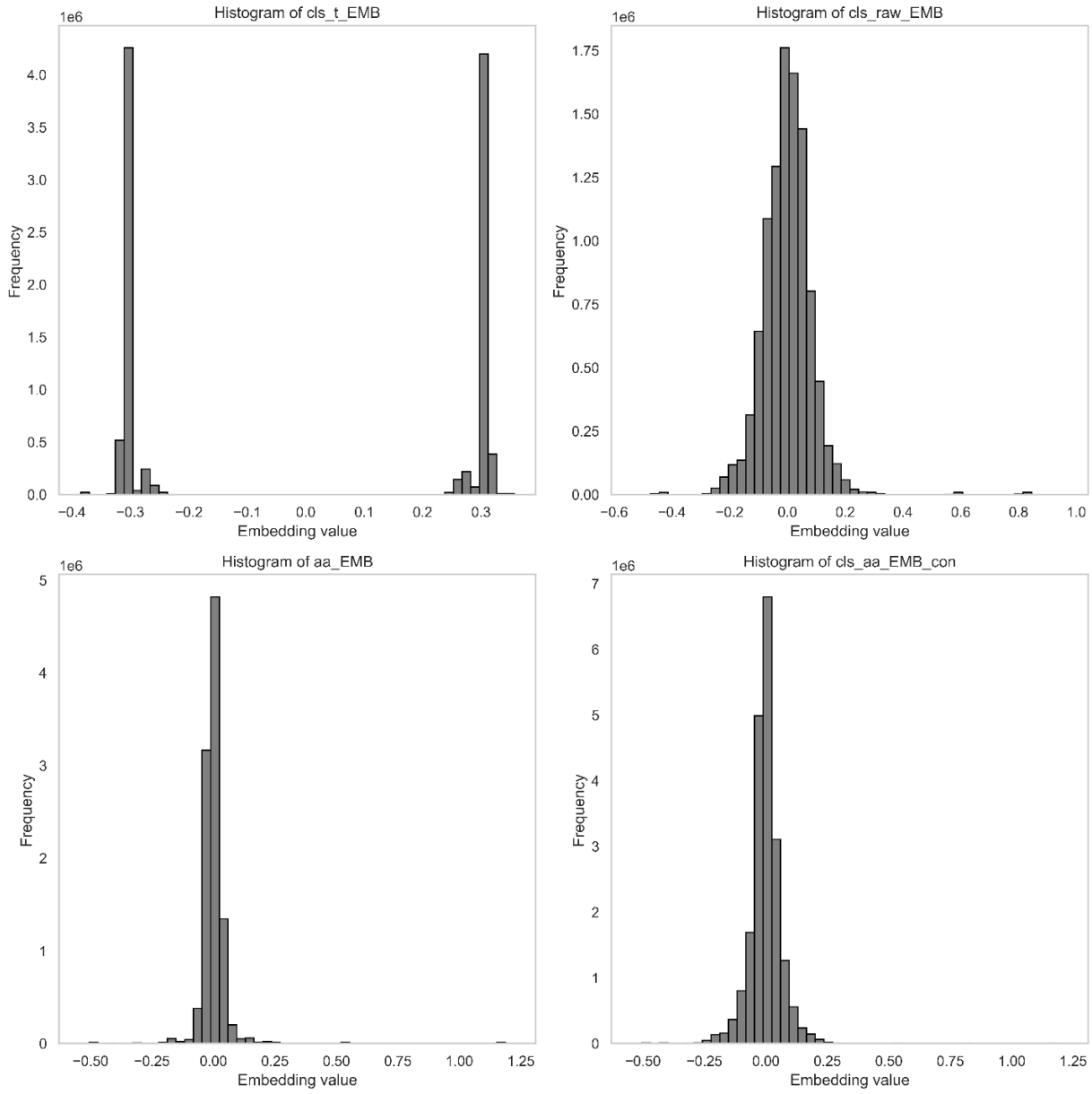

Figure A1. Value distribution of ProtBERT-generated embeddings