University of Stuttgart
Germany

SimTech

Institute of Biochemistry
Department of Technical Biochemistry

Simulation Technology

Bachelor Thesis

SIMULATION TECHNOLOGY DEGREE COURSE
Submitted to the University of Stuttgart

Understanding antibiotic resistance by machine learning

Niklas Abraham

Supervisor:	Prof. Dr. Jürgen Pleiss
Second Supervisor:	Prof. Dr. Steffen Staab
Matriculation No.:	3674080
SimTech Nr.:	203
Submission date:	30. September 2025

Abstract

Antibiotic resistance represents a critical global health challenge, with antimicrobial resistance directly responsible for 1.27 million deaths in 2019. This study addresses the prediction of antibiotic resistance phenotypes from protein sequence data using machine learning approaches, focusing on TEM-1 β-lactamases where single mutations dramatically alter resistance profiles.

We developed a comprehensive framework integrating biological databases (NCBI, CARD, UniProt) with state-of-the-art protein language models to predict resistance phenotypes. Using 139 TEM variants across four phenotype classes (2b, 2be, 2ber, 2br), we achieved prediction accuracies up to 97.6% through transfer learning approaches.

Key findings demonstrate that protein language models encode functionally relevant information without explicit biochemical supervision. Self-similarity weighted pooling strategies identified functionally critical regions, including catalytic residues and active site loops, through purely data-driven analysis. Biochemically informed aggregation strategies significantly outperformed traditional mean pooling approaches, with weighted position strategies achieving 84.5% accuracy compared to 61.9% for mean pooling.

Latent space analysis revealed that protein language models organize biochemically relevant information in a highly concentrated manner, with the top 10 embedding dimensions accounting for over 80% of predictive importance. A supervised basis transformation showed strong correlation (0.794) between the first principal component and self-similarity weights, indicating functional organization within the embedding space.

These results provide first evidence for a "protein grammar" governing how amino acid sequences encode functional properties, offering new insights into protein sequence-function relationships. The work demonstrates that integration of domain knowledge with machine learning methodologies can yield insights greater than the sum of their parts, opening new avenues for addressing antibiotic resistance challenges through computational approaches.

Table of Contents

1. Introduction

One of the major challenges in the field of global healthcare is antibiotic resistance. Estimates suggest that the antimicrobial resistance was directly responsible for 1.27 million deaths in 2019 [1]. Through the widely spread practice of prescribing antibiotics the number of resistant bacteria is expected to rise in the coming years [2].

The most commonly used antibiotics are so called β-lactam antibiotics. These antibiotics work by interfering with the synthesis of the bacterial cell wall [3]. One of the first enzymes which were found and classified into the same class are the TEM-1 β-lactamases, today, 250 TEM-like enzymes are known. Many of these enzymes vary only in one or two mutations, which makes them sequence wise similar, but have different phenotypes when it comes to antibiotic resistance [4]. This makes the sequence space very dense and therefore new analytical approaches are needed to be able to make predictions about the antibiotic resistance of the enzymes.

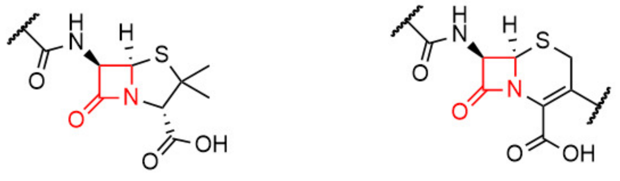
New popular approaches in machine learning have been used to make a variety of predictions concerning protein structure (e.g. AlphaFold [5]), protein function (e.g. ESM-3 [6]) and protein classification. These methods make it possible to analyze and extract insights from data that has been collected over decades, potentially leading to new discoveries in the field of antibiotic resistance. This has led to a revolution in the field of enzymes and proteins with the use of machine learning.

This thesis combines established biological databases such as the NCBI non-redundant database, CARD, and UniProt with state-of-the-art protein language models to develop computational tools for predicting resistance phenotypes from sequence data. By integrating domain knowledge with machine learning approaches, this work validates that protein language models encode functionally relevant information for biochemical prediction tasks. The findings provide evidence for a "protein grammar" that governs how amino acid sequences encode functional properties, opening new avenues for addressing challenges in medicine while advancing computational biology.

2. Theoretical Background

2.1. Antibiotic Resistance

Antibiotic resistance is defined as the ability of bacteria, viruses, fungi and parasites to resist the antimicrobial medicines like for example antibiotics [1]. Most commonly used antibiotics are the so called β-lactam antibiotics which contain a β-lactam ring in their chemical structure [7]. The β-lactam ring is a four membered ring with a carbonyl group. This includes penicillin, cephalosporin, carbapenems, which are the most common antibiotics used today [2]. The core structure of penicillin derivatives **1** and cephalosporin derivatives **2** are shown in Figure 2.1.



1 **2**

Figure 2.1.: Core structure of β-lactam antibiotics. The four-membered β-lactam ring is the key structural element responsible for the antibiotic activity (marked in red). Structure drawn with ChemDraw by Arthur Sieglin.

The β-lactam antibiotics exert their bactericidal effect by targeting the bacterial cell wall. They mimic the D-Ala-D-Ala moiety of the peptidoglycan layer of the cell wall, which is the main component of the cell wall in bacteria. The peptidoglycan layer is built from carbohydrates of repeating units N-acetylglucosamine (GlcNAc) and N-acetylmuramic acid (MurNAc), which are linked by a diverse component. Normally, penicillin-binding proteins (PBPs) catalyze the transpeptidation reaction where the active site serine attacks the carbonyl of the terminal D-Ala, releasing the last D-Ala and forming an acyl-enzyme intermediate. β-lactam antibiotics act as mechanism-based inhibitors because they are structural mimics of this D-Ala-D-Ala moiety. The weakened cell is no longer able to withstand the osmotic pressure, leading to the cell lysis and death [8]. Due to evolutionary pressure, bacteria can develop resistance to the β-lactam antibiotics. The enzymes in both Gram-positive and Gram-negative bacteria which are responsible for the resistance are called β-lactamases. Through hydrolysis, the enzymes break the β-lactam ring, which is the key structure of the β-lactam antibiotics [9]. Based on their catalytic mechanism, the β-lactamases are classified into serine β-lactamases and metallo-β-lactamases. The former employ an active site with a serine residue, while the latter have a metal ion in their active site (typically zinc) [8] [9]. Based on a proposed classification by sequence homology the β-lactamases are classified into 4 classes, A, B, C and D. The classes A, C, and D include the serine β-lactamases, while the class B contains the metallo-β-lactamases [7]. One of the first β-lactamases found was an enzyme called TEM-1, named after the first three letters of the patient's name [10]. Today there are 263 TEM-like β-lactamases described and classified based on their phenotypes, those are available in databases like BLDB [4]. Although many of the enzymes vary only by one or two mutations, they have different phenotypes, different protein stabilities and different susceptibilities to antibiotics.

The functional classification of β-lactamases is based on their substrate specificity and inhibitor susceptibility, leading to distinct phenotypes that have important clinical implications.

Table 2.1.: Comparison of β-lactamase phenotypes and their substrate specificity

Phenotype	**Substrate Specificity**	**Examples**
2b	Penicillins, early cephalosporins	TEM-1, TEM-2, SHV-1
2br	Penicillins	Inhibitor-resistant TEMs (IRTs)
2be	Penicillins, 3rd-gen cephalosporins, monobactams	ESBLs (TEM, SHV, CTX-M)
2ber	Penicillins, 3rd-gen cephalosporins, monobactams	TEM-50, TEM-68, SHV variants

The most common phenotypes observed in TEM type β-lactamases are described below: Phenotype 2b refers to the classical TEM enzymes that are capable of hydrolyzing penicillins and early cephalosporins. These enzymes are inhibited by clavulanate, sulbactam, and tazobactam. Well known examples of this phenotype include TEM-1, TEM-2, and SHV-1. In clinical settings, bacteria expressing this phenotype are resistant to ampicillin, amoxicillin, and first generation cephalosporins, but they remain susceptible to combinations of β-lactam antibiotics with β-lactamase inhibitors (BLIs) [11]. Phenotype 2br, where "r" indicates resistance to inhibitors, describes enzymes that have evolved from the 2b group through point mutations that confer resistance to clavulanate and other inhibitors. These inhibitor resistant TEMs (IRTs) continue to hydrolyze penicillins but are only poorly inhibited by clavulanic acid, sulbactam, and sometimes tazobactam. As a result, this phenotype can lead to treatment failures when using amoxicillin clavulanate or ticarcillin clavulanate combinations [12]. Phenotype 2be, with "e" standing for extended spectrum, encompasses extended spectrum β-lactamases (ESBLs) that have evolved from 2b TEMs. Mutations in these enzymes broaden their activity to include third generation cephalosporins such as cefotaxime and ceftazidime, as well as monobactams like aztreonam, while their activity against carbapenems remains low. These enzymes are still inhibited by clavulanate in vitro, which is an important diagnostic feature for ESBLs. Clinically, this phenotype is a major resistance mechanism in Enterobacterales against expanded spectrum cephalosporins [13]. Phenotype 2c is characterized by TEM enzymes that hydrolyze carbenicillin, showing activity against both carbenicillin and ticarcillin, while still being inhibited by clavulanate [11]. Phenotype 2d refers to oxacillinases, also known as OXA type enzymes, which hydrolyze cloxacillin and oxacillin more efficiently than other substrates and are only weakly inhibited by clavulanate [11].

2.1.1. Mutations of β-lactamases and their effects

To understand how mutations affect the function of TEM-type class A β-lactamases, it is important to first consider their conserved catalytic architecture. These enzymes are built around key structural features, including Ser70 (the nucleophile), Lys73 and Lys234 (which form part of the general acid/base network), the SDN loop (Ser130-Asp131-Asn132), and the Ω-loop (Arg164-Asp179) that contains the general base Glu166 (Figure 2.2).

Changes at these critical sites, as well as compensatory substitutions elsewhere in the protein, can influence the enzyme's substrate spectrum, susceptibility to inhibitors, and overall protein stability.

Glu166 activates the deacylating water via a Glu166-(H_2O)-Asn170 hydrogen-bond network and also participates in acylation state proton transfers. Substituting Glu166 severely reduces turnover: Glu166Ala/Cys/Asp lower catalytic efficiency despite preserved binding, consistent with a deacylation defect [14, 15]. Mechanistically, the cis peptide at Glu166-Pro167 and the short 167-170 α-turn are required to hold the catalytic water in place [15].

Arg164 forms a clamp with Asp179 (and Glu171) that narrows the active-site entrance. Clinical ESBL substitutions at this position Arg164Ser/His/Cys disrupt or weaken that clamp, enlarge the cavity, and selectively boost hydrolysis of third-generation cephalosporins. Arg164Ser increases catalytic efficiency for cefotaxime and ceftazidime, while reducing ampicillin activity; Arg164His/Cys behave similarly [15]. Loop destabilization and reduced conformational heterogeneity of the active site likely contribute to these ESBL phenotypes. Trp165 substitutions are rarer but functionally significant. Trp165Arg rewires loop electrostatics (forming an Arg165-Glu168 salt bridge), which alters ligand recognition and can confer resistance to the suicide inhibitor clavulanate (an IRT-like behavior) by perturbing inhibitor binding while retaining penicillinase activity [15].

M69I, adjacent to the catalytic Ser70 and SDN loop, is a hallmark of inhibitor-resistant TEMs (IRTs). It decreases the acylation efficiency of β-lactamase inhibitors (e.g., clavulanate),

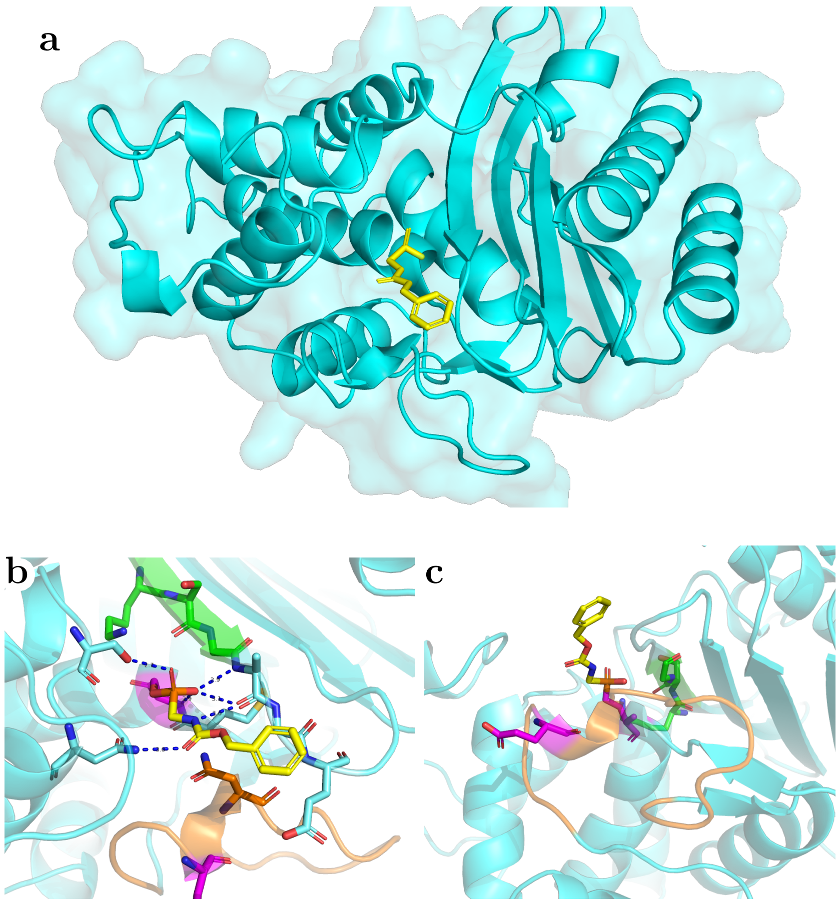


Figure 2.2.: a) Overall 3D structure with the enzyme in cyan and penicillin in yellow. b) Binding site residues (green) in contact with penicillin (yellow) within 4 Å, active site in magenta. c) The Ω-loop region (orange) is shown. The tool used to generate the structures is Pymol.

thereby reducing susceptibility to mechanism-based inhibition. The trade-off is reduced global stability and, in some backgrounds, a modest penalty on penicillin turnover [16, 17]. In clinical and directed-evolution lineages, M69I is frequently paired with the global stabilizer M182T to restore folding robustness. G238S, located on the β-strand bordering the active site, reshapes the oxyanion region and increases flexibility near the Ω-loop, improving accommodation of oxyimino side chains. E104K, on the outer wall of the binding pocket, creates favorable electrostatics for cephalosporin C7 substituents. Together, E104K+G238S synergize to yield high cefotaxime/ceftazidime activity; the clinically common TEM-52 (E104K/G238S/M182T) exemplifies this combination [18, 15]. Both E104K and G238S destabilize the fold; M182T is commonly co-selected to compensate [17].

M182T, distant from the active site, is a global suppressor that increases apparent stability in TEM-1 and rescues diverse destabilizing lesions, including ESBL (G238S, Arg164Ser) and IRT (M69I) backgrounds and even a core-defect lethal mutation L76N [19, 20, 17]. Crystallography and MD indicate no single invariant local H-bond explains the effect; instead, M182T tunes long-range packing and solvent networks across the interdomain interface [17, 14]. Its repeated emergence in directed evolution for third-generation cephalosporin hydrolysis underscores the generality of this stabilizing role [18, 21]. L76N, buried in the hydrophobic core, is folding-lethal by itself but becomes viable when combined with M182T, demonstrating epistasis between global stability centers and active-site remodeling mutations [19, 20]. Lys234 (with neighbors in the 232-234 region) participates in the proton-relay network during acylation and is highly constrained; viable substitutions are rare. Together with Lys73 and the SDN loop, this region couples to the Ω-loop and helps set the catalytic water geometry. Perturbations here tend to be deleterious rather than adaptive, consistent with the strong conservation observed across class A enzymes [15].

Across TEM evolution, catalytic-spectrum gains (ESBL via Arg164→Ser/His/Cys, G238S; IRT via M69I; third-generation cephalosporins via E104K+G238S) frequently incur a stability cost. Selection repeatedly pairs such activity-enhancing yet destabilizing mutations with global stabilizers (M182T), yielding functional enzymes within the narrow stability-activity window accessible to marginally stable proteins [17]. The Ω-loop acts as an allosteric gate: its topology is conserved but locally flexible; mutations that loosen its "bottleneck" (Arg164 variants) expand spectrum, whereas disabling its general base (Glu166) cripples catalysis [15].

2.2. Machine Learning

Machine Learning is a subfield of artificial intelligence (AI) and can be broadly classified into three categories: supervised learning, unsupervised learning and reinforcement learning. In all of these tasks a model is trained on a dataset and then used to predict a target. The process of training a model is called learning and when using a model to predict a target, it is called inference. Depending on the dataset the model is trained on, the task can be classified in one of the three before mentioned categories. In supervised learning, the model is trained on a labeled dataset, where the input often called features and the output called labels are given. The model is trained to predict the labels for new inputs [22]. A simple example could be a classifier which predicts whether an image depicts a dog or a cat. Unsupervised learning is the task of detecting patterns in a dataset, which only consists of features and holds no labels. The model therefore tries to find meaningful differences in the data and group those together [22]. An example are the clustering of movies into genres. In reinforcement learning the model (called agent) is placed in an environment and is given a set of actions it can perform at each step. The agent changes the state of the environment by these actions, and gets a reward after each step. The goal of the agent is to maximize the reward over time [22]. A self driving car is an example for a reinforcement learning agent. There exist some other categories of learning, but they are mixtures of the three before mentioned categories. In this work a mixture of supervised and unsupervised learning is used.

2.2.1. Short Overview of Unsupervised Learning Methods

Throughout this work various unsupervised learning methods are used to analyze and visualize data. One of the used methods is the Support Vector Machine (SVM), it introduces the maximum margin principle, which is a way to find the best hyperplane between data in a high dimensional feature space [23]. For the analysis of high dimensional spaces both PCA, t-SNE and UMAP are used. The principal component analysis (PCA) is an orthogonal linear transformation, that maximises the sample variance in the low dimensional space [24]. t-SNE is a probabilistic non-linear dimensionality reduction technique, that is used to visualize neighbouring points in a high dimensional space [25]. UMAP builds a k-nearest neighbour graph and interprets it as a fuzzy topological representation of a manifold (Uniform Manifold Approximation and Projection), the low dimensional embedding is found by minimising the cross-entropy between the fuzzy topological representation and the low dimensional embedding [26].

The key method used in this thesis are transformer models, which are a type of neural network. Transformer models have gained popularity in the last years through their ability to process sequence data of any kind. The most famous examples are models like ChatGPT (the GPT stands for Generative Pre-trained Transformer).

2.2.2. Introduction to Neural Networks

Neural networks are models inspired by animal brains, functioning as interconnected graphs while the neurons act as nodes, similar to their counterparts in nature [27]. In these networks, the neurons are connected, forming a graph where the edges represent the weights of the connections between them. With the help of these weights and an additional parameter, the biases, the network is capable of learning complex patterns and hidden relationships in the data.

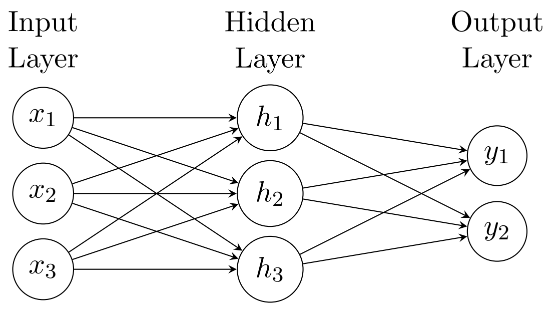


Figure 2.3.: Schematic representation of a neural network with three input neurons, three hidden neurons, and two output neurons. The arrows represent weighted connections between neurons.

Each neuron in the network receives a set of inputs from the previous layer and applies a weighted sum of these inputs plus a bias. This result is then passed through an activation function σ, which allows the network to learn non-linear relationships between the inputs. The most common activation functions are the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ and the rectified linear unit (ReLU) function $\sigma(x) = \max(0, x)$. The calculation is done with the weight matrix $\mathbf{W}$, in which each element w_{ij} represents the weight of the connection between the i-th input neuron and the j-th hidden neuron. The bias vector $\mathbf{b}$ is added to the weighted sum to account for the bias of the neuron. The value from the previous layer is represented by a scalar a. Thus the output for the j-th hidden neuron is calculated as:

$$h_j = \sigma\left(\sum_{i=1}^{n} w_{ij} a_i + b_j\right) = \sigma\left((\mathbf{Wa})_j + b_j\right) \tag{2.1}$$

this result h_j is then passed on to the next layer and so forth. This process is called forward propagation and is repeated until the final output is reached.

The goal of the training process is to find the weights and biases which result in the best performance of the network. The network's performance is measured by a loss function, which is a measure of the difference between the predicted and the true output $L(y_{predicted}, y_{true})$. The loss function is then minimized by adjusting the weights and biases of the network. This is done using the gradient descent algorithm, which calculates the derivative of the loss function with respect to the weights and biases and adjusts the weights and biases in the direction of the negative gradient. This is done using the chain rule of calculus [28]. In this work the optimizer used was Adam [29], it uses the first and second moment of the gradient to adjust the learning rate of the network. The issue with a simple neural network like the one shown in Figure 2.3 is that at a depth of ~ 50 layers the gradients vanish and numerical instabilities occur.

Another architectural design issue from these feed forward networks was the missing of a way to capture temporal dependencies between inputs, aka a word comes before a word in a sentence and therefore changes the meaning of the sentence. A solution was the Recurrent neural network (RNNs) where the output of the network is fed back into the network as input. This allows the network to capture the temporal dependencies between inputs [30].

2.2.3. Mathematical formulation of sequence representation

One of the most important tasks in machine learning and this work is the representation of sequences. A sequence is a set of elements, which are ordered and have a temporal dependency, a sequence of length $n \in \mathbb{N}$ can be denoted as $\mathbf{p} = (p_1, p_2, \ldots, p_n) \in \mathcal{P}^n$. The possible elements of the sequence are in the set $\mathcal{P}$, which can hold words from a sentence or amino acids from a protein sequence. In most cases the input is not a mathematical object like a vector, but a list of string elements. To represent these string elements as a vector an embedding is used in which each element is mapped to a predefined vector. In the case of natural language the words are mapped based on their similarity in meaning to other words, e.g. the word "dog" is mapped to a vector which is similar to the vector of the word "cat". In the case of protein sequences the amino acids are mapped to a vector which is similar to the vector of the amino acid which is most similar to it. All the possible sequences of arbitrary lengths are found in the set

$$\mathcal{X} = \bigcup_{n=1}^{\infty} \mathcal{P}^n. \tag{2.2}$$

The goal of any sequence-level representation is to find a function

$$\psi(\cdot\,;\theta_\psi) : \mathcal{X} \to \mathbb{R}^d \tag{2.3}$$

which maps a sequence to a vector in a dimensional space. This function is parameterized by a finite set of parameters $\theta_\psi \in \Theta_\psi$. The function or model ψ must be able to handle sequences of arbitrary length $n \in \mathbb{N}$ and shall therefore be independent of the dimension of the output space (called embedding space) and parameter set size $|\theta_\psi| < \infty$ [31].

To solve this problem, encoder decoder RNNs were introduced [32]. The encoder takes the input sequence $\mathbf{p} \in \mathcal{X}$ and maps it to a vector $\mathbf{h} \in \mathbb{R}^d$. The decoder is then tasked to map the vector $\mathbf{h}$ back to a sequence $\mathbf{q} \in \mathcal{Q}^m$. With the goal of the output sequence to be the same as the input sequence. The obstacle is that the encoder and decoder are not able to utilize parallel processing of all the elements of the sequence at once. Additionally the fixed length of the hidden states of the encoder and decoder are not able to capture the long term dependencies in the sequence. The elements of the sequence are processed one by one, which is why the encoder decoder RNNs are not able to handle long sequences efficiently. To tackle this problem the content based attention mechanism was introduced [33]. It lets the decoder focus on some parts of the input sequence more than others, this is done through a weight for each hidden state of the encoder. If every position attends to all positions in the same sequence, the weights are

$$\alpha_{ij} = \frac{e^{q_i \cdot k_j}}{\sum_{k=1}^{n} e^{q_i \cdot k_k}} \tag{2.4}$$

where q_i is the query vector for the i-th position in the sequence, k_j is the key vector for the j-th position in the sequence. The attention mechanism is then used to weight the hidden states of the encoder. The weighted hidden states are then summed up to get the context vector c.

$$c = \sum_{i=1}^{n} \alpha_i h_i \tag{2.5}$$

where α_i is the weight for the i-th hidden state. The context vector c is then used as input for the decoder. This attention mechanism was then introduced in transformer models [34].

2.2.4. Transformer Architecture

Transformer models dispense with the recurrent nature of the encoder decoder RNNs. They rely on the self-attention mechanism to capture mid or long range relationships in the sequence $\mathbf{p} \in \mathcal{X}$ between positions p_i and p_j with $i, j \in \mathbb{N}$ and $i \neq j$ [34]. Given a sequence $\mathbf{p} \in \mathcal{X}$ of length n, three learnable projection matrices

$$\mathbf{Q} = \mathbf{W}^Q \mathbf{p}, \quad \mathbf{K} = \mathbf{W}^K \mathbf{p}, \quad \mathbf{V} = \mathbf{W}^V \mathbf{p}$$

are trained to project the sequence into the query, key and value spaces. The query tells the model "what it is looking for", the key tells the model "what is contained in the sequence" and the value holds the information of the sequence. The self-attention of the scaled dot product is

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{2.6}$$

where d_k is the dimension of the key and value vectors. The output of the self-attention is then passed through a feed forward network.

Multi-head attention repeats this self-attention h times with different projection matrices $\mathbf{W}_h^Q, \mathbf{W}_h^K, \mathbf{W}_h^V$ for each head $h \in \{1, \ldots, h\}$. The output of the multi-head attention is then the concatenation of the outputs of the h heads.

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O \tag{2.7}$$

where $\mathbf{W}^O$ is a learnable projection matrix. The output of the multi-head attention is then passed through a feed forward network. Each head can specialize in a different abstract concept of the sequence, which allows the model to capture different relationships in the sequence, e.g. (in the case of language) syntax, semantics, etc.

Each transformer block consists of a multi-head attention layer, a position wise feed forward network and a residual connection followed by a layer normalization. The position of the tokens is encoded by a positional encoding, which is added to the input tokens before the first layer of the transformer. All of the tokens are processed in parallel, which allows modern hardware to run these kinds of layers efficiently during inference and training [34]. All of these architectural advantages allowed the advent of pre-trained transformer models, for language or protein sequences.

2.2.5. Protein large language models

As large language models (LLMs) have gained in popularity so have protein large language models (pLMs). These models are more capable of handling protein sequences and can represent these in meaningful ways in high dimensional spaces. A common method for pLMs

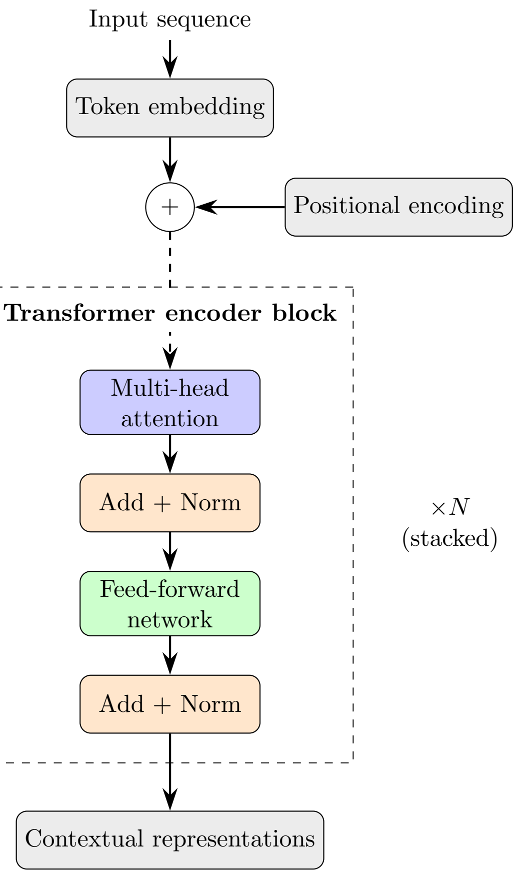


Figure 2.4.: Schematic of one Transformer encoder layer. The same block is repeated N times to form the full encoder.

is to represent them as a $\Phi(\cdot\,;\,\theta_\Phi) : \mathcal{X} \to (\mathbb{R}^l)^n$ for protein sequences. In this case the $\mathcal{X}$ is the set of all possible protein sequences in which the alphabet is the 20 amino acids. The implication is that the pLM maps each amino acid to a vector in the embedding space $\mathbb{R}^l$ and the output is a list of vectors of length n which represent the protein sequence. The θ_Φ are the parameters, which are trained before by masking out the amino acids and letting the model predict them, and are provided by the model provider. Since the resulting representation is a list of vectors, which vary in length depending on the protein sequence, the representation is not a function in the sense of (2.3). Therefore in most cases the output is aggregated into a single vector of shape $\mathbb{R}^l$ by a pooling operation. The aggregation function

$$\pi(\cdot\,;\,\theta_\pi) : (\mathbb{R}^l)^n \to \mathbb{R}^d \text{ with } l, n, d \in \mathbb{N} \tag{2.8}$$

is used to aggregate the list of vectors into a single vector. The simplest aggregation functions are mean pooling $\pi_{\text{mean}}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$ or max pooling $\pi_{\max}(\mathbf{x}) = \max_{i=1}^{n} \mathbf{x}_i$. Taken together the pLM $\Phi(\cdot\,;\,\theta_\Phi)$ and the aggregation function $\pi(\cdot\,;\,\theta_\pi)$ can be used to represent a protein sequence as a vector in the embedding space

$$\psi(\mathbf{p}\,;\,\theta_\psi) = \psi(\mathbf{p}\,;\,\theta_\psi, \theta_\pi) = \pi(\Phi(\mathbf{p}\,;\,\theta_\Phi)\,;\,\theta_\pi) \in \mathbb{R}^d \tag{2.9}$$

where θ_π are the parameters of the aggregation function and θ_ψ are the parameters of the pLM [31].

One of the first pre-Transformer models was UniRep, an LSTM (Long Short-Term Memory) network, which is a type of RNN [35]. UniRep demonstrated for the first time that sequence-only self-supervised learning is possible and that such models can generate meaningful embeddings for protein sequences. With the introduction of the transformer architecture, models like ProTrans were able to create even more informative embeddings for protein sequences. For the first time, these models were trained on hundreds of billions of residues, with each residue represented as a single token. During training, 15% of the tokens were masked, and the model was trained to predict these masked tokens. The T5 model was able to capture biophysical features of amino acids, including hydrophobicity, polarity, charge, and side chain volume. These features were evident in tSNE plots of the embeddings, where mean pooling was applied using the π_{mean} function [36]. In a different task the goal was to predict the protein's sub-cellular localization, here 4 different aggregation functions were tested, the mean, max, min and the concatenation of the embeddings, with the mean pooling being the best performing one [36] [37]. Besides differing aggregation functions, a pLM can be fine tuned on specific tasks. For this the model's parameters $\theta_\psi \in \Theta_\psi$ are fine tuned, this has been shown to improve prediction of GO annotations, which are standardized terms that describe the molecular function, biological process, and cellular component associated with a protein, providing a structured vocabulary for protein function across species [38]. The results showed that via a simple MLP on top of the embeddings the model was able to predict the sub-cellular localization with a higher accuracy than other tools, which specialized in this task (e.g. DeepLoc which needs MSA as input) [36]. This method of using protein sequence representations for downstream prediction is called transfer learning. With the ESM model class originally built by Meta Research and later with ESM-3 built by "evolutionaryscale.ai" a new era of protein language models was opened. ESM-2 was trained on 65 million protein sequences from UniRef50 and UniRef90, the model's size was varying from 8 million to 15 billion parameters. The training was done in the masked language modeling task, where 15 % of the tokens were masked out and the model was trained to predict them [39]. With ESM-3, a new type of model architecture was introduced. In this model, training was performed not only on masked protein sequences, but also included structural information and functional annotations. The ESM-C model class, which is the main focus of this work, was developed as an alternative to ESM-3 and was specifically designed for the task of sequence representation. The ESM-C model was trained in three sizes: 300M, 600M, and 6B parameters (M for million, B for billion). The 300M version achieved performance similar to the 650M ESM-2 model, while the 600M ESM-C surpassed the performance of the 3B ESM-2 model [6].

2.2.6. MLP, Decision Trees and NGBoost for phenotype prediction

After obtaining protein sequence representations through pLMs and aggregation functions, the next step is to use these embeddings for downstream prediction tasks. Two main approaches are commonly employed: Multi-Layer Perceptrons (MLPs) and tree-based methods, particularly Natural Gradient Boosting (NGBoost).

Multi-Layer Perceptrons

Multi-Layer Perceptrons are feedforward neural networks that can be used for both classification and regression tasks. An MLP typically consists of multiple fully connected layers with non-linear activation functions between them [40]. For protein phenotype prediction, the aggregated protein embeddings serve as input to the MLP, which learns to map these high-dimensional representations to the target phenotype labels.

The MLP can be formulated as a composition of functions:

$$f(\mathbf{x}) = f_L \circ f_{L-1} \circ \ldots \circ f_1(\mathbf{x}) \tag{2.10}$$

where each layer f_i applies an affine transformation followed by a non-linear activation function. For classification tasks with mutually exclusive phenotype labels, the final layer typically uses a softmax activation to output class probabilities, and the model is trained to minimize the negative log-likelihood.

Decision Trees and Ensemble Methods

Decision trees are interpretable machine learning models that make predictions through a series of binary splits on input features [41]. Each internal node represents a decision based on a feature x_j and threshold t, creating splits of the form $x_j \leq t$ versus $x_j > t$. The leaves contain the final predictions, typically the most frequent class for classification tasks.

Formally, a decision tree defines a function $f : \mathbb{R}^p \rightarrow \mathcal{Y}$ where $\mathcal{Y}$ is the set of target phenotype labels. Split selection maximizes the reduction in node impurity, measured by Gini impurity for classification:

$$G(S) = 1 - \sum_c p_c^2 \tag{2.11}$$

where p_c is the proportion of class c in node S. The information gain is calculated as:

$$\Delta I = I(S) - \frac{|S_L|}{|S|} I(S_L) - \frac{|S_R|}{|S|} I(S_R) \tag{2.12}$$

where I represents the impurity measure [42].

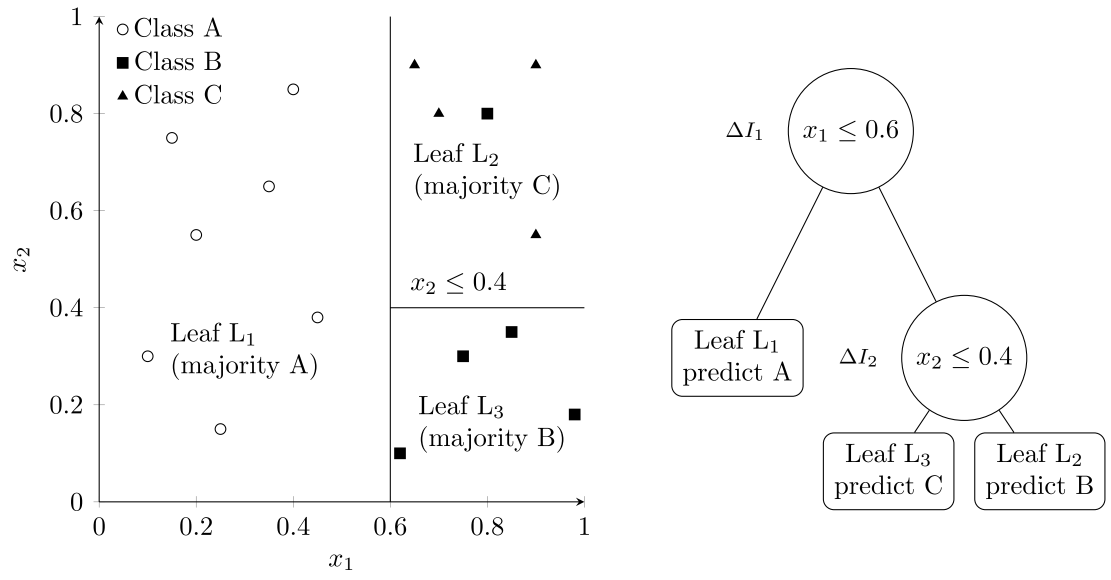


Figure 2.5.: Example of a binary decision tree. The left figure shows the axis-aligned partition of a 2-D feature space, while the right figure shows the corresponding binary decision tree. Splits match the drawn thresholds. Leaf majorities determine class predictions.

While individual decision trees suffer from high variance, ensemble methods address this limitation. Random forests reduce variance by training multiple trees on bootstrapped samples with random feature subsets and averaging their predictions [43]. Gradient boosting methods take a different approach, combining many weak learners sequentially to reduce bias while maintaining manageable variance.

Natural Gradient Boosting

Natural Gradient Boosting (NGBoost) represents a modern extension of gradient boosting that enables probabilistic predictions by modeling the full conditional distribution $p_\theta(y \mid x)$ rather than producing only point estimates [44]. This approach learns distributional parameters $\theta(x)$ such as mean and variance for Normal distributions or probability parameters for Bernoulli distributions as additive functions of input features.

The central objective minimizes the expected negative log-likelihood:

$$\mathcal{L}(\theta) = \mathbb{E}_{(x,y)}\left[-\log p_\theta(y \mid x)\right] \tag{2.13}$$

A key innovation is the use of natural gradients, which incorporate the local geometry of the parameter space through the Fisher information matrix $\mathbf{F}(\theta)$:

$$\tilde{g} = \mathbf{F}(\theta)^{-1}\nabla_\theta \mathcal{L} \tag{2.14}$$

This makes optimization invariant to smooth reparameterizations and provides more stable training when learning multiple coupled parameters.

The training procedure initializes distributional parameters $\theta^{(0)}$ using a constant model that maximizes likelihood. At each boosting iteration m, the algorithm computes per-sample natural gradients, fits regression trees to predict these gradients from input features, and updates parameters as:

$$\theta^{(m)}(x) \leftarrow \theta^{(m-1)}(x) + \eta \cdot f_m(x) \tag{2.15}$$

where $f_m(x)$ represents the base learner output and η is the learning rate.

NGBoost supports various predictive distributions: Bernoulli for binary classification, categorical for multiclass problems, and Normal, Log-Normal, Poisson, or Negative Binomial for continuous outcomes. Link functions enforce parameter constraints as needed.

The major advantage of NGBoost lies in providing calibrated uncertainty estimates alongside predictions. Unlike classical gradient boosted decision trees (XGBoost, LightGBM) that optimize convex losses for point estimates, NGBoost generalizes boosting to full predictive distributions, often improving calibration without sacrificing accuracy [45, 46, 44].

2.3. Hardware Resources

The computational infrastructure for this project consisted of two distinct server systems, each optimized for different computational tasks: the CPU server and the GPU server. This heterogeneous setup was designed to efficiently handle both data storage and processing requirements while accommodating the demanding computational needs of machine learning and graph database operations.

The CPU server served as the primary computational server, specifically configured for data processing and database operations. The system was equipped with an Intel Xeon E5-2680 v4 processor operating at 2.40GHz, housed in an HPE ProLiant DL360 Gen9 chassis. The server featured 256 GB of RAM to handle large-scale data operations and graph database queries efficiently. Storage was provided by a 232 GB solid-state drive (SSD), ensuring fast read/write operations critical for database performance. The system operated on Ubuntu 22.04 LTS, providing a stable and well-supported Linux environment for scientific computing applications.

In contrast, the GPU server functioned as the dedicated graphics processing unit server, specifically designed to handle computationally intensive tasks including large language model (LLM) operations and artificial intelligence workloads. The system was built around an AMD Ryzen Threadripper 1900X 8-core processor, providing substantial multi-threading capabilities for parallel processing tasks. The key computational resource was an NVIDIA

GeForce RTX 3090 graphics card with 24 GB of video RAM (VRAM), enabling the processing of large neural networks and complex machine learning models. The GPU was connected via PCIe Generation 1 interface operating at 8x bandwidth.

Both servers were integrated within the same local network infrastructure, enabling efficient data transfer and resource sharing. A network mount configuration was established between the two systems, allowing for seamless file system access across both machines. This architecture facilitated a distributed computing approach where the Neo4j graph database could be executed within Docker containers on the CPU server while leveraging the faster SSD storage of the GPU server for data persistence. This configuration optimized both computational performance and storage efficiency, ensuring that database operations could benefit from high-speed storage while maintaining separation of concerns between data processing and machine learning workloads.

3. Methods

Modern bioinformatics increasingly involves the integration of heterogeneous data types: protein sequences, functional annotations, metadata, structural information, and phenotypic observations, all of which originate from a variety of sources and formats. One of the major challenges in this field is handling such complex data, as biological datasets often consist of a combination of different data types, such as measurements and labels, which are frequently mixed together. This complexity makes biological data difficult to manage, since standard table formats are often insufficient and single files can become unwieldy. Classical flat file formats such as FASTA [47] offer a robust mechanism for representing linear nucleotide or amino acid sequences but are inherently limited in their capacity to encapsulate rich metadata or complex semantic relationships inherent in biological systems. While it is possible to include extra data through extended sequence names or auxiliary CSV tables, these approaches tend to scale poorly and complicate downstream computational analyses. Furthermore, traditional tools have primarily focused on analyzing sequences alone, with the main goal being to predict or determine properties based solely on sequence information. Consequently, there exists a strong need for a data architecture capable of representing biological entities and their interrelations in a flexible and extensible manner, particularly in the context of machine learning driven enzyme engineering applications. To effectively combine different types of data sources, new data management strategies are required. Our group has developed new tools, such as PyEED[1], which are designed to make all types of biological data ready for machine learning applications and facilitate training on them [48].

3.1. PyEED Framework

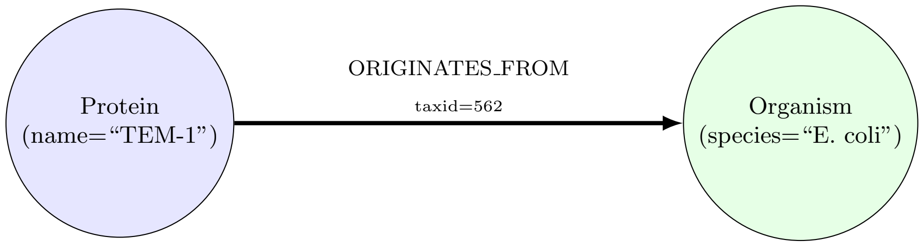
To address these challenges, we developed PyEED, a Python based framework designed to connect (i) publicly available biological databases, (ii) a local graph based data backend, and (iii) user developed pipelines in Python. Python was selected due to its widespread adoption in the scientific community and its rich ecosystem of machine learning and bioinformatics packages. The databases integrated by PyEED vary widely in structure and scope but typically expose programmatic access via well documented REST APIs. This architecture enables seamless ingestion of heterogeneous data such as protein sequences from one source

[1] `https://github.com/PyEED/pyeed`

and organism specific metadata from another and represents them in a unified internal data model.

PyEED uses a graph database rather than a traditional relational (SQL) database as its core. In graph databases, individual entities are represented as nodes and their relationships as edges. Both nodes and edges can carry arbitrary attributes (key-value pairs), enabling the storage of complex context such as residue level annotations, structural positions, or experimental outcomes. Figure 3.1 shows a minimal example illustrating how a protein can be connected to an organism with a "`ORIGINATES_FROM`" relationship, including a property specifying the NCBI taxonomy identifier.

Figure 3.1.: Minimal graph database example used to explain the basic modelling objects in PyEED.



All such individual facts are stored in one coherent graph, typically referred to as a knowledge graph. Machine learning models tailored to graph structured data (e.g., graph convolutional networks or graph attention networks) can be trained on this graph to infer novel biochemical properties. PyEED uses the open-source graph database engine Neo4j [49] (Version 5.26) [50], which supports efficient storage and traversal of large biological graphs and exposes a domain specific query language called Cypher. Cypher is analogous to SQL but is designed for pattern based querying of nodes and edges in a graph (see Example 3.2).

```
MATCH (p:Protein {name:"TEM-1"})-[r:ORIGINATES_FROM]->(o:Organism {species:
"E. coli"}) RETURN o.taxid, p.sequence;
```

Figure 3.2.: Example Cypher query

The schema used by PyEED can be user defined and extended, yet in this work the base schema was used. The idea is to have a minimal amount of different node types and relationships. A subset of the full schema is shown in Figure 3.3. The Protein node is the central node in the graph and holds attributes such as the protein name, the sequence, the accession id, a protein embedding of the sequence and more. In the insertion of a new protein from for example NCBI GenBank, the organism is also inserted and connected. Such that the protein node is connected to the organism node via the ORIGINATES_FROM relationship. While the organism holds attributes such as the species name, the taxonomic id. The DNA sequence is stored in a DNA node, which connects to a protein node via an ENCODE relationship, which also holds the start and end of the coding region of the DNA sequence for this specific protein. Between two protein nodes or two DNA nodes relationships like MUTATION are

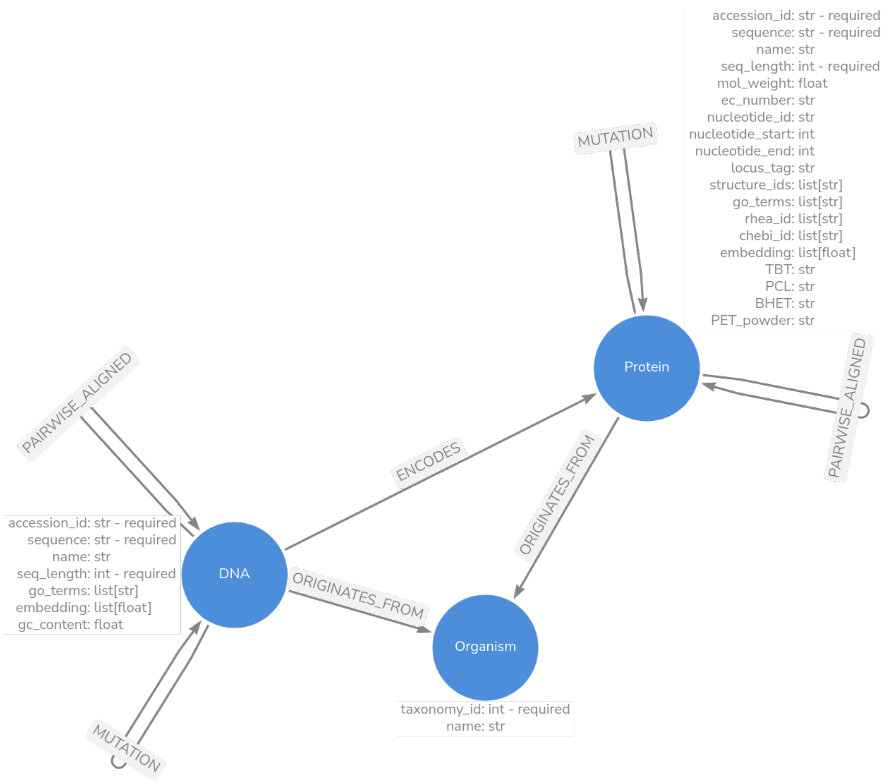


Figure 3.3.: Minimal PyEED graph schema.

used to define a mutation from one amino acid to another, the attributes on the edge contain a list of from and to positions of the mutation.

The full schema used in this work is shown in Figure 3.4. The full schema is more complex and contains more node and relationship types. Here besides protein, DNA, Organism nodes, annotations for Regions and Sites exist. Moreover the GO terms from UniProt are stored as GO Term nodes and connected to protein nodes via ASSOCIATED_WITH relationships. The CARD ontology can be loaded in the same way into the graph, the classes are stored in their hierarchical structure and order and connect to the protein nodes via SUBCLASS_OF relationships and CUSTOM_RELATIONSHIPS (defined by the Ontology e.g. CONFERS_RESISTANCE_TO)

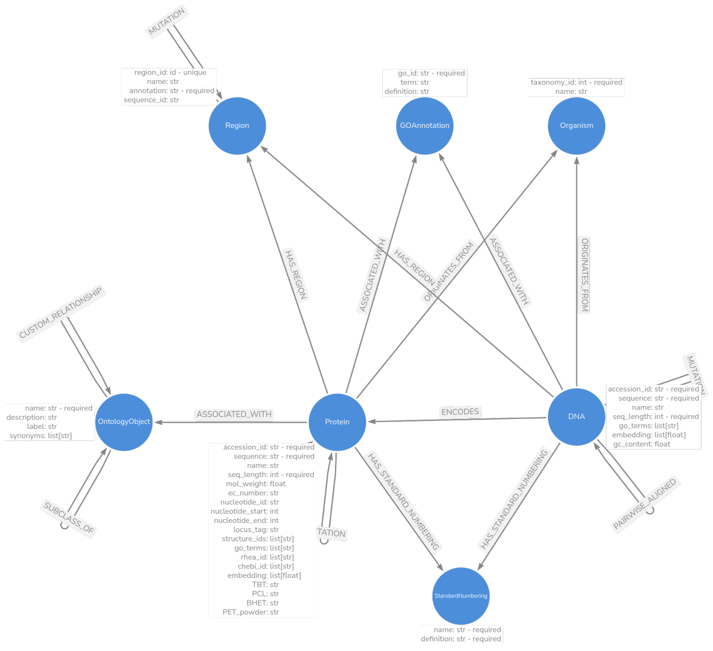


Figure 3.4.: Detailed PyEED schema, showing all node and relationship types used in the enzyme engineering context.

3.1.1. Databases Integrated by PyEED

PyEED aggregates heterogeneous biological knowledge from the following high-quality curated repositories. Each database contributes distinct types of biochemical, evolutionary, or functional information, all mapped into the PyEED graph schema via standardized node and edge types.

- NCBI GenBank and RefSeq:
  Established by the National Center for Biotechnology Information (NCBI, USA), GenBank contains publicly submitted nucleotide sequences for over 500,000 formally described organisms, whereas RefSeq provides a curated, non-redundant set of reference genomes, transcripts, and proteins [51]. Both serve as major sources of sequence data and taxonomic metadata. The taxonomic lineage (NCBI TaxID), host organism, and source feature annotations are extracted to form Organism and Protein/Nucleotide nodes in the PyEED graph and are connected via explicit ORIGINATES_FROM relationships.

- UniProt Knowledgebase (UniProtKB):
  UniProtKB is a universal protein knowledgebase containing over 230 million protein entries enriched with manually reviewed functional annotations in UniProtKB/Swiss-Prot and computationally annotated records in UniProtKB/TrEMBL [52]. Functional data such as enzyme commission (EC) numbers, molecular function descriptions, post-translational modifications, and domain architectures are imported into the graph as

node attributes or via relationships toward Function or Domain node types.

- Comprehensive Antibiotic Resistance Database (CARD):
  CARD provides a rigorously curated ontology of antimicrobial resistance determinants, resistance mechanisms, drug classes, and molecular targets [53]. Each entry represents a gene/protein experimentally linked to a resistance phenotype. PyEED integrates CARD by adding ResistanceGene nodes that connect to Drug, Mechanism, and Organism nodes. This enables machine learning models to correlate sequence variations with phenotypic drug resistance patterns directly on the graph.

- Gene Ontology (GO):
  The Gene Ontology is a controlled hierarchical vocabulary consisting of exactly three sub-ontologies: Molecular Function, Biological Process, and Cellular Component [54]. PyEED incorporates GO terms as GO Term nodes and connects them to protein nodes through ASSOCIATED_WITH edges. This enables semantic similarity analysis, functional enrichment, and graph-based feature learning on a principled ontological backbone.

- Beta-Lactamase DataBase (BLDB):
  BLDB is a manually curated specialized repository focusing entirely on beta-lactamases, enzymes that hydrolyze beta-lactam antibiotics [55]. It covers Ambler class assignment (A-D), substrate/inhibitor profiles, resolved 3D structures, and active site annotations. PyEED uses BLDB to annotate enzyme nodes with structurally relevant detail such as Ambler class, active-site residue numbering, inhibitor spectra, and evolutionary clades. This domain-specific information is critical for large language model-driven directed evolution campaigns involving beta-lactamases.

By harmonizing these diverse data structures within a single graph, PyEED provides a unified framework suited for advanced graph learning tasks (e.g. node classification, link prediction, and subgraph embedding) in enzyme engineering.

3.2. Data Acquisition and Preprocessing

A list of 270 TEM NCBI IDs was downloaded from the BLDB [55], and the respective protein and DNA sequences were retrieved from the NCBI nonredundant protein and nucleotide databases, respectively [51]. Of the 270 protein sequences, 209 had an accession ID; these 209 TEM protein sequences were retrieved from the NCBI nonredundant protein database. For each of the 209 TEM protein sequences, a BLASTP [56] search was performed in the NCBI nonredundant protein database (maximum E-value 0.001, maximum of 5000 hits). The hits were checked for duplicates by accession ID. The resulting 12,560 unique protein entries were added to a Neo4j Graph Database version 5.26. Of the 209 proteins from the BLDB, 207 had a DNA sequence linked in the NCBI entry. For all of these, a BLASTN search (maximum E-value 0.001, maximum of 5000 hits) was performed in the NCBI nonredundant nucleotide database, with a cut-off date of July 14, 2024. The 1,034,793 hits were checked for duplicates by accession ID, resulting in 45,521 unique DNA entries, which were added to the Neo4j database.

Since not all retrieved DNA sequences had a relationship to a protein, the link was created via a codon table. If the protein did not exist for the corresponding DNA, a new protein entry was created, termed "Predicted_From_DNA." To further clean the data, a sequence identity check was performed across all proteins—those from CARD, BLDB, BLAST search, and the predicted proteins from DNA. If a pair of duplicate entries based on protein sequence was found, a cleaning process was initiated so that only one of the two was retained in the database. For example, if protein A and protein B had the same sequence and protein A was kept, all relationships previously associated with protein B were removed and linked to protein A. In this way, knowledge such as DNA entries, CARD ontology, etc., was not lost. After completing this step for the proteins, the same process was applied to the DNA entries, considering only DNA sequences connected to a protein. This meant that multiple DNA entries could be linked to one protein. Additionally, a check was performed to ensure that all DNA was forward-coded (5' to 3'). In 77 cases where the complement was deposited, the reverse complement was generated. In the BLAST search and through the import of the entire CARD ontology, too many proteins were included to allow for meaningful mutational analysis. To focus on the TEM family, we chose only the proteins with a length of 286 amino acids. This resulted in 10721 proteins being selected and used in the further analysis.

The selected proteins from the clustering analysis were then used for mutational analysis. For all these proteins, a residue numbering was created based on TEM-1 (according to the CARD ontology). This was done by aligning each protein to TEM-1 and then creating a numbering based on the alignment. Thus, each residue in the protein has a number allowing it to be compared to the TEM-1 numbering, which ranges from 1 to 286 (3 to 290 according to [9]).

Although this specific numbering was used to conform with the literature on class A enzymes, the PyEED framework allows for flexible numbering, so that the numbering can be changed to one of choice, for instance, the Ambler standard numbering [9]. In the CARD numbering used here, the TEM-1 signal sequence starts at 1 and ends at 24, whereas in the Ambler standard numbering it starts at 3 and ends at 26. Based on the standard numbering, a mutation was defined as a change in the amino acid at a specific position, relative to the standard numbering (in our case, TEM-1). Thus, a mutation has a "from" and "to" position, as well as the amino acid that is changed from and to. The same can be done for DNA, so that a mutation is defined as a change in the nucleotide at a specific position, relative to the standard numbering (in our case, TEM-1 DNA). For the selected protein sequences, networks were generated with nodes representing unique sequences (including the signal sequence) and edges connecting two sequences if they differ by one amino acid.

3.3. Phenotype prediction for the TEM β-lactamases

The goal of the phenotype prediction is to predict the phenotype of a TEM β-lactamase based on its sequence. The data for the prediction is the sequence of the TEM β-lactamase and the phenotype of the TEM β-lactamase. The phenotype is the class of the TEM β-lactamase based on the BLDB database. The class is based on the substrate specificity and inhibitor susceptibility of the TEM β-lactamase. The classes are '2b' '2be' '2ber' '2br' (see Table 2.1). The training and testing data was always split with a 70/30 split and the random state was set to 42, the class distribution was kept in the training and testing data via the stratified split from sklearn [57]. To predict the phenotype of a TEM β-lactamase, its sequence is embedded into a vector space using a protein language model. This embedding is then used to predict the phenotype of the TEM β-lactamase. This approach is called transfer learning, as the goal is to extract properties from the vector embedding that describe the protein [58] [59] [60]. For this different machine learning methods can be used, such as MLP, SVMs, Random Forests, etc [40].

3.3.1. Different aggregation functions $\pi(\cdot\,;\,\theta_\pi)$

All of the downstream analyses required a single protein embedding vector for one sequence, both in training and inference. The output from a pLM $\Phi(\cdot\,;\,\theta_\Phi)$ is a list of vectors of shape $\mathbb{R}^l$. The aggregation function $\pi(\cdot\,;\,\theta_\pi)$ is used to aggregate the list of vectors into a single vector of shape $\mathbb{R}^d$, the fixed dimensionality is needed for the machine learning methods. The aggregation function can be mean pooling, max pooling, self-similarity weighted pooling, etc (2.8) (2.9).

The most used aggregation function is mean pooling, due to its simplicity and effectiveness. In almost all cases reviewed in the literature there were no other pooling methods tested or compared to mean pooling [35] [61] [62] [63] [64].

One of the main concerns about mean pooling is that in protein sequences certain positions can play a critical role for the function of a protein, like for example binding and active site positions. Mean pooling does not take this into account and treats all positions equally. This might result in a signal loss in certain dimensions. In order to have a baseline against which test can be conducted the mean and max pooling aggregation function where used, $\pi_{\text{mean}}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$ or max pooling $\pi_{\max}(\mathbf{x}) = \max_{i=1}^{n}\mathbf{x}_i$ respectively.

As an alternative pooling method, which uses no prior biochemical knowledge, self-similarity weighted pooling was employed. This pooling method weights positions based on their dissimilarity to all other positions in the sequence, with the underlying hypothesis that positions that deviate from the consensus may carry more functional information.

The self-similarity weighted pooling strategy follows a systematic three-step computational process. First, the approach calculates the cosine similarity matrix S between all pairs of position embeddings to quantify the relatedness between token embeddings:

$$S_{ij} = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2} \tag{3.1}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the embeddings of positions i and j respectively. This calculation results in a similarity matrix where each entry represents the cosine similarity between token embeddings. The resulting matrix can be visualized as a heatmap to identify embedding space organization patterns, where regions of low similarity (represented as blue bands in visualization) indicate positions that exhibit dissimilarity from other positions, suggesting these positions carry unique functional information.

Second, the average cosine similarity for each position is calculated as:

$$\bar{s}_i = \frac{1}{n} \sum_{j=1}^{n} S_{ij} \tag{3.2}$$

where n is the sequence length. Third, to emphasize positions that are dissimilar to other positions, weights are computed by raising the dissimilarity to the power of α:

$$w_i = \frac{(1 - \bar{s}_i)^\alpha}{\sum_{k=1}^{n} (1 - \bar{s}_k)^\alpha} \tag{3.3}$$

where α is a hyperparameter controlling the degree of emphasis on dissimilar positions. The final aggregated sequence representation is calculated as:

$$\pi_{\text{self-sim}}(\mathbf{x}) = \sum_{i=1}^{n} w_i \mathbf{x}_i \tag{3.4}$$

This approach identifies potentially important residues without prior biochemical knowledge by leveraging the assumption that positions exhibiting greater dissimilarity from other positions in the embedding space carry unique functional information relevant for phenotype determination.

In the particular case of β-lactamase, there is extensive biochemical knowledge about which positions are important for the enzyme's function. This information can be leveraged to assign weights to specific positions in the sequence, either by selecting single positions or multiple positions for aggregation. To identify the most informative combination of positions, both random selections of known biochemically relevant positions and completely random position selections were used as controls.

The following array of biochemically important positions was used for targeted selection (see also Section 2.1.1 for discussion of their biochemical relevance):

Important positions: 69, 70, 73, 76, 104, 164, 165, 166, 167, 168, 170, 171, 179, 182, 234, 235, 236, 238

These positions were chosen based on prior biochemical studies and literature, as they are known to play critical roles in the structure and function of β-lactamase. By comparing aggregation strategies that focus on these positions to those using randomly selected positions, the impact of incorporating biochemical knowledge into the aggregation process can be systematically evaluated.

3.3.2. MLP and NGBoost for phenotype prediction

To identify optimal model configurations for antibiotic resistance prediction, comprehensive hyperparameter tuning was conducted for both Natural Gradient Boosting (NGBoost) and Multi-Layer Perceptron (MLP) models. The hyperparameter optimization strategy employed systematic grid search and random sampling approaches, exploring multiple dimensions of model configuration including protein language model selection, embedding extraction strategies, and model-specific architectural parameters.

NGBoost Hyperparameter Optimization

The NGBoost hyperparameter space was systematically explored across three main dimensions: protein language models, embedding extraction strategies, and boosting-specific parameters.

For the protein language models, nine state-of-the-art models were evaluated. These included variants from the ESM-2 family (facebook/esm2_t6_8M_UR50D, facebook/esm2_t12_35M_UR50D, facebook/esm2_t30_150M_UR50D, facebook/esm2_t33_650M_UR50D, facebook/esm2_t36_3B_UR50D), ProtT5 models (prot_t5_xl_bfd, prot_t5_xl_uniref50), and ESM-C models (esmc_300m, esmc_600m). These models cover a range of computational complexity and training strategies, which allowed for assessment of the trade-off between model size and prediction performance.

Embedding extraction approaches were systematically evaluated as follows:

- **Single-position strategies:** Targeting key catalytic residues (positions 67, 165, 232–234 in adjusted numbering).
- **Multi-position strategies:** Combining multiple functionally important sites.
- **Windowed approaches:** Extracting local sequence context around critical positions,

with window sizes of 2, 3, and 8.

- **Weighted strategies:** Applying learned importance weights to specific positions.
- **Global pooling methods:** Using mean and max pooling over the entire sequence.
- **Self-similarity weighted approaches:** Employing varying alpha parameters (-1, -0.1, 0.5, 1.0, 2.0, 3.0).
- **Random sampling strategies:** Used for baseline comparison.

The selection of positions for these strategies was informed by structural and functional analysis of TEM β-lactamases, with a focus on residues known to influence substrate specificity and catalytic activity. This included the catalytic triad, active site loops, and allosteric networks (see Section 2.1.1).

For NGBoost-specific parameters, the boosting hyperparameters were optimized through grid search. The number of estimators was varied from 100 to 1000 in increments of 100. The learning rate was tested at 0.005, 0.008, and 0.01. The maximum tree depth was varied from 2 to 6. Column sampling and minibatch fractions were kept constant at 0.7 and 0.6, respectively, to maintain model stability.

A total of 5,000 experiment configurations were sampled from the complete hyperparameter space, which allowed for a comprehensive exploration while keeping the computational requirements feasible. The implentation was done via the pyeed framework and the python package ngboost [65].

MLP Hyperparameter Optimization

The MLP hyperparameter tuning followed a similar multi-dimensional approach, exploring neural network architectures, training configurations, and the same embedding strategies as used for NGBoost.

For the network architectures, multiple feed-forward designs were systematically evaluated. These included single hidden layer networks with 32, 64, 128, or 256 neurons; two-layer networks with [128, 64], [64, 64], or [256, 128] neurons; and three-layer networks with [128, 64, 32] or [256, 128, 64] neurons. Regularization variants were also tested, incorporating dropout with rates of 0.1 or 0.2, as well as batch normalization. The implentation was done via the pyeed framework and the python package torch [66].

For training configurations, several optimization parameters were explored. Learning rates of 0.0001, 0.001, and 0.01 were tested, along with batch sizes of 16, 32, and 64. Both Adam and AdamW optimizers were used, with AdamW including a weight decay of 0.01. Class weighting strategies were compared between balanced and unbalanced options. Training was conducted for 200 to 300 epochs, with early stopping applied using a patience of 15 epochs.

The same comprehensive set of embedding extraction methods used for NGBoost was also applied to the MLP models to ensure a fair and comparable evaluation across model types.

Experimental Design and Evaluation Framework

A 70/30 train-test split was used for all experiments, as described above. Of the 209 TEM β-lactamases, 140 of them had an annotated phenotype in the BLDB database and an accession id. This resulted in 4 target classes '2b' '2be' '2ber' '2br' for the prediction of the phenotype (see Table 2.1). One of the 140 TEM β-lactamases had a phenotype 'ESBL' which due to it's occurrence of only one sample was excluded from the prediction. The remaining 139 TEM β-lactamases have occurrences of 15 for '2b', 83 for '2be', 10 for '2ber' and 31 for '2br'. Model performance was evaluated using several metrics. These included classification accuracy for overall performance, macro-averaged precision, recall, and F1-score to account for class imbalance, as well as training time and computational efficiency. Robust error handling was implemented to categorize and analyze failure modes, especially for NGBoost experiments, which were sensitive to numerical instability. Error types were systematically classified, including singular matrix errors, numerical instability, and memory limitations. This information was used to refine the parameter space.

4. Results

4.1. Phenotype Prediction Results

Analysis of token-level embeddings from the ESM-C 600M model examined how protein language models capture TEM-1 β-lactamase organization. The UMAP projection of individual token embeddings with mean pooling aggregation (Figure 4.1) shows embedding space organization characteristics.

Each amino acid residue occupies a distinct position in the latent space. While residues that are close together in the linear sequence often form clusters in the embedding space, there are also distinct jumps between clusters. These transitions may indicate regions associated with different functional roles. The signal sequence, for example, forms a separate cluster in the lower right portion of the projection.

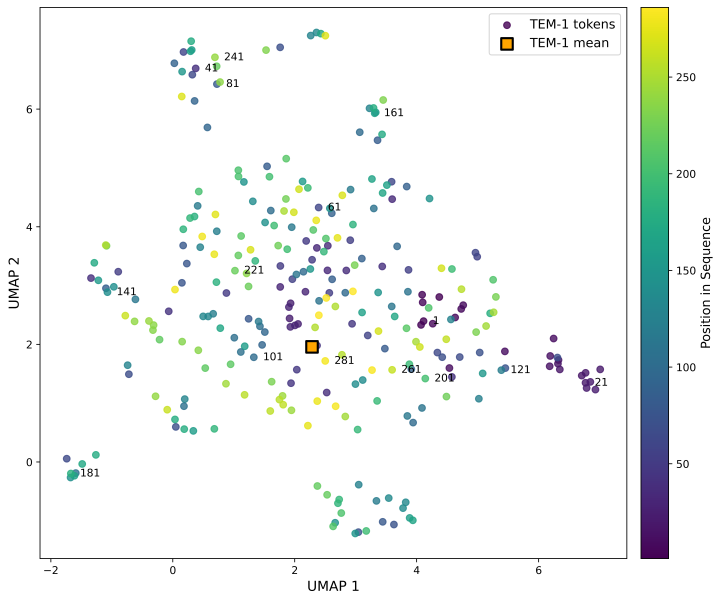


Figure 4.1.: UMAP (Mean Pooling), every token of TEM1 with the esmc 600m model, the mean is shown in orange and the tokens are colored by their position in the sequence

The mean pooling strategy (orange point in Figure 4.1) positions centrally among token embeddings, providing a summary representation of the sequence. This supports the use of mean pooling in protein sequence analysis, as discussed in Section 2.2.5.

Mean pooling may not capture protein sequence complexity. Certain positions play critical roles in protein function, such as binding sites, active site residues, and catalytic triads (Section 2.2.5). Mean pooling treats all positions equally, potentially losing residue-level information for phenotype prediction.

This limitation motivated exploration of alternative pooling strategies. Two approaches were tested for improved pooling methods. The first approach uses biochemical knowledge, selecting tokens based on functional importance. For TEM-1 β-lactamase, this includes residues within the active site, catalytic triad, substrate binding regions, and the Ω-loop (Section 2.1.1). Random token selections served as controls.

The second approach addresses the requirement for prior biochemical knowledge, which is often unavailable for novel proteins. A self-similarity weighted pooling strategy automatically identifies important positions based solely on embedding characteristics, as described in Section 3.3.1.

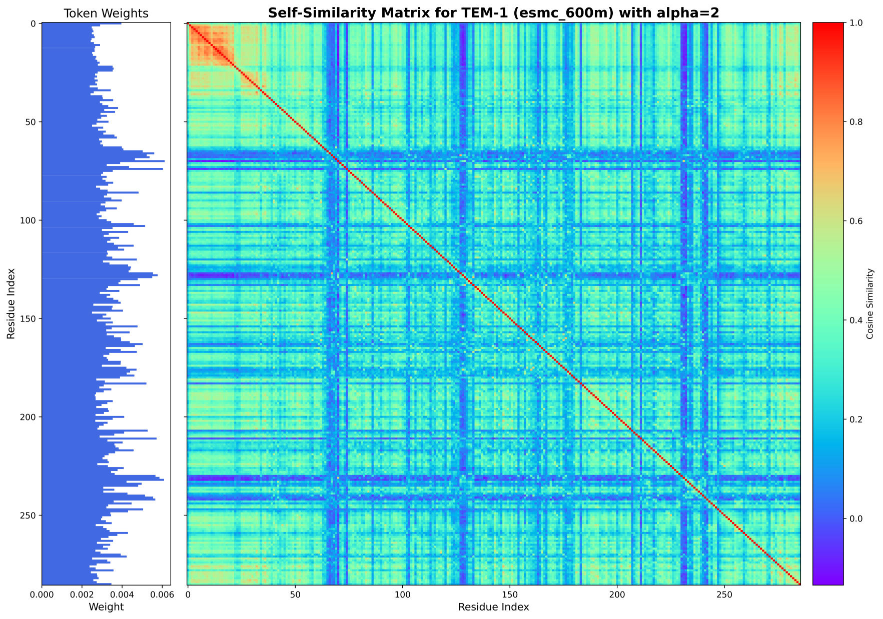


Figure 4.2.: Self-similarity weighted pooling analysis for TEM-1 using ESM-C 600M model. The heatmap displays cosine similarity between all token embeddings, with blue indicating dissimilar regions (0) and red indicating similar regions (1). The left panel shows the computed weights for each position with alpha parameter set to 2, highlighting regions deemed functionally important by the self-similarity analysis.

Figure 4.2 presents the cosine similarity heatmap analysis for TEM-1 using the ESM-C 600M model. The heatmap visualizes the similarity matrix calculated between all token embeddings, where blue regions indicate low similarity (approaching 0) and red regions indicate high similarity (approaching 1). The embedding space organization patterns reveal distinct blue bands that identify regions where tokens exhibit dissimilarity from other positions, suggesting these positions carry unique functional information.

The self-similarity analysis shows regions exhibiting dissimilarity from others in the embedding space (blue bands in Figure 4.2). The top 15 highest-weighted positions were examined (Figure 4.3).

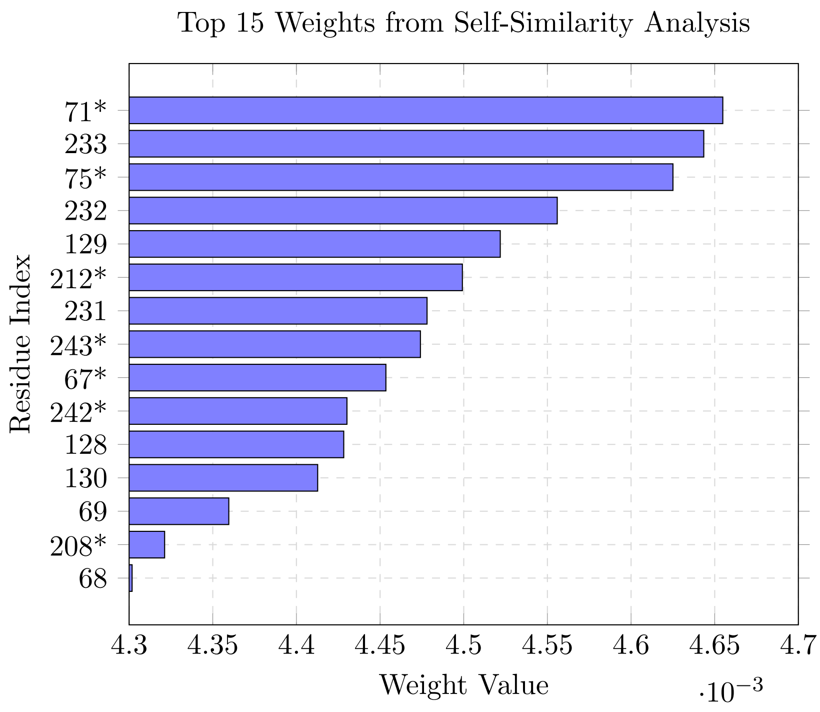


Figure 4.3.: Bar plot showing the top 15 weights from the self-similarity analysis for TEM-1 ESMC 600M model with alpha=1. The ranking is from top to bottom (1-15) with corresponding residue indices displayed on the x-axis. Residue indices marked with * indicate positions identified as important by self-similarity analysis but not yet characterized in the literature, suggesting potential targets for further investigation.

The identification of positions 71, 233, 75, 232, and 129 among the top-weighted residues is particularly noteworthy when considered in the context of TEM-1 biochemistry. Position 71 corresponds to a residue adjacent to the catalytic Ser70, placing it in the immediate vicinity of the nucleophilic center essential for β-lactam hydrolysis. The clustering of positions 232-234 in the top rankings aligns with the known importance of this region in the proton-relay network during acylation, as discussed in Section 2.1.1. These positions participate in the coupling between the general acid/base network and the Ω-loop, directly influencing catalytic water geometry.

Position 75 is located near the catalytically important Lys73, which forms part of the general acid/base network alongside Lys234. The high weighting of position 129 is also biochemically relevant, as it lies adjacent to the SDN loop (Ser130-Asp131-Asn132), a conserved structural motif critical for substrate binding and catalysis in class A β-lactamases.

The appearance of several Ω-loop positions (212, 208) among the weighted residues further validates the approach, given the established role of this region in substrate specificity and catalytic efficiency. As detailed in Section 2.1.1, mutations in the Ω-loop, particularly around Arg164-Asp179, are frequently associated with extended-spectrum and inhibitor-resistant phenotypes.

The fact that this knowledge-free approach successfully identifies residues with established biochemical importance suggests that protein language models have learned to encode functionally relevant information in their embeddings, even without explicit structural or functional training data.

Hyperparameter optimization experiments evaluated different embedding strategies and protein language models for phenotype prediction (Section 3.5). MLP and NGBoost models were evaluated across 5,000 configurations each.

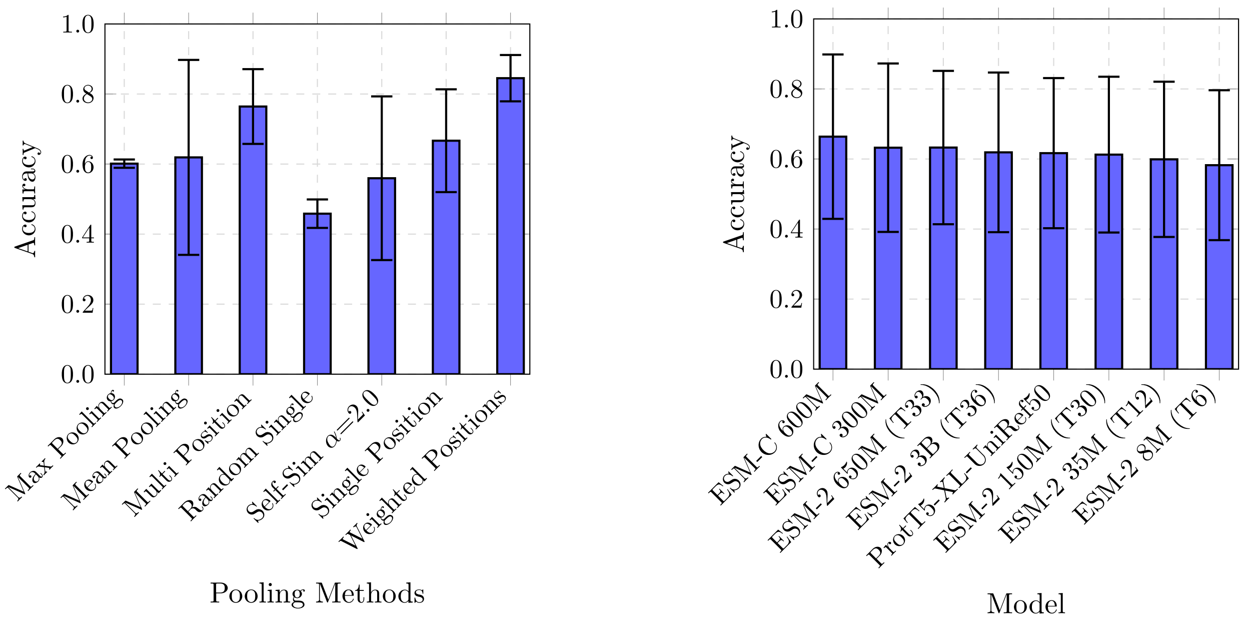


Figure 4.4.: Comparison of phenotype prediction accuracy for different pooling methods (left) and different protein language models (right) using MLP classifiers. Error bars represent standard deviation across multiple runs.

Phenotype classification of TEM-1 β-lactamase variants achieved accuracies up to 97.6% on the test set for weighted positions and multi-position strategies using MLP models. This accuracy enables prediction of resistance phenotypes (2b, 2be, 2ber, 2br) from sequence information alone.

For MLP models, weighted positions strategy achieved the highest average accuracy of 84.5%

± 6.6%, followed by multi-position selection at 76.4% ± 10.7%. Mean pooling achieved 61.9% ± 27.8% and max pooling 60.1% ± 1.2%. Self-similarity weighted approach with $\alpha = 2.0$ reached 56.0% ± 23.4%, while random single position selection achieved 45.8% ± 4.1%.

NGBoost models showed similar performance across strategies. Weighted positions achieved 73.3% ± 7.0% accuracy, multi-position selection 72.2% ± 7.1%. Mean pooling achieved 61.4% ± 7.3% and max pooling 63.3% ± 8.3%. Self-similarity weighted approach achieved 63.3% ± 9.4%.

Model comparison showed modest differences. For MLP models, ESM-C 600M achieved the highest average accuracy of 66.4% ± 23.5%, followed by ESM-2 650M (T33) at 63.3% ± 21.9%. The larger ESM-2 3B model achieved 61.9% ± 22.8%. For NGBoost, ESM-C 600M achieved 69.5% ± 10.0%, followed by ESM-2 650M at 67.9% ± 9.9%.

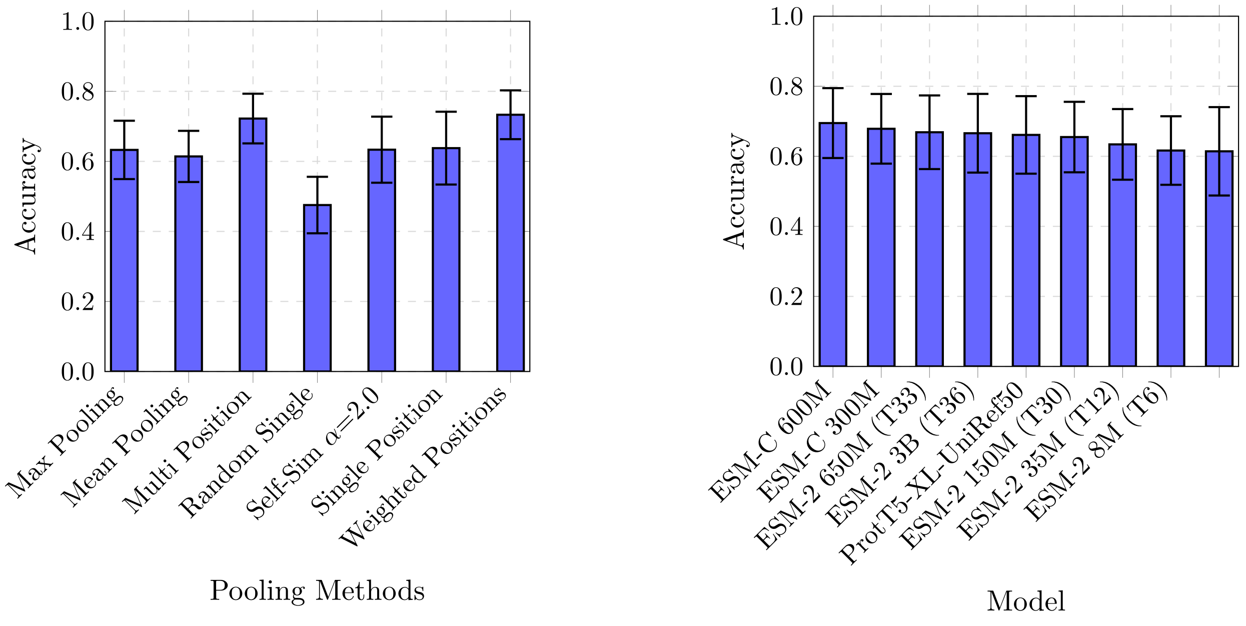


Figure 4.5.: Comparison of different pooling methods (left) and different protein language models (right) for phenotype prediction accuracy using NGBoost classifiers. Error bars represent standard deviation across multiple runs.

The violin plots (Figures 4.6 and 4.7) provide additional insight into the distribution of performance across different embedding strategies. These visualizations reveal not only central tendencies but also the variance and shape of performance distributions across the 5,000 hyperparameter configurations tested for each model type.

For both model types, biochemically informed strategies (weighted positions, multi-position) showed higher mean performance and more consistent results compared to traditional pooling methods.

The top-1 accuracy analysis (Figure 4.8) examines the best-performing configuration for each embedding strategy across all 5,000 runs, providing insight into the peak potential of

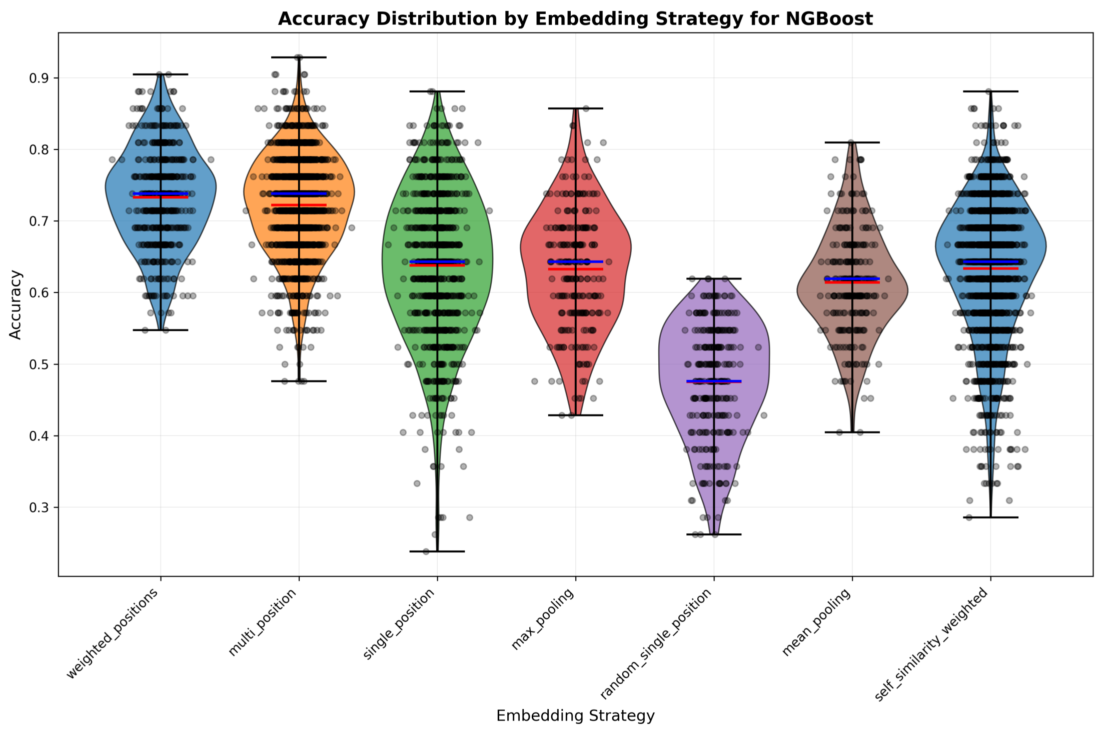


Figure 4.6.: Violin plot showing accuracy distribution by embedding strategy for NgBoost models.

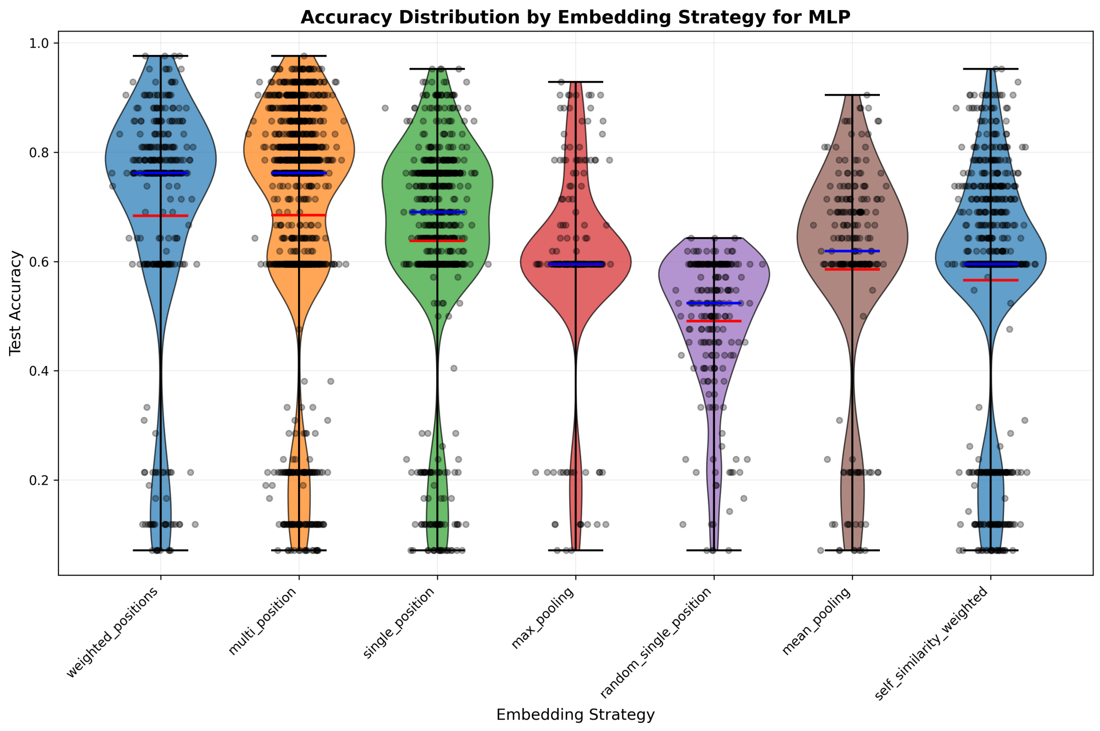


Figure 4.7.: Violin plot showing accuracy distribution by embedding strategy for MLP models.

each approach. This analysis reveals the theoretical upper bounds achievable when optimal hyperparameters are selected for each strategy.

For MLP models, weighted positions and multi-position strategies both achieved top-1 accuracies of 97.6%. Self-similarity weighted, and single position strategies achieved 95.2% accuracy. Traditional pooling methods achieved 92.9% (max pooling) and 90.5% (mean pooling). Random strategies achieved 69.0% (multi-position) and 64.3% (single position).

For NGBoost models, multi-position strategy achieved 92.9% top-1 accuracy. Weighted positions approach achieved 90.5%, while single position and self-similarity weighted approaches achieved 88.1%. Traditional pooling methods achieved 85.7% (max pooling) and 81.0% (mean pooling).

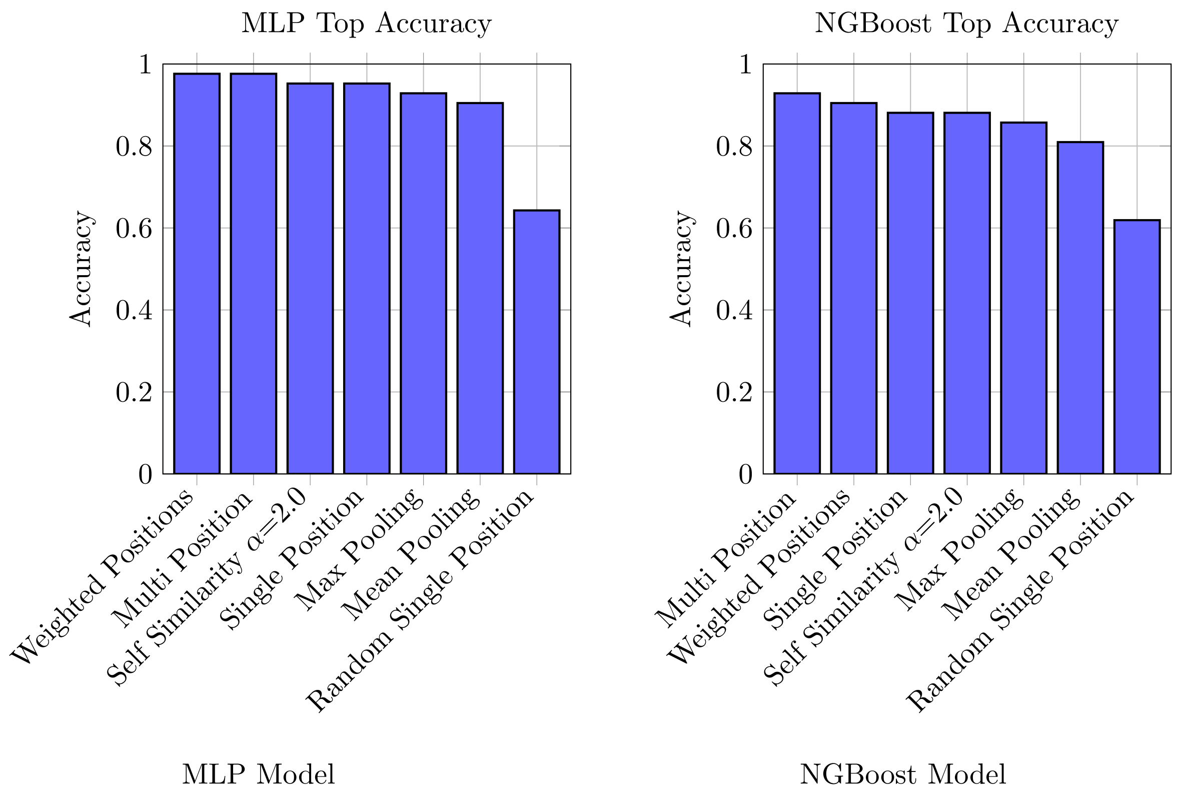


Figure 4.8.: Comparison of top-1 accuracy across embedding strategies for MLP and NGBoost models based on 5000 runs each

Biochemically informed strategies showed consistent superiority across both model types.

4.2. Latent Space Analysis and Dimensionality Reduction

Further investigation examined the mechanisms underlying self-similarity weighted pooling performance. Three best-performing ESM-C 600M model configurations with self-similarity weighted pooling ($\alpha = 3.0$) were selected for analysis. These configurations achieved test accuracies of 78.57%, 76.19%, and 73.81%.

Permutation importance analysis quantified the contribution of different embedding dimensions to phenotype prediction [**?**]. This approach permutes individual features and measures the resulting degradation in model accuracy. The permutation importance was calculated using the `permutation_importance` function from scikit-learn [57].

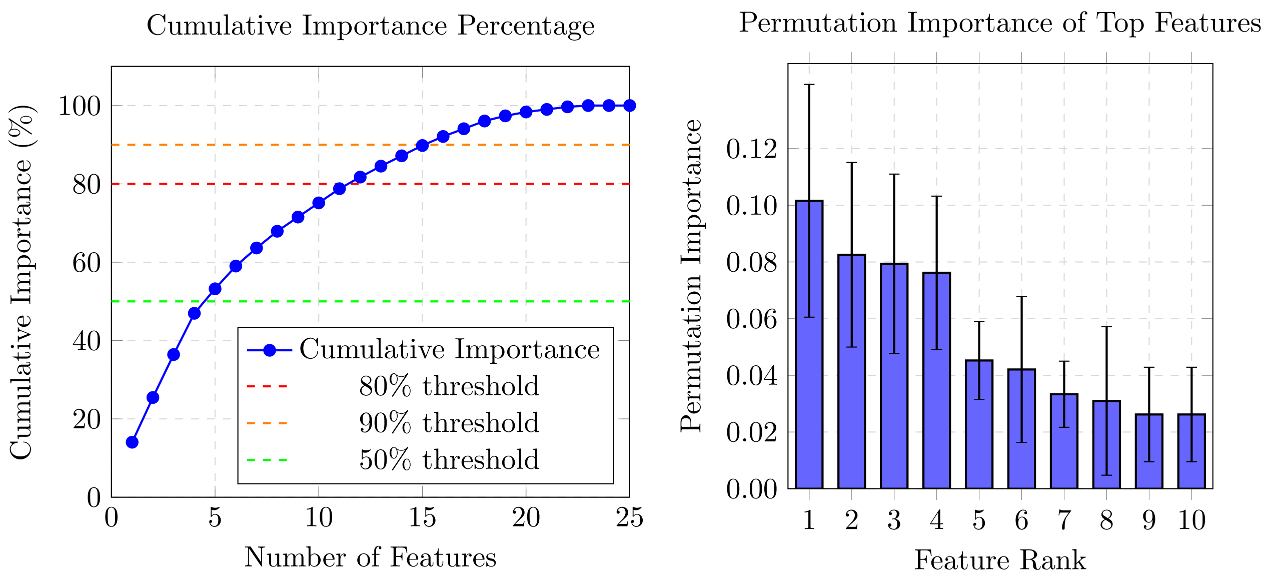


Figure 4.9.: Permutation importance analysis of latent space features. Left: Bar chart showing the permutation importance of the top 10 features with error bars representing standard deviation across models. Right: Cumulative importance percentage showing how many features are needed to capture the majority of the predictive power. The dashed lines indicate 80%, 90%, and 50% thresholds.

The permutation importance analysis examined the complete downstream task performance, encompassing the weak learner F, aggregation function π, and protein language model Φ. A small subset of features was responsible for the majority of predictive performance.

The relationship between embedding dimensions and self-similarity patterns was examined. The heatmap (Figure 4.2) shows that regions exhibiting dissimilarity from the majority of positions (blue regions) also demonstrate dissimilarity among themselves.

A supervised basis transformation was applied to identify the minimal set of dimensions explaining variance in self-similarity weighted pooling patterns across the TEM enzyme family. The transformation procedure followed a weighted principal component analysis (PCA) approach, where a diagonal weight matrix $\mathbf{W} = \mathrm{diag}(\mathbf{w})$ was constructed from the self-similarity weights. The weighted covariance matrix was computed as:

$$\mathbf{C}_{\text{weighted}} = \mathbf{X}_{\text{centered}}^T \mathbf{W} \mathbf{X}_{\text{centered}} + \lambda \mathbf{I} \tag{4.1}$$

where $\mathbf{X}_{\text{centered}}$ represents the mean-centered embedding matrix, and regularization was applied with $\lambda = 10^{-6} \times \text{trace}(\mathbf{C}_{\text{weighted}})/d$ to ensure numerical stability. The rotation matrix $\mathbf{R}$ was obtained from the eigenvectors corresponding to the largest eigenvalues, and the final transformation was applied as $\mathbf{X}_{\text{transformed}} = \mathbf{X}_{\text{scaled}}\mathbf{R}$.

This weighted PCA procedure was applied family-wide, combining all 209 TEM proteins to derive a unified latent space representation. The resulting projection of TEM-1 in the transformed space, colored according to self-similarity weighted pooling values, is presented in Figure 4.10.

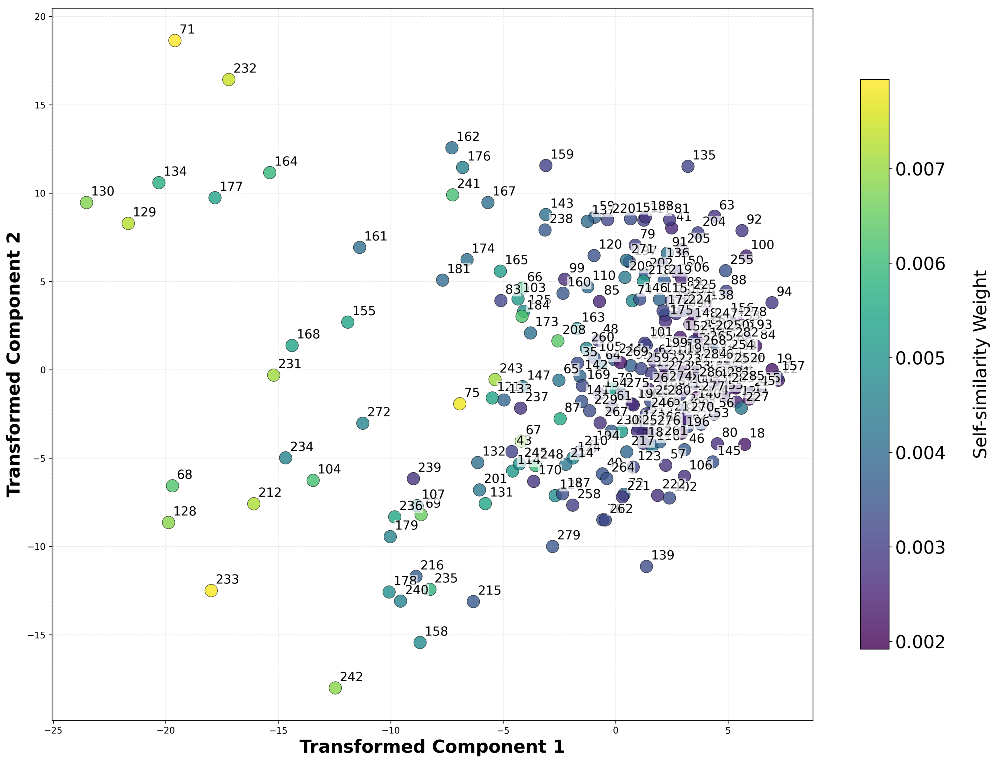


Figure 4.10.: Projection of TEM1 in the transformed space in a per token matter colored by the self similarity weighted pooling.

The first two principal components separate tokens according to their self-similarity pooling weights. Tokens positioned at the negative extreme of the first component (approximately −15 and below) correspond to residues with established biochemical functions.

Tokens 129 and 130 appear as outliers in the projection despite their proximity to the SDN loop region. Token 75 exhibits a high self-similarity weight but lacks clearly established biochemical function in current literature.

The variance analysis (Figure 4.11) shows monotonic decrease in explained variance with each additional component, consistent with principal component analysis. Approximately 300 features explain the cumulative variance in the latent space.

To validate that captured variance correlates with self-similarity weights, correlation analysis

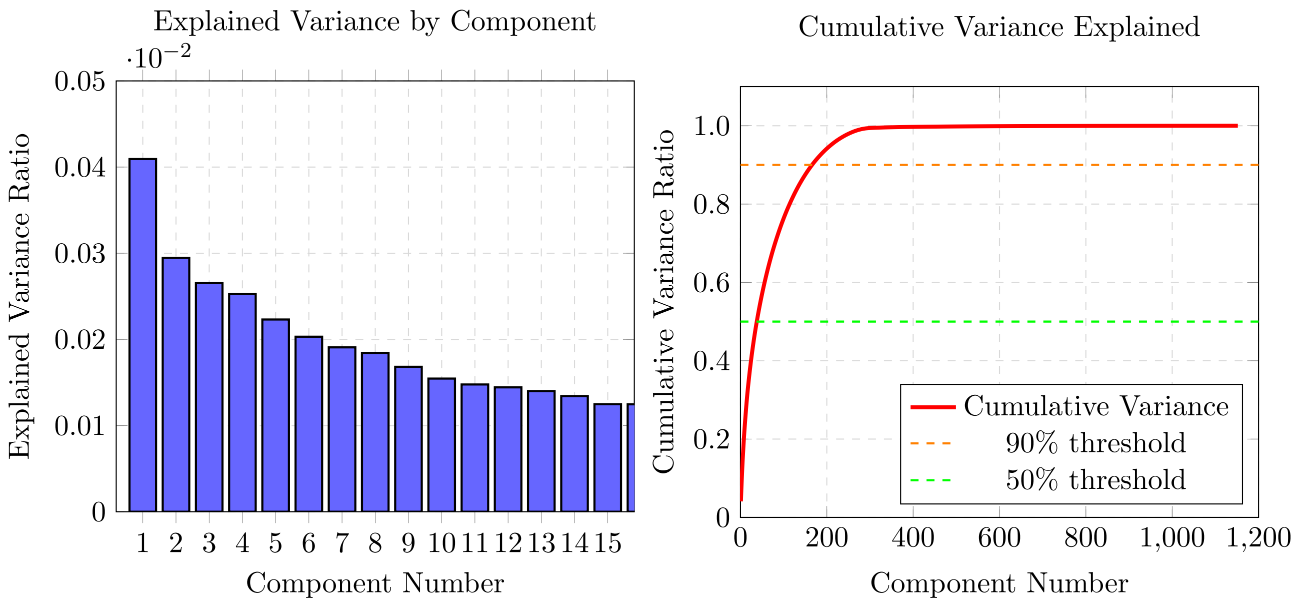


Figure 4.11.: Explained variance analysis after basis transformation. Left: Individual explained variance ratio for the first 15 components. Right: Cumulative variance explained across all components.

was performed between self-similarity weights and explained variance of individual components.

The correlation analysis involved computing the absolute correlation coefficient between the transformed embeddings along each principal component and the original self-similarity weights:

$$\text{correlation}_i = |\text{corr}(\mathbf{X}_{\text{transformed}}[:, i], \mathbf{w})| \tag{4.2}$$

where $\mathbf{X}_{\text{transformed}}[:, i]$ represents the projection of all tokens onto the i-th principal component and $\mathbf{w}$ represents the self-similarity weights.

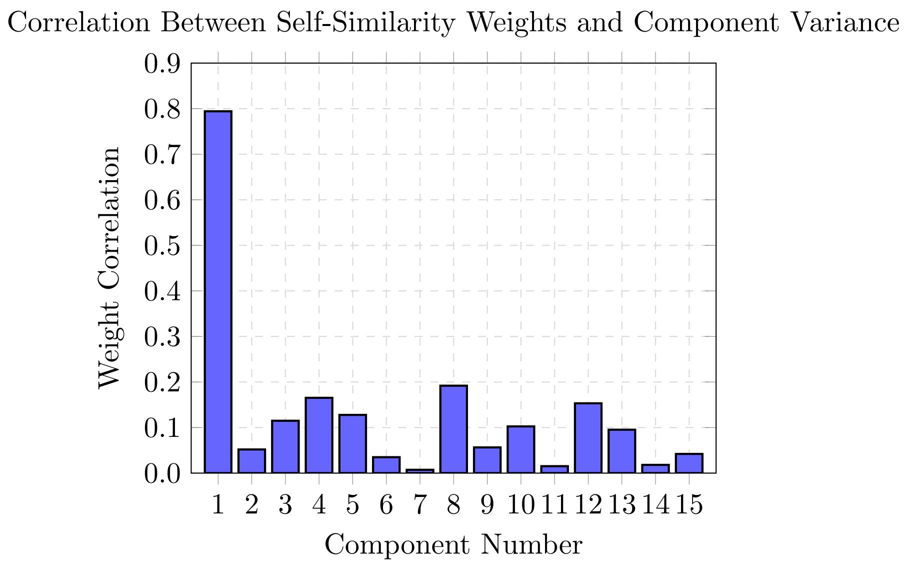


Figure 4.12.: Correlation between self-similarity weights and explained variance of the first 15 components after basis transformation. The first component shows the highest correlation (0.794), indicating it captures the most variance related to self-similarity weights.

The correlation analysis (Figure 4.12) demonstrates that the first principal component exhibits the strongest relationship with self-similarity weights, achieving a correlation coefficient of 0.794. This high correlation confirms that the first component effectively captures the variance most relevant to the self-similarity weighted pooling strategy, validating the supervised basis transformation approach. Subsequent components show substantially lower correlations, with the second component achieving only 0.052, indicating that the primary signal related to self-similarity patterns is concentrated in the first dimension of the transformed space.

These findings suggest that the embedding space contains a dominant axis of variation that aligns with functionally relevant self-similarity patterns. The strong correlation observed for the first component provides empirical support for the hypothesis that protein language models learn to encode biochemically meaningful information in their latent representations. Furthermore, the concentration of self-similarity-related variance in a single principal component underscores the potential for future investigations focusing on the projection along this primary axis, particularly for understanding the unique behavior of positions such as residue 75, which exhibits high self-similarity weights but unclear biochemical attribution in current literature.

5. Conclusion

Validation of Transfer Learning for Protein Phenotype Prediction

Transfer learning approaches achieved accuracies up to 97.6% for predicting antibiotic resistance phenotypes in TEM β-lactamases. This validates that protein language models encode functionally relevant information for biochemical prediction tasks.

This approach requires modest training requirements. With only 139 TEM variants spanning four phenotype classes (2b, 2be, 2ber, 2br), the models achieved high prediction accuracy using standard computational resources and straightforward implementation through the PyEED framework.

The widespread reliance on mean pooling in existing literature represents a limitation. Incorporating biochemical knowledge into pooling strategies showed improved performance, demonstrating that aggregation methodology can be as influential as model architecture selection.

Rethinking the Hierarchy of Optimization in Machine Learning Pipelines

The findings challenge conventional priorities in machine learning optimization. While the AI community typically focuses on model architecture and hyperparameters, embedding aggregation strategies can have equal or greater impact on performance. Performance differences between pooling methods often exceeded variance between different protein language models.

The self-similarity weighted pooling approach offers a computationally efficient alternative that can be integrated into existing workflows. Its implementation as a differentiable operation allows for end-to-end training and joint learning with downstream prediction tasks.

This approach extends beyond protein language models to other domains involving sequential data processing. Applications in natural language processing could benefit from similar attention-based aggregation strategies.

Emergent Biochemical Knowledge from Self-Supervised Learning

Protein language models spontaneously learn to encode functionally relevant information without explicit biochemical supervision. The self-similarity weighted pooling analysis identified functionally critical regions in TEM-1 β-lactamase through a purely data-driven approach, demonstrating that these models learn from contextual relationships between amino acid positions.

This finding provides a step toward making protein language models more interpretable. The self-similarity analysis offers insight into the functional logic encoded within their representations, suggesting these models may have learned a form of ”protein grammar”.

The implications for drug discovery and protein engineering include identifying novel functional sites in poorly characterized enzymes or predicting functional consequences of mutations in unstudied protein families.

Latent Space Organization and Functional Dimensionality

Analysis of latent space representations showed how protein language models organize biochemically relevant information. Permutation importance analysis demonstrated that certain embedding dimensions contribute disproportionately to phenotype prediction accuracy, with the top 10 features accounting for over 80% of cumulative predictive importance.

The convergence between positions identified through self-similarity weighted pooling and those selected based on biochemical knowledge validates that protein language models encode functional relevance aligned with experimental observations.

To investigate why a small subset of dimensions could capture the majority of predictive power, a supervised basis transformation was applied. This transformation explored how functional information is distributed across the 1,152-dimensional embedding space.

The basis transformation analysis identified position 75, which exhibited high self-similarity weights despite lacking clear biochemical attribution in current literature. This finding exemplifies how computational analysis can generate testable hypotheses for experimental validation.

The variance analysis revealed that the first 300 components capture the majority of variance related to cosine similarity patterns. The critical insight emerges from the correlation analysis rather than the variance distribution alone.

The weighted correlation analysis provided evidence for functional organization within the latent space. The first principal component showed strong correlation with self-similarity weights, indicating that functionally relevant information is highly concentrated rather than distributed across multiple directions.

The analysis suggests that resistance phenotypes are encoded through discrete functional modules distributed across the protein sequence. This modular organization implies that antibiotic resistance emerges from the coordinated action of spatially separated sequence elements.

These findings provide evidence for a ”protein grammar” that governs how amino acid sequences encode functional properties. The ability of self-similarity analysis to identify functionally relevant positions without prior biochemical knowledge suggests that protein language models have learned generalizable principles of protein function.

Implications and Future Directions

The convergence of results from multiple analytical approaches points toward a shift in biochemical sequence analysis. The integration of biological data through the PyEED framework, combined with aggregation strategies for protein embeddings, represents a paradigm that addresses limitations in the field.

The success of biochemically informed pooling strategies suggests future research should develop more sophisticated aggregation functions that can adaptively weight sequence positions based on task-specific requirements. Other approaches might incorporate structural information, evolutionary conservation patterns, or functional annotations.

This work highlights the importance of systematic evaluation across multiple dimensions of model configuration. Optimal performance requires careful consideration of the entire pipeline, not just individual components in isolation.

The implications extend to the global health challenge of antibiotic resistance. Computational tools that can rapidly predict resistance phenotypes from sequence data become valuable for clinical decision-making and surveillance efforts. The high accuracy achieved suggests such tools are approaching the reliability required for practical implementation.

Critical questions emerge from this research: Can the principles uncovered in TEM β-lactamases be generalized to other enzyme families and resistance mechanisms? How can the interpretability insights guide the design of next-generation antibiotics? Can the self-similarity weighting approach be extended to incorporate multiple types of biochemical information simultaneously?

This thesis demonstrates that integrating domain knowledge with machine learning approaches can yield insights greater than the sum of their parts. By bridging computational methodology and biochemical understanding, this work opens new avenues for addressing challenges in medicine while advancing computational biology.

List of Figures

List of Tables

Bibliography

[1] WorldHealthOrganization. Antimicrobial resistance. https://www.who.int/newsroom/fact-sheets/detail/antimicrobial-resistance, 2023. Accessed: 2025-05-12.

[2] A. Versporten, P. Zarb, I. Caniaux, et al. Antimicrobial consumption and resistance in adult hospital inpatients in 53 countries: results of an internet-based global point prevalence survey. *Lancet Glob Health*, 6:e619–e629, 2018.

[3] DJ Waxman, RR Yocum, and JL Strominger. Penicillinsand cephalosporins are active site-directed acylating agents: evidence in support of the substrate analogue hypothesis. *Philos Trans R Soc Lond B Biol Sci*, 289:257–271, 1980.

[4] T. Naas, S. Oueslati, RA. Bonnin, et al. Beta-lactamase database (bldb) -structure and function. *J Enzyme Inhib Med Chem*, 32:917–919, 2017.

[5] J. Jumper, R. Evans, A. Pritzel, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583–589, 2021.

[6] T. Hayes, R. Rao, H. Akin, et al. Simulating 500 million years of evolution with a language model. *Science*, 387:850–858, 2025.

[7] K. Bush. The abcd's of beta-lactamase nomenclature. *J Infect Chemother*, 19:549–559, 2013.

[8] D. Kim, S. Kim, Y. Kwon, et al. Structural insights for beta-lactam antibiotics. *Biomol Ther (Seoul)*, 31:141–147, 2023.

[9] RP. Ambler. The structure of beta-lactamases. *Philos Trans R Soc Lond B Biol Sci*, 289:321–331, 1980.

[10] JG. Sutcliffe. Nucleotide sequence of the ampicillin resistance gene of escherichiacoli plasmid pbr322. *Proc Natl Acad Sci USA*, 75:3737–3741, 1978.

[11] K. Bush and GA. Jacoby. Updated functional classification of beta-lactamases. *Antimicrob Agents Chemother*, 54:969–976, 2010.

[12] E.B. Chaïbi, D. Sirot, G. Paul, and R. Labia. Inhibitor-resistant tem beta-lactamases: phenotypic, genetic and biochemical characteristics. *J Antimicrob Chemother*, 43:447–58, 1999.

[13] D.L. Paterson and RA. Bonomo. Extended-spectrum beta-lactamases: a clinical update. *Clin Microbiol Rev*, 18:657–686, 2005.

[14] George Minasov, Xiaojun Wang, and Brian K. Shoichet. An ultrahigh resolution structure of tem-1 β-lactamase suggests a role for glu166 as the general base in acylation. *Journal of the American Chemical Society*, 124(19):5333–5340, 2002.

[15] A. Egorov, M. Rubtsova, V. Grigorenko, I. Uporov, and A. Veselovsky. The role of the ω-loop in regulation of the catalytic activity of tem-type β-lactamases. *Biomolecules*, 9:854, 2019.

[16] S. Farzaneh, E.B. Chaibi, J. Peduzzi, M. Barthelemy, R. Labia, J. Blazquez, and F. Baquero. Implication of ile-69 and thr-182 residues in kinetic characteristics of irt-3 (tem-32) beta-lactamase. *Antimicrob Agents Chemother*, 40(10):2434–2436, Oct 1996.

[17] X.J. Wang, G. Minasov, and B.K. Shoichet. Evolution of an antibiotic resistance enzyme constrained by stability-activity trade-offs. *J. Mol. Biol.*, 320:85–95, 2002.

[18] M. Orencia, J. Yoon, J. Ness, W.P.C. Stemmer, and R.C. Stevens. Predicting the emergence of antibiotic resistance by directed evolution and structural analysis. *Nature Structural & Molecular Biology*, 8:238–242, 2001.

[19] W. Huang and T. Palzkill. A natural polymorphism in beta-lactamase is a global suppressor. *Proc Natl Acad Sci USA*, 94(16):8801–8806, Aug 1997.

[20] V. Sideraki, W. Huang, T. Palzkill, and H.F. Gilbert. A secondary drug resistance mutation of tem-1 beta-lactamase that suppresses misfolding and aggregation. *Proc Natl Acad Sci USA*, 98(1):283–288, Jan 2001.

[21] J. Hecky and K.M. Müller. Structural perturbation and compensation by directed evolution at physiological temperature leads to thermostabilization of beta-lactamase. *Biochemistry*, 44(38):12640–12654, Sep 2005.

[22] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning.* Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, MA, second edition, 2018. This work is subject to a Creative Commons CC BY-NC-ND license. Subject to such license, all rights are reserved. © 2018 Massachusetts Institute of Technology. All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher. This book was set in LATEX by the authors. Printed and bound in the United States of America.

[23] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[24] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441, 1933.

[25] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[26] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018.

[27] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[28] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[29] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv*, 1412.6980, 2014.

[30] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[31] Navid Naderi, Navid Alizadeh, and Rohit Singh. Aggregating residue-level protein language model embeddings with optimal transport. *Bioinformatics Advances*, 1:vbaf060, 2024.

[32] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[33] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv*, 1409.0473, 2014.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv*, 1706.03762, 2017.

[35] E.C. Alley, G. Khimulya, S. Biswas, et al. Unified rational protein engineering with sequence-based deep representation learning. *Nature Methods*, 16:1315–1322, 2019.

[36] Ahmed Elnaggar, Michael Heinzinger, Christian Dallago, Ghalia Rehawi, Yu Wang, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Martin Steinegger, Debsindhu Bhowmik, and Burkhard Rost. Prottrans: Towards cracking the language of life's code through self-supervised deep learning and high performance computing. *CoRR*, abs/2007.06225, 2020.

[37] L. C. Vieira, M. L. Handojo, and C. O. Wilke. Medium-sized protein language models perform well at transfer learning on realistic datasets. *bioRxiv*, 2024.

[38] Andrew Dickson and Mohammad R K Mofrad. Fine-tuning protein embeddings for functional similarity evaluation. *Bioinformatics*, 40(8):btae445, 07 2024.

[39] Z. Lin et al. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379:1123–1130, 2023.

[40] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[41] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[42] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[43] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[44] T. Duan, A. Anand, D. Ding, K. Thai, S. Basu, A. Y. Ng, and A. Schuler. Ngboost: Natural gradient boosting for probabilistic prediction. *ICML*, 2020.

[45] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. *KDD*, 2016.

[46] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, and Q. Ye. Lightgbm: A highly efficient gradient boosting decision tree. *NeurIPS*, 2017.

[47] Wikipedia. Fasta format, 2023. Accessed: 2025-05-12.

[48] PyEED. Pyeed, 2023. Accessed: 2025-05-12.

[49] Neo4j. Neo4j, 2025. Accessed: 2025-05-12.

[50] Neo4j. Neo4j 5.26 docker image, 2025. Accessed: 2025-05-12.

[51] NCBI Resource Coordinators. Database resources of the national center for biotechnology information. *Nucleic Acids Research*, 46(D1):D8–D13, 2018.

[52] The UniProt Consortium. Uniprot: the universal protein knowledgebase in 2023. *Nucleic Acids Research*, 51(D1):D523–D531, 2023.

[53] A. Alcock et al. Card 2023: Expanded curation, support for machine learning, and resistome prediction at the comprehensive antibiotic resistance database. *Nucleic Acids Research*, 51(D1):D690–D699, 2023.

[54] The Gene Ontology Consortium. The gene ontology resource: enriching a gold mine. *Nucleic Acids Research*, 49(D1):D325–D334, 2021.

[55] T. Naas, S. Oueslati, R. A. Bonnin, M. L. Dabos, A. Zavala, L. Dortet, P. Retailleau, and B. I. Iorga. Beta-lactamase database (bldb) - structure and function. *J. Enzyme Inhib. Med. Chem.*, 32(6):917–919, 2017.

[56] C. Camacho, G. Coulouris, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer, and T.L. Madden. Blast+: architecture and applications. *BMC Bioinformatics*, 10(421), 2008.

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[58] A. Rives, J. Meier, T. Sercu, S. Goyal, Z. Lin, J. Liu, D. Guo, M. Ott, C.L. Zitnick, J. Ma, and R. Fergus. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proc. Natl. Acad. Sci. U.S.A.*, 118(15):e2016239118, 2021.

[59] I. Pudžiuvelytė, K. Olechnovič, E. Godliauskaite, K. Sermokas, T. Urbaitis, G. Gasiunas, and D. Kazlauskas. Temstapro: protein thermostability prediction using sequence representations from protein language models. *Bioinformatics*, 40(4):btae157, 2024.

[60] Z. Wang, D. Xie, D. Wu, et al. Robust enzyme discovery and engineering with deep learning using catapro. *Nature Communications*, 16:2736, 2025.

[61] S. Biswas, G. Khimulya, E.C. Alley, et al. Low-n protein engineering with data-efficient deep learning. *Nature Methods*, 18:389–396, 2021.

[62] A. Kroll, S. Ranjan, MKM Engqvist, and MJ Lercher. A general model to predict small molecule substrates of enzymes based on machine and deep learning. *Nature Communications*, 14:2787, 2023.

[63] H. Yu, H. Deng, J. He, et al. Unikp: a unified framework for the prediction of enzyme kinetic parameters. *Nature Communications*, 14:8211, 2023.

[64] H. Stärk, C. Dallago, M. Heinzinger, and B. Rost. Light attention predicts protein location from the language of life. *Bioinformatics Advances*, 1(1):vbab035, 2021.

[65] T. Duan, A. Anand, D. Ding, K. Thai, S. Basu, A. Y. Ng, and A. Schuler. Ngboost: Natural gradient boosting for probabilistic prediction. *arXiv*, 1910.03225, 2020.

[66] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Acknowledgments

I would like to thank first Prof. Jürgen Pleiss for his supervision, all the help provided, and his guidance throughout this work.

Next, I thank Max Häußler, Jan Range, Alina Lachheim, and Torsten Gieß for the helpful discussions and their experienced support in creating my first proper scientific work.

I also thank Arthur Sieglin and Kriti Sharma for their helpful explanations regarding biochemistry and their continued motivation throughout this work.

Finally, I would like to thank Isaac Choi and Annika Brauchler for providing helpful insight into new machine learning approaches.

Niklas Abraham
Stuttgart, 30. September 2025

Declaration of Authorship

I hereby certify that I have written the present thesis entitled **Understanding antibiotic resistance by machine learning** independently and that the work contained herein is my own. All formulations and concepts taken verbatim or in substance from printed or unprinted material have been cited according to the rules of good scientific practice and indicated by exact references to the original source. The same applies to all illustrations. The present thesis has not been submitted to another university for the award of an academic degree in this form.

Place, Date:	Stuttgart, 30. September 2025
Signature:	______________________________
	Niklas Abraham

A. Supporting Information

A.1. TEM-1 Amino Acid Sequence with Standard Numbering

For the complete table, please refer to the CSV file provided in the digital appendix.

Amino acid in TEM-1,Amino acid position in TEM-1,standard numbering based on lahey,
M,1,3,start of signal peptide S,2,4, I,3,5, Q,4,6, H,5,7, F,6,8, R,7,9, V,8,10, A,9,11, L,10,12,
I,11,13, P,12,14, F,13,15, F,14,16, A,15,17, A,16,18, F,17,19, C,18,20, L,19,21, P,20,22, V,21,23,
F,22,24, A,23,25,end of signal peptide H,24,26,start of mature protein P,25,27, E,26,28,
T,27,29, L,28,30, V,29,31, K,30,32, V,31,33, K,32,34, D,33,35, A,34,36, E,35,37, D,36,38,
Q,37,39, L,38,40, G,39,41, A,40,42, R,41,43, V,42,44, G,43,45, Y,44,46, I,45,47, E,46,48,
L,47,49, D,48,50, L,49,51, N,50,52, S,51,53, G,52,54, K,53,55, I,54,56, L,55,57, E,56,58, S,57,59,
F,58,60, R,59,61, P,60,62, E,61,63, E,62,64, R,63,65, F,64,66, P,65,67, M,66,68, M,67,69,
S,68,70, T,69,71, F,70,72, K,71,73, V,72,74, L,73,75, L,74,76, C,75,77, G,76,78, A,77,79,
V,78,80, L,79,81, S,80,82, R,81,83, V,82,84, D,83,85, A,84,86, G,85,87, Q,86,88, E,87,89,
Q,88,90, L,89,91, G,90,92, R,91,93, R,92,94, I,93,95, H,94,96, Y,95,97, S,96,98, Q,97,99,
N,98,100, D,99,101, L,100,102, V,101,103, E,102,104, Y,103,105, S,104,106, P,105,107, V,106,108,
T,107,109, E,108,110, K,109,111, H,110,112, L,111,113, T,112,114, D,113,115, G,114,116,
M,115,117, T,116,118, V,117,119, R,118,120, E,119,121, L,120,122, C,121,123, S,122,124,
A,123,125, A,124,126, I,125,127, T,126,128, M,127,129, S,128,130, D,129,131, N,130,132,
T,131,133, A,132,134, A,133,135, N,134,136, L,135,137, L,136,138, L,137,139, T,138,140,
T,139,141, I,140,142, G,141,143, G,142,144, P,143,145, K,144,146, E,145,147, L,146,148, T,147,149,
A,148,150, F,149,151, L,150,152, H,151,153, N,152,154, M,153,155, G,154,156, D,155,157,
H,156,158, V,157,159, T,158,160, R,159,161, L,160,162, D,161,163, R,162,164, W,163,165,
E,164,166, P,165,167, E,166,168, L,167,169, N,168,170, E,169,171, A,170,172, I,171,173, P,172,174,
N,173,175, D,174,176, E,175,177, R,176,178, D,177,179, T,178,180, T,179,181, M,180,182,
P,181,183, A,182,184, A,183,185, M,184,186, A,185,187, T,186,188, T,187,189, L,188,190,
R,189,191, K,190,192, L,191,193, L,192,194, T,193,195, G,194,196, E,195,197, L,196,198,
L,197,199, T,198,200, L,199,201, A,200,202, S,201,203, R,202,204, Q,203,205, Q,204,206,
L,205,207, I,206,208, D,207,209, W,208,210, M,209,211, E,210,212, A,211,213, D,212,214,
K,213,215, V,214,216, A,215,217, G,216,218, P,217,219, L,218,220, L,219,221, R,220,222,
S,221,223, A,222,224, L,223,225, P,224,226, A,225,227, G,226,228, W,227,229, F,228,230,
I,229,231, A,230,232, D,231,233, K,232,234, S,233,235, G,234,236, A,235,237, G,236,238,
E,237,240, R,238,241, G,239,242, S,240,243, R,241,244, G,242,245, I,243,246, I,244,247, A,245,248,

A,246,249, L,247,250, G,248,251, P,249,252, D,250,254, G,251,255, K,252,256, P,253,257, S,254,258, R,255,259, I,256,260, V,257,261, V,258,262, I,259,263, Y,260,264, T,261,265, T,262,266, G,263,267, S,264,268, Q,265,269, A,266,270, T,267,271, M,268,272, D,269,273, E,270,274, R,271,275, N,272,276, R,273,277, Q,274,278, I,275,279, A,276,280, E,277,281, I,278,282, G,279,283, A,280,284, S,281,285, L,282,286, I,283,287, K,284,288, H,285,289, W,286,290,end of mature protein