The present work was submitted to the
Institute of Software and Tools for Computational Engineering (RWTH Aachen University)
and to the
Institute of Biological Information Processing (Forschungszentrum Jülich).

Evaluation of various machine learning approaches to predicting enzyme mutation data

Bachelor Thesis

Presented by Tilman Hoffbauer
Matr.-No. 396570

Supervised by Univ.-Prof. Dr. rer. nat. Uwe Naumann
Univ.-Prof. Dr. rer. nat. Birgit Strodel

Aachen, September 6, 2021

I Abstract

Enzyme engineering plays a crucial role for industry and research, but the expensive and time-consuming evaluation of mutants in the laboratory limits the number of variants that can be explored. Therefore, researchers started to investigate the usage of machine learning for predicting enzyme mutation effects. Many different algorithms and embeddings have been developed in this context.
This work contains a comparison of multiple different embedding strategies and machine learning algorithms. It was shown that embeddings based on transfer learning improve the accuracy of the predictions across a variety of different proteins. Both a novel technique for processing multiple sequence alignments with autoencoders and the application of state-of-the-art natural language processing techniques have been investigated. On the model side, a novel k-convolutional neural network that uses the three-dimensional structure of the protein was developed and compared to the other models. The application of support vector machines that use transfer learning provided accurate predictions of protein mutation effects. All hyperparameters have been optimized automatically with distributed computing.
Furthermore, the comparison is accompanied by a thorough sensitivity analysis, which discovered multiple interesting effects, including starting points for further improvements. Additionally, the effects of various properties inherent to the datasets were investigated. It was shown that a larger dataset size strongly correlates to higher performance and that the lack of mutants with multiple mutations in the training dataset degrades performance significantly. Finally, a measure of epistasis was defined and its negative correlation to model performance was shown. Using the sensitivity analysis, it was demonstrated that the proposed methods can learn some of the epistasis in the dataset. In conclusion, this work provides multiple contributions to the field which will aid in further improvement of protein mutation effect prediction and its application in industry and research.

Contents

This document contains hyperlinks for indices, cross-references, URLs, and citations in its digital form. Additionally, for a limited time, there are **interactive versions of the plots available online** by clicking on them or browsing to `https://online-figures.thoffbauer.de/bachelor-thesis`.

1 Introduction to mutagenesis studies for enzyme optimization

1.1 Topic relevance

A natural organism has a variety of tasks it must accomplish, e.g. processing food to usable energy for movement. Many of these require a chain of chemical reactions to transform the reactants into the desired products [1–3]. However, many of these reactions would not happen at the required speed naturally. To catalyze these reactions, organisms use i.a. enzymes, that are specifically tuned to the reaction and the environmental conditions and accelerate the reaction without being consumed in the process by lowering the activation free energy. For example, lactase is an enzyme that can separate lactose (milk sugar) into galactose and glucose. While the reactant cannot be absorbed by the gastrointestinal wall, the products can be transferred into the bloodstream. Thus, this enzyme enables a reaction that is crucial to the processing of milk, and humans without this enzyme suffer lactose intolerance [4]. Besides enzymes, other proteins are engineered, too. For example, the green fluorescent protein (GFP) is used as a marker in many studies [5–7].

Enzymes are proteins, and as such, they can be defined as a chain of amino acids. There are 20 dominant naturally occurring amino acids, called canonical amino acids. The DNA encodes the protein sequences and is transcribed to mRNA, which is finally translated to the corresponding primary amino acid sequence. A functional protein or enzyme is obtained after folding to its native three-dimensional structure. Thus, the properties of a protein can be changed by altering some of the amino acids in its sequence [8]. A single alteration is called a mutation, while the resulting sequence is a mutant of the wild type. This process is an important part of evolution, where proteins mutate to suit the changing environmental properties [9].

For industry and research, enzymes are of great interest due to their catalytic properties. However, taking a wild type (WT) enzyme from a suitable organism might not be the best choice. Instead, it can be beneficial to create mutants from it. Then, each mutant can be assessed in terms of the target property, e.g. activity or stability, and the best mutant can finally be used in subsequent applications. This process is known as enzyme engineering and especially the method of directed evolution and has become increasingly important in the past years [10]. For her pioneering work in this field, Frances H. Arnold was awarded the Nobel Prize in chemistry in 2018 together with George P. Smith and Sir Gregory P. Winter [11]. The basic mutagenesis study template is as follows (Figure 1): First, select a set of mutants. Second, evaluate the target value for each mutant. Third, repeat this process until no further optimization is possible or a sufficiently good enzyme was found. When selecting the next set of mutants during repetition, the mutants are selected based on the existing knowledge of the mutational landscape derived in prior iterations.

This process comes with several problems: On the one hand, due to the length of protein sequences (> 100 amino acids), the directed evolution approach suffers from an exponential blowup, i.e. the number of possible mutants grows exponentially with the number of mutations. On the other hand, experimentally evaluating a single mutant is expensive, as the protein must be expressed by bacteria and evaluated with an appropriate experimental setup. Consequently, there are two ways of making directed evolution feasible: First, one can limit the mutations considered in the first iteration by careful selection [12]. Second, one can try to predict a specific mutant's target value from the knowledge obtained in prior iterations before conducting the experimental evaluation. For the latter, machine learning has become increasingly important in recent years [13].

Machine learning promises to derive a model from existing enzyme mutation data that predicts the target value for new mutants. For example, one can start the first round of the mutagenesis

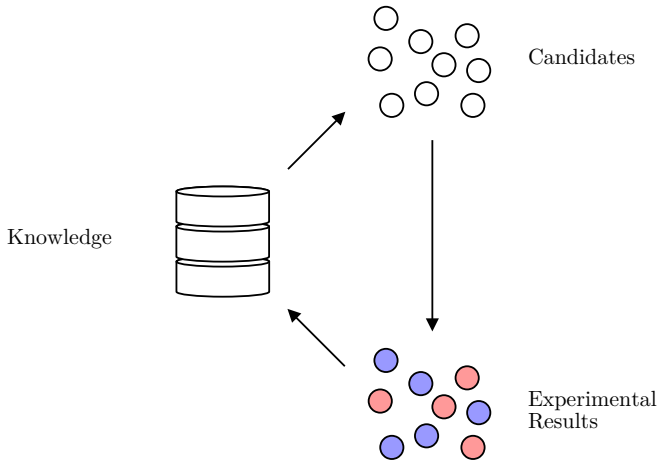


Figure 1: The directed evolution scheme.

study with a random selection of mutants. Then, the derived data can be used to train a model which predicts the target value from the sequence. This way, the model provides *a priori* information on the expected mutant quality which can be used as a heuristic in the selection of the next mutants to try experimentally [14].
However, there are several challenges in applying machine learning to enzyme mutation data. First, mutations are not additive. If one combines two mutations that positively influence the target value, the target value of their combination might be both higher or lower than the addition of the two values, i.e. there is not necessarily a linear correlation. This is called positive or negative epistasis, respectively, and is a problem for predicting the target value for manifold mutants [8]. Because the protein folding can bring amino acids close to each other which are far apart in the sequence, these interactions can span over distant amino acids.
Additionally, the sequence-function data collected in a mutagenesis study is usually small due to the high cost of evaluating many different mutants. This is further constrained by the time of application of machine learning: Due to the use of the method in the study design, it should be applied as early as possible in the study, where not all sequence-function pairs have been collected, yet. The order of the study induces the additional problem of inhomogeneous training and test data. While a study typically starts with a set of single mutants, whose function values provide the training data for the algorithm, the model will be used to predict variants with multiple mutations for later iterations. Details on the designs of the studies used in this thesis will be presented later.

1.2 Related work

In a comprehensive study, Xu et al. [15] compared many different preprocessing techniques and machine learning algorithms for the prediction of target values for four publicly available datasets with 81 - 51,715 data points. They compared many classical machine learning methods and two neural network architectures. From these, the best performing models were a support vector machine and a stacked convolutional neural network. They also used advanced ways of embedding the amino acids into numerical feature vectors. Most notably, the best performing embeddings include features derived from AAindex [16–19], a database containing various numerical features for each of the 20 canonical amino acids and their pairwise combinations. However, Xu et al. have found the embedding selection to be less important than the machine learning algorithm selection.

Cadet et al. [8] took a different way in their development of the innovative sequence-activity relationship model (innov'SAR) by using a significantly smaller dataset with just 43 data points. They found that when using a partial least-squares regressor, i.e. a linear regressor, applying the fast Fourier transformation (FFT) on the input data encoded by just one of the AAindex values yields significantly better results than without applying the FFT. They report a high coefficient of determination for their test dataset that comprised five arbitrarily selected mutants.

In a literature overview, Yang et al. [20] provided a high-level comparison of different machine learning methods and two case studies. One of the case studies included innov'SAR. They developed a flow chart for selecting a machine learning algorithm. Specifically, the flow chart only recommends neural networks for more than 10 000 samples. In many other cases, they recommend support vector machines or decision trees. They promoted the usage of transfer learning, where knowledge from large databases of sequences with unknown functions should be transferred to the specific problem at hand.

One approach in this direction is the usage of evolutionary data. To this end, multiple evolutionary similar sequences are aligned. On manual inspection, one can find interesting information in these alignments [9]. Alternatively, one can use unsupervised machine learning methods to extract useful information from them automatically. Sinai et al. [21] used a variational autoencoder for learning a latent representation of the evolutionary landscape which consisted of just 5 variables. The idea behind this is that although there are 20^n sequences with length n, only a small fraction of these sequences are functionally relevant and in the considered evolutionary neighborhood. From the autoencoder, a probability of every single sequence occurring in this evolutionary neighborhood can be derived. Sinai et al. showed that this probability correlates with enzyme activity. Additionally, one can inspect the internal latent representation learned and observe clusters of similar sequences.

Similar to the local evolutional landscape of a single mutant, one can train models on the global landscape of all known protein sequences. In this context, transferring methods from natural language processing (NLP) to protein data is popular. Language models like BERT [22] try to approximate the grammar of a language by modeling the probability of a masked word given its context[1]. For example, given the context "The chef [mask] the meal.", the language model should predict a high probability for the word "cooked", but a low probability for the word "ate" (Example adapted from Clark et al. [23]). The unique advantage of these models is that they can train on unlabeled data, i.e. on raw sentences with unknown meaning. There are millions of sentences available on the internet. Although this task is not directly useful for most problems, the features the language model learns can be transferred to downstream tasks, e.g. question answering. These transfer learning approaches are the current state-of-the-art for many NLP problems [22]. The popular word2vec model [24, 25] was transferred to protein sequences in the development of ProtVec [26]. However, Xu et al. have found its embeddings to be not useful for mutation effect prediction [15]. But, in the past few years, many new language models have been developed, that provide an increasingly better understanding of natural language beyond word co-occurrence. Recently, Elnaggar et al. [27] transferred many state-of-the-art language models from NLP to protein sequences, including Bert [22], Albert [28], ELECTRA [23], T5 [29], and XLNet [30]. They trained recent state-of-the-art language models on large genome databases of millions of unlabeled amino acid sequences, resulting in a set of protein language models. The learned features can be extracted as numerical vectors from the protein model. They showed the use of these embeddings in downstream tasks like protein location classification. Additionally, basic properties like protein type or acid charge can be observed in a two-dimensional projection of the embedding space.

[1]There are variations of this problem, but they have in common that the model tries to infer missing information from context.

1.3 Aims of this work

The aim of this work is the investigation of various machine learning approaches to predicting the target value of a mutant sequence, given a set of known sequence / target-value relationships. To this end, multiple ways of feature engineering and different machine learning algorithms will be combined and their performances are assessed on various datasets. The primary aim is to compare the performance of these methods to each other and find their strengths and weaknesses. The datasets differ in size, target protein, target value, mutation distribution, and mutagenesis study type. Secondary, the usage of transfer learning will be explored, i.e. the transfer of knowledge derived from unlabeled sequences to the specific protein and its mutants at hand. Here, both a global approach that uses many arbitrary sequences and a local approach that uses an evolutionary neighborhood will be employed. Additionally, comparisons will be made regarding the distribution of single and manifold mutants and the size of the dataset itself. Finally, a sensitivity analysis will be conducted on suitable models. This analysis allows for the attribution of the prediction to the individual inputs.

2 Main definitions and problem statement

2.1 Problem definition

As stated before, every protein can be defined by its amino acid sequence, where only the 20 canonical amino acids are relevant for most proteins. These amino acids have a one-letter encoding standardized by IUPAC. In this work, only substitutions of amino acids, and no deletions nor insertions, are considered.

Definition 1 *A* sequence *is a list of amino acids* $a_1 \dots a_m$ *with* $m \in \mathbb{N}$ *and* $a_i \in \mathcal{A} = (A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y)$ *for all* $i \in \{1, \dots, m\}$. *Then,* $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ *with* $n \in \mathbb{N}$ *is a* dataset, *where* $x_i \in \mathcal{A}^m$ *is a sequence and* $y_i \in \mathbb{R}$ *its target value for all* $i \in \{1, \dots, n\}$. *There is exactly one sample* (x_{wt}, y_{wt}) *in the dataset called the* wild type. *A mutation is a substitution of one amino acid* a_i *at position* i *in the wild type sequence by another amino acid* a_i', *which is written* $a_i i a_i'$, *e.g. A53Y. The set of all distinct mutations in a dataset and a sequence will be denoted as* $mut(\mathcal{D})$ *and* $mut(x)$ *respectively.*

Now, the goal is to find an algorithm that computes a model from this dataset predicting the target value given a sequence.

Definition 2 *A* model *is a function* $f : \mathcal{A}^m \to \mathbb{R}$ *which predicts the target value of a given sequence.*

2.2 Performance measures

Given a model, an immediate question arises: Is this model a good approximation or not? For this, a dataset with sequences and target values is necessary. There are several ways of measuring performance:

Definition 3 *The* root-mean-square error (RMSE) *is defined as*

$$RMSE(f, \mathcal{D}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2}.$$

Definition 4 *The* coefficient of determination (R^2) *is defined as*

$$R^2(f, \mathcal{D}) = 1 - \frac{\sum_{i=1}^{n} (y_i - f(x_i))^2}{\sum_{i=1}^{n} (y_i - \overline{y})^2},$$

where $\overline{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$ *is the mean of all target values.*

These two measures both work on the squared deviation between the predicted value and the real value, where higher deviations are worse. The RMSE is non-negative, and a value of 0 identifies a perfect model. Note that $R^2 \leq 1$ and a value of 1 identifies a perfect correlation. The R^2 value can become negative[2] which indicates a model which is worse than predicting the mean $\overline{y}$ of the dataset[3]. While RMSE focuses on the deviations, R^2 considers the explained variance. For an $R^2 = 0.6$, 60% of the variance in the target value can be explained by the model from the sequence. This can be derived from the formula by interpreting the numerator as the unexplained variance and the denominator as the total variance. The R^2 value is normalized

[2] R^2 is not necessarily the square of some real-valued R.

[3] However, note that $\overline{y}$ might be unknown *a priori*, e.g. for a test data set.

and can be compared between datasets. In contrast to this, the RMSE values depend on the scale of y and are thus incomparable between datasets.
A new set of measures can be defined if one considers the given regression problem as a binary classification problem, where the task is to predict whether the target value is greater than a given threshold. In this setting, one obtains both a predicted class and a true class for each sample.

Definition 5 *The confusion matrix of the binary classification problem for a threshold $c \in \mathbb{R}$ is a 2x2 matrix counting the individual cases:*

		Prediction	
		$f(x_i) > c$	$f(x_i) \leq c$
True	$y_i > c$	*True positives (TP)*	*False negatives (FN)*
	$y_i \leq c$	*False positives (FP)*	*True negatives (TN)*

With the associated measures:

$$\begin{aligned} \mathit{Sensitivity}(f, \mathcal{D}, c) &= \frac{TP}{TP + FN} \\ \mathit{Precision}(f, \mathcal{D}, c) &= \frac{TP}{TP + FP} \\ \mathit{Accuracy}(f, \mathcal{D}, c) &= \frac{TP + TN}{TP + FN + FP + TN} \end{aligned}$$

This alternative problem formulation ignores errors that are caused by imprecise predictions as long as they point in the same direction, i.e. above or below the threshold. The three derived measures vary in their meaning:

- A high sensitivity value means that good mutants are detected by the model.
- A high precision value means that if a model predicts a mutant to be good, it is truly good.
- A high accuracy value hints at a good performance of the model in general on both positive and negative classifications.

The derived performance measures might be more interesting for some mutagenesis studies, as the precise value will be evaluated experimentally anyway, but a general heuristic *a priori*, e.g. better or worse than the wild type, is sufficient.

2.3 Model optimization with gradient descent

Throughout this thesis, there are multiple applications of gradient descent for model optimization. These models are all trained similarly:

Definition 6 *Given a model f_θ as in Definition 2 that is parameterized by $\theta \in \mathbb{R}^{|\theta|}$, a dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and a loss function $L(f_\theta(x_i), y_i)$ that computes the error of a prediction and is differentiable w.r.t. θ, gradient descent training names the following procedure:*

randomly split $\mathcal{D}$ into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{stop}}$ with a fixed proportion
initialize θ with small random values
while $\sum_{(x,y)\in\mathcal{D}_{\text{stop}}} L(f_\theta(x), y)$ does not increase consistently **do**
  for $\mathcal{D}_{\text{train}}$ in batches (X, Y) of size b **do**
    $l := \frac{1}{b}\sum_{i=1}^{b} L(f_\theta(x_i), y_i)$
    calculate $\frac{\mathrm{d}l}{\mathrm{d}\theta}$
    update θ along $\frac{\mathrm{d}l}{\mathrm{d}\theta}$ according to the optimizer to minimize l
  end for
end while

Note that the technique of stopping if the error on $\mathcal{D}_{\text{stop}}$ increases is called early-stopping. Due to stochasticity in the training process, training is only stopped if the error did not decrease in a fixed number of epochs. After stopping, θ is reset to the best intermediate value. In this work, the Adam optimizer [31] is used. The learning rate of the optimizer, i.e. the step size for weight updates, and the batch size are hyperparameters.
One has multiple losses to choose from. On the one hand, the RMSE is a common choice for regression tasks. On the other hand, classification problems occur in the context of transfer learning from unlabeled sequences. In contrast to the binary classification problem introduced earlier, these problems are often related to other tasks, e.g. predicting a sequence, where the selection of an amino acid for a specific position can be considered a multi-class classification problem. These outputs are usually represented as a probability distribution across the available classes. For this, the cross-entropy loss is better suited as it accounts for the probabilities assigned by the prediction to each class. For example, if the model predicts the wrong class but with low confidence, i.e. probability scores are high for multiple classes, the model will be rewarded for expressing its uncertainty.

Definition 7 *The* cross-entropy *loss for a true value y and a prediction $\hat{y}$ with C classes is defined as*

$$H(\hat{y}, y) = -\sum_{i=1}^{C} y_i \log \hat{y}_i.$$

As the cross-entropy loss expects a probability for each class, the individual probabilities must sum up to one. However, common models only produce so-called logits that do not sum up to one. To achieve a proper probability distribution, one can apply the softmax function [32], that normalizes these probabilities:

Definition 8 *The* softmax *function is an activation function $\sigma : \mathbb{R}^n \to [0, 1]^n$ with*

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_i}}.$$

A common technique to improve the generalization performance of a model trained with gradient descent is dropout [33]. This is exclusively applied during training and skipped during prediction. If dropout is applied to an intermediate vector, a random fraction $p \in [0, 1)$ of its elements is set to 0 during training. To balance the output between training and prediction, all other values are multiplied with $\frac{1}{p}$ during training. According to Hinton et al. [33], who proposed dropout initially, this technique reduces overfitting and thus improves model performance.

3 Methods for predicting enzyme mutation data

3.1 Data preprocessing

The raw amino acid sequence is not usable for most machine learning methods because these methods require a numerical feature vector. For this, embeddings can be used. In the following sections, the embeddings compared in this thesis are explained. If a model is trained on multiple embeddings, the individual embeddings are concatenated together.

3.1.1 Basic embeddings

Because we have categorical input data, as the amino acids are neither continuous nor naturally ordered, the most simple and common encoding is one-hot encoding.

Definition 9 *By* one-hot encoding *a sequence $x_i = a_1 \dots a_m \in \mathcal{A}^m$ every a_j is replaced by a vector $(0, \dots, 1, \dots, 0)$ with a 1 at the index of the amino acid in $\mathcal{A}$. This leads to one feature matrix $F \in \mathbb{R}^{m \times |\mathcal{A}|}$ per sample.*

Another basic encoding can be derived by considering only the mutations in the dataset and ignoring all other constant residues in the sequence.

Definition 10 *The* mutation indication *encoding is based on the set of mutations $mut(\mathcal{D}) = \{m_1, \dots, m_l\}$ with $l \in \mathbb{N}$. Each sequence x_i is replaced by a vector $v \in \{0,1\}^l$ with*

$$v_j = \begin{cases} 1 & \text{if } m_j \in mut(x_i) \\ 0 & \text{else.} \end{cases}$$

3.1.2 AAindex-based embeddings

In the AAindex database [16–19], there are a total of 553 different numerical values for each of the 20 canonical amino acids. Before further usage, every one of the 553 values is normalized to mean 0 and standard deviation 1. In a simple formulation, one uses all the 553 values for each amino acid:

Definition 11 *The* AAindex *embedding for a sequence $x_i = a_1 \dots a_m \in \mathcal{A}^m$ is obtained by replacing every a_j with the 553 AAindex values $v_a^T \in \mathbb{R}^{553}$ for this amino acid. This leads to one feature matrix $F \in \mathbb{R}^{m \times 553}$ per sample.*

However, it might be beneficial to use a subset of these values to reduce the number of input parameters and avoid overfitting. For this, Xu et al. [15] propose sScales, which is a subset of AAindex that was selected based on literature research. In contrary to their work, this thesis does not concern the derived values based on principal component analysis (PCA), as preliminary studies by the author of this thesis have shown that this approach is less effective.

Definition 12 *The* sScales *embedding for a sequence $s = a_1 \dots a_m \in \mathcal{A}^m$ is obtained by replacing every a_i with 13 AAindex values $v_a \in \mathbb{R}^{13}$ for this amino acid. This leads to one feature matrix $F \in \mathbb{R}^{m \times 13}$ per sample. The selected descriptors are listed in Table 1.*

identifier	publication	short description
BIOV880101	Biou et al., 1988 [34]	Information value for accessibility; average fraction 35%
BLAM930101	Blaber er al., 1993 [35]	Alpha helix propesnity of position 44 in T4 lysozyme
NAGK730101	Nagano, 1973 [36]	Normalized frequency of alpha-helix
TSAJ990101	Tsai et al., 1999 [37]	Volumes including the crystallographic waters using the ProtOr
NAKH920106	Nakashima-Nishikawa, 1992 [38]	AA composition of CYT of multi-spanning proteins
NAKH920107	Nakashima-Nishikawa, 1992 [39]	AA composition of EXT of multi-spanning proteins
NAKH920108	Nakashima-Nishikawa, 1992 [40]	AA composition of MEM of multi-spanning proteins
CEDJ970104	Cedano et al., 1997 [41]	Composition of amino acids in intracellular proteins (percent)
LIFS790101	Lifson-Sander, 1979 [42]	Conformational preference for all beta-strands
MIYS990104	Miyazawa-Jernigan, 1999 [43]	Optimized relative partition energies - method C
ARGP820101	Argos et al., 1982 [44]	Hydrophobicity index
DAWD720101	Dawson, 1972 [45]	Size
FAUJ880109	Fauchere et al., 1988 [46]	Number of hydrogen bond donors

Table 1: The list of AAindex descriptors used in the sScales embedding.

$\cdots$	C	V	R	K	F	$\cdots$
$\cdots$	C	W	R	Q	F	$\cdots$
$\cdots$	C	V	-	K	X	$\cdots$
$\cdots$	C	I	R	E	F	$\cdots$

Figure 2: An excerpt of an example MSA.

3.1.3 Structure

After an amino acid sequence is expressed in a cell, it folds to its three-dimensional structure. For a given wild type structure, it is assumed that the yet unknown mutant structures and therefore the positions of their atoms are not significantly different. Thus, the positions of the atoms are considered identical for all mutants. In some of the methods, structural information is used. Here, the position of an amino acid is defined as follows:

Definition 13 *For a sequence $x_i = a_1 \dots a_m \in \mathcal{A}^m$ the position $p_j \in \mathbb{R}^3$ of a_j is the position of the corresponding $C\alpha$ atom in the wild type structure.*

3.1.4 Evolutionary features

To augment the available data $\mathcal{D}$, one can incorporate similar sequences in the model training. For this, one can distill a set of sequences, that are similar to the sequences present in $\mathcal{D}$ and occur naturally. These sequences can then be aligned to each other, such that shared subsequences are matched (cf. Figure 2). This results in a multiple sequence alignment (MSA) that can contain gaps "-" and rare, non-canonical amino acids "X". In this thesis, amino acid insertions in aligned sequences are ignored. In Figure 2, one can observe that the amino acids in the second and fourth columns seem to be mutated at the same time.
The next step is to obtain knowledge like this that can be used by following models. For this, a linear autoencoder is used to model the dependencies between the different amino acids. To this end, each sequence x_i is one-hot encoded into a matrix $F_i \in \mathbb{R}^{m \times 22}$ which can be reshaped to a vector $v_i \in \mathbb{R}^{22m}$. Note that we now have 22 classes instead of 20 as in Definition 9 due to the gap "-" and rare amino acid "X" symbols. The autoencoder contains an encoder matrix $W_e \in \mathbb{R}^{l \times 22m}$, an encoder bias $b_e \in \mathbb{R}^l$, a decoder matrix $D \in \mathbb{R}^{22m \times l}$, and a decoder bias $b_d \in \mathbb{R}^{22m}$ where l is called the latent size. With this, the model is defined as

$$f_\theta(x_i) = \text{softmax}(\text{reshape}_{m \times 22}(D(W_e v_i + b_e) + b_d)) \text{ with } \theta = (W_e, b_e, W_d, b_d).$$

Here, softmax normalizes the output of the autoencoder to have a sum of 1 per position in the sequence, i.e. to output a probability distribution of amino acids for each position in the output (Definition 8). Now, both matrices are trained using gradient descent (Definition 6) to learn the identity function $v_i = f_\theta(x_i)$ under the cross-entropy loss (Definition 7). During training, dropout [33] is applied to the latent representation for regularization.
In this process, the latent size l is a hyperparameter. Usually, one chooses $l < 22m$, and as such, the encode-decode process is lossy and cannot learn the identity function for arbitrary sequences. Instead, the autoencoder will learn to exploit dependencies in the local evolutionary landscape. For example, if two mutations only occur together and not individually, just one variable would be sufficient for encoding this in the latent representation. For a new sequence, e.g. an artificial mutant from a mutagenesis study, which only has one of these two mutations, the autoencoder might still predict a high probability for both mutations. This discrepancy might provide insight to a model which is then trained on these probability distributions for target value prediction. This leads to the evolutionary embedding based on linear autoencoders as it will be used in this thesis:

Definition 14 *The* evolutionary embedding *replaces every sequence x_i with $f_\theta(v_i)$ where $v_i \in \mathbb{R}^{22m}$ is the one-hot encoding of this sequence into the 20 canonical amino acid classes plus the rare amino acid class "X" and the gap class "-". This leads to one feature matrix $F \in \mathbb{R}^{m \times 22}$ per sample.*

3.1.5 ProtTrans

For the ProtTrans embedding, the protein language models by Elnaggar et al. [27] are used. These models are trained to predict masked amino acids from the rest of the sequence. For example, given the sequence `...CV_KF...` adapted from Figure 2, the protein language model is trained to predict the masked amino acid `R` with high probability. To conduct this training, the only necessary input data are the amino acid sequences of naturally occurring proteins. Fortunately, there are millions of natural sequences in the Uniclust database [47] and billions of sequences in the BFD database [48, 49]. ELECTRA [23] varies this training goal, such that the model tries to find artificial mutations in the sequence which are inserted by an adversarial network. As the training task requires the model to reconstruct the masked/replaced amino acids, it must capture the "grammar" of natural amino acid sequences.
For every amino acid in a sequence, a numerical feature vector (embedding) can be extracted from the last layer of these protein language models[4]. These vectors can then be used as the input to the following models. Here, the protein language model selection (one of Bert [22], Albert [28], ELECTRA [23], T5 [29] or XLNet [30]) is a hyperparameter.

Definition 15 *The* ProtTrans embedding *replaces every amino acid a_i in a sequence x with v_i where $v_i \in \mathbb{R}^e$ is the output of the selected protein language model for this amino acid. This leads to one feature matrix $F \in \mathbb{R}^{m \times e}$ per sample.*

3.1.6 Spectral Embedding

Of all previous embeddings, a spectral version can be derived. Here, the "spectral embedding" refers to a transformation of the embedding from sequence space to frequency space. This idea is transferred from innov'SAR [8].

Definition 16 *The* spectral version *of an embedding is defined as the result of the following process:*

1. *Use a discrete Fourier transformation (DFT) to get the coefficients $X_0, \ldots, X_{n-1} \in \mathbb{C}$ from the underlying embedding $x_0, \ldots, x_{n-1} \in \mathbb{R}$ like the following:*

$$X_l = \sum_{k=0}^{n-1} x_k e^{-2\pi i \frac{l}{n} k},\ i = \sqrt{-1},\ l \in \{0, \ldots, n-1\}$$

   If the underlying embedding is two-dimensional, repeat this process for each series along the sequence length.

2. *Only use the first half of the output:*

$$n_{\frac{1}{2}} = \begin{cases} \frac{n}{2} + 1 & \text{if } n \text{ even} \\ \frac{n+1}{2} & \text{if } n \text{ odd} \end{cases}$$

[4] In this thesis, the alternative technique of fine-tuning the model on the downstream task showed negative results in preliminary studies. The author hypothesizes that the high number of parameters in the language model encourages overfitting during fine-tuning.

3. Compute the amplitudes by taking the normalized magnitude:

$$\hat{X}_l = \frac{2|X_l|}{n},\ l \in \{0, \ldots, n_{\frac{1}{2}} - 1\}$$

In the first step, the input is represented as a mixture of complex oscillations with frequencies $0, \frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}$, weighted by the complex factors $X_0, \ldots, X_{n-1}$ [50]. This can be seen from the inverse transformation, which reconstructs the sequence $x_0, \ldots, x_{n-1}$ from $X_0, \ldots, X_{n-1}$:

$$x_k = \sum_{l=0}^{n-1} \underbrace{X_l}_{\text{weight}} \cdot \underbrace{e^{2\pi i \frac{l}{n} k}}_{\text{oscillation } f=\frac{l}{n}}$$

The forward transformation can be computed efficiently using variations of the fast Fourier transform (FFT) algorithm. As the input sequence $x_0, \ldots, x_{n-1}$ is real-valued, $X_{n-l} = \overline{X_l}$ (complex conjugate) [50]. Thus, one can ignore the second half of the spectrum in step two as it does not contain new information. This is a special property of the DFT. In the third step, only the amplitudes of the individual frequencies are kept and the phase information is discarded.

3.2 Machine learning models

The previously presented techniques create either a one- or a two-dimensional embedding for each input sample. Given these embeddings, one can now perform machine learning on this data. There are many different machine learning algorithms and new ones are published regularly. In this work, many models are based on existing literature. However, the kCNN is a novel technique developed in this work. Some of the methods presented here only process one-dimensional data. For these, two-dimensional embeddings must be reshaped to one-dimensional embeddings. However, some of the methods can only process two-dimensional embeddings and cannot be used with one-dimensional embeddings.

3.2.1 Linear regression

In this thesis, linear regression is implemented by GLMNET [51]. As such, the model expects a feature vector $v_i \in \mathbb{R}^l$ for each sample sequence x_i that can be produced with one of the embedding methods defined in Subsection 3.1. The model function is defined as

$$f_{b,w}(v_i) = b + w^T v_i$$

with bias $b \in \mathbb{R}$ and weights $w \in \mathbb{R}^l$. The bias and weights are optimized to find the global minimum of the loss

$$L(v, y) = \underbrace{\frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} (y_i - f_{b,w}(v_i))^2}_{\text{mean squared error}} + \underbrace{\lambda \left((1-\alpha)||w||_2^2/2 + \alpha||w||_1 \right)}_{\text{regularization}}.$$

Here, the loss does not only contain the actual error but an additional regularization term. This is added to reduce the overfitting of noise in the training dataset by penalizing large values in w. The regularization term is a weighted sum between the L1- and L2-norm of the weight vector controlled by the hyperparameter α, as both norms provide different benefits to the final regression performance. Additionally, the regularization strength can be controlled with the hyperparameter λ. This is the most simple model used in this work and serves as a baseline.

3.2.2 Support vector machine

A support vector machine (SVM) is a machine learning tool that was originally developed for classification. However, its concepts can be broadened to ϵ support vector regression (ϵ-SVR) as defined by LIBSVM [52]. Here, the model expects a feature vector $v_i \in \mathbb{R}^l$ for each sample, too. The ϵ-SVR also fits a linear regression to the data points to minimize the error. It also has a regularization hyperparameter C similar to linear regression, where lower C values correspond to higher regularization. There are three main differences: First, the error is the mean absolute error. Second, the ϵ-SVR ignores errors that are below a certain threshold ϵ, which is another hyperparameter. Third, the input features are projected to a higher-dimensional space before fitting the linear regression. Interestingly, the solution to the linear regression problem only depends on the dot product between the projected data points. This enables the kernel trick [53], where one computes the dot product between two projected input samples in the higher-dimensional space without doing the projection. Instead, the dot product in the high-dimensional space is simplified analytically to a term that does not depend on the projected data points.

There are multiple kernel functions to choose from. This includes the linear kernel (no transformation), polynomial kernel (transform x to $(a_0, a_1x, a_2x^2, \ldots, a_dx^d)$ with $a_0, a_1, a_2, \ldots, a_d \in \mathbb{R}$ as coefficients), and the radial basis function (RBF) kernel (transform x to $(a_0, a_1x, a_2x^2, \ldots)$ with $a_0, a_1, a_2, \cdots \in \mathbb{R}$ as coefficients)[5]. This way, one can see the RBF kernel as an extension of the polynomial kernel which is, in turn, an extension of the linear kernel. Hence, only the RBF kernel is considered in this work. The RBF kernel function, that calculates the infinite-dimensional dot product, is

$$k(x, x') = e^{-\gamma||x-x'||^2}.$$

Alternatively to the infinite-dimensional interpretation, one can consider the kernel function as a similarity measure. For the RBF kernel function, we can observe that the similarity is reduced with increasing Euclidean distance between the two data points. Thus, the RBF kernel considers close data points for the approximation of the target value of a new data point. Here, γ is a scaling parameter for the distance between the data points and a hyperparameter.

3.2.3 Multi-layer perceptron

The multi-layer perceptron is a basic type of neural network. It consists of several chained neurons, where each neuron has inputs $x_1, \ldots, x_n \in \mathbb{R}$ and one output $y \in \mathbb{R}$ (cf. Figure 3 left). The output of a neuron is defined as

$$y = g\underbrace{\left(b + \sum_{i=1}^{n} x_i w_i\right)}_{=:a} \text{ with } b, w_1, \ldots, w_n \in \mathbb{R}$$

where g is the activation function. The activation function is added to the neuron to allow for non-linear outputs. In this work, the Rectified Linear Unit (ReLU) activation function is used [54]. It is defined as $g(a) = \max(0, a)$. To compute higher-order functions, one can stack multiple layers of neurons together. A network, where each neuron is connected to all neurons in the preceding layer, is called a multi-layer perceptron (cf. Figure 3 right).

[5]Here, the representations are only given for a scalar input feature $x \in \mathbb{R}$ instead of the input vector $v_i \in \mathbb{R}^l$, because the formulation for multi-dimensional input features is more complicated.

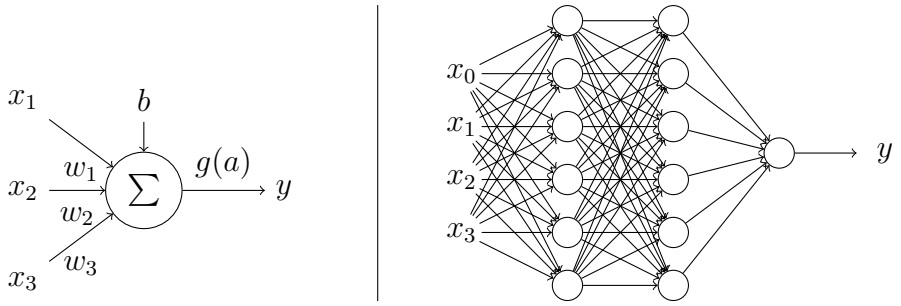


Figure 3: Left: A single neuron with several inputs, the bias, and one output. Right: A small multi-layer perceptron with four inputs, two hidden layers of six neurons each, and one output.

Each layer of neurons l_j can be calculated with efficient matrix operations (g is applied element-wise):

$$x_{j+1} = l_j(x_j) = g(W_j x_j + b_j) \text{ with } W_j \in \mathbb{R}^{\#\text{out} \times \#\text{in}} \text{ and } b_j \in \mathbb{R}^{\#\text{out}}$$

This allows for the formulation of the model function

$$f_\theta(x) = l_h(l_{h-1}(\dots l_2(l_1(x)) \dots)) \text{ with } \theta = (W_1, b_1, \dots, W_h, b_h)$$

of an MLP with $h \in \mathbb{N}$ layers. Similarly, the gradients $\frac{\mathrm{d}x_{j+1}}{\mathrm{d}W_j}$ and $\frac{\mathrm{d}x_{j+1}}{\mathrm{d}b_j}$ can be computed efficiently[6]. Thus, the gradient $\frac{\mathrm{d}f_\theta(x)}{\mathrm{d}\theta}$ can be computed using the chain rule, which is called backpropagation. This allows for the training of the MLP using gradient descent as in Definition 6. During training, dropout [33] is applied to the inputs of each layer to reduce overfitting.
For an MLP, the number and size of the hidden layers are hyperparameters. Interestingly, a two-layer MLP is able to approximate any function $f : X \to Y$ where X and Y are closed and bounded subsets of $\mathbb{R}^n$ for an arbitrary $n \in \mathbb{N}$ [55]. In practice, MLPs with more layers are common, however [56].

3.2.4 k-Convolutional neural network

In MLPs, one neuron is connected to all outputs of the previous layer or the complete input, but this can be generalized to allow for different connection schemes. This leads to the idea of convolutional neural networks (CNN) [57], which have first proven useful in handwritten digit recognition, where the input is a two-dimensional image. Here, each neuron connects to a crop of the input image. This concept can be further broadened to one- and three-dimensional data and CNNs have proven to be effective in many areas [32]. In this work, the convolution operation is further generalized to consider three-dimensional windows of amino acids in the corresponding enzyme structure instead of parts of the sequence. This leads to the formulation of the kCNN. The kCNN begins with an embedding that converts the input sequence $x = a_1 \dots a_m$ to a feature matrix $F \in \mathbb{R}^{m \times l}$ with $l \in \mathbb{N}$, too. Then, the following procedure is applied:

1. **k nearest neighbors:** Add the k nearest amino acids in the three-dimensional structure to each amino acid in the feature matrix, such that in the j-th layer of the feature tensor one finds the $(j-1)$-th nearest residue (cf. Figure 4). In the first layer, the base sequence is retained.

[6] Although the ReLU function $\max(0, a)$ is not differentiable at $a = 0$, this does not impose a problem in practice, as the exact value of 0 occurs rarely. If a value of 0 is encountered during training, the gradient is set to 0.

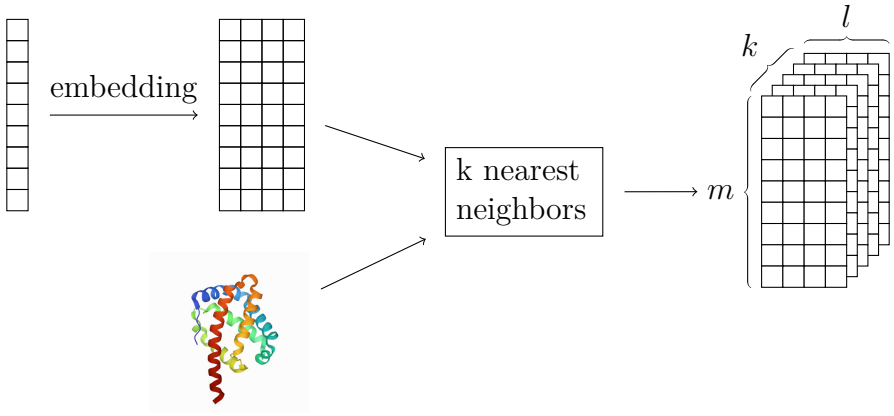


Figure 4: The kCNN feature embedding for $m = 9$, $l = 4$, and $k = 5$. In this example, the structure of the *Rhodothermus marinus* nitric oxide dioxygenase is used. The structure is taken from the PDB [58].

2. **filter application:** For each filter matrix $W_i \in \mathbb{R}^{m \times l}$ and slice F_j in the feature tensor, the element-wise product is taken. Then, the resulting values are summed per slice and filter and a bias $b_i \in \mathbb{R}$ is added, which leads to one activation vector $v_j = b_i + \sum F_j \circ W_i \in \mathbb{R}^m$ per filter (cf. Figure 5).

3. **max-pooling:** This vector is reduced via max-pooling and the ReLU [54] activation function is applied to it, calculating the filter output $o_i = \mathrm{ReLU}(\max_j(v_j))$. This yields one output vector $o \in \mathbb{R}^{\#\text{filters}}$ per input sequence.

4. **final layer:** The final output is a linear combination of the individual filter outputs, so $f_\theta(x) = W_{\text{final}} o + b_{\text{final}}$.

Therefore, $\theta = (W_{\text{final}}, b_{\text{final}}, W_1, b_1, \ldots, W_{\#\text{filters}}, b_{\#\text{filters}})$. The max-pooling over the sequence with just a single filter layer is an adaption from the work of Kim et al. for sentence classification in natural language processing [59]. Similar to the MLP, all filter applications and the final linear combination can be expressed as matrix operations. The same holds for the gradients w.r.t. the weights and biases. Accordingly, this function is trained using gradient descent as in Definition 6. To reduce overfitting, dropout is applied to both the activation vectors v_i and the filter outputs o. Here, the number of filters is a hyperparameter.

3.3 Hyperparameter optimization

All the machine learning algorithms and some of the embeddings have associated hyperparameters. These are parameters that cannot be tuned by the method itself but must be set externally. Additionally, the choice of embeddings for a method can be considered a hyperparameter itself. Hyperparameter optimization (HPO) refers to the tuning of these hyperparameters. In this work, automatic HPO will be used. For this, many different hyperparameter configurations must be evaluated. Here, a hyperparameter configuration contains one set of values for all hyperparameters. This requires two parts:

- One must be able to evaluate the quality of one hyperparameter configuration, i.e. how good it is in terms of the measures defined in Section 2.2.

- A good choice of the hyperparameter values to try is crucial for an efficient search. If the HPO is more likely to select good hyperparameter values, a good model will be found earlier.

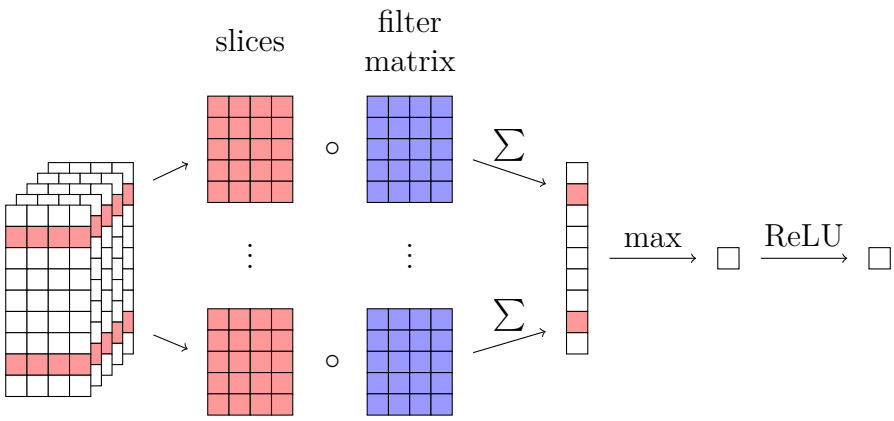


Figure 5: Application of a single kCNN filter where the filter matrix is colored blue and the two example slices are colored red.

3.3.1 Trial evaluation

A trial refers to the evaluation of one hyperparameter configuration. The user of the model is most interested in the performance of the model on new data. To simulate new data, one can split the dataset $\mathcal{D}$ into a training dataset $\mathcal{D}_{\text{train}}$ and a validation dataset $\mathcal{D}_{\text{val}}$. Then, the model can be trained on $\mathcal{D}_{\text{train}}$ with a specific hyperparameter configuration. After that, the performance can be assessed on $\mathcal{D}_{\text{val}}$.
If one chooses a hyperparameter configuration based on $\mathcal{D}_{\text{val}}$, indirect optimization, i.e. "training", on this dataset happens. As such, $\mathcal{D}_{\text{val}}$ is not anymore completely new to the model. To overcome this, another dataset split is made before, such that the dataset $\mathcal{D}$ is randomly split into $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{val}}$ and $\mathcal{D}_{\text{test}}$. Then, the model is trained on $\mathcal{D}_{\text{train}}$ and its performance is assessed on $\mathcal{D}_{\text{val}}$ during HPO. After HPO, the performance of the final model is assessed on $\mathcal{D}_{\text{test}}$. This performance can then be used as a good approximation of the performance on similar, new data. Splitting the data into three parts reduces the amount of data available to training significantly. To reduce this problem, k-fold cross-validation is employed[7]. For k-fold cross-validation, one first splits $\mathcal{D}$ into $\mathcal{D}_{\text{HPO}}$ and $\mathcal{D}_{\text{test}}$. Then $\mathcal{D}_{\text{HPO}}$ is split into k parts $\mathcal{D}_{\text{HPO},1}, \ldots, \mathcal{D}_{\text{HPO},k}$. For each $i \in \{1, \ldots, k\}$, the model is trained on $\mathcal{D}_{\text{train},i} = \bigcup_{j=1, i \neq j}^{k} \mathcal{D}_{\text{HPO},j}$ and its performance is assessed on $\mathcal{D}_{\text{val},i} = \mathcal{D}_{\text{HPO},i}$. After these k iterations, the mean of the individual scores correlates to the performance of the model on the test dataset. The mean score is called the trial value. Before testing on $\mathcal{D}_{\text{test}}$, the model is trained on $\mathcal{D}_{\text{HPO}}$. k-fold cross-validation significantly increases the computational cost, but provides better, i.e. less noisy, estimates of the hyperparameter configuration quality while maintaining a large training dataset.
For the regression problems, i.e. predicting the target value from the sequence, 10-fold cross-validation is used and the trial value is the mean of the 10 individual RMSEs. For tuning the autoencoders that are used in the evolutionary embedding, 5-fold cross-validation is used and the trial value is the mean of the 5 individual cross-entropies. The number of folds is reduced here because the datasets are significantly larger which leads to reduced noise in the value estimate. For each trial, additional data is collected which might be useful to evaluate the HPO progress. This includes various metrics of the quality of the k individual models.

3.3.2 Hyperparameter selection

For hyperparameter selection, the tree-structured Parzen estimator (TPE) [60] algorithm is used. The TPE approximates the probability distributions $p(x|y)$ and $p(y)$, where x is the

[7]This k should not be confused with the k in the kCNN.

hyperparameter configuration and y is the trial value. By Bayes-theorem, this allows for the approximation of $p(y|x)$.
During HPO, hyperparameter configurations which are expected to maximize y are sampled from these probability distributions. Then, these configurations are evaluated in a trial as described in subsection 3.3.1. After a trial completes, the probability estimates can be updated and a new hyperparameter configuration can be sampled. This allows for an iterative improvement of the probability estimates. In the end, the most promising trial is chosen. This is done by the user, who considers both the trial values and the additional data of the trials.

3.4 Implementation of the proposed methods

3.4.1 General

The proposed methods were implemented in a Python 3 [61] program. Data handling happens using NumPy [62], which adds n-dimensional and fast numerical operations to Python. To read the three-dimensional structures, the PDB module from BioPython was used [63]. For additional acceleration of performance-critical algorithms, compiled sub-routines were added. Parts of the Python code are compiled just-in-time using Numba [64]. For more complex routines, a native module written in the Rust programming language [65] was added. The Rust programming language is a low-level programming language like C/C++, but it offers a modern design and additional safety guarantees. It uses PyO_3 [66] to generate the Python bindings. Plots are generated using PlotLy [67], which enables the interactive plots available online. The DFT is computed with FFTW [68]. For confidence interval calculation via bootstrapping, arch [69] is used.
For GLMNET, the corresponding GLMNET Python package from the authors was used, which uses Fortran for efficient computations internally [51]. The SVM, as well as the used performance metrics and routines for k-fold cross-validation, are implemented using scikit-learn [70]. PyTorch [71] was used for the implementation of the neural networks, i.e. the MLP, the kCNN, and the autoencoder. PyTorch uses C or CUDA (NVIDIA GPU) kernels for efficient computations of these neural networks. To facilitate an easier and more unified training loop, PyTorch Lightning was employed [72]. The ProtTrans models [27] were accessed via the Huggingface Transformers library [73].
To ensure code quality, automatic formatting and linting were used. Additionally, unit tests were implemented for parts of the data preprocessing code to ensure the correct behavior of these routines.

3.4.2 Distributed hyperparameter optimization

The hyperparameter optimization procedure described in section 3.3 was implemented using Optuna [74], a hyperparameter optimization framework. To accelerate the HPO studies, distributed computing was employed. Optuna natively supports the storage of all study-related data to a central database. Additional data was added to every study to allow for the reconstruction of all relevant settings from the database entry. This way, the user can create a study in the central database and the workers can then collect the relevant settings from the database and start calculations. During the computation of a trial, the workers report intermediate values back to the database, allowing for live inspection of the progress. At the end of the computation, the final values are saved to the database and a new trial is started. To simplify the creation of studies, a command-line interface was added.
For long-running trials, i.e. the neural networks, a PostgreSQL database [75] was chosen due to its reliability and structured data organization. This allows for the inspection of the database state via SQL statements. Due to the central SQL database and the low amounts of data

transferred for long-running trials, the workers can be inhomogenous. To facilitate this even further, a Docker [76] image for the worker program was created, which allows for the easy addition of new nodes to the computing infrastructure.
For short running trials with many workers per node, i.e. for GLMNET and SVM, the Redis in-memory database [77] was preferred due to its low latency and high performance under concurrent accesses. After computation, the study data is copied to the PostgreSQL database for persistence.

4 Applying the methods to real data

4.1 Preparation

4.1.1 Overview

In this work, five different datasets are used. A statistical overview of them can be found in Figure 6. The sizes of the datasets and the lengths of the target sequences are listed in Table 2. The first dataset `A` is taken from Engqvist et al. [78] and contains data about the shift of the absorption maximum of the photosensitive proton-pumping *Gloeobacter violaceus* rhodopsin. Here, the aim is to produce a high shift, independent of its direction. Engqvist et al. first performed site-saturation mutagenesis on the amino acids within 5 Å of the binding pocket. Then, they tried various combinations of the individual mutations. As can be seen from Figure 6, dataset `A` contains few mutations per mutant, with most of the mutants having only one mutation. The target values follow a skewed distribution, where there are more red-shifts (larger wavelength) than blue-shifts (smaller wavelength).
The second dataset `B` comes from the work of Gumulya et al [79]. They optimized the enantioselectivity of the *Aspergillus niger* epoxide hydrolase. Two molecules are enantiomers if they have the same formula but have mirrored three-dimensional structures and are not superposable by translation and rotation. An enzyme with high enantioselectivity produces one enantiomer significantly more often than the other. The enantioselectivity is measured via the enantiomeric ratio, referred to as the E-value. This is the ratio of the one enantiomer over the other enantiomer in the product. For example, with $E = 3$, there would be three times as many molecules of the one enantiomer than of the other enantiomer in the product. To optimize the enantioselectivity, Gumulya et al. used the technique of iterative saturation mutagenesis (ISM). In this process, the considered mutations are partitioned into multiple groups, e.g. I, II, and III. The ISM follows a path, e.g. I $\rightarrow$ III $\rightarrow$ II. In the first step, saturation mutagenesis is conducted on the mutations in the first group, i.e. all combinations of the mutations in the first group are evaluated. Then, the best mutant is used as a template for a saturation mutagenesis study on the mutations in the next group. This process is repeated until all groups have been optimized. As the choice of the path is critical for the optimization outcome due to local minima, Gumulya et al. performed ISM on all possible paths with four groups of mutations to explore the effect of path choice. This led to many mutants with multiple mutations as can be seen from Figure 6, as the number of pathways grows exponentially with path length and thus mutation count. Additionally, only improved mutants were reported, and thus the target value distribution only contains increased values. This dataset is larger than dataset `A` (cf. Table 2).
The third dataset `C` is taken from Wu et al. [14]. This work used various machine learning methods (multiple linear and kernel regression methods, decision tree algorithms, and MLPs) for predicting the expected target value of a mutant. They applied these techniques for a machine learning-assisted mutagenesis study to optimize the activity of the *Rhodothermus marinus* nitric oxide dioxygenase in catalyzing the reaction of phenyldimethyl silane with ethyl 2-diazopropanoate. In contrast to previous datasets, this led to a non-linear progression in the mutation landscape resulting in many mutants with a high number of mutations (cf. Figure 6). As can be seen from the target value distribution for dataset `C`, many mutants are worse than the wild type. However, a small amount of mutants is significantly better than the wild type. This dataset is the largest of the four directed mutagenesis studies (cf. Table 2).
The fourth dataset `D` was produced by Sarkisyan et al. [80]. The mutagenesis study was applied to the *Aequorea victoria* green fluorescent protein, where the fluorescence of the protein was the target value. Most notably, this dataset is the only unbiased dataset in this thesis, as it is based on a random mutagenesis study where mutations were created by an error-prone sequence reproduction mechanism. This led to a Poisson distribution of mutations per mutant as shown

Figure 6: An overview of the five datasets before splitting. Top: Number of mutants with the specified number of mutations. Bottom: Histogram of the mutant target values, where the target value of the wild type is marked in black.

in Figure 6. However, the target value distribution contains two peaks, one around the wild type and one near zero, i.e. no fluorescence. The fitness landscape of the protein is thus very narrow and many mutations disrupt function almost completely. Few mutants are more fluorescent than the wild type. Finally, this dataset is significantly larger than the other datasets (cf. Table 2).

The fifth dataset `E` from Cadet et al. [8] is similar to dataset `B`. This dataset uses the same enzyme and experimental setup and comes from the same research group. In the work of Cadet et al., the previously mentioned technique innov'SAR was proposed. The dataset contains nine mutants with single mutations and multiple mutants with combinations of these mutations. In similarity to the work of Cadet et al. and in contrast to dataset `B`, the enantioselectivity is measured in $\Delta\Delta G^{\ddagger}$, i.e. the difference in activation energy between the enantiomers. This value is proportional to the logarithm of the E-value in dataset `B`. Here, a negative $\Delta\Delta G^{\ddagger}$ indicates that the activation energy for producing the one enantiomer is lower than the activation energy for producing the other enantiomer. Thus, the one enantiomer is produced more often in this case. This dataset is the smallest of all considered datasets (cf. Table 2).

Some of the described methods require three-dimensional structures. The structures were downloaded from the PDB, namely `6NWD` [81], `3G0I` [82], `6WK3` [58], `5N9O` [83], and `3G0I` [82] for datasets `A`, `B`, `C`, `D`, and `E`, respectively. For each structure containing multiple replicas of the protein, only chain A was used for extracting the locations of the Cα atoms. As the sequences of the structure and the dataset did not always match perfectly, a pairwise alignment was calculated. This way, there are missing coordinates for a few, non-mutated amino acids. These are ignored during training of the kCNN, as this is the only method that uses the structure.

Dataset	Sequence length	Number of samples	Number of distinct mutations
A	298	71	60
B	398	141	52
C	145	567	84
D	237	51715	1810
E	398	43	9

Table 2: Sequence length, number of samples and number of distinct mutations of the respective datasets.

dataset	MSA size	compared to dataset size	rare amino acids fraction
A	1942	27.35x	0.0054%
B	35896	254.58x	0.0024%
C	5008	8.83x	0.0072%
D	348	0.01x	0.0412%
E	23668	550.42x	0.0027%

Table 3: MSA quality for the different datasets. The MSA size is the number of distinct sequences in the MSA, the next column contains the relative size of the MSA to the dataset, and the last column contains the fraction of rare amino acids "X" in the MSA.

For most comparisons, the dataset was split randomly into $\mathcal{D}_{\text{HPO}}$ and $\mathcal{D}_{\text{test}}$ such that approximately 75% of the data is in $\mathcal{D}_{\text{HPO}}$ and 25% of the data is in $\mathcal{D}_{\text{test}}$. These will be referred to as A75, B75, C75, D75, and E75, respectively. For many comparisons, the additional dataset Dr500 will be used, which consists of a random subset of 500 samples of dataset D for training and the remaining samples for testing. This dataset has a number of training samples similar to the order of magnitude of the other datasets, allowing for a better comparison and a reduced computational cost.

4.1.2 Evolutionary features

For finding similar sequences and generating the corresponding multiple sequence alignment (MSA), HHblits [84] as implemented in the HHsuite [85] was used. For every sequence x_i in the dataset $\mathcal{D}$, a search with three iterations and a maximum Expect value of 0.001 was performed on the Uniclust30 database [47], resulting in a set of aligned sequences S_i. Here, the Expect value of a hit denotes the expected number of sequences with the same alignment that would be found in a database of similar size by chance. This ensures that the hits are significant and not random. The search parameters are copied from AlphaFold [86], a deep learning technique developed by DeepMind that uses MSAs for protein structure prediction. Finally, duplicate sequences were pruned to obtain the final set of sequences $S = \bigcup_{i=1}^{n} S_i$ for the MSAs that are used by the autoencoders. The statistics for these MSAs are shown in Table 3. Most notably, the MSAs for dataset B and E are significantly larger than the dataset, while the MSA for dataset D is very small.
Then, the autoencoders from Section 3.1.4 were trained on these MSAs. Note that the MSA contains both sequences found via the training data set and sequences found via the test data set. This is valid because the test sequences are known *a priori*, although their target value is not known. By using both parts for the MSA, additional knowledge can be learned by the autoencoder. Here, one HPO per latent size $l \in \{10, 20, 50, 100, 200, 500, 1000, 2000\}$ was conducted. The maximum latent size was chosen such that the autoencoders have to compress

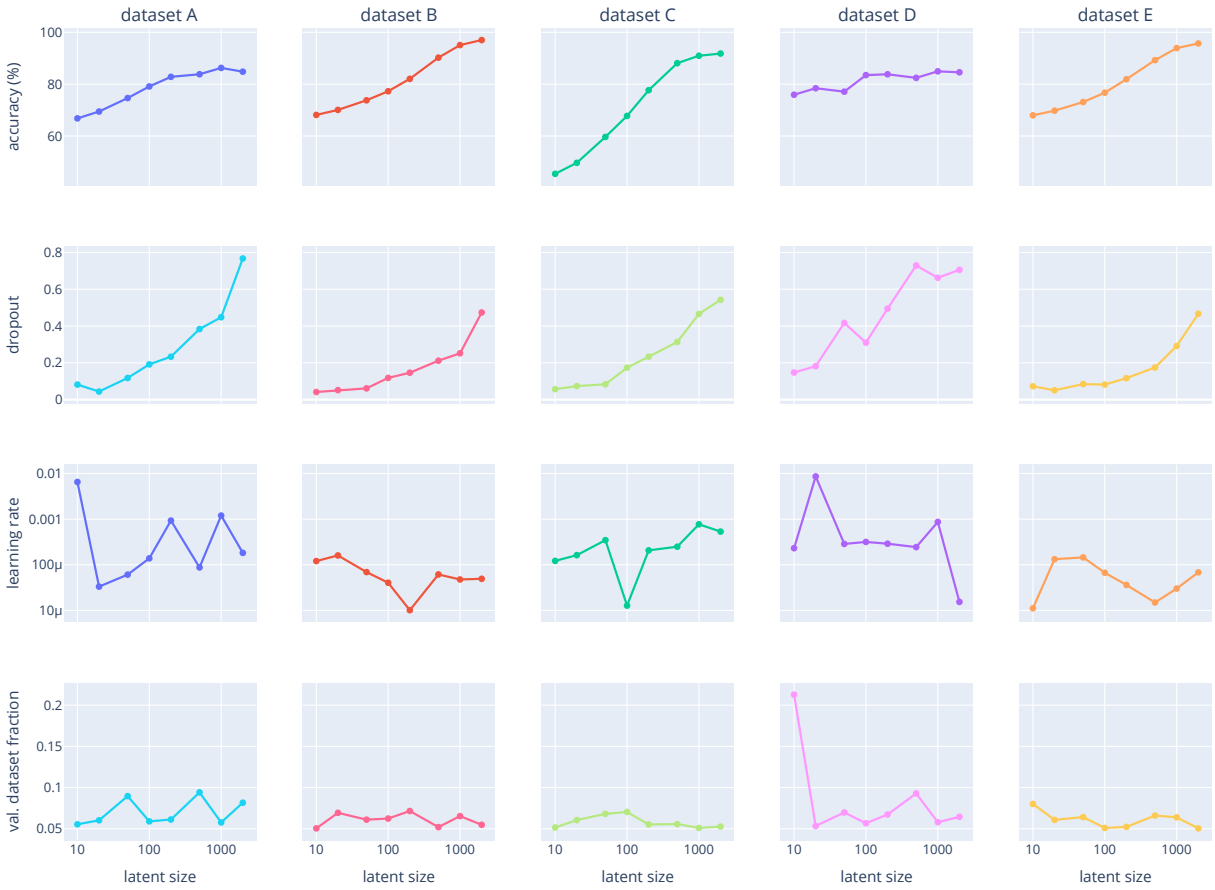


Figure 7: The best hyperparameter values for the different autoencoders according to the HPO. The y-axis of the learning rate (third row) and the x-axes are on a logarithmic scale.

the input sequence one-hot representation to less than 25% of the original size. During HPO, different hyperparameter values were selected for the various latent sizes and different accuracies were achieved. These are shown in Figure 7. While the learning rate and the validation dataset fraction do not correlate with latent size, the accuracy and dropout increase with latent size. Both are to be expected. With a higher latent size, the autoencoder has more space to store information necessary for proper reconstruction. At the same time, a higher latent size leads to earlier overfitting, which can be mitigated with a higher dropout.

4.1.3 ProtTrans embeddings

The ProtTrans models were downloaded from the Huggingface Transformers hub [73]. For T5-XXL, half-precision floating-point numbers had to be used because the model did not fit into the GPU memory with full precision. Each model was run on all sequences of each dataset. The corresponding embeddings were saved to high-performance storage to facilitate a fast lookup during training.
In a preliminary analysis, the embeddings of the individual protein models were reduced to two dimensions using t-SNE [87]. The t-SNE method is used to transform the initial representation of $m \cdot l$ dimensions, where m is the sequence length and l the embedding length per amino acid, to a two-dimensional representation. In this process, t-SNE tries to preserve distances between samples, i.e. if two samples are close in the high-dimensional space, they should be close in the two-dimensional space. Here, closeness is defined by the Euclidean distance. In Figure 8 the two-dimensional embeddings are plotted, where each sample is colored by its target value. As can be seen from the plot, the two-dimensional embeddings are mostly consistent between different protein models except for rotation. Some of the embeddings separate the data well with respect to the target value. This is the case for datasets `A`, `C`, and `E`. However, for datasets `B` and `D`, no clear separation can be observed. Interestingly, dataset `C` is clustered by all protein models.

4.2 Influence of different embeddings

4.2.1 Experimental setup

In this part, the differences between the embeddings will be investigated. To refer to combinations of embeddings quickly, a four-digit binary notation is introduced. Here, the first digit corresponds to the one-hot embedding, the second to the sScales embedding, the third to the evolutionary embedding, and the fourth to the ProtTrans embedding. The embeddings selected by a `1` will be concatenated together. For example, the notation `1001` refers to a concatenation of the one-hot and the ProtTrans embedding. All embeddings were trained with an SVM. It was preferred over the other ones because it has both a low runtime and a high accuracy. This restriction is necessary to keep the computation feasible. Each embedding can optionally be replaced with its spectral version. Whether or not the spectral version is taken was considered a hyperparameter. The test was carried out on the datasets `A75`, `B75`, `C75`, `Dr500`, and `E75`. Together, this accumulates to 5 datasets $\cdot$ (2^4 combinations -1 invalid) $= 75$ different setups. For each setup, multiple separate HPOs were run to account for variance, each for 500 trials. This way, the maximum improvement of the trial value over the last 50 trials of each HPO is 0.26% on average with a maximum of 14.85%, indicating a good convergence.

4.2.2 Direct comparison

After the HPO, the best trial of each study was selected and evaluated on the test dataset for R^2 values. The results are plotted in Figure 9. As can be seen from the plot, there is a high

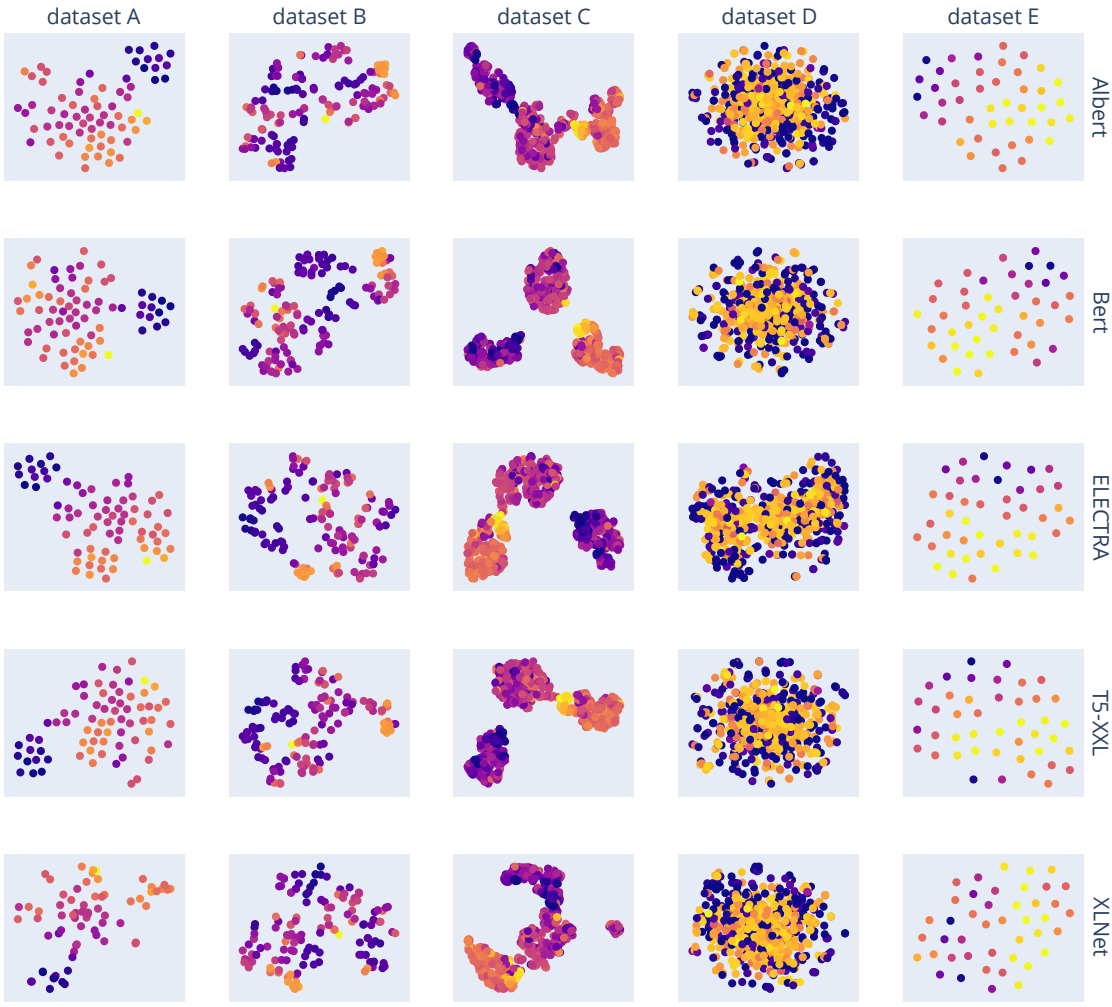


Figure 8: The embeddings of a selection of protein models projected to two dimensions via t-SNE. Each point corresponds to one mutant, where its color indicates the target value (warmer color = higher target value). For dataset D, a random sample of 500 data points was used.

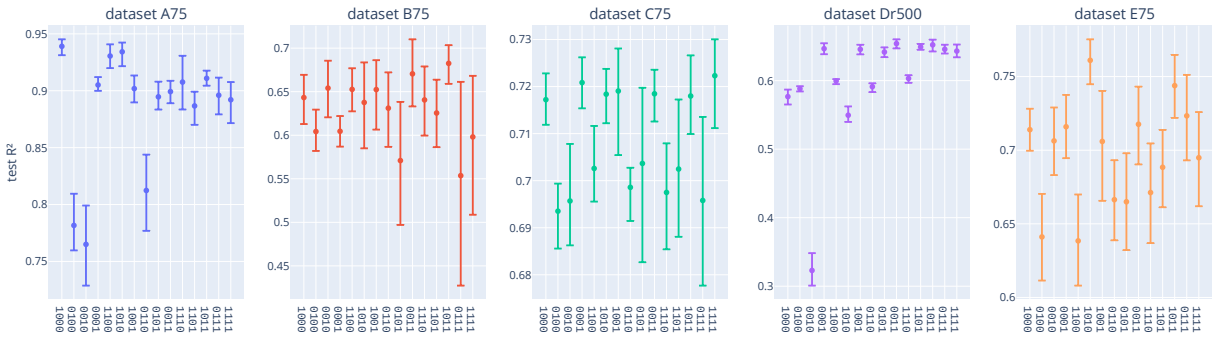


Figure 9: The mean test R^2 values and their 95% confidence intervals (via bootstrapping) of the various embedding combinations per dataset.

in %	A75	B75	C75	Dr500	E75	mean
one-hot	43.8	36.5	50.0	70.8	60.9	52.4
sScales	79.2	79.2	72.9	68.8	44.3	68.9
evolutionary	60.4	46.9	62.5	31.2	37.5	47.7
ProtTrans	42.7	40.6	39.6	41.7	49.0	42.7
mean	56.5	50.8	56.2	53.1	47.9	52.9

Table 4: Fraction of HPOs on the given dataset that used the embedding and chose to use the spectral version of the embedding in favor of the normal version.

variance in test R^2 between the different combinations of embeddings and datasets. The best scores were achieved on dataset A75, while the worst scores resulted for dataset Dr500.
On dataset A75, the one-hot embedding is the best among the tested combinations. All top three combinations use the one-hot embedding. On the other datasets, the one-hot embedding gives results that are close to or below average. The sScales embedding performs poorly across all datasets. On all datasets, the sole sScales embedding has low R^2 values, and many combinations that use the sScales embedding have low R^2 values, too.
The evolutionary embedding shows a varying performance. Although it gives low R^2 values on datasets A75, C75, and Dr500, it is among the top embedding combinations for datasets B75 and E75. On dataset B75, the top three embeddings are 1011, 0011, and 0010, and, on dataset E75, 1010 is the best embedding combination. This is consistent with the MSA sizes (Table 3), where larger MSAs correlate to better performance of the evolutionary embedding.
On dataset C75, the top embedding combinations use the ProtTrans embedding. This is in line with the findings from the t-SNE plot (Figure 8). Additionally, the average results of the ProtTrans embedding on dataset A75 and the below-average performance on dataset B75 are to be expected from this plot, too. However, on dataset Dr500, the top embedding combinations use the ProtTrans embedding although the t-SNE plot showed no separation. The t-SNE algorithm can only approximate the relationships in the data when converting from the high-dimensional space to the two-dimensional space. Thus, there might be a separation in dataset Dr500 that is not representable in two dimensions by t-SNE. Additionally, this dataset is the most diverse dataset in terms of mutations, which makes a two-dimensional representation even more challenging. Finally, the performance of the ProtTrans embedding is below expectation on dataset E75. This might be due to the high number of dimensions of the ProtTrans embedding, which leads to sparsity on a small dataset like E75. Interestingly, although both the evolutionary and the ProtTrans embeddings use unlabeled sequences produced by evolution, the use of the two embeddings seems to be independent: There are both datasets where only one embedding is beneficial as well as datasets where both embeddings are beneficial.
Concatenating all embeddings (1111) is not always a good strategy to simplify the feature selection. While it produces good results for datasets C75, and Dr500, it is below average for datasets A75, B75, and E75. Interestingly, datasets C75 and Dr500 have the most training samples. This indicates that a careful selection of the embeddings is crucial for a good performance on smaller datasets. The ideal choice of embeddings does not seem to be obvious *a priori*.

4.2.3 Spectral embeddings

To analyze the use of the spectral embedding, one has to investigate the HPO results and not the performance of the best model on the test dataset, as the selection of spectralization is part of the HPO itself. Doing the spectralization selection outside the HPO would increase the number of HPOs significantly because this needs separate HPOs for the spectral embeddings and all their combinations. Note that the trial value is measured as the root-mean-square error

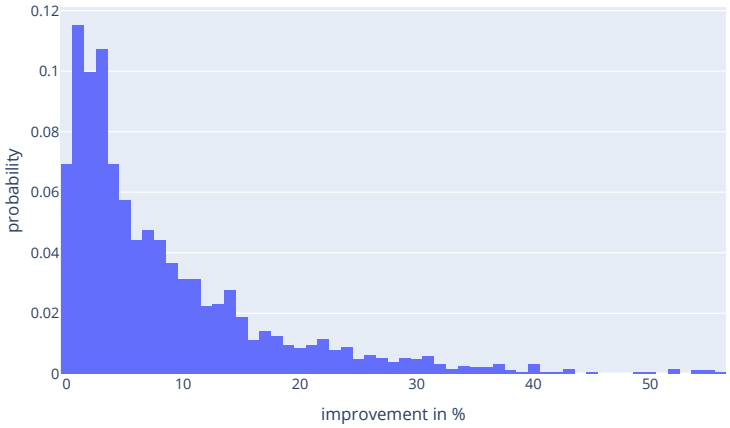


Figure 10: Distribution of spectral tuning improvements over all HPO / embedding pairs.

(RMSE), while the previously compared measure was the coefficient of determination (R^2). Here, two comparisons were carried out:
First, the value of the best trial with and without spectralization was compared for each HPO and embedding. For each such pair, the relative improvement of selecting the better one was calculated. The improvements are plotted in Figure 10. This results in a median improvement of 5.17% (95% confidence interval $[4.88\%, 5.58\%]$ via bootstrapping), i.e. by tuning the spectralization selection the HPO was able to find trials with an at least 5.17% better value for half of the studies. There are a few studies with improvements higher than 10%. This indicates a moderate improvement by tuning.
Second, the fraction of HPOs on a given dataset that used an embedding as part of their combination and that chose to use the spectral version over the normal version of this embedding was measured. These values are shown in Table 4. One can observe that the sScales embedding is preferred to be spectral significantly more often than the other three embeddings. Additionally, the evolutionary and ProtTrans embedding are spectralized slightly less often. In total, the spectral version is chosen in 52.9% of all cases.
In general, this is in line with the findings of Cadet et al. [8]. However, there is a notable difference: While Cadet et al. use linear regression, the SVM already has a non-linear transformation through the RBF kernel. Therefore, it is new that spectralization provides benefits even to non-linear models. This might be because there are repeating patterns in the sScales embedding, e.g. in α-helices, which can be represented by the magnitude of a frequency. Surprisingly, the spectralization is chosen least often on dataset `E75`, although this is the dataset Cadet et al. used in their work. The low use of spectralization for the evolutionary and ProtTrans embedding might be related to the fact that these embeddings are already contextualized in contrast to the sScales embedding.

4.2.4 Autoencoder latent size selection

To analyze the latent size selection of the evolutionary embedding, a similar approach as in the previous analysis on spectral embedding selection was used. For each dataset and latent size, the mean trial value of all trials of `0010` HPOs was calculated. Note that the trial value is the mean RMSE obtained in cross-validation. The results are plotted in Figure 11. On dataset `B75`, where the evolutionary embedding achieved high test R^2 values, there seems to be a clear minimum at a latent size of 500. On dataset `E75`, where the results were good, too, it is not clear if the global minimum is at a latent size of 100 or below 10. The other datasets were not analyzed further here, because the corresponding SVMs did not achieve a good test R^2 score.

Figure 11: The mean RMSE achieved by each HPO that uses the evolutionary embedding and over the latent size on each dataset. Note that the RMSE depends on the target value scale and is thus not comparable between datasets.

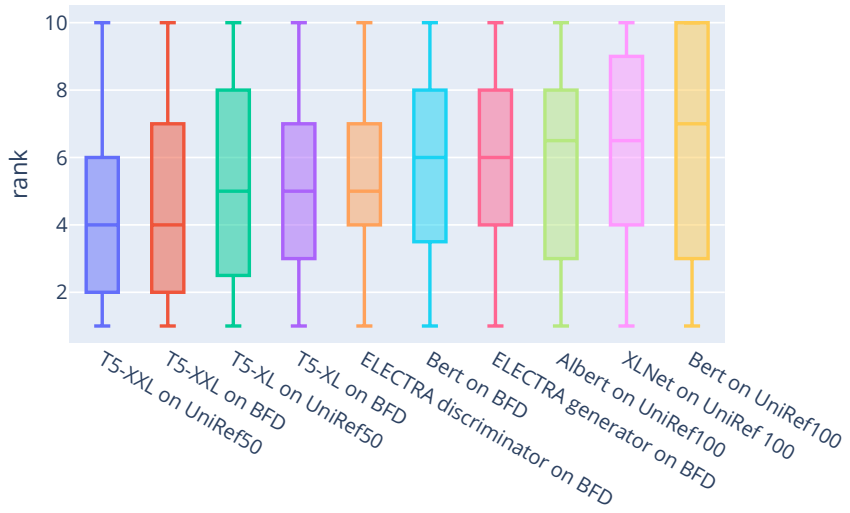


Figure 12: Boxplots of the ranks of the various protein models. The sticks of the Boxplots mark the minimum and maximum of the data, the box the interquartile range, and the line indicates the median. The Boxplots are sorted by median rank, first quartile, and third quartile in that order.

4.2.5 ProtTrans model selection

As the selection of the protein language model for embedding was part of the HPO, one has to inspect the trial values for analyzing the performance differences. For each dataset (`A75`, `B75`, `C75`, `Dr500`, and `E75`) and protein language model (T5-XL on BFD / UniRef50, T5-XXL on BFD / UniRef50, Bert on BFD / UniRef100, Albert on UniRef100, ELECTRA generator / discriminator on BFD, XLNet on UniRef100), the mean trial value, i.e. mean RMSE, of all trials in all `0001` HPOs was calculated. Here, UniRef50, UniRef100 and BFD are genome databases. The BFD database [48, 49] is the largest, while UniRef50 and UniRef100 [47] contain data of higher quality. At the time of writing, only 10 of 14 protein language models described by Elnaggar et al. [27] have been published, yet. To make the RMSE values comparable between datasets, the protein language models were ranked per dataset by their mean RMSE. These ranks are shown in Boxplots in Figure 12. As can be observed from the plot, the T5-XXL model achieved the best results on average. Interestingly, the T5-XXL model is the largest model of all considered protein models, measured by the number of trained parameters. However, the lead

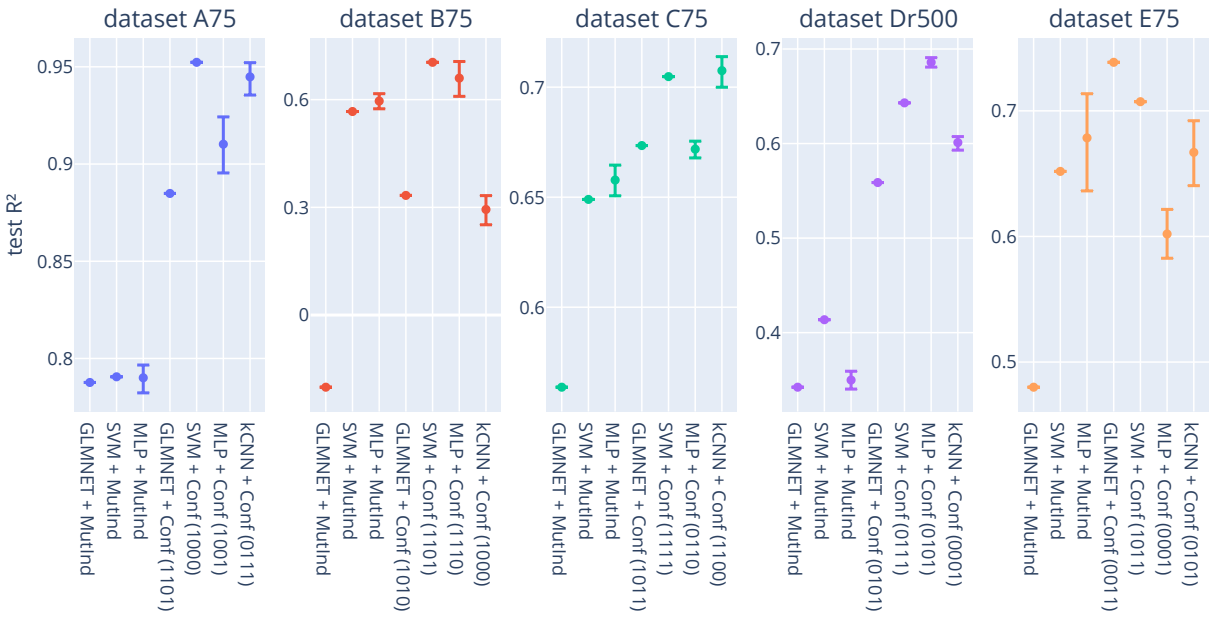


Figure 13: The mean test R^2 values for all models and their 95% confidence intervals (via bootstrapping). Note that the deterministic algorithms (GLMNET and SVM) have zero standard deviation. For the configurable embeddings, the chosen embedding is shown in parentheses behind the model name. The binary numbers have the same meaning as in Section 4.2.

of the T5-XXL model is not consistent across all datasets, i.e. there are multiple datasets where other protein language models outperform the T5-XXL model. Finally, the models trained on UniRef50 seem to have a slight benefit.

4.3 Performance of the different machine learning algorithms

In this section, the performance differences of the various machine learning algorithms will be investigated. For this, the GLMNET, SVM, MLP, and kCNN algorithms will be compared on the datasets `A75`, `B75`, `C75`, `Dr500`, and `E75`. Each model will be optimized once with the mutation indication embedding (MutInd, Definition 10) and once with a special configurable embedding (Conf). The configurable embedding is a concatenation of the one-hot, sScales, evolutionary and ProtTrans embeddings, where the inclusion of each embedding is a hyperparameter as well as its spectralization. This allows the HPO to select the embeddings and thus reduces the computational effort because unpromising embedding combinations can be avoided early in the HPO. As the kCNN cannot be used with the mutation indication embedding because it requires a sequence dimension in the input, this leads to 5 datasets $\cdot$ (2 embeddings $\cdot$ 4 algorithms $-$ 1 invalid) $=$ 35 HPOs. Each HPO was conducted for 300 trials. This way, the maximum improvement of the trial value over the last 50 trials of each HPO is 0.56% on average with a maximum of 8.96%, indicating a moderate convergence. For the non-deterministic models MLP and kCNN, the evaluation was repeated for 20 iterations to achieve a good estimate.

4.3.1 Direct comparison

The mean test R^2 values of all models are plotted in Figure 13. Note that these values are not as reliable as the embedding comparison in Section 4.2 because the HPOs were not repeated due to computational limitations. However, it is still interesting to compare the test R^2 values and the

embedding selections. Although the HPO did not select the best embedding combination in all cases, it always selected a good embedding combination. Interestingly, the selected embeddings vary between the different models, suggesting that the use of an embedding is dependent on the model and vice versa. Note that selecting the embeddings in the HPO is significantly faster than doing separate HPOs for each embedding combination.

There is a major difference between the neural network models and the other two algorithms: While training via gradient descent (Definition 6) is non-deterministic, i.e. introduces variance, the GLMNET and SVM training algorithms are deterministic. The determinism significantly reduces the evaluation overhead, as there is no variance that one has to account for.

On all datasets, the models using the mutation indication embedding performed worse than most of the other models. This effect is most severe on datasets `A75` and `Dr500`. There are multiple possible explanations for this: First, the mutation indication embedding is unable to generalize to new mutations not seen in the training data, as there are no input features for the new mutations in this embedding. Second, this embedding cannot leverage the additional information provided by the more sophisticated embeddings like the sScales, evolutionary, and ProtTrans embeddings. However, the performance of the mutation indication embedding on datasets `B75`, `C75`, and `E75` is close to the best models for SVM and MLP.

The kCNN outperforms the MLP with a configurable embedding on datasets `A75`, `C75`, and `E75`. In all these cases, the kCNN is still outperformed by the best SVM from the embedding comparison (Section 4.2). The only case, where a neural network performs best, is the MLP with a configurable embedding on dataset `Dr500`. Note that dataset `Dr500` is the largest dataset. The GLMNET models are usually outperformed by the SVM for configurable embeddings. Note that on dataset `E75`, i.e. the smallest dataset, the performance of GLMNET is close to the performance of the best SVM from the embedding comparison (Section 4.2).

Therefore, there seems to be a preference according to dataset size: On very small training datasets, linear regression (GLMNET) might be beneficial. On a medium training dataset size, support vector regression performs best. On large datasets, neural network methods are the best choice. Note that this is similar to the findings from Yang et al. [20], but the threshold for neural networks is significantly lower with only 500 instead of 10 000 data samples.

4.3.2 Runtime of the different methods

There are major differences in runtime between the different algorithms and embeddings. The mean runtime per trial (10 folds) are shown in Figure 14. Due to the heterogeneous computing devices (HPC cluster vs. personal computer), these values do not form a proper benchmark, but the differences measured here are larger than the differences caused by the computing hardware. In all cases, the GLMNET and SVM algorithms are ten times faster than the MLP and kCNN. The mutation indication embedding has a lower runtime impact than the configurable embedding. This is most likely due to the low number of input parameters from the mutation indication embedding relative to the configurable embedding. Additionally, the configurable embedding itself takes longer to compute, too. Finally, the runtime increases with large datasets across all algorithms and embeddings, reaching 10 000 seconds per trial for the kCNN on dataset `Dr500`.

4.4 Effect of dataset size

In this analysis, the effect of the dataset size was examined. To this end, dataset `D` was split in different proportions to assess the performance across dataset size. Here, `D` was chosen over the other datasets because it is the largest and the only unbiased dataset of the five considered. Similar to the embedding analysis, the SVM algorithm was used for training the models. The

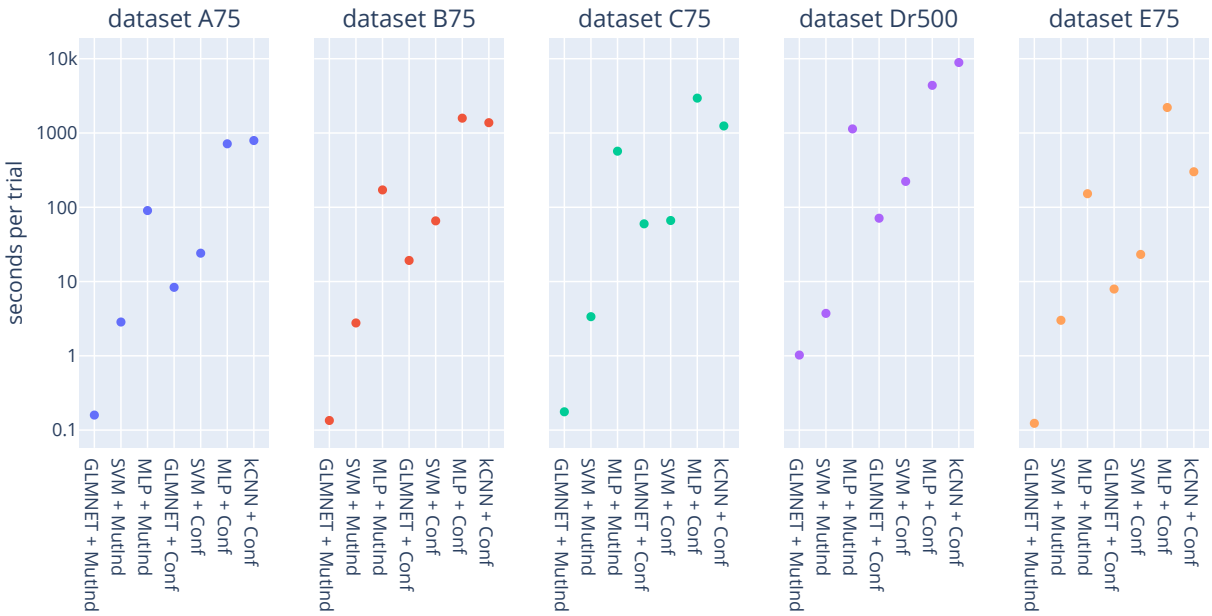


Figure 14: The mean runtime of the different models. Note that the y-axis has a logarithmic scale.

dataset `D` was shuffled randomly once, and then split into `DrX_i` with the following procedure, where `X` is the number of training samples and `i` is the index:

1. Split the dataset into strides of length `X`, where the last stride is truncated to the dataset size.
2. Take stride `i` as the training dataset and the rest as the test datasets.

This method is similar to cross-validation (Section 3.3.1). It generates training datasets of the given size that are disjoint, i.e. `Dr50_0` and `Dr50_1` share no training data samples, but have equally many.

4.4.1 Individual hyperparameter optimizations

In the first analysis, the SVM was optimized with the one-hot embedding on all training datasets described previously individually. The corresponding HPOs had an average improvement over the last 50 trials of 0.14% with a maximum of 2.94%, indicating a moderate convergence. The number of different values for `i` was at least three and incremented if the confidence intervals were too large for meaningful interpretation. At a dataset size of 1 000, the computation became infeasible with the given resources due to the non-linear complexity of the SVM algorithm. The test R^2 values are shown in subplot I of Figure 15. From the shape of the curve, one can assume a logarithmic growth of the performance with dataset size on an unbiased dataset like `D`.

4.4.2 Fixed parameters

To use the SVM for larger training datasets, one might try to use the hyperparameters obtained on a subset of the dataset for training on the complete dataset. This way, no HPO has to be conducted on the large dataset which reduces hundreds of costly training operations to a single training operation. However, this might lead to a sub-optimal choice of hyperparameters for the large dataset. To investigate the differences, the parameters found by the HPOs on datasets of size 100 were evaluated on all other datasets. The results of this comparison are shown in subplot

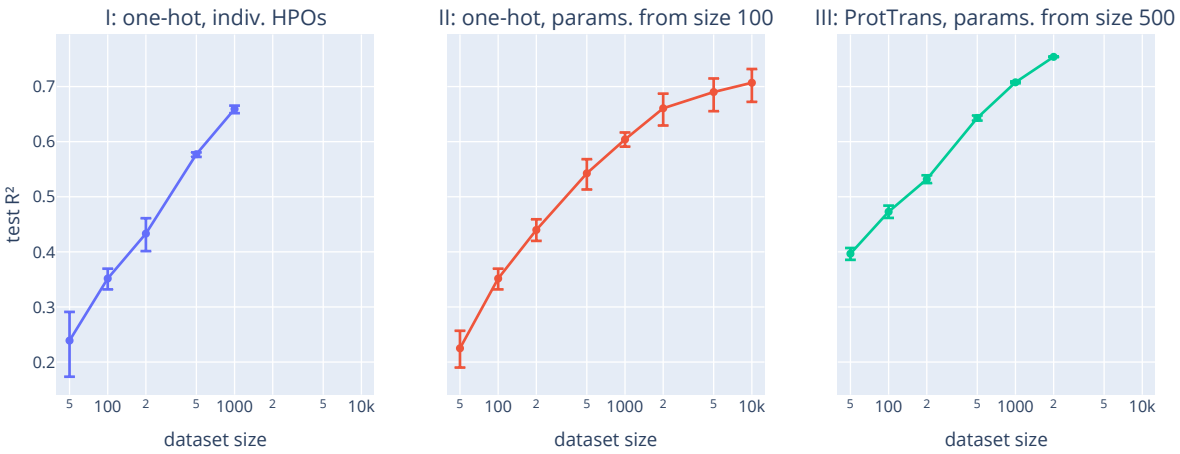


Figure 15: The mean test R^2 values of the various studies with their 95% confidence interval (via bootstrapping). The x-axis has a logarithmic scale.

II of Figure 15. In comparison to the first subplot, performance degradation is observable that grows with the difference in dataset size. However, the performance degradation is not major, e.g. for a size of 1 000, the difference in test R^2 is 0.0552 (95% confidence interval: $[0.0700, 0.0407]$). With this method, the evaluation of larger dataset sizes became possible. Here, one can observe the development of a performance plateau at around 5 000 training samples. For reference, the complete dataset contains 51 715 samples (Table 2). This result cannot be directly transferred to the other algorithms, as the hyperparameters are dependent on the machine learning algorithm in use.

4.4.3 ProtTrans embedding

To investigate whether the scaling behavior can be influenced by selecting a different embedding, the one-hot embedding was replaced with the ProtTrans embedding because it uses additional knowledge from unlabeled sequences and has shown success on dataset `D` in the embedding comparison (Section 4.2). As the hyperparameters can be transferred across dataset sizes according to the previous analysis, the best known hyperparameters for this SVM and embedding combination on dataset `D` were used: The configuration determined by the HPO for `0001` on `Dr500` in the embeddings comparison (Section 4.2). The test R^2 for the other dataset sizes are shown in subplot III of Figure 15. Most notably, the test R^2 values are higher than those for the one-hot embedding, which is in line with the findings from the embeddings comparison (Figure 9). Apart from that, the curve is approximately parallel to the curve of the one-hot embedding with fixed parameters, i.e. without individual HPOs (subplot II).

4.5 Effect of dataset split

As explained in the introduction and as described in the dataset overview (Section 4.1.1), mutagenesis studies often start with mutants containing a single mutation (single-mutant) and progress to mutants with multiple mutations (multi-mutant) later. Therefore, it would be beneficial to have an algorithm that can predict the target value for multi-mutants from single-mutants. However, generalizing from the training data to unknown test data of a different distribution might be challenging. To determine the generalizability of machine learning methods to different mutation distributions, the four datasets `A`, `B`, `C`, and `E` that represent the directed mutagenesis

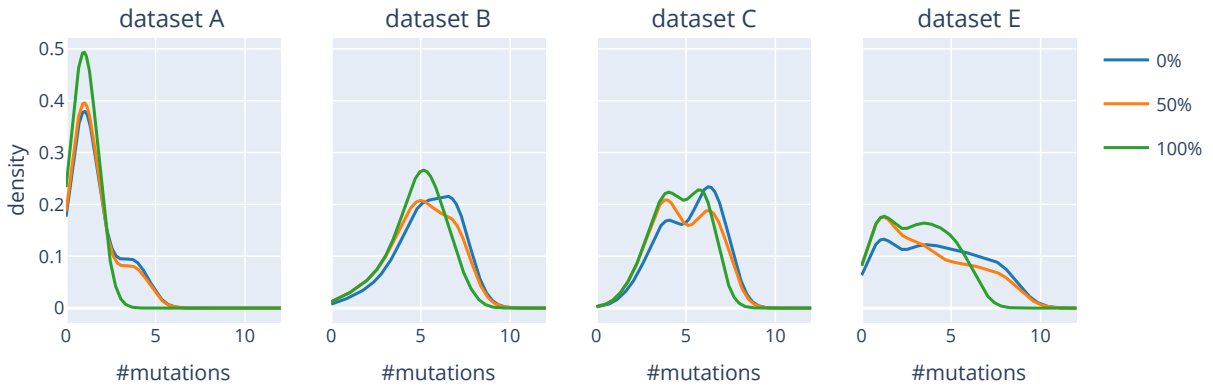


Figure 16: The density of mutations in the training datasets `Xsp_i` for `p` $\in \{0, 50, 100\}$ averaged over all `i`. Calculated using a Gaussian kernel density estimator of bandwidth 0.8.

studies in this work were split in `Xsp_i`, by the following procedure, where `X` is the dataset, `i` the random seed and `p` the percentage of non-random samples (e.g. `Cs50_3`):

1. Shuffle the n samples in dataset `X` randomly with seed `i`.
2. Order the samples by the number of mutations ascending.
3. $n_{\text{train}} = \lfloor 0.75 \cdot n \rfloor$, $n_{\text{by\_mut}} = \lfloor n_{\text{train}} \cdot \texttt{p}\% \rfloor$
4. Take the first $n_{\text{by\_mut}}$ samples from the ordered dataset and add them to the training dataset.
5. Sample $n_{\text{train}} - n_{\text{by\_mut}}$ samples from the rest uniformly and add them to the training dataset.
6. The rest is the test dataset.

This procedure leads to the distributions in Figure 16. As can be seen from the plot, the various splits change the distribution of the number of mutations to a less uniform distribution with ascending `p`. With higher `p`, the number of mutants with a large number of mutations becomes smaller, while the number of mutants with a small number of mutations becomes larger. However, the total number of training samples stays identical for all dataset splits.

4.5.1 Effect on test R^2

To assess the impact of `p` on the model performance, SVMs were optimized on the `Xsp_i` datasets individually with the best embedding combination for this from the embedding comparison conducted earlier (Section 4.2). This was repeated for 12 different values of `i`. The higher `p`, i.e. the fewer multi-mutants are present in the training data, the worse the test R^2 becomes. Surprisingly, this is not the case for dataset `A`, where the performance difference is slightly positive (0.0675 mean difference in test R^2 with 95% confidence interval $[0.0047, 0.1635]$, via bootstrapping). Note that dataset `A` contains mostly single-mutants and only few multi-mutants (Figure 6). However, on all other datasets the differences are severe: -0.4004 with interval $[-0.3124, -0.4760]$ for dataset `B`, -0.6364 with interval $[-0.5323, -0.7673]$ for dataset `C`, and -0.7285 with interval $[-0.4583, -1.1244]$ on dataset `E`. This suggests that the generalization from single- to multi-mutants proves difficult for the proposed machine learning methods. On dataset `C`, a small peak in performance can be observed before the drop.

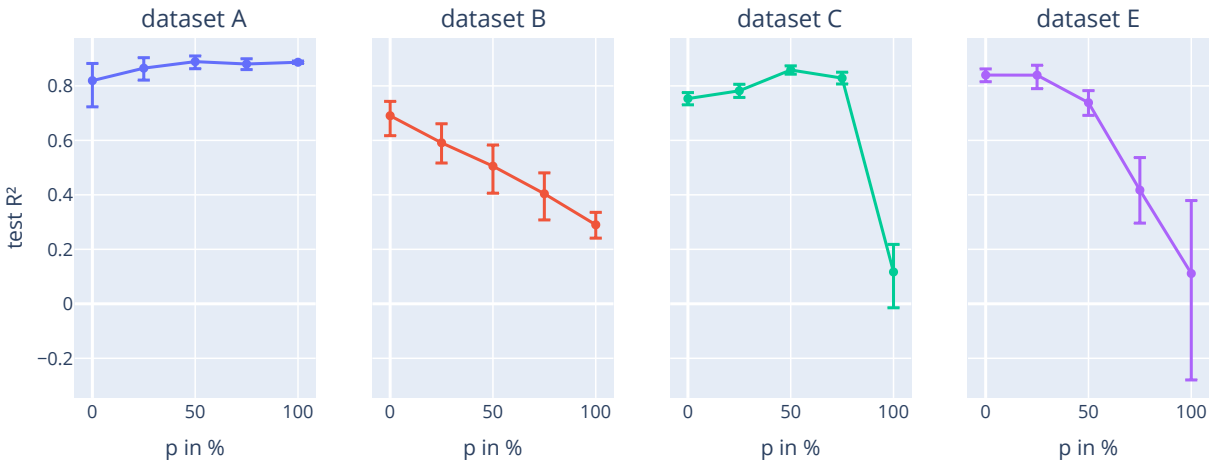


Figure 17: The mean test R^2 of the SVMs on datasets `Xsp_i` averaged over `i` with their 95% confidence interval (via bootstrapping).

dataset	model	R^2	sensitivity in %	precision in %	accuracy in %
`A75`	SVM `1000`	0.9534	90.00	90.00	88.89
`B75`	SVM `0011`	0.7967	100.00	76.19	86.11
`C75`	SVM `1111`	0.7364	76.47	89.66	84.51
`Dr500`	MLP `0101`	0.6958	76.88	87.74	83.58
`E75`	SVM `1111`	0.8323	81.82	83.33	83.33

Table 5: The R^2, sensitivity, precision, and accuracy values of the best models on the test datasets.

4.6 General prediction performance

In this chapter, the general performance of the best models found will be investigated on the five datasets `A75`, `B75`, `C75`, `Dr500`, and `E75`. For this, the following models were selected (The embedding combination is specified using a binary code as in Section 4.2):

- `A75`: best SVM with `1000` from Section 4.2
- `B75`: best SVM with `0011` from Section 4.2
- `C75`: best SVM with `1111` from Section 4.2
- `Dr500`: MLP with configurable embedding (`0101`) from Section 4.3
- `E75`: best SVM with `1010` from Section 4.2

As can be seen from this list, the long-optimized SVMs from the embedding comparison are selected for all datasets except for `Dr500` due to their high test R^2 values. For the largest dataset `Dr500`, the MLP achieved a better test R^2 value. The sensitivity, precision, and accuracy values as specified in Definition 5 were calculated on each test dataset. Here, the threshold was set to the median of the training data target values. Therefore, the classification problem is to decide, whether a new mutant is better than half of the already observed mutants or not. These scores and the test R^2 values are shown in Table 5. To further illustrate the performance, additional correlation plots are provided in Figure 18. Note that the scatter plot was replaced with a 2D histogram contour plot for dataset `Dr500` due to the high number of samples. If all samples are shown in a scatter plot, all areas with more than a certain threshold of samples look equally

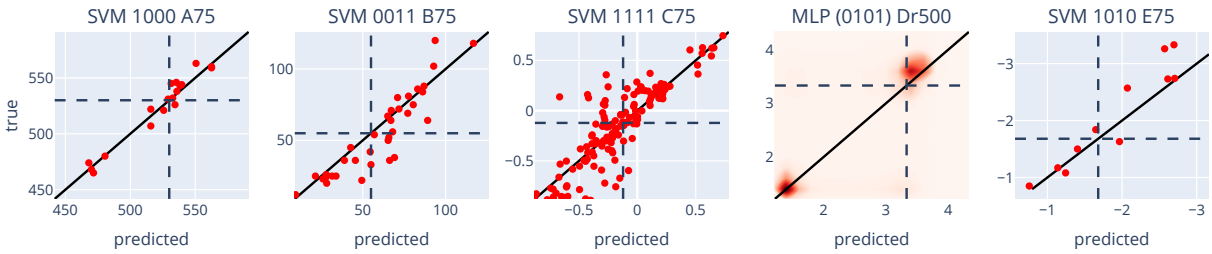


Figure 18: Correlation plots for the chosen models. For each sample, the predicted value is the x-coordinate and the true value is the y-coordinate. The continuous line represents a perfect correlation ($R^2 = 1$), while the dashed lines represent the classification threshold. Therefore, true positives are in the top right, false positives in the bottom right, false negatives in the top left, and true negatives in the bottom left quadrant. Due to a large number of samples, the scatter plot was replaced by a contour histogram for dataset `Dr500`, where darker colors correspond to more samples. For dataset `E75`, the axes are reversed because smaller values are better for this dataset.

dense. The special structure consisting of two peaks is consistent with Figure 6 (bottom). For dataset `E75`, where lower scores are better, the classes and axes were reversed.

Due to the small number of samples in the test datasets of `A75` and `E75`, their values should be handled with care. On some datasets, there are significant differences between sensitivity and precision. While precision is higher than sensitivity for datasets `C75` and `Dr500`, they are equal for datasets `A75` and `E75`, and for dataset `B75`, sensitivity is significantly higher than precision. Remember that high precision means that the model is very accurate on samples it classifies positive and that high sensitivity means that the model finds all positive samples. Therefore, the high sensitivity and low precision for dataset `B75` lead to many false positives and few false negatives. In contrast, the high precision for dataset `C75` leads to few false positives and many false negatives. The training dataset of `Dr500` was artificially limited to 500 samples. However, it seems that random mutagenesis studies as applied in dataset `D` require significantly more samples to achieve a performance comparable to the directed evolution studies underlying the four other datasets. Another possible explanation is that this might be a property of the protein used in dataset `D`, i.e. that mutations of this protein are hard to predict in general. Due to the limited number of training samples, 62.87% of the mutations do not occur in the training dataset but only in the test dataset of `Dr500`. Increasing the training dataset size improves performance (cf. Section 4.4).

4.7 Sensitivity analysis

After training a model, it is interesting to interpret the functionality of the model. This allows for insights into the reasoning the model conducts when predicting the target value of a sample. For simple models, like GLMNET, this is easy to accomplish: The coefficient for every input parameter immediately specifies its importance in prediction. For more complex models, this becomes increasingly difficult. In an extreme case, models like the MLP cannot be interpreted just from the trained weights. This makes these models a “black box”.

Despite their “black box” nature, these models are often differentiable with respect to their input, i.e. $\frac{\partial f(x)}{\partial x_i}$ can be determined numerically. This allows for estimating the effect of an input variable on the output. In this work, Integrated Gradients [88] will be used for this purpose. The following parts will be covered:

1. Introduction to Integrated Gradients
2. Proportion of the embedding parts in combinations for SVMs
3. Attribution to mutated and non-mutated positions for the SVMs on different embedding combinations
4. Three-dimensional attribution for the SVM
5. Importance of the training samples for the SVM for the one-hot and ProtTrans embeddings
6. Various properties of kCNN on dataset `A75`
7. Various properties of MLP on dataset `B75`
8. Various properties of MLP on dataset `Dr500`
9. Various properties of GLMNET on dataset `E75`

The four non-SVM models at the end are selected such that they have a good performance and represent a variety of different embedding combinations.

4.7.1 Technique

To calculate the attribution of an input feature, Integrated Gradients [88] approximates the following integral:

$$a_i = (x_i - x_i') \cdot \int_{\alpha=0}^{1} \frac{\partial f(x' + \alpha \cdot (x - x'))}{\partial x_i} \mathrm{d}\alpha$$

where x' is a baseline to which the input x is compared. Here, the wild-type was used as the baseline. Thus, a_i is the total effect of input x_i on the difference in prediction when moving along a straight path from the wild type to the mutant, where $\alpha \in [0, 1]$ models the progression on this path. Furthermore, the sum of all a_i is the total difference in prediction between the wild type and the mutant. For the following analysis, the attribution values were normalized by the total attribution:

$$\hat{a}_i = \frac{a_i}{\sum_i a_i}$$

This way, the $\hat{a}_i$ values can be compared between different samples with differing predictions. Because this technique only requires that the model function is differentiable, and all models in this work are differentiable (including GLMNET and SVM), it allows for a unified sensitivity analysis framework that can be applied to all discussed models. Additionally, the FFT, autoencoders, and ProtTrans embeddings are differentiable, too, allowing for end-to-end sensitivity analysis with just one technique. The one-hot and sScales embedding do not contain internal transformations that must be differentiated, because they are a fixed mapping from amino acids to numerical descriptors. Note that the computational complexity can become severe for large protein language models like T5-XXL.
In the following analysis, the attributions will be calculated for the test datasets. Because the test dataset of `Dr500` is very large, a special dataset split `Dr500_red` (*red*uced) was developed. This split has the same training dataset but uses a 500 samples subset of the test dataset which was stratified for the number of mutations and the target value, i.e. it was ensured that the distributions are the same between the complete test dataset and its subset.

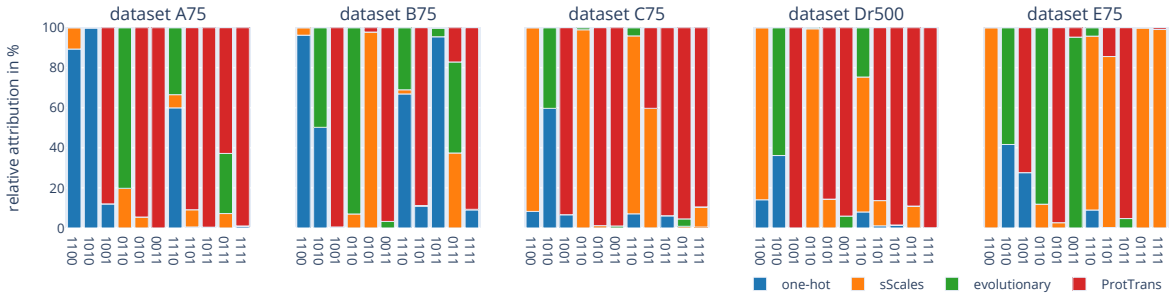


Figure 19: Relative attribution of the individual embeddings in all 11 embedding combinations one the different datasets.

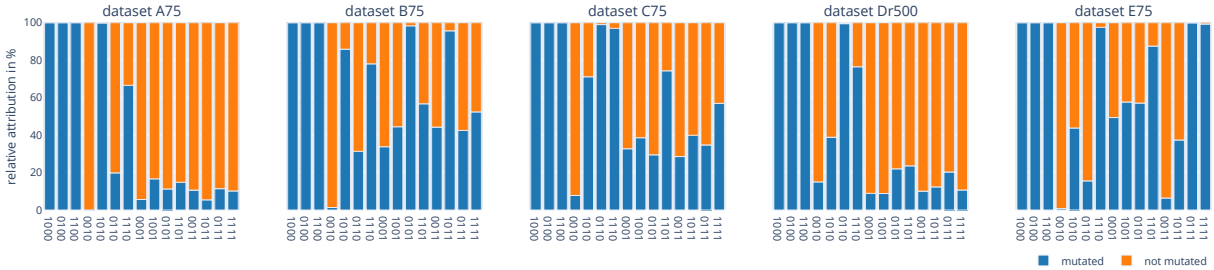


Figure 20: Relative attribution of mutated and non-mutated positions for the various embedding combinations.

4.7.2 SVM on embedding combinations

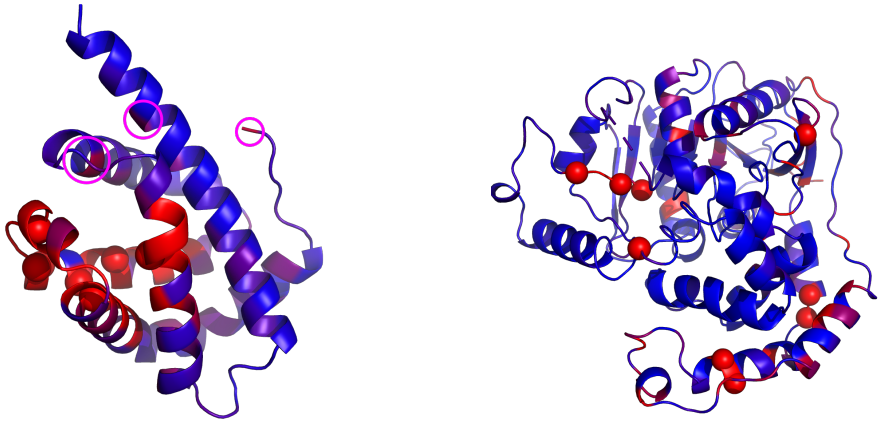
In a first analysis, the attribution of all input features on the predictions of the SVMs from the embedding comparison (Section 4.2) was computed. This was only done for one SVM per embedding combination and dataset. Then, the attribution of each embedding on the prediction was normalized by the total attribution. The results are plotted in Figure 19. One can observe that many embedding combinations are dominated by a single embedding. For example, `1111` is dominated by ProtTrans on datasets `A75`, `B75`, `C75`, and `Dr500`, and by sScales on dataset `E75`. There are a few exceptions from this rule, e.g. `0111` on dataset `B75`. Considering that the ProtTrans embedding achieves below average scores on dataset `B75` and that the sScales embedding achieves below average scores on dataset `E75` according to Section 4.2, one can conclude that the SVM has a prioritization problem: It cannot prioritize the useful embeddings over the misleading ones. This is due to the RBF kernel function $k(x, x')$:

$$k(x, x') = e^{-\gamma ||x - x'||_2^2}$$

The L2-norm used here cannot differentiate between the different dimensions of the concatenated embedding and treats them all as equally important.

4.7.3 Mutated vs. non-mutated positions for SVM

The attributions from the previous analysis can also be aggregated over the amino acid's position in the sequence instead of the embedding part. In this analysis, the attributions were summed per position and partitioned into two groups: Positions, at which the residue is mutated, and positions without mutations. The relative attributions of mutated and non-mutated positions are shown in Figure 20. For `1000` and `0100`, the SVM only considers the mutated positions. This is to be expected, as there are no changes in the embedding of the non-mutated positions. The



(a) `1111` on `C75` **(b)** `1010` on `E75`

Figure 21: The relative attribution of each amino acid to the prediction of the SVM. The values are capped at 0.5% (red). Mutated residues are shown with spheres.

`0010` and `0001` embeddings are different in this respect: Both embeddings show changes not only at the mutated positions but also at the non-mutated positions. These changes are used by the SVM for prediction (cf. `0010` and `0001`). This effect is stronger for the evolutionary embedding than for the ProtTrans embedding. For the embedding combinations, the non-mutated positions are still important for the prediction in most cases.

4.7.4 Three-dimensional activation of SVM

For two selected SVMs, `1111` on `C75` (Figure 21a) and `1010` on `E75` (Figure 21b), the attributions are shown on the structure using PyMOL [89]. These two were chosen for two reasons: They achieve the best score of all embedding combinations on their dataset and they have a high attribution to non-mutated positions. For `1111` on `C75`, which is mostly dominated by the ProtTrans embedding according to the previous analysis, one can observe a clear prioritization of the mutated positions and the neighboring amino acids in the sequence. Additionally, some of the prediction is attributed to amino acids which are not close in the sequence, but in the structure, and the N-terminus is considered, too (marked in magenta). This is different for `1010` on `E75`, which uses a combination of the one-hot and evolutionary embeddings. Although there is still a high attribution to the mutated positions, the remaining attribution is scattered across the whole structure.

4.7.5 SVM training samples importance

In support vector regression, one can infer the importance of a training sample on a prediction directly from the coefficients and kernel function values. This does not require Integrated Gradients. The model function for support vector regression is

$$f(x) = c_0 + \sum_{i=1}^{n} c_i k(x_i, x)$$

where $c_0, \ldots, c_n \in \mathbb{R}$ are the learned parameters, k is the kernel function, x_i are the training samples and x is the sample to predict. Recall that the kernel function is a measure of similarity. Therefore, one can define the influence w_i of training sample x_i on the prediction as

$$w_i = c_i k(x_i, x)$$

which can be normalized to the importance

$$\hat{w}_i = \frac{|w_i|}{\sum_{i=1}^{n} |w_i|}.$$

This analysis was conducted for SVMs on `Dr500_red`, which is the only unbiased dataset, with the one-hot and ProtTrans embedding each. The results are shown in Figures 22 and 23, respectively. First, one can observe that both embeddings prioritize some training samples more than others. For example, the first 100 training samples account for only 1% of all predictions (top plot). This suggests that many training samples were of no use for the prediction of the test dataset. Second, there is significantly more variance and structure in the importance values of the ProtTrans embedding than the one-hot embedding (bottom plot). Therefore, the ProtTrans embedding seems to be more precise in the notion of similarity, i.e. it can differentiate samples that cannot be differentiated by the one-hot embedding. This might be a reason for the higher test R^2 values observed in the embedding comparison (Section 4.2). Notably, the large importance values on the left of Figure 23 are caused by a high similarity of mutants with the E94G mutation. Why the E94G mutation produces such high importance values remains unclear.

4.7.6 kCNN on dataset A75

In this section, the kCNN trained on dataset `A75` (Section 4.3) is investigated more closely, as it represents one of the three cases where the kCNN outperforms the MLP. First, the positional attribution is shown in Figure 24. As can be seen from the rendering, the kCNN attributes the prediction to the mutated residues and their structural surroundings. This effect is even stronger than for the SVMs discussed earlier. For further analysis, six different plots were generated from the attribution values obtained via Integrated Gradients. These are shown in Figure 25 and discussed in lexicographic order: First, one can observe that the attribution is almost entirely dominated by the ProtTrans embedding, a small fraction is attributed to the sScales embedding, and the attribution to the evolutionary embedding is negligible. Second, the ProtTrans attributions follow an approximate normal distribution, i.e. there are a few dimensions in the ProtTrans embedding which have a very low or high attribution, but most attributions are between 0.1% and 0.2%.
The remaining four plots consider the kCNN filters (Section 3.2.4). In the third plot, the attribution over the k-dimension of the feature matrix is shown. One can see that the kCNN does not prioritize nearby amino acids. A histogram of the number of distinct locations in the sequence, where a filter activates, is shown in the fourth plot. Many filters activate on at most two distinct locations in the sequence. This indicates a specialization of the filters to the individual mutations, i.e. filters seem to be designed to only capture one mutation. However, the importance of the filters varies significantly as shown in the histogram in the fifth plot. Most of the filters have a very low attribution, i.e. are unused when predicting the test dataset. To answer the question, whether the important filters activate at multiple positions, the sixth plot, a correlation plot of the previous two values, was generated. One can observe that the filters that activate at multiple distinct locations have a very low attribution. This indicates that the kCNN does not learn general features that can be applied to multiple parts of the sequence, as the author of this thesis expected, but instead degenerates to using dedicated filters per mutation.

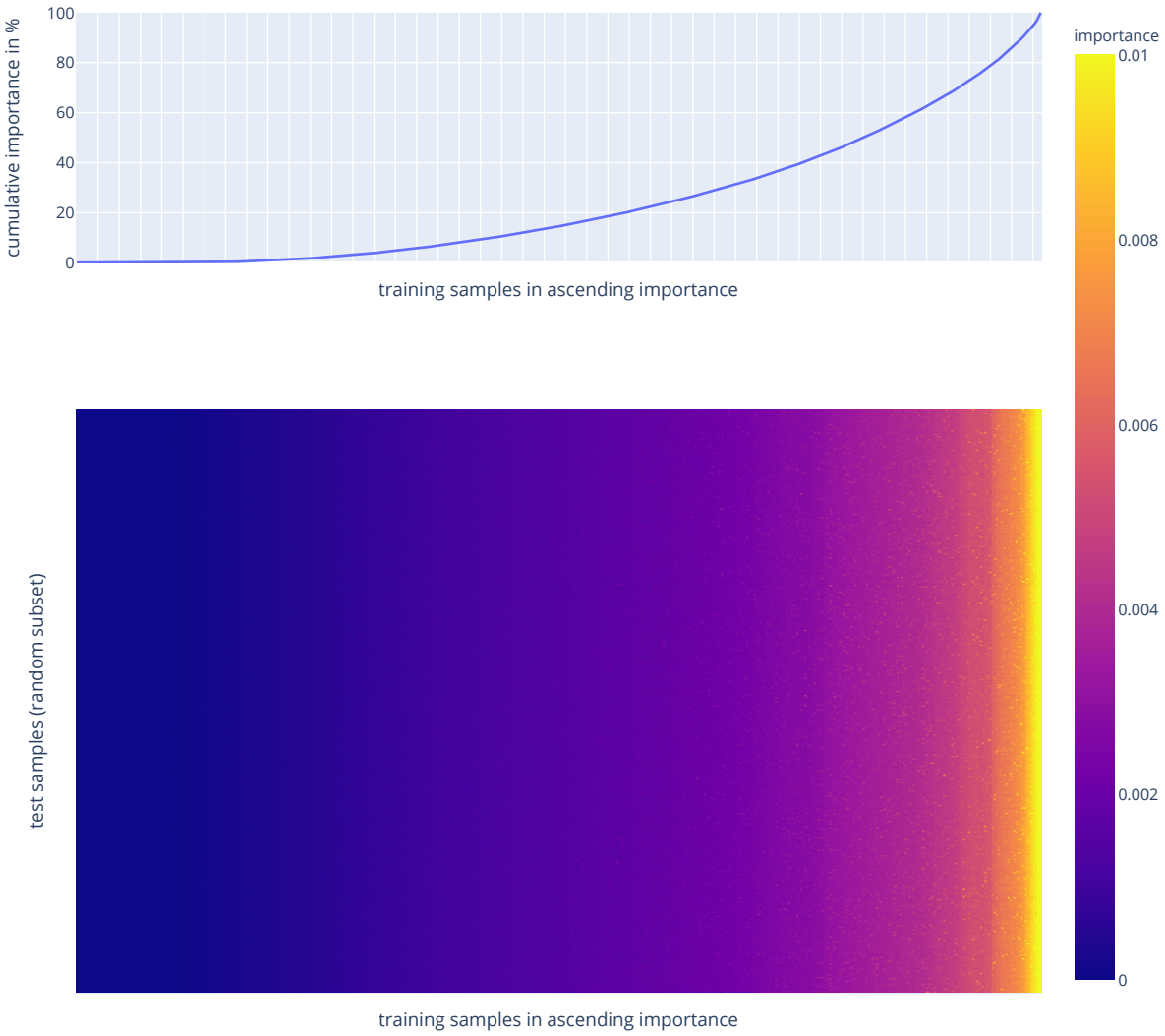


Figure 22: Bottom: The importance values $\hat{w}_i$ of the training samples on the test samples prediction with the one-hot embedding. On the y-axis, each row corresponds to a sample in the test dataset (e.g. E16G+T96A+D179G). On the x-axis, each column corresponds to a sample in the training dataset (e.g. E16G+T96A+V175A). The value of every cell shows the importance $\hat{w}_i$ of the corresponding training sample for the prediction of the test sample. The training samples are sorted in ascending total importance and the heatmap colors are capped to 0.01. The x-axis and y-axis labels are omitted due to the high number of samples in the dataset, but they are available in the online version of this plot. For example, at the intersection of row E16G+T96A+D179G and column E16G+T96A+V175A, one can find the importance of this training sample (column) for the prediction of the test sample (row). Top: The cumulative total importance of the training samples, i.e. the cumulative column sum.

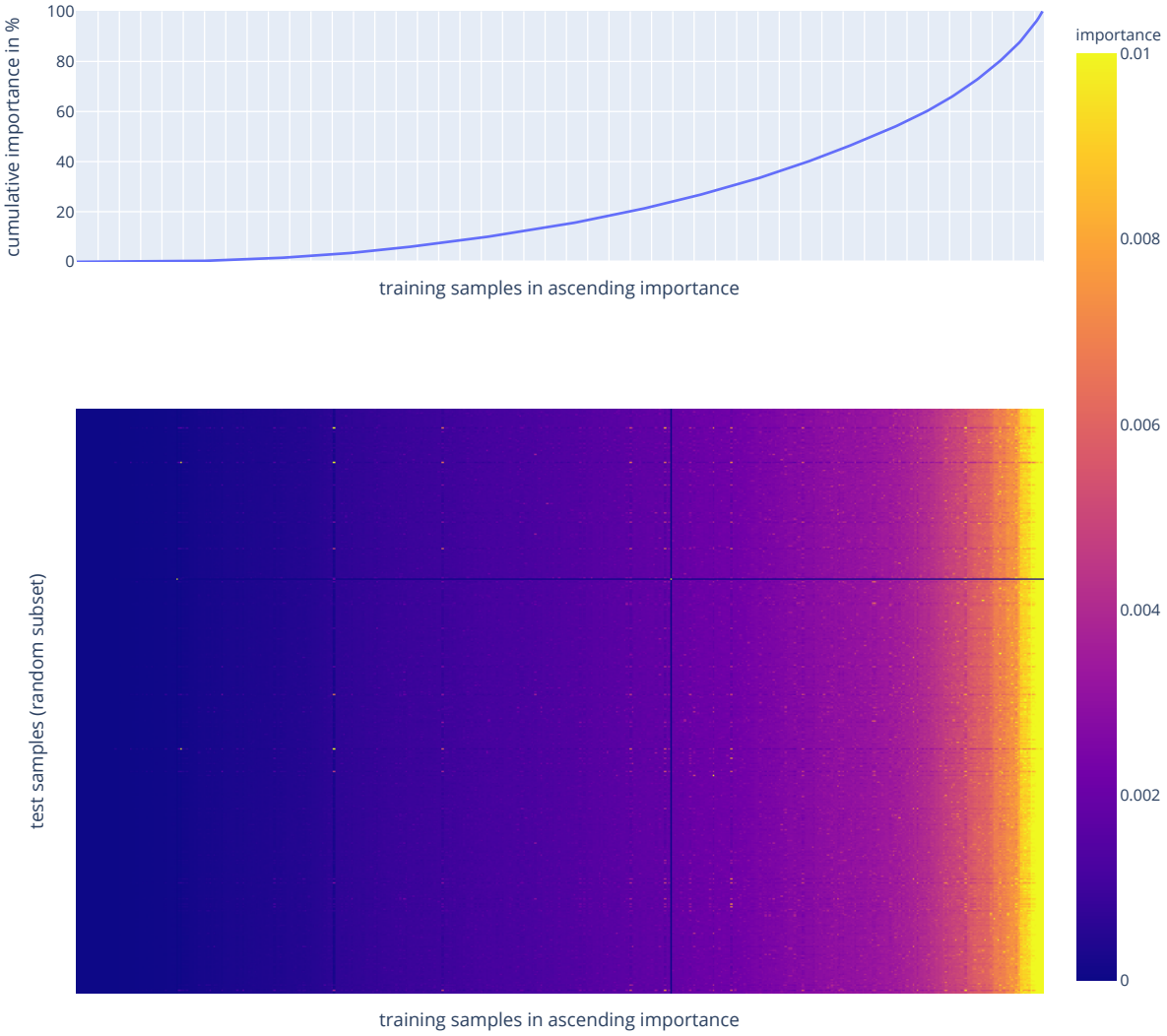


Figure 23: Same as Figure 22 but for the ProtTrans embedding.

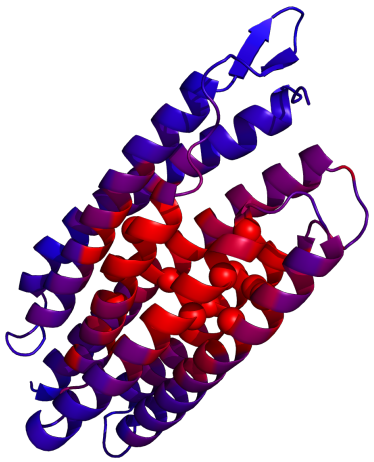


Figure 24: The relative attribution of each amino acid to the prediction of the kCNN on the test dataset of `A75`. The values are capped at 0.5% (red). Mutated residues are shown with spheres.

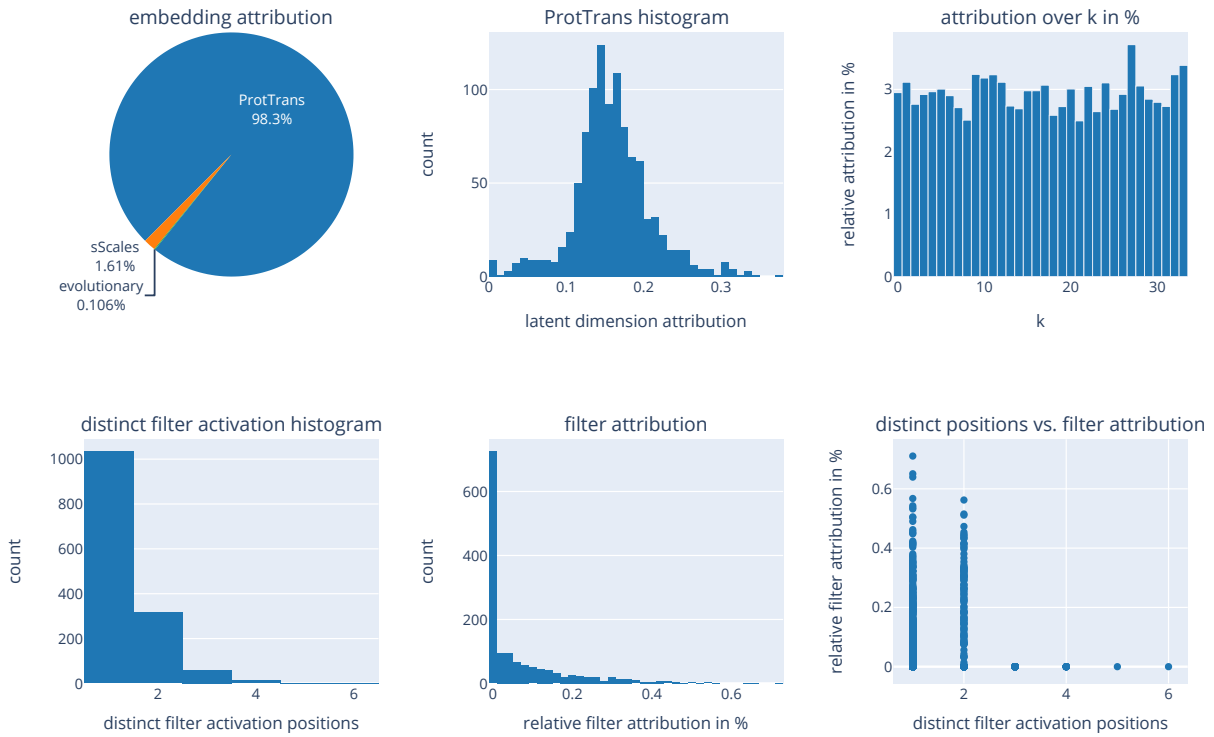


Figure 25: Various attributions for the kCNN on dataset `A75`. In lexicographic order: 1. Relative attribution of the individual embedding parts. 2. Histogram of the relative attributions to the ProtTrans embedding. 3. The relative attribution over the k-dimension of the kCNN filters. 4. Histogram over the number of distinct positions at which a filter has activated. 5. Histogram of relative filter importances. 6. Correlation plot between distinct filter activation positions (4.) and relative filter importance (5.).

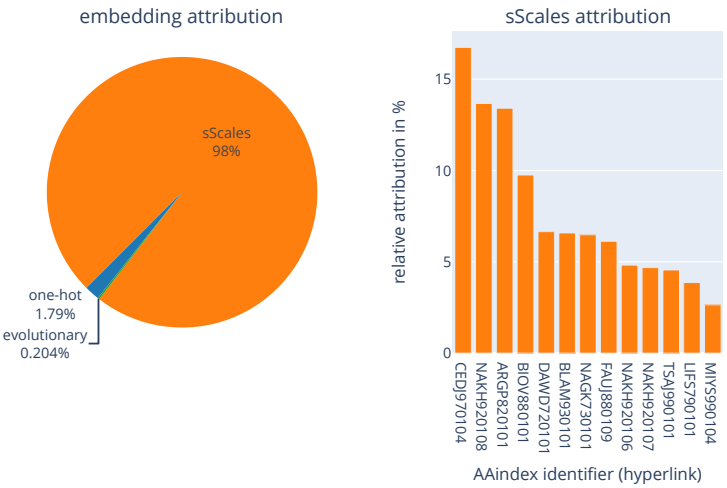


Figure 26: Various attributions for the MLP on dataset `B75`. In lexicographic order: 1. Relative attribution of the individual embedding parts. 2. Relative attribution of the 13 AAindex descriptors in sScales.

4.7.7 MLP on dataset B75

Similar to the previous analysis, the MLP trained on dataset `B75` (Section 4.3) was inspected, too. The two corresponding plots are shown in Figure 26. For this model, the prediction is dominated by the sScales embedding, as can be seen in the first plot. Thus, the attribution of the 13 AAindex descriptors in sScales was calculated. The results are shown in the second plot. There is a clear prioritization of the three descriptors `CEDJ970104`, `NAKH920108`, and `ARGP820101`, which account for 44% of the total prediction. These descriptors are described as "Composition of amino acids in intracellular proteins (percent)" [41], "AA composition of MEM of multi-spanning proteins" [40], and "Hydrophobicity index" [44] in the AAindex database. Therefore, the two most important descriptors are non-physical but describe general frequencies of amino acids in proteins. Interestingly, the frequency values come from both membrane-spanning and intracellular proteins, which agrees to the suggestion by Zou et al. [90] that the protein in question is intracellular with a membrane-anchor. The third most important descriptor for this model is related to the physical properties of the amino acid.

4.7.8 MLP on dataset Dr500

In a third analysis, the MLP trained on dataset `Dr500` (Section 4.3) was considered. The four plots are shown in Figure 27. As can be seen from the first plot, the prediction of the MLP is dominated by the ProtTrans embedding, which is spectralized. The relative attributions of the ProtTrans dimensions (second plot) show a similar distribution as in the previous analysis of the kCNN on dataset `A75`. From the third plot, one can infer that most of the prediction is based on the positions which were not mutated in the test dataset. In the fourth plot, one can see the relative attribution of the individual frequencies from the spectralized ProtTrans embedding. Besides the peak at frequency 0 Hz, which corresponds to the mean of the data, no clear prioritization can be observed. It is unclear why there is no prioritization in this data.

4.7.9 GLMNET on dataset E75

Finally, the GLMNET model trained on dataset `E75` (Section 4.3) was analyzed. The four plots are shown in Figure 28. From the first plot, one can deduce that both embeddings, evolutionary and ProtTrans, are equally important for the prediction. Note that the evolutionary embedding

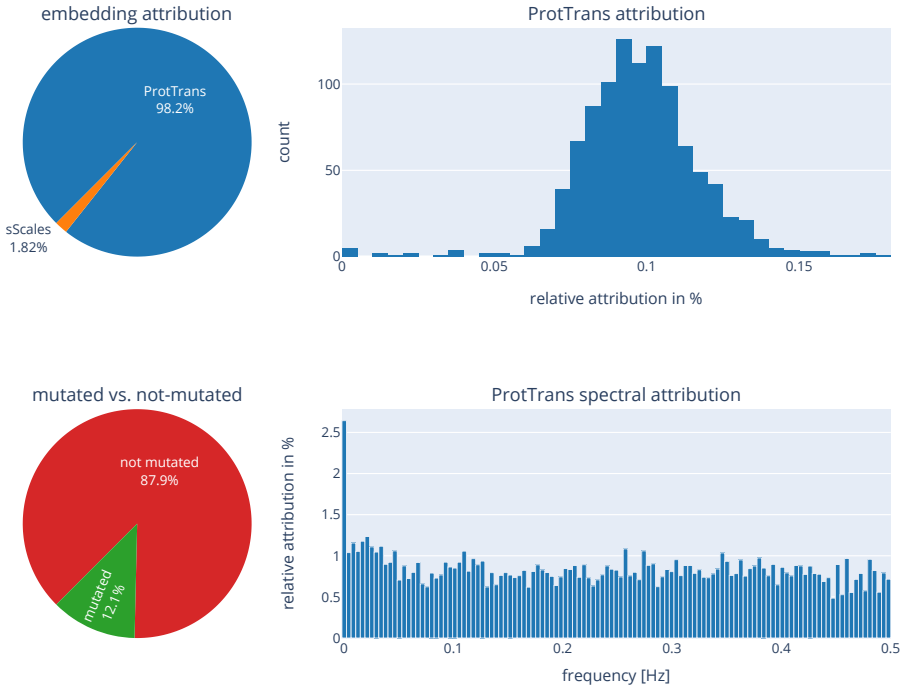


Figure 27: Various attributions for the MLP on dataset `Dr500_red`. In lexicographic order: 1. Relative attribution of the individual embedding parts. 2. Histogram of the relative attribution of the ProtTrans dimensions. 3. Relative attribution of mutated and not mutated positions. 4. Relative attribution of the frequencies in the spectral version of the evolutionary embedding.

was spectralized. In the third plot, one can see a clear dominance of the not mutated positions for the final prediction, as it is common for models that use the evolutionary and ProtTrans embeddings (cf. Figure 20). In the second plot, a histogram of the relative attributions of the ProtTrans dimensions is shown. One can see that there is significantly more sparsity here than in other models: Most of the dimensions are unimportant for the prediction. This effect continues in the fourth plot, where the attribution of the different frequencies from the spectral version of the evolutionary embedding is shown. One can observe that most frequencies have an attribution value of 0 and that lower frequencies account for the majority of the prediction. This sparsity is most likely due to the use of the L1-norm in GLMNET, which encourages sparsity.

To make the final statement more plausible, consider the following L1 and L2 norms for small $\epsilon, \delta > 0$ and $x = (1, \epsilon)$ [91]:

$$
\begin{aligned}
||x||_1 &= 1 + \epsilon \\
||x - (\delta, 0)||_1 &= 1 - \delta + \epsilon = ||x||_1 - \delta \\
||x - (0, \delta)||_1 &= 1 - \delta + \epsilon = ||x||_1 - \delta \\
||x||_2^2 &= 1 + \epsilon^2 \\
||x - (\delta, 0)||_2^2 &= 1 - 2\delta + \delta^2 + \epsilon^2 = ||x||_2^2 - 2\delta + \delta^2 \\
||x - (0, \delta)||_2^2 &= 1 - 2\epsilon\delta + \delta^2 + \epsilon^2 = ||x||_2^2 - 2\epsilon\delta + \delta^2
\end{aligned}
$$

Note that the decrease in the norm by subtracting δ from the first, large component or the second, small component is equal for the L1-norm, but varies for the L2-norm. Therefore, the

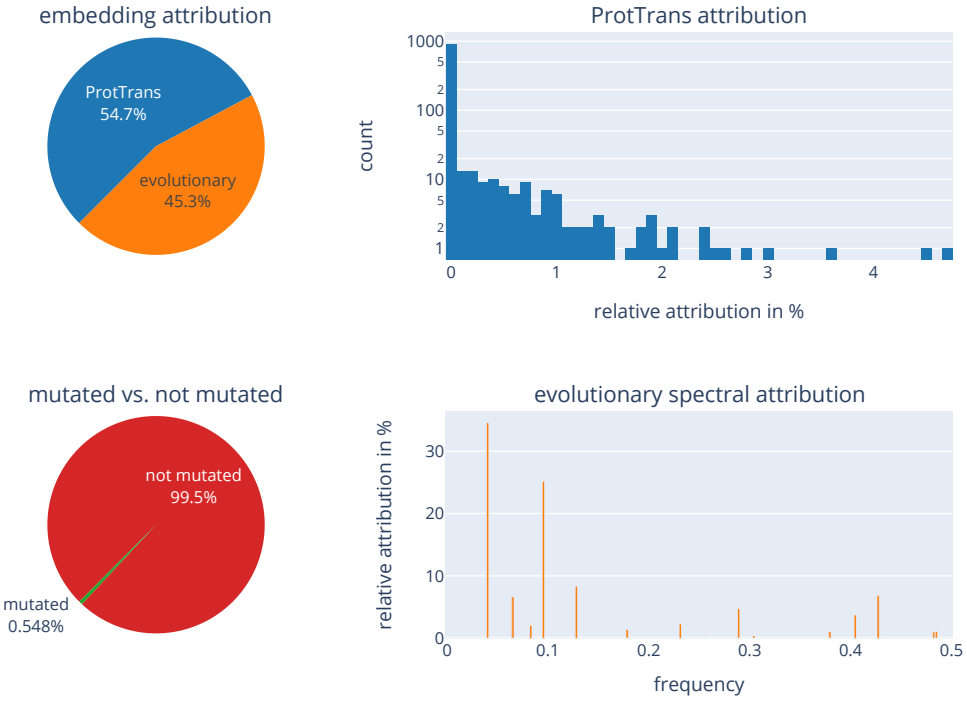


Figure 28: Various attributions for GLMNET on dataset E75. In lexicographic order: 1. Relative attribution of the individual embedding parts. 2. Histogram of the relative attribution of the ProtTrans dimensions. 3. Relative attribution of mutated and not mutated positions. 4. Relative attribution of the frequencies in the spectral version of the evolutionary embedding.

L2-norm does not encourage the reduction of small values, which harms sparsity in regularization because values of 0 are rarely achieved.

4.8 Mutation effect prediction

4.8.1 Measuring epistasis

If there is no epistasis in the dataset, the effect of individual mutations is independent, and thus constant. This exactly corresponds to the assumptions of linear regression with a mutation indication embedding. To proof this, consider the model function for linear regression:

$$f(x) = w_0 + \sum_{i=1}^{n} w_i x_i$$

As the $x_1, \ldots, x_n \in \{0, 1\}$ indicate the presence or absence of a mutation in the mutant, and each mutation is weighted by a constant factor $w_i \in \mathbb{R}$, each mutation has the same effect in every mutant. Therefore, this model assumes no epistasis.
To use this model as a measure, the linear regression problem is solved for a given dataset without regularization. The removal of regularization makes this model different from the GLMNET technique and is necessary to avoid the disturbances of this process. This way, the linear regression model learns the optimal solution for the training dataset and not for the test dataset, as GLMNET tries to achieve it. Linear regression without regularization is also known as "ordinary least squares" regression. Due to this, the R^2 score on the training dataset serves as

dataset	$h(\mathcal{D}_{\text{train}})$	$h(\mathcal{D}_{\text{test}})$	$h(\mathcal{D})$
A75	0.0212	0.0000	0.0252
B75	0.1124	0.1145	0.1977
C75	0.2607	0.1265	0.2834
Dr500	0.0000	0.2814	0.2899
E75	0.0565	0.0012	0.0907

Table 6: Epistasis value h for training, test, and complete dataset of the splits A75, B75, C75, Dr500, and E75.

an indicator for the epistasis: If $R^2 = 1$, the linear regression can find a perfect fit and therefore there cannot be any epistasis present in the dataset. Therefore, a measure of epistasis h is proposed:

Definition 17 *For a linear regression model f trained on the mutation indication embedding (Definition 10) of dataset $\mathcal{D}$, the* epistasis $h(\mathcal{D})$ *is defined by the following term:*

$$h(\mathcal{D}) = 1 - R^2(f, \mathcal{D})$$

The values of h were calculated for the training, test, and complete dataset of the splits A75, B75, C75, Dr500, and E75 (Table 6).
For interpretation, one can compare these values to the R^2 values of the best models as shown in Table 5. One can see that there is a perfect negative rank correlation: The i-th best score is achieved on the i-th lowest epistasis value ($i \in \{1, 2, 3, 4, 5\}$). In other words, high epistasis as measured by h degrades the performance of the tested models. Additionally, there are differences between the epistasis measured in the training and test datasets. The low epistasis in the training dataset of Dr500, but the high epistasis in the test dataset of Dr500, might be the reason for the poor performance of the models on this dataset.

4.8.2 Detection of epistasis by the models

As described in Section 4.7.1, the Integrated Gradients technique can be used for end-to-end analysis of the attribution of each mutation on the prediction of a mutant. To achieve the best possible estimates for these values, this analysis was conducted on the best models from Section 4.3 on each dataset.
The attribution of each mutation on each mutant is shown in Figure 29. Dataset Dr500 is omitted from this plot because its attribution matrix is very large and sparse, making visual interpretation difficult. Note that the attributions explain the difference between the wild type and the mutant. As the mutant is not always at the center of the target value range (cf. Figure 6), the attributions are sometimes overwhelmingly positive or negative. These heatmaps explain the predictions of the model: If users are interested in what effect each mutation has on the predicted target value, they can look into the corresponding row of the heatmap. For example, the mutant Y32G+F46R+L56D+V97G in the test dataset C75 has a total attribution of 0.037, i.e. slightly positive, but the individual mutations have both positive and negative attributions: Y32G 0.100, F46R 0.025, L56D −0.077, and V97G −0.011.
For a more aggregated view, histograms of the standard deviations of all attributions to one mutation, i.e. the columns of Figure 29, are shown in Figure 30. The attribution values were normalized to a scale of −1 to 1 of the target value for comparability. In comparison to the measured test dataset epistasis $h(\mathcal{D}_{\text{test}})$ shown in Table 6, similarities and dissimilarities can be observed. First, the standard deviations are the lowest for dataset A75, which is in line with the

Figure 29: Mutation attribution of the best model on the test datasets of `A75`, `B75`, `C75`, and `E75`. On the x-axis, each column corresponds to one mutation, e.g. G162L. On the y-axis, each row corresponds to one mutant, e.g. G162L+E166W+A256C. Each cell contains the attribution of the mutation (x-axis) to the predicted mutant (y-axis) target value, on a normalized scale. Red cells have a value of 0, blue cells are negative, and yellow cells are positive. For example, at the intersection of column G162L and row G162L+E166W+A256C, one can find the attribution of G162L to the prediction for G162L+E166W+A256C. The x-axis and y-axis labels are omitted due to the high number of samples in the datasets, but they are available in the online version of this plot.

h-value. Both the h-value and the standard deviations grow for datasets `B75` and `C75`, and peak at dataset `Dr500`. This suggests that these models can partially model the different amounts of epistasis in these datasets. However, the standard deviations are high for dataset `E75`, although the h-value is very low. This indicates that the model overestimates the epistasis on dataset `E75`.

4.9 Negative Results

During the course of this work, multiple preliminary studies were conducted. These were used to explore the problem and as initial guidance for the design of the comparisons presented in the previous chapters. Note that for most preliminary studies the computational effort was reduced. For this, the studies were often limited to dataset `C` exclusively. Additionally, many comparisons (e.g. number of hidden layers) were not done using separate HPOs for each value (e.g. one HPO per number of hidden layers), but as another parameter of a single HPO. In the following, the not or only marginally successful approaches are summarized.

MSA search On a first try, the multiple sequence alignments (MSAs) were searched using BLAST [92] instead of HHblits [84]. Additionally, only the wild-type was used for searching for similar sequences. This delivered significantly smaller MSAs that did not contain the mutations of interest that are present in the dataset. With moving from the search for a single sequence to all sequences, a new algorithm was required due to the high runtime of BLAST. This led to HHblits which reports being both faster and more sensitive [84].

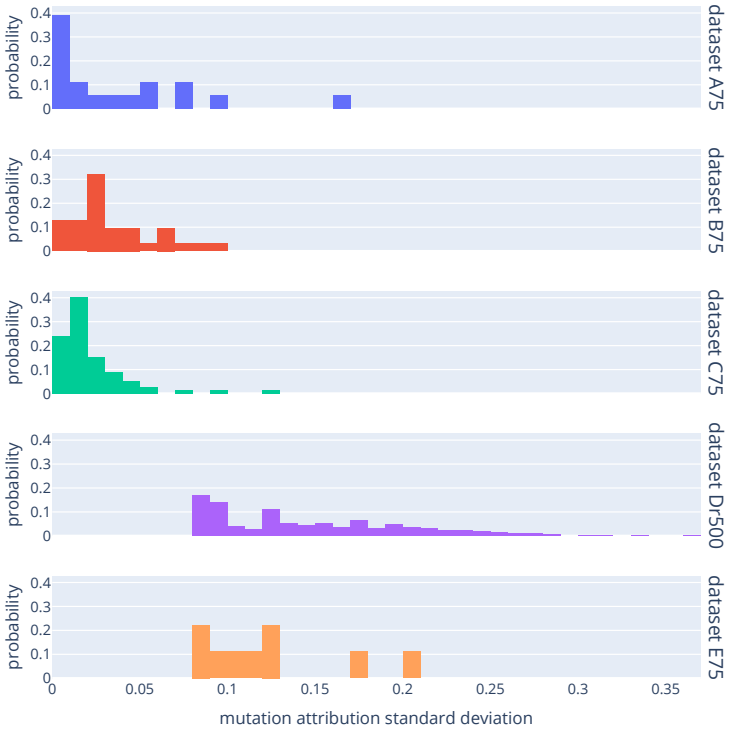


Figure 30: Histograms of the mutation attribution standard deviations from the best model on each dataset.

MSA variational autoencoding Initially, variational autoencoders (VAEs) were tried for encoding the MSA information. This was inspired by the work of Sinai et al. [21]. VAEs represent the latent space not as a fixed vector, but as a set of independent Gaussian random variables with a mean and a standard deviation that can be chosen by the encoder. To keep the intermediate values constrained to a normal Gaussian distribution, a second loss term is added that encourages means close to zero and standard deviations close to one. Otherwise, the encoder could just set the standard deviations to zero and use the means as a fixed vector. The author hypothesized that this variational approach better suits the idea of using the MSA to model the latent space in a continuous way that is also applicable to the new sequences from the dataset that are not present in the MSA and of different origin. However, the VAEs only adhered to the second loss term and provided poor reconstruction accuracies of the mutated residues. This still allowed for a low loss value, because many residues of the MSA are mostly constant and can be guessed with high accuracy without additional information.

MSA representation In the beginning, the downstream models (e.g. SVM) were trained on the latent representation of the autoencoder because the author expected this to be a condensed representation of the evolutionary landscape. With the usage of the latent space, additional down-projection methods other than autoencoders became possible. The author also tried to use a PCA for the projection, but this provided bad results. Switching to the probability estimation for the sequence as described in Definition 14 provided significantly better results.

MSA encoder/decoder architecture For both the VAE and the normal autoencoder, various MLP architectures were explored. While the VAE preferred MLPs with multiple hidden layers, the autoencoder was reduced to the linear autoencoder presented in Definition 14 during HPO. Therefore, the simplified version of the linear autoencoder was used for further studies due to its significantly better runtime and lower complexity.

Autoencoder optimization Initially, the autoencoders were optimized jointly with the downstream models. To accelerate this process, a sub-HPO was performed for every trial in the main HPO, where the main HPO controls the autoencoder parameters and the sub-HPO controls the SVM parameters (This was only tested with the SVM algorithm.) because the SVM optimization is significantly faster than the Autoencoder (especially VAE) training. However, this was still computationally expensive. With the switch to the linear autoencoders, the only hyperparameter that was of interest to the downstream models was the latent size, and as such, precomputation became possible, eliminating the need for the joint HPO.

AAindex selection The sScales embedding is a selection of indices from the AAindex database [16–18] adapted from the work of Xu et al. [15]. However, this selection is not guaranteed to be optimal. To overcome this limitation, multiple strategies were investigated which all shown less effective:

- Doing no selection, i.e. taking all parameters.
- Training a linear projection of the AAindex parameters to a smaller set of values jointly with the neural network (not applicable to GLMNET and SVM).
- Using one of the other AAindex descriptors from Xu et al. [15], including a PCA of these values.

3D CNN To utilize the structure of the protein even better, a three-dimensional CNN was evaluated. It operates on a raster of 2 Å resolution, where each cell encodes the amino acid present in this cell. This model was very resource-intensive and did not show good results in the beginning. Therefore, research on this model was abandoned early.

Graph convolutional neural networks On the route to the development of the kCNN, a more general form of convolutional neural networks was tried: Graph convolutional neural networks (GNNs). While the neighborhood is defined by neighboring pixels in the simple CNN and by the k nearest neighbors in the kCNN, this concept can be broadened to allow for graphs with arbitrary neighborhoods. For example, edges in the graph can be based on the actual distance of the residues instead of taking the k nearest neighbors. The GNNs were implemented using PyTorch Geometric [93]. Although multiple different architectures were explored, the kCNN always performed significantly better than the GNNs. This might be due to insufficient training data.

Pairwise AAindex descriptors The AAindex database [16–18] also contains descriptors for pairs of amino acids. It was tried to provide these as inputs to the kCNN and the GNNs, but this did not show an improvement over the sScales selection.

kCNN-MLP-hybrid As the MLP and the kCNN showed good results, a hybrid version of the two methods was developed. Here, the kCNN and MLP are first applied separately to the input. Then, their last layer is combined such that the output is a linear combination of both the kCNN filter and the MLP's last layer activations. During training, both parts are jointly optimized with gradient descent. However, this produced worse results than the two models individually.

5 Conclusion and outlook

5.1 Key contributions of this thesis

There are a wide variety of machine learning methods to predict mutation effects on proteins [8, 15, 20, 21]. In this work, four different embeddings (one-hot, sScales, evolutionary, and ProtTrans) and four different models (GLMNET, SVM, MLP, kCNN) have been compared. Additionally, the effect of using the spectral version of an embedding was investigated. It was shown that it is indeed possible to predict the effect of mutations with good accuracy for a variety of different proteins (Section 4.6).

From the model perspective, the strengths and weaknesses of multiple models have been shown (Section 4.3). For the considered datasets, support vector machines (SVM) are a good choice in many cases due to their short runtime and easy usage, which allows for thorough hyperparameter optimization and embedding comparison. In total, SVMs provided the best performance on four of the five datasets. The multi-layer perceptron (MLP) shows benefits for large datasets, where it excels in performance over the SVM. Additionally, training via gradient descent scales better for large datasets than the non-linear complexity of the SVM algorithm. However, it was shown that the hyperparameter optimization (HPO) can be performed on a subset of the dataset for increased scalability (Section 4.4.2).

From an embedding perspective, the feasibility of transfer learning was demonstrated (Section 4.2). Both the evolutionary embedding, which draws information from evolutionary similar sequences, and the ProtTrans embedding, which uses millions of unlabeled sequences, provide useful information to the prediction of mutation effects. These two methods are thus able to augment the dataset with additional knowledge derived from unlabeled sequences. Here, the usage of linear autoencoders for the embedding of the multiple sequence alignment is a novel technique.

The automatic HPO used in this work provided a unified framework for the selection of good hyperparameters, simplifying the comparison of many different methods with a varying number of hyperparameters. The high computational cost of the HPO was handled with distributed computation and an advanced hyperparameter tuning algorithm. Selecting the embedding as part of the HPO proved useful to reduce the load even further (Section 4.3). However, it was noted that the HPO induces variance in the final performance, which required repetition of the HPO for the experiments. Finally, it was still computationally infeasible to optimize the neural network techniques as thoroughly as the SVMs due to their significantly higher runtime per trial.

This work contains a thorough discussion of the effects induced by the dataset and the dataset split. First, it was shown that performance increases significantly with an increase in training dataset size (Section 4.4). Second, it has been demonstrated that it is important to have both multi-mutants in the training dataset for sufficient performance (Section 4.5). If there are mostly single-mutants in the training dataset, and the test dataset contains multi-mutants, performance degrades. Third, a measure of epistasis was defined and its negative correlation to model performance has been empirically validated (Section 4.8). High epistasis in the dataset degrades performance, too.

For the sensitivity analysis, a unified framework has been developed which allows for the end-to-end analysis of the attribution of the mutations and their embeddings to the predicted target value (Section 4.7.1). This allowed for a thorough analysis of the inner workings of the models which provided several insights.

First, the attribution to the predictions of the SVM was often dominated by a single embedding, even though many embedding combinations were tried (Section 4.7.2). It was therefore concluded that the SVM has a prioritization problem due to the L2-norm in the RBF kernel function. Second, the positional attribution of the evolutionary and ProtTrans embeddings was

investigated (Section 4.7.3). It was shown that both embeddings facilitate the attribution to non-mutated positions due to the context-awareness of their embeddings. While the ProtTrans embedding focuses on the mutated positions and their structural surroundings, the attribution of the evolutionary embedding is scattered across the sequence. Third, the importance of the training samples for the prediction of the SVM was inspected. It was shown that many training samples are of little importance and that the ProtTrans embedding allowed for a more detailed similarity measure.
The k-convolutional neural network (kCNN) was proposed in this thesis as a novel alternative to the regular CNN (Section 3.2.4). While the kCNN outperformed the MLP on multiple datasets (Section 4.3), initial assumptions did not hold during sensitivity analysis: The kCNN does not learn general filters that can be applied to multiple parts of the sequence, but the filters specialize in individual mutations (Section 4.7.6). However, the kCNN strengthens the attribution to the structural surroundings of the mutated residues. The analysis of the MLP uncovered prioritization (Section 4.7.7). For example, it was possible to show that of the 13 sScales descriptors, 3 are used primarily by the MLP on dataset `B`, of which only one describes a physical property. The two primary descriptors instead provide frequency information of the amino acids in proteins. For GLMNET, an inherent sparsity was shown in the attribution, that was traced back to the L1-norm (Section 4.7.9).
Finally, the end-to-end sensitivity analysis allows for the attribution of the prediction of a mutant to the mutations it is composed of. Therefore, the effect of individual mutations can be determined. This allows users to investigate the plausibility of the otherwise "black box" models. It was shown that the variance in these attributions correlates to the epistasis for four of the five datasets. In the fifth and smallest dataset, an overestimation of epistasis was uncovered.

5.2 Future work

The main part of future work naturally consists of the improvement of the predictions. To this end, multiple different approaches can be taken:
First, the prioritization problem of the SVM could be counteracted with an advanced kernel function definition. As the sum of multiple kernel functions is a valid kernel function [53], the kernel could be set to the weighted sum of one RBF kernel per embedding part. Other approaches might include a weighting of the embedding dimensions before application of the kernel.
It might be interesting to compare models that are trained for regression but used as classifiers, like in this work, to models that are immediately trained as classifiers. In certain cases, there might be benefits to training classifiers, if the only output required is the classification. Note that a regression model solves a more difficult task because it has to predict the exact value and not just the class. A unified target value scale might be beneficial for prediction, too. For example, the target value from dataset `E` is proportional to the logarithm of the target value from dataset `B`. At the moment, it is unclear what effect this has on the performance of the model.
As the transfer learning approach has shown promising results, additional research in that direction could improve results even further. For example, the encoding of the multiple sequence alignments could be improved with other approaches, e.g. with the Potts model [94]. Additionally, the alignment search methods could be extended to find more similar sequences for the currently sparse alignments of datasets `A`, `C`, and `D`. Similarly, the ProtTrans models could be improved by using more state-of-the-art natural language processing techniques like additional training goals during pretraining on the unlabeled sequences despite masked amino acid reproduction. An interesting combination of these two approaches is given by the embeddings that are learned by AlphaFold 2 [95].

The HPO procedure can be improved, too. Besides better hyperparameter selection algorithms and more computational hardware, the design of the hyperparameter space plays an important role, too. For example, it could be better investigated if the selection of the embeddings during HPO achieves as good results as the direct comparison with multiple HPOs. If so, even the model selection could be considered part of the HPO to further reduce the computational cost. Additionally, with a higher computational budget or better efficiency, neural networks could be investigated further. As these models have significantly more hyperparameters, a longer HPO might be necessary for good optimization.
For the application of the methods in a wide area, automation plays a crucial role. To simplify the usage of the models, an automated pipeline could be designed which spans from the data input to the predictions and attributions, covering dataset splitting, training, HPO, and sensitivity analysis. This would accelerate the use of these methods and broaden the field of application to practitioners inexperienced in machine learning.
Finally, the dataset creation could be investigated further. There is existing work on the field of integrating the mutation effect prediction with the directed evolution study design [14]. As the training sample importance analysis has shown that many samples are unimportant for the prediction, the careful selection of training samples might accelerate the directed evolution process. Because the model has to be trained on existing data, which has to be produced in the laboratory, it would be beneficial to have guidance on what samples are relevant for prediction. This does not only include the usage of the mutants which are predicted to be the best, but also the inclusion of non-perfect mutants for further exploration. In an extreme case, this broadens the prediction problem to a reinforcement learning problem, where the goal is to find a good mutant with as few experimental mutant evaluations as possible.

II Acknowledgments

I would like to thank the people who contributed to the work presented in this thesis. First, my thanks go to my academic supervisors, Prof. Birgit Strodel (Forschungszentrum Jülich), Prof. Uwe Naumann (RWTH Aachen University), and M.Sc. Anna Jäckering (Forschungszentrum Jülich), for their support in creating this work. I would also like to thank Jan Vincent Hoffbauer for proofreading the thesis and providing computational resources and M.Sc. Monika Keutmann for the countless discussions on the content of this thesis. As the work was computationally expensive, I like to thank the HPC center of the RWTH Aachen University: Simulations were performed with computing resources granted by RWTH Aachen University under projects `thes0972` and `rwth0794`. This work uses the ProtTrans models provided by the Rostlab at TU Munich. Training these models requires computational resources not available to me, which makes me grateful for the open access publication of these pre-trained models. Finally, this work would not have been possible without the many contributors to the open-source software used. A special thank goes to the contributors of the Torch, Plotly, scikit-learn, NumPy, Optuna, Transformers, FFTW, and Numba libraries and the PostgreSQL database, all of which play a crucial role in this work.

III User guide

III.1 Installation from source

The software is available for download on GitHub (here):

`https://github.com/Turakar/mutation-prediction`

Additional files that are too large for version control can be downloaded here:

`https://mutation-prediction.thoffbauer.de`

The software has only been tested on Linux operating systems, but it might work on Windows, too. To use the software, you have to install some required packages first:

- Python $\geq$ 3.8 (here).
- Poetry $\geq$ 1.1 (here).
- The Rust toolchain, e.g. via rustup (here).
- CUDA $\geq$ 10.2 (here), if you want to use NVIDIA GPU acceleration (highly recommended). While AMD GPUs are supported by Torch, they were not tested with this software.
- Redis (here), if you want to use it for studies with short trial runtime.
- PostgreSQL (here), if you want to use it for distributed studies.
- A Fortran compiler.

Most dependencies are managed with Poetry. To install the dependencies, run the following command in a shell inside the project directory (without the `$`):

```
$ poetry install
```

If you use a CUDA version different from the default of Torch, you might want to consider updating Torch to that version as shown in the Torch documentation (here). In this case, choose the installation method via Pip and prefix the commands with `poetry run ...`. You do not need `torchvision` and `torchaudio`.
Then, the Rust native module must be compiled and installed. For this, execute the following commands:

```
$ cd mutation_prediction_native
$ poetry run maturin develop --release
```

This will compile the Rust module with all optimizations and install it into the Poetry environment. Finally, the Torch Geometric package can be installed to add support for GNNs (optional). Please follow the official installation instructions (here).
To test the installation, you can show the help page of the CLI with this command:

```
$ poetry run python cli.py -h
```

III.2 Docker

Alternatively, you can use Docker (here). After installation, you only have to run the following command for building the container image. Note that the Docker image uses the `Europe/Berlin` timezone by default.

```
$ docker build -t mutation-prediction .
```

If you want to use NVIDIA GPU acceleration (highly recommended for MLP and kCNN), you have to install the NVIDIA container toolkit (here). To build the container image with GPU acceleration, execute the following command:

```
$ docker build -t mutation-prediction -f Dockerfile.gpu .
```

To test the image, you can show the help page of the command-line interface with this command:

```
$ docker run --rm -it --name mutation_prediction mutation-prediction -h
```

If you want to use GPUs, add the `--gpus all` flag to the Docker command before the image name `mutation-prediction`. You should use either the Redis or the PostgreSQL database, as the container is removed on exit with the `--rm` option.

III.3 Command-line interface

The command-line interface (CLI) provides multiple sub-commands that help with the creation, management, optimization, and evaluation of studies. The CLI is structured into multiple sub-commands. All sub-commands have an individual help page accessible via the following command:

```
$ poetry run python cli.py <subcommand> -h
```

To get started, one can create a simple study:

```
$ poetry run python cli.py --db sqlite:///studies.db new test \
    EpsilonSvrMutInd C75 { \
        C: {low: 1e-3, high: 1e6, log: true}, \
        epsilon: 0, \
        gamma: {low: 1e-6, high: 1e3, log: true} \
    }
```

This command consists of multiple components:

- `--db <database-url>` specifies the URL to the database. Here, a SQLite database is created on disk because this requires no additional installation. Refer to the dcoumentation of SQLAlchemy (here) for creating URLs to SQL storages, and to the Python Redis documentation (here) for creating URLs to Redis storages.
- `new` is the sub-command.
- `test` is the study name.
- `EpsilonSvrMutInd` is the model name.
- `C75` is the dataset name.

- `{...}` specifies the hyperparameter ranges. This is specified in Hjson (here), an extension of JSON that strives to be better human-readable. One can also use JSON here.

The dataset and model names are defined in `mutation_prediction/cli/registry.py`. Each model comes with a set of hyperparameters, that are defined in the corresponding source file under `mutation_prediction/models`. If the embedding takes hyperparameters, too, they are added to the model hyperparameters under the key `embedding`. Please have a look at the source for details on the various models and embeddings. There are additional flags available to control various parameters of the study, e.g. the sampling strategy or the number of cross-validation folds.
To run the optimization of the study `test`, use the following command:

```
$ poetry run python cli.py --db sqlite:///studies.db optimize test -n 50
```

This will run the study until a total of 50 successful trials are completed. Please refer to the help page of the `optimize` sub-command for additional termination criteria.
After optimization, the trial with the best value can be evaluated on the test dataset with the following command:

```
$ poetry run python cli.py --db sqlite:///studies.db evaluate test
```

With the additional flags of the `evaluate` sub-command, the number of test iterations, as well as the trial to test, can be changed.

III.3.1 Precomputation

The evolutionary and ProtTrans embeddings are precomputed. For precomputation, the HPOs as described in the main text must be completed first. The precomputation is managed with the `precompute_embeddings.py` script. For example, to precompute the ProtAlbert embedding for dataset A, execute the following command:

```
$ poetry run python -m mutation_prediction.cli.precompute_embeddings \
    prottrans <ProtTrans-ID> <dataset>
```

Where the `ProtTrans-ID` is one of the IDs of the protein language models listed on the Rostlab Huggingface page (here). The `dataset` can be one of `A`, `B`, `C`, `D`, or `E`. The precomputed embeddings are saved to `precomputed/<ProtTrans-ID>-<dataset>.npy`.
For precomputing the evolutionary embedding, this changes to the following command:

```
$ poetry run python -m mutation_prediction.cli.precompute_embeddings \
    autoencoder <database-url> <study-name> <output-name> <dataset>
```

Here, `database-url` is the URL to the database, `study-name` the name of the HPO study for this autoencoder, `output-name` the desired output name, and `dataset` the dataset like above. The precomputed embeddings are saved to `precomputed/<output-name>.npy`.

IV Developer guide

IV.1 Code structure

The code is organized in the `mutation_prediction` folder and divided into four central modules. The `cli` module provides the command-line interface (CLI) the user can interact with. Its most important sub-component is the registry in `registry.py`, where all models and datasets are defined by name. It also includes a utility `scheduler.py` that allows for scheduling commands on different CPUs in parallel.
The `data` module provides all functionality related to the loading and preprocessing of the datasets. The module file `__init__.py` contains the basic dataset definition. There, a dataset consists of the following members:

- `sequence`: A linear NumPy array that defines the wild type sequence. Each amino acid is encoded by an integer. The mappings are defined by the functions `acid_to_index()` and `index_to_acid()`.
- `positions`: A two-dimensional NumPy array that defines the positions of the mutations. Each row corresponds to one mutant. If a mutant has fewer mutations than the length of one row, the rest of the array is ignored.
- `acids`: A two-dimensional NumPy array containing the substituted amino acids at the positions defined by the previous array. The amino acids are encoded in the same way as `sequence`.
- `num_mutations`: This linear NumPy array contains the number of mutations per mutant.
- `y`: A linear NumPy array with the target values (optional).
- `structure`: The BioPython structure (optional, needed for the kCNN).
- `msa_path`: The path to the MSA archive file (optional, needed for autoencoder training).
- `name`: The name of the dataset (optional, needed for loading the embeddings from precomputed arrays).
- `ids`: IDs of the individual mutants in the dataset (optional, needed for loading the embeddings from precomputed arrays).

Dataset loading from the `data` directory is managed by `loader.py`, while shuffling, splitting, and other preprocessing are done by `preprocessing.py`. For MSA and structure loading, additional helpers are in `msa_helper.py` and `structure_utils.py`, respectively. The MSA helper calls the compiled Rust code in `mutation_prediction_native`.
The `embeddings` module implements the various embeddings. An embedding has two main methods: `embed_update()` and `embed()`. The first function should be used when embedding the training data, while the second function should be used in all other cases. This allows for realistic modeling of the test data as new data. For example, the normalization constants are solely based on the training data but applied to both training and test data. The evolutionary and the ProtTrans embeddings allow for precomputation. While the precomputed evolutionary embeddings are contained in the repository, the ProtTrans embeddings were too large for this. To precompute the ProtTrans embeddings, use `precompute_embeddings.py` in `cli`.
The actual models are defined in the `models` module. A model provides two main methods: `fit()` and `predict()`. The `fit()` method is used to train the model on the dataset, while

the `predict()` method predicts the target values of a dataset. Additionally, a model has hyperparameters. The hyperparameters are organized in a tree structure. The individual hyperparameters are defined in the model constructor and optimized by Optuna during HPO. If an embedding has additional hyperparameters, the model constructor adds them to the model hyperparameters under the key `embedding`.

IV.2 Tooling

The dependencies of the project are managed by Poetry and are listed in `pyproject.toml`. After editing the file or to update the dependencies, run the following command to resolve the dependency graph and install all updates:

```
$ poetry update
```

After updating, Torch might have to be reinstalled as described in the user guide. The Rust dependencies are defined in the file `mutation_prediction_native/Cargo.toml`. After changing the requirements, re-run Maturin.
For formatting and linting, this project uses black, isort, and flake8. To run all checks, execute the following command:

```
$ poetry run ./check.sh
```

The black and isort tools support automatic reformatting of the code. To execute reformatting, run the following commands:

```
$ poetry run black .
$ poetry run isort .
```

The tests are run with pytest, and stored in the `tests` folder. To run the test suite, execute the following command:

```
$ poetry run pytest
```

IV.3 Other folders

The `experiments` folder contains various experiments that were conducted during development. This includes the comparisons that are part of the main text, but also some preliminary experiments. Many of the experiments are written as Jupyter Notebooks and have to be opened with this software for viewing and editing. The `images` folder contains the plots used in this thesis as static PDF and interactive HTML files.

V References

[1] Michael R. Bedford. "The Effect of Enzymes on Digestion". In: *Journal of Applied Poultry Research* 5.4 (Dec. 1996), pp. 370–378. DOI: 10.1093/japr/5.4.370 (cit. on p. 1).

[2] S M Morris. "Regulation of Enzymes of Urea and Arginine Synthesis". In: *Annual Review of Nutrition* 12.1 (July 1992), pp. 81–101. DOI: 10.1146/annurev.nu.12.070192.000501 (cit. on p. 1).

[3] Anita H. Payne and Dale B. Hales. "Overview of Steroidogenic Enzymes in the Pathway from Cholesterol to Active Steroid Hormones". In: *Endocrine Reviews* 25.6 (Dec. 2004), pp. 947–970. DOI: 10.1210/er.2003-0030. URL: https://doi.org/10.1210/er.2003-0030 (cit. on p. 1).

[4] Catherine J. E. Ingram, Charlotte A. Mulcare, Yuval Itan, Mark G. Thomas, and Dallas M. Swallow. "Lactose digestion and the evolutionary genetics of lactase persistence". In: *Human Genetics* 124.6 (Nov. 2008), pp. 579–591. DOI: 10.1007/s00439-008-0593-6 (cit. on p. 1).

[5] T. Spellig, A. Bottin, and R. Kahmann. "Green fluorescent protein (GFP) as a new vital marker in the phytopathogenic fungusUstilago maydis". In: *Molecular and General Genetics MGG* 252.5 (Oct. 1996), pp. 503–509. DOI: 10.1007/bf02172396. URL: https://doi.org/10.1007/bf02172396 (cit. on p. 1).

[6] Lucas Marie-Orleach, Tim Janicke, Dita B Vizoso, Micha Eichmann, and Lukas Schärer. "Fluorescent sperm in a transparent worm: validation of a GFP marker to study sexual selection". In: *BMC Evolutionary Biology* 14.1 (2014), p. 148. DOI: 10.1186/1471-2148-14-148. URL: https://doi.org/10.1186/1471-2148-14-148 (cit. on p. 1).

[7] Birgit Koch, Linda Elise Jensen, and Ole Nybroe. "A panel of Tn7-based vectors for insertion of the gfp marker gene or for delivery of cloned DNA into Gram-negative bacteria at a neutral chromosomal site". In: *Journal of Microbiological Methods* 45.3 (July 2001), pp. 187–195. DOI: 10.1016/s0167-7012(01)00246-9. URL: https://doi.org/10.1016/s0167-7012(01)00246-9 (cit. on p. 1).

[8] Frédéric Cadet, Nicolas Fontaine, Guangyue Li, Joaquin Sanchis, Matthieu Ng Fuk Chong, Rudy Pandjaitan, Iyanar Vetrivel, Bernard Offmann, and Manfred T. Reetz. "A machine learning approach for reliable prediction of amino acid interactions and its application in the directed evolution of enantioselective enzymes". In: *Scientific Reports* 8.1 (Nov. 2018). DOI: 10.1038/s41598-018-35033-y (cit. on pp. 1–3, 12, 22, 28, 51).

[9] Ingemar von Ossowski, Georg Hausner, and Peter C. Loewen. "Molecular evolutionary analysis based on the amino acid sequence of catalase". In: *Journal of Molecular Evolution* 37.1 (July 1993), pp. 71–76. DOI: 10.1007/bf00170464 (cit. on pp. 1, 3).

[10] Ryan E. Cobb, Ran Chao, and Huimin Zhao. "Directed evolution: Past, present, and future". In: *American Institute of Chemical Engineering Journal* 59.5 (Jan. 2013), pp. 1432–1440. DOI: 10.1002/aic.13995 (cit. on p. 1).

[11] NobelPrize.org. *Frances H. Arnold – Facts – 2018*. URL: https://www.nobelprize.org/prizes/chemistry/2018/arnold/facts/ (visited on 06/28/2021) (cit. on p. 1).

[12] Manfred T. Reetz, Marco Bocola, José Daniel Carballeira, Dongxing Zha, and Andreas Vogel. "Expanding the Range of Substrate Acceptance of Enzymes: Combinatorial Active-Site Saturation Test". In: *Angewandte Chemie International Edition* 44.27 (July 2005), pp. 4192–4196. DOI: 10.1002/anie.200500767 (cit. on p. 1).

[13] Stanislav Mazurenko, Zbynek Prokop, and Jiri Damborsky. "Machine Learning in Enzyme Engineering". In: *American Chemical Society Catalysis* 10.2 (Dec. 2019), pp. 1210–1223. DOI: 10.1021/acscatal.9b04321 (cit. on p. 1).

[14] Zachary Wu, S. B. Jennifer Kan, Russell D. Lewis, Bruce J. Wittmann, and Frances H. Arnold. "Machine learning-assisted directed protein evolution with combinatorial libraries". In: *Proceedings of the National Academy of Sciences* 116.18 (Apr. 2019), pp. 8852–8858. DOI: 10.1073/pnas.1901979116 (cit. on pp. 2, 21, 53).

[15] Yuting Xu, Deeptak Verma, Robert P. Sheridan, Andy Liaw, Junshui Ma, Nicholas M. Marshall, John McIntosh, Edward C. Sherer, Vladimir Svetnik, and Jennifer M. Johnston. "Deep Dive into Machine Learning Models for Protein Engineering". In: *Journal of Chemical Information and Modeling* 60.6 (Apr. 2020), pp. 2773–2790. DOI: 10.1021/acs.jcim.0c00073 (cit. on pp. 2, 3, 9, 50, 51).

[16] Kenta Nakai, Akinori Kidera, and Minoru Kanehisa. "Cluster analysis of amino acid indices for prediction of protein structure and function". In: *Protein Engineering, Design and Selection* 2.2 (1988), pp. 93–100. DOI: 10.1093/protein/2.2.93 (cit. on pp. 2, 9, 50).

[17] Kentaro Tomii and Minoru Kanehisa. "Analysis of amino acid indices and mutation matrices for sequence comparison and structure prediction of proteins". In: *Protein Engineering, Design and Selection* 9.1 (1996), pp. 27–36. DOI: 10.1093/protein/9.1.27 (cit. on pp. 2, 9, 50).

[18] S. Kawashima, H. Ogata, and M. Kanehisa. "AAindex: Amino Acid Index Database". In: *Nucleic Acids Research* 27.1 (Jan. 1999), pp. 368–369. DOI: 10.1093/nar/27.1.368 (cit. on pp. 2, 9, 50).

[19] S. Kawashima. "AAindex: Amino Acid index database". In: *Nucleic Acids Research* 28.1 (Jan. 2000), pp. 374–374. DOI: 10.1093/nar/28.1.374 (cit. on pp. 2, 9).

[20] Kevin K. Yang, Zachary Wu, and Frances H. Arnold. "Machine-learning-guided directed evolution for protein engineering". In: *Nature Methods* 16.8 (July 2019), pp. 687–694. DOI: 10.1038/s41592-019-0496-6 (cit. on pp. 3, 31, 51).

[21] Sam Sinai, Eric Kelsic, George M Church, and Martin A Nowak. "Variational auto-encoding of protein sequences". In: (2017). arXiv: 1712.03346 (cit. on pp. 3, 49, 51).

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: (2018). arXiv: 1810.04805 (cit. on pp. 3, 12).

[23] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators". In: (2020). arXiv: 2003.10555 (cit. on pp. 3, 12).

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space". In: (2013). arXiv: 1301.3781 (cit. on p. 3).

[25] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. "Distributed Representations of Words and Phrases and their Compositionality". In: (2013). arXiv: 1310.4546 (cit. on p. 3).

[26] Ehsaneddin Asgari and Mohammad R. K. Mofrad. "Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics". In: *Public Library of Science - PLOS ONE* 10.11 (Nov. 2015). Ed. by Firas H Kobeissy, e0141287. DOI: 10.1371/journal.pone.0141287. URL: https://doi.org/10.1371/journal.pone.0141287 (cit. on p. 3).

[27] Ahmed Elnaggar et al. "ProtTrans: Towards Cracking the Language of Lifes Code Through Self-Supervised Deep Learning and High Performance Computing". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021). DOI: 10.1109/tpami.2021.3095381 (cit. on pp. 3, 12, 18, 29).

[28] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations". In: (2019). arXiv: 1909.11942 (cit. on pp. 3, 12).

[29] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: (2019). arXiv: 1910.10683 (cit. on pp. 3, 12).

[30] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: (2019). arXiv: 1906.08237 (cit. on pp. 3, 12).

[31] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG] (cit. on p. 7).

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org (cit. on pp. 7, 15).

[33] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. arXiv: 1207.0580 [cs.NE] (cit. on pp. 7, 11, 15).

[34] V. Biou, J.F. Gibrat, J.M. Levin, B. Robson, and J. Garnier. "Secondary structure prediction: combination of three different methods. Information value for accessibility; average fraction 35%". In: *Protein Engineering 2, 185-191* (1988). URL: https://www.genome.jp/entry/aaindex:BIOV880101 (cit. on p. 10).

[35] M. Blaber, X.J. Zhang, and B.W. Matthews. "Structural basis of amino acid alpha helix propensity. Alpha helix propensity of position 44 in T4 lysozyme". In: *Science 260, 1637-1640* (1993). URL: https://www.genome.jp/entry/aaindex:BLAM930101 (cit. on p. 10).

[36] K. Nagano. "Local analysis of the mechanism of protein folding. I. Prediction of helices, loops, and beta-structures from primary structure. Normalized frequency of alpha-helix". In: *Journal of Molecular Biology 75, 401-420* (1973). URL: https://www.genome.jp/entry/aaindex:NAGK730101 (cit. on p. 10).

[37] J. Tsai, R. Taylor, C. Chothia, and M. Gerstein. "The packing density in proteins: standard radii and volumes. Volumes including the crystallographic waters using the ProtOr". In: *Journal of Molecular Biology 290, 253-266 (Cyh 112.8)* (1999). URL: https://www.genome.jp/entry/aaindex:TSAJ990101 (cit. on p. 10).

[38] H. Nakashima and K. Nishikawa. "The amino acid composition is different between the cytoplasmic and extracellular sides in membrane proteins. AA composition of CYT of multi-spanning proteins". In: *Federation of European Biochemical Societies Letter 303, 141-146* (1992). URL: https://www.genome.jp/entry/aaindex:NAKH920106 (cit. on p. 10).

[39] H. Nakashima and K. Nishikawa. "The amino acid composition is different between the cytoplasmic and extracellular sides in membrane proteins. AA composition of EXT of multi-spanning proteins". In: *Federation of European Biochemical Societies Letter 303, 141-146* (1992). URL: https://www.genome.jp/entry/aaindex:NAKH920107 (cit. on p. 10).

[40] H. Nakashima and K. Nishikawa. "The amino acid composition is different between the cytoplasmic and extracellular sides in membrane proteins. AA composition of MEM of multi-spanning proteins". In: *Federation of European Biochemical Societies Letter 303, 141-146* (1992). URL: https://www.genome.jp/entry/aaindex:NAKH920108 (cit. on pp. 10, 44).

[41] J. Cedano, P. Aloy, J.A. Perez-Pons, and E. Querol. "Relation between amino acid composition and cellular location of proteins. AA composition of MEM of multi-spanning proteins". In: *Journal of Molecular Biology 266, 594-600* (1997). URL: https://www.genome.jp/entry/aaindex:CEDJ970104 (cit. on pp. 10, 44).

[42] S. Lifson and C. Sander. "Antiparallel and parallel beta-strands differ in amino acid residue preference. Conformational preference for all beta-strands". In: *Nature 282, 109-111* (1979). URL: https://www.genome.jp/entry/aaindex:LIFS790101 (cit. on p. 10).

[43] S. Miyazawa and R. L. Jernigan. "Self-consistent estimation of inter-residue protein contact energies based on an equilibrium mixture approximation of residues. Optimized relative partition energies - method C". In: *Proteins 34, 49-68* (1999). URL: https://www.genome.jp/entry/aaindex:MIYS990104 (cit. on p. 10).

[44] S. Miyazawa and R. L. Jernigan. "Structural prediction of membrane-bound proteins. Hydrophobicity index". In: *European Journal of Biochemistry 128, 565-575* (1982). URL: https://www.genome.jp/entry/aaindex:ARGP820101 (cit. on pp. 10, 44).

[45] D.M. Dawson. "Size". In: *The Biochemical Genetics of Man (Brock, D.J.H. and Mayo, O., eds.), Academic Press, New York, pp.1-38 (1972)* (1972). URL: https://www.genome.jp/entry/aaindex:DAWD720101 (cit. on p. 10).

[46] J.L. Fauchere, M. Charton, L.B. Kier, A. Verloop, and V. Pliska. "Amino acid side chain parameters for correlation studies in biology and pharmacology. Number of hydrogen bond donors". In: *International Journal of Peptide and Protein Research 32, 269-278* (1988). URL: https://www.genome.jp/entry/aaindex:FAUJ880109 (cit. on p. 10).

[47] Milot Mirdita, Lars von den Driesch, Clovis Galiez, Maria J. Martin, Johannes Söding, and Martin Steinegger. "Uniclust databases of clustered and deeply annotated protein sequences and alignments". In: *Nucleic Acids Research* 45.D1 (Nov. 2016), pp. D170–D176. DOI: 10.1093/nar/gkw1081 (cit. on pp. 12, 23, 29).

[48] Martin Steinegger, Milot Mirdita, and Johannes Söding. "Protein-level assembly increases protein sequence recovery from metagenomic samples manyfold". In: *Nature Methods* 16.7 (June 2019), pp. 603–606. DOI: 10.1038/s41592-019-0437-4 (cit. on pp. 12, 29).

[49] Martin Steinegger and Johannes Söding. "Clustering huge protein sequence sets in linear time". In: *Nature Communications* 9.1 (June 2018). DOI: 10.1038/s41467-018-04964-5 (cit. on pp. 12, 29).

[50] Stephen Roberts. *Signal Processing Lecture 7 - The Discrete Fourier Transform*. URL: https://www.robots.ox.ac.uk/~sjrob/Teaching/SP/l7.pdf (visited on 06/30/2021) (cit. on p. 13).

[51] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. "Regularization Paths for Generalized Linear Models via Coordinate Descent". In: *Journal of Statistical Software* 33.1 (2010). DOI: 10.18637/jss.v033.i01 (cit. on pp. 13, 18).

[52] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM". In: *Association for Computing Machinery - Transactions on Intelligent Systems and Technology* 2.3 (Apr. 2011), pp. 1–27. DOI: 10.1145/1961189.1961199 (cit. on p. 14).

[53] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. "Kernel methods in machine learning". In: *The Annals of Statistics* 36.3 (2008), pp. 1171–1220. DOI: 10.1214/009053607000000677 (cit. on pp. 14, 52).

[54] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. Journal of Machine Learning Research - Workshop and Conference Proceedings. 2011, pp. 315–323 (cit. on pp. 14, 16).

[55] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314. DOI: 10.1007/bf02551274 (cit. on p. 15).

[56] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. June 2016 (cit. on p. 15).

[57] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (Dec. 1989), pp. 541–551. DOI: 10.1162/neco.1989.1.4.541 (cit. on p. 15).

[58] Bruce J. Wittmann, Anders M. Knight, Julie L. Hofstra, Sarah E. Reisman, S. B. Jennifer Kan, and Frances H. Arnold. "Diversity-Oriented Enzymatic Synthesis of Cyclopropane Building Blocks". In: *American Chemical Society Catalysis* 10.13 (June 2020), pp. 7112–7116. DOI: 10.1021/acscatal.0c01888 (cit. on pp. 16, 22).

[59] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2014. DOI: 10.3115/v1/d14-1181. URL: https://doi.org/10.3115/v1/d14-1181 (cit. on p. 16).

[60] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. "Algorithms for hyper-parameter optimization". In: *25th annual conference on neural information processing systems (NIPS 2011)*. Vol. 24. Neural Information Processing Systems Foundation. 2011 (cit. on p. 17).

[61] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697 (cit. on p. 18).

[62] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2 (cit. on p. 18).

[63] T. Hamelryck and B. Manderick. "PDB file parser and structure class implemented in Python". In: *Bioinformatics* 19.17 (Nov. 2003), pp. 2308–2310. DOI: 10.1093/bioinformatics/btg299 (cit. on p. 18).

[64] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. "Numba". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC - LLVM '15*. ACM Press, 2015. DOI: 10.1145/2833157.2833162 (cit. on p. 18).

[65] Steve Klabnik and Carol Nichols. *The Rust Programming Language*. USA: No Starch Press, 2018. ISBN: 1593278284 (cit. on p. 18).

[66] Alexander Niederbühl, Martin Larralde, Yuji Kanagawa, konstin, mejrs, messense, and Paul Ganssle. *PyO3 v0.13.2: Rust bindings for the Python interpreter*. URL: https://pyo3.rs/v0.13.2/ (cit. on p. 18).

[67] Plotly Technologies Inc. *Collaborative data science*. 2015. URL: https://plot.ly (cit. on p. 18).

[68] Matteo Frigo and Steven G. Johnson. "The Design and Implementation of FFTW3". In: *Proceedings of the IEEE* 93.2 (2005). Special issue on "Program Generation, Optimization, and Platform Adaptation", pp. 216–231 (cit. on p. 18).

[69] Kevin Sheppard et al. *bashtage/arch: Release 5.0.1*. 2021. DOI: 10.5281/ZENODO.593254 (cit. on p. 18).

[70] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 18).

[71] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on p. 18).

[72] W.A. Falcon et al. "PyTorch Lightning". In: *GitHub* 3 (2019). URL: https://github.com/PyTorchLightning/pytorch-lightning (cit. on p. 18).

[73] Thomas Wolf et al. “Transformers: State-of-the-Art Natural Language Processing”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6 (cit. on pp. 18, 25).

[74] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. “Optuna”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, July 2019. DOI: 10.1145/3292500.3330701 (cit. on p. 18).

[75] M. Stonebraker, L.A. Rowe, and M. Hirohama. “The implementation of POSTGRES”. In: *IEEE Transactions on Knowledge and Data Engineering* 2.1 (Mar. 1990), pp. 125–142. DOI: 10.1109/69.50912 (cit. on p. 18).

[76] Docker Inc. *Docker*. URL: https://docker.com/ (cit. on p. 19).

[77] Salvatore Sanfilippo and the Redis community. *Redis*. URL: https://redis.io/ (cit. on p. 19).

[78] Martin K.M. Engqvist, R. Scott McIsaac, Peter Dollinger, Nicholas C. Flytzanis, Michael Abrams, Stanford Schor, and Frances H. Arnold. “Directed Evolution of Gloeobacter violaceus Rhodopsin Spectral Properties”. In: *Journal of Molecular Biology* 427.1 (Jan. 2015), pp. 205–220. DOI: 10.1016/j.jmb.2014.06.015 (cit. on p. 21).

[79] Yosephine Gumulya, Joaquin Sanchis, and Manfred T. Reetz. “Many Pathways in Laboratory Evolution Can Lead to Improved Enzymes: How to Escape from Local Minima”. In: *ChemBioChem* 13.7 (Apr. 2012), pp. 1060–1066. DOI: 10.1002/cbic.201100784 (cit. on p. 21).

[80] Karen S. Sarkisyan et al. “Local fitness landscape of the green fluorescent protein”. In: *Nature* 533.7603 (May 2016), pp. 397–401. DOI: 10.1038/nature17995. URL: https://doi.org/10.1038/nature17995 (cit. on p. 21).

[81] Takefumi Morizumi, Wei-Lin Ou, Ned Van Eps, Keiichi Inoue, Hideki Kandori, Leonid S. Brown, and Oliver P. Ernst. “X-ray Crystallographic Structure and Oligomerization of Gloeobacter Rhodopsin”. In: *Scientific Reports* 9.1 (Aug. 2019). DOI: 10.1038/s41598-019-47445-5 (cit. on p. 22).

[82] Manfred T. Reetz et al. “Directed Evolution of an Enantioselective Epoxide Hydrolase: Uncovering the Source of Enantioselectivity at Each Evolutionary Stage”. In: *Journal of the American Chemical Society* 131.21 (May 2009), pp. 7334–7343. DOI: 10.1021/ja809673d (cit. on p. 22).

[83] G.S. Kachalova, A.P. Popov, A.A. Simanovskaya, M.V. Krukova, and A.V. Lipkin. EGFP (enhanced green fluorescent protein) mutant - L232H. Mar. 2018. DOI: 10.2210/pdb5n9o/pdb (cit. on p. 22).

[84] Michael Remmert, Andreas Biegert, Andreas Hauser, and Johannes Söding. “HHblits: lightning-fast iterative protein sequence searching by HMM-HMM alignment”. In: *Nature Methods* 9.2 (Dec. 2011), pp. 173–175. DOI: 10.1038/nmeth.1818 (cit. on pp. 23, 48).

[85] Martin Steinegger, Markus Meier, Milot Mirdita, Harald Vöhringer, Stephan J. Haunsberger, and Johannes Söding. "HH-suite3 for fast remote homology detection and deep protein annotation". In: *BioMed Central Bioinformatics* 20.1 (Sept. 2019). DOI: 10.1186/s12859-019-3019-7 (cit. on p. 23).

[86] Andrew W. Senior et al. "Improved protein structure prediction using potentials from deep learning". In: *Nature* 577.7792 (Jan. 2020), pp. 706–710. DOI: 10.1038/s41586-019-1923-7. URL: https://doi.org/10.1038/s41586-019-1923-7 (cit. on p. 23).

[87] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html (cit. on p. 25).

[88] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. "Axiomatic Attribution for Deep Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. Aug. 2017, pp. 3319–3328. URL: http://proceedings.mlr.press/v70/sundararajan17a.html (cit. on pp. 36, 37).

[89] Schrödinger, LLC. "The PyMOL Molecular Graphics System, Version 1.8". Nov. 2015 (cit. on p. 39).

[90] Jinyu Zou, B Martin Hallberg, Terese Bergfors, Franz Oesch, Michael Arand, Sherry L Mowbray, and T Alwyn Jones. "Structure of Aspergillus niger epoxide hydrolase at 1.8 Å resolution: implications for the structure and function of the mammalian microsomal class of epoxide hydrolases". In: *Structure* 8.2 (Feb. 2000), pp. 111–122. DOI: 10.1016/s0969-2126(00)00087-3 (cit. on p. 44).

[91] bnaul (https://stats.stackexchange.com/users/2111/bnaul). *Why L1 norm for sparse models*. Cross Validated. URL:https://stats.stackexchange.com/q/45644 (version: 2016-04-04). eprint: https://stats.stackexchange.com/q/45644. URL: https://stats.stackexchange.com/q/45644 (cit. on p. 45).

[92] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. "Basic local alignment search tool". In: *Journal of Molecular Biology* 215.3 (Oct. 1990), pp. 403–410. DOI: 10.1016/s0022-2836(05)80360-2 (cit. on p. 48).

[93] Matthias Fey and Jan Eric Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: (2019). arXiv: 1903.02428 (cit. on p. 50).

[94] Stefan Seemayer, Markus Gruber, and Johannes Söding. "CCMpred—fast and precise prediction of protein residue–residue contacts from correlated mutations". In: *Bioinformatics* 30.21 (July 2014), pp. 3128–3130. DOI: 10.1093/bioinformatics/btu500. URL: https://doi.org/10.1093/bioinformatics/btu500 (cit. on p. 52).

[95] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* (2021). Accelerated article preview. DOI: 10.1038/s41586-021-03819-2 (cit. on p. 52).