Predict & Improving Protein Stability

Parsa Shahidi

GitHub

Project Overview

Phase 1: Predicting Proteins Stability

At this stage, the goal is to develop a computational model for predicting the **melting temperature (T_m)** of proteins based on their amino acid sequences.

Phase 2: Improve Protein Stability

In this phase, we use models created in phase 1 to improve the protein stability by **changing one amino acid** in the sequence while preserving the original 3D structure. (single point mutation)

Data preparation

Data Collection

The dataset for this project is obtained from **three main sources**:

1. **ProThermDB**
   The ProThermDB database allows users to select features for display and perform searches.
   Database Link (Data can be explored via the "Browse -> Search" section.)
   This dataset does not include protein sequences, but the sequences can be retrieved using the UniProt_ID.
2. **Kaggle - Novozymes Enzyme Stability Prediction**
   This dataset was released as part of a Kaggle competition focused on enzyme

stability prediction — a task closely aligned with the goals of this project.
🔗 https://www.kaggle.com/competitions/novozymes-enzyme-stability-prediction/data

3. **Kaggle - FireProtDB (Enhanced Dataset)**
   This dataset, hosted on Kaggle, is a curated and improved version of enzyme stability data. It was compiled by a user for a previous competition and serves as a high-quality resource for training and benchmarking.
   🔗 https://www.kaggle.com/datasets/dschettler8845/fireprotdb-database

Sequence Extraction

To extract the protein sequences, we download the **uniprot_sprot.fasta** database. This database contains UniProt_IDs along with their corresponding protein sequences. Next, we execute **seq_extract.py** to retrieve sequences.

This script employs two methods to find protein sequences:

1. **Searching in the Local Database:** The script first attempts to extract the required sequence from the uniprot_sprot.fasta database.
2. **Using the API:** If the UniProt_ID is not found in the local database, the sequence is retrieved through the UniProt REST API.

Processing Summary:

- Processed **923** unique UniProt IDs
- Success rate: 923/942 (**98.0%**)

This script converts raw_data.tsv to protein_seq.tsv.

Pre-Processing

You can see pre-processing steps in **datasets/pre_processing.ipynb**

In the notebook we explained pre-processing steps with details and relevant **histograms** and **correlation functions.**

Features Explanation

In the **Null handling** section of notebook, we observed that many features have high null percentage. So we decided to just continue with features that have **less than 75%** null values.

The total sample count after cleaning is **28456** with **926** unique sequences.

	Null Count	Null %	Non-Null Count	Non-Null %
UniProt_ID	0	0.00	28456	100.00
PDB_wild	1716	6.03	26740	93.97
SEC_STR	12023	42.25	16433	57.75
ASA	14299	50.25	14157	49.75
pH	69	0.24	28387	99.76
T_(C)	14413	50.65	14043	49.35
Tm_(C)	14436	50.73	14020	49.27
ΔTm_(C)	22069	77.55	6387	22.45
ΔH_(kcal/mol)	23507	82.61	4949	17.39
ΔCp_(kcal/mol)	25391	89.23	3065	10.77
ΔHvH_(kcal/mol)	23364	82.11	5092	17.89
ΔG_(kcal/mol)	27074	95.14	1382	4.86
ΔΔG_(kcal/mol)	25640	90.10	2816	9.90
m_(kcal/mol/M)	21303	74.86	7153	25.14
Cm_(M)	21082	74.09	7374	25.91
ΔG_H2O_(kcal/mol)	19566	68.76	8890	31.24
ΔΔG_H2O_(kcal/mol)	22687	79.73	5769	20.27
STATE	22879	80.40	5577	19.60
REVERSIBILITY	1346	4.73	27110	95.27
Protein_Sequence	0	0.00	28456	100.00
SEC_STR_ENCODED	0	0.00	28456	100.00

Some columns have a lot of null values, in the following we just use columns with less than 75% null values including:

- ASA
- pH
- T_(C)
- Tm_(C)
- m_(kcal/mol/M)
- Cm_(M)
- ΔG_H2O_(kcal/mol)
- REVERSIBILITY
- SEC_STR_ENCODED
- Protein_Sequence

Hese is brief explanation of each feature:

- **Protein Sequence:** The amino acid sequence of a protein, determining its structure and function.

- **Sec Str (Secondary Structure):** refers to the local folding patterns of a protein's backbone, mainly including **alpha-helices (α-helices)** and **beta-sheets (β-sheets)**, which are stabilized by hydrogen bonds.

- **ASA (Accessible Surface Area):** The surface area of a biomolecule that is accessible to solvent molecules.
- **pH & T:** These are experimental conditions, and most experiments were conducted under neutral conditions and room temperature (pH = 7, T = 25°C).

- $\boldsymbol{T_m}$ **(C) (Melting Temperature):** The temperature at which 50% of a protein is unfolded, indicating thermal stability.

- $\boldsymbol{\Delta G_{H2o}}$ **(kcal/mol) (Free Energy Change in Water):** The free energy of unfolding in water, measuring protein stability in the absence of denaturants.
  A **denaturant** is a chemical that **unfolds proteins**.

- **m (kcal/mol/M):** It's the **slope** of the line when you graph protein stability vs. denaturant concentration. The m-value tells us how sensitive the protein is to the denaturant.

- $\boldsymbol{C_m}$ **(M) (Midpoint Concentration in Molarity):** The concentration of the denaturant where **half of the protein is unfolded** and the other half is still folded.
  It's a key number because it tells us **how stable** the protein is. A higher C_m means the protein is harder to unfold (more stable).

- **Reversibility:** Indicates whether the protein folding/unfolding process is reversible.

Encoding

Excluding sequences, two other features don't contain numerical values. So, we used different encodings for each one of them:

1- **Secondary structure: count-base encodings**

There are 4 secondary structure types: **Coil, Helix, Sheet, Turn**

Each sample has a number of each secondary structure type, we can count the number of types and represent it as a vector.

Encoded vector: [Coil-count, Helix-count, Sheet-count, Turn-count]

For example:

Secondary Structure	Count-Based Encoding
Coil	[1, 0, 0, 0]
Helix	[0, 1, 0, 0]
Sheet	[0, 0, 1, 0]
Turn	[0, 0, 0, 1]
Helix, Sheet	[0, 1, 1, 0]
Helix, Turn, Turn	[0, 1, 0, 2]
-	[0, 0, 0, 0]

2- **Reversibility: mapped to numerical values between -1 and 1**
Reversibility values are mostly **yes** or **no** Booleans.

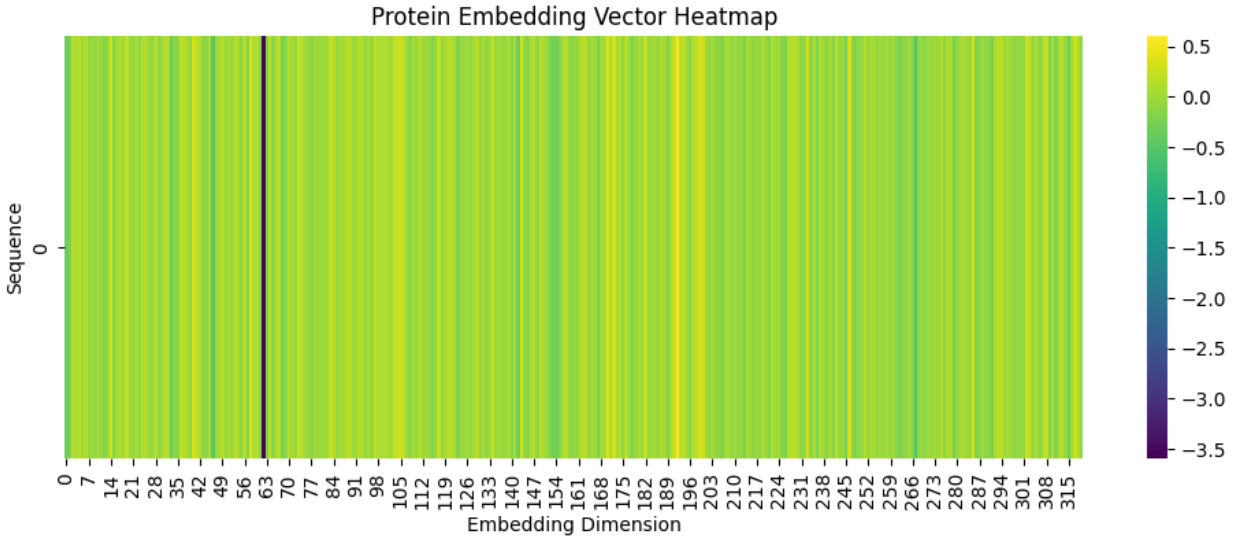
Embedding

You can see embedding steps in **notebooks/1-embedding.ipynb**

We use **Facebook ESM2** pre-trained model for sequence embedding. It's designed to understand and represent protein sequences in the same way language models (like GPT) understand natural language. Instead of words, it learns the "language" of amino acids.

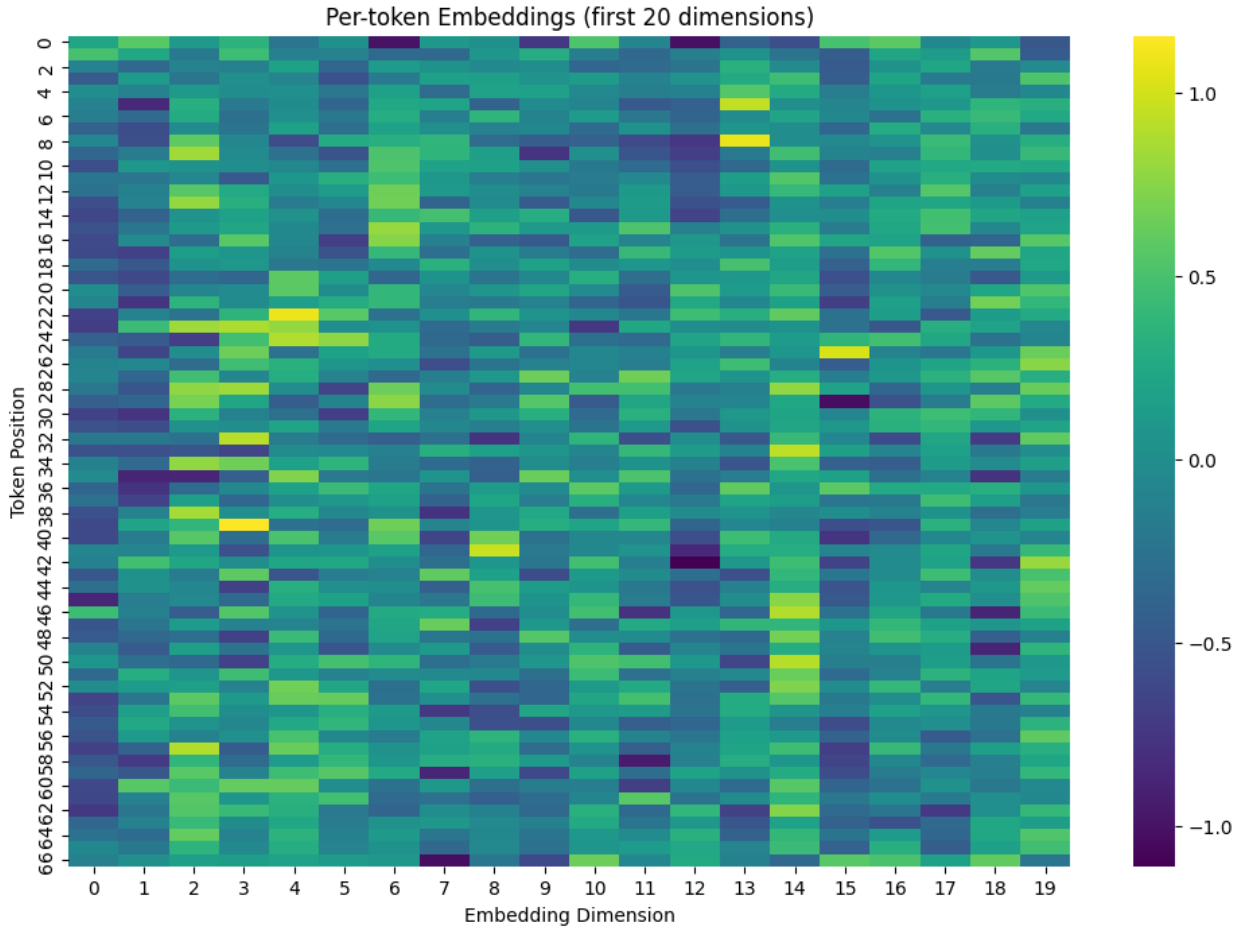
This model takes a sequence string for input and outputs a **320 dimensional** embedding vector.

In the following we embedded an example sequence with a length of 65 for better understanding.



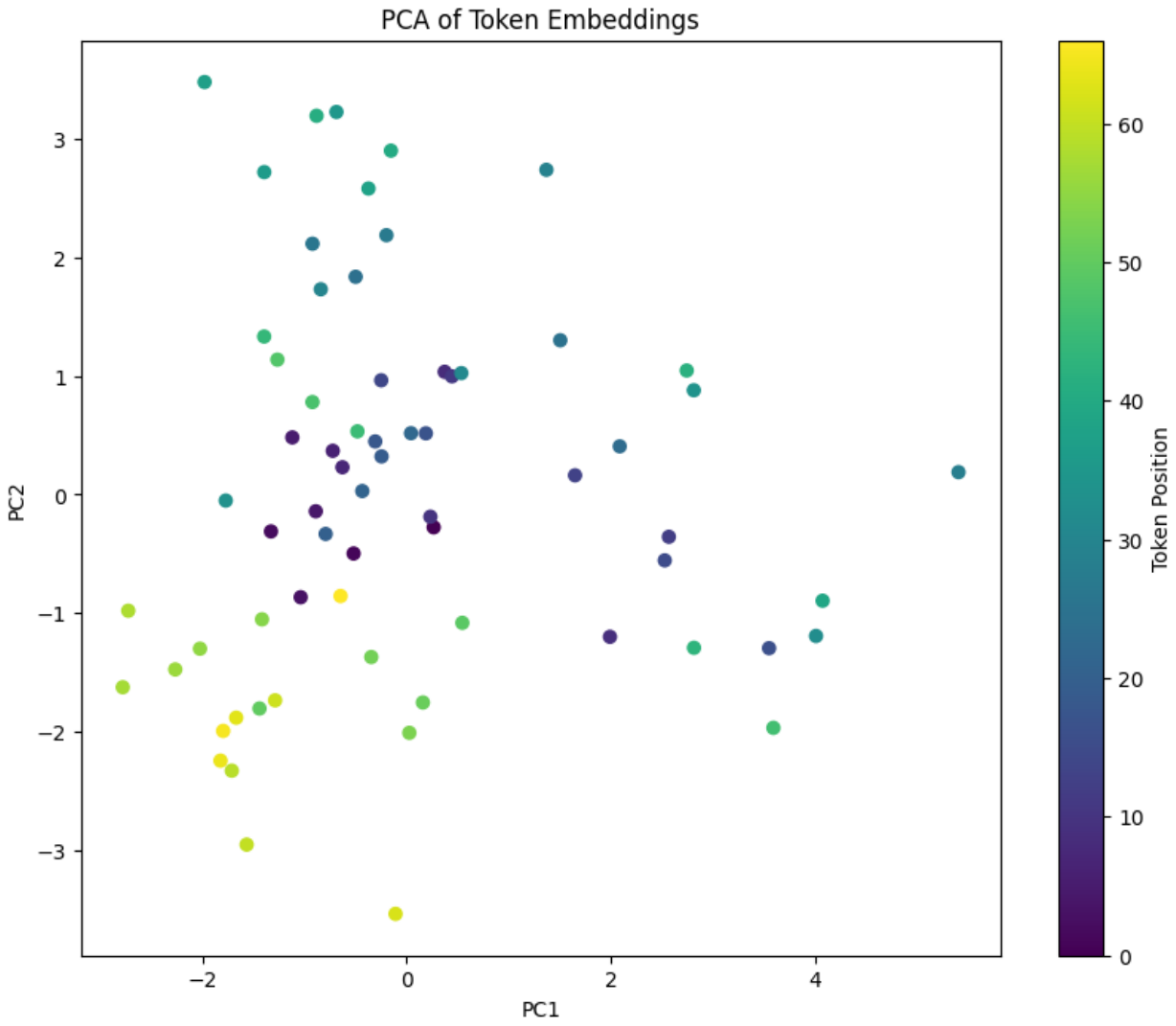
Tokenizing

Tokenizing in protein language is fairly easy, Each amino-acid or letter in the sequence represent as a token. You can see per-token embeddings for first 20 dimensions below.



This heatmap shows that the pre-trained model understands the relations between amino-acids duo to their positioning. For example the last two amino-acids of example sequence are both "G", But in the heatmap you can see the last two tokens have different embeddings, This happens because the model is a transformer and it's picking up long range relationships between amino-acids.

We use PCA to visualize high-dimensional token representations in a lower-dimensional space.



- **Clustering:** Some groups of tokens are close to each other, indicating that they have similar representations in the model's embedding space.
- **Separation:** Tokens from different parts of the sequence may occupy different regions, hinting at a possible progression or shift in semantic or syntactic roles.
- **Outliers:** A few tokens are far from the main cloud — possibly unique or semantically distinct tokens.

At the end of the notebook, we saved unique sequences and their embeddings in **sequence_embeddings.csv** for later use in pre-processing.

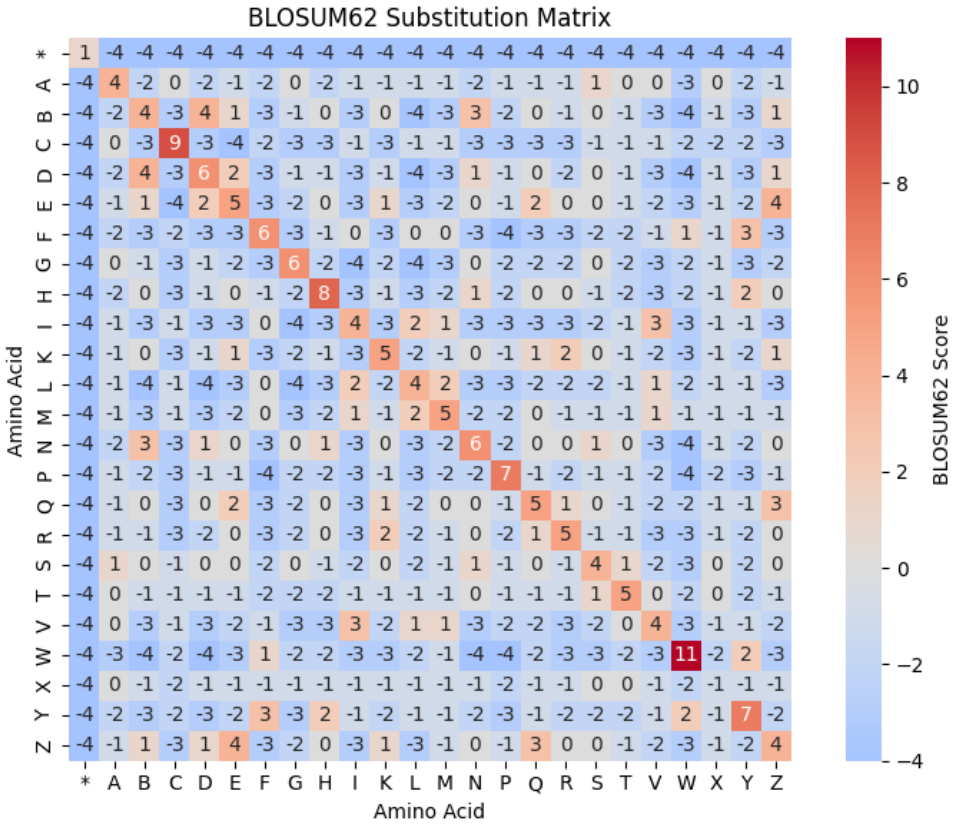
Augmentation

Check out **notebooks/2-augmentation.ipynb** for more details on the augmentation process.

After training several classical models, we found that the dataset was too small to achieve strong performance. To overcome this, we decided to generate new sequences by randomly mutating amino acids while aiming to preserve the original **melting temperature (Tm)**.

Since Tm depends on the **3D structure** of the protein, we applied two key constraints to help preserve structural stability during augmentation:

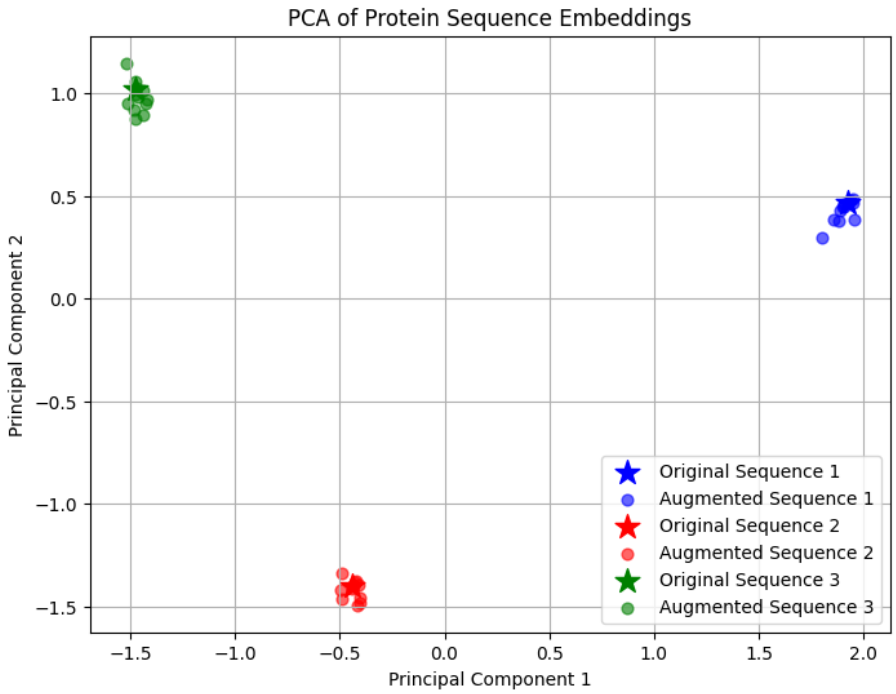
1. **BLOSUM62 Substitution Matrix**
   We used the BLOSUM62 matrix to guide substitutions, allowing only those with high similarity scores to ensure evolutionary and structural compatibility, as BLOSUM scores reflect **how often one amino acid is naturally replaced by another** in related proteins, making it a reliable constraint for biologically plausible mutations.



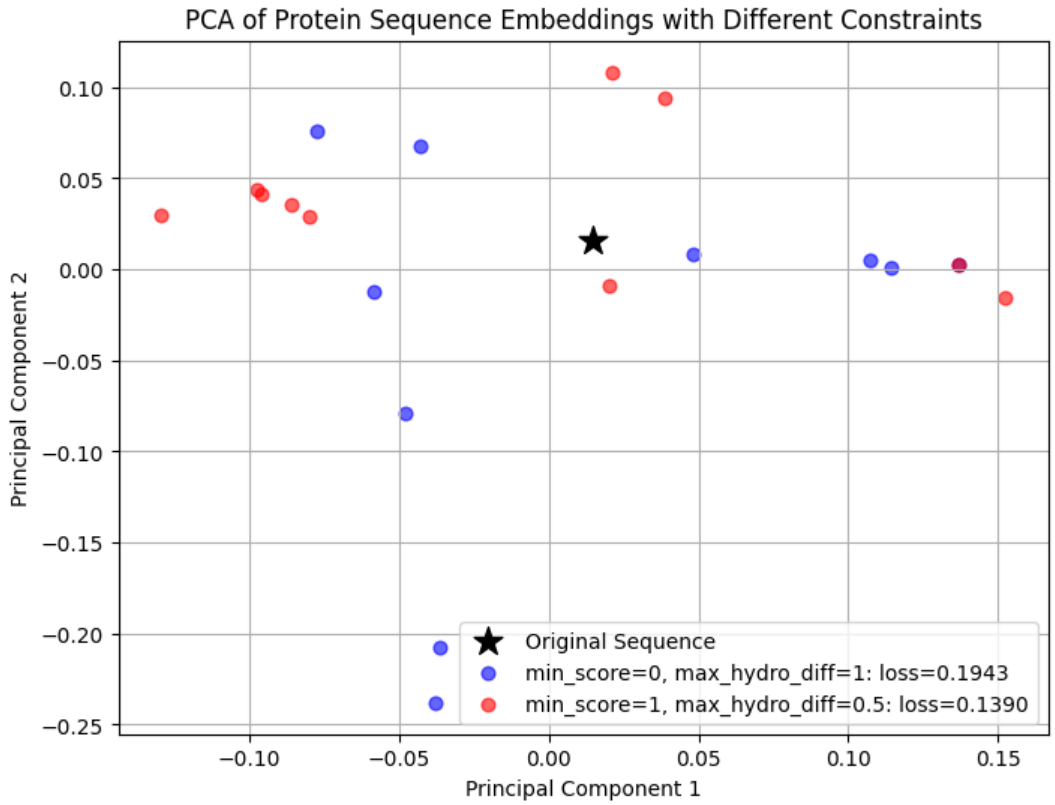
2. **Hydrophobicity Constraint**
   We limited the hydrophobicity difference between original and mutated residues to avoid disrupting the protein's physicochemical balance, since changes in **hydrophobicity can affect how the protein folds and interacts with water**, potentially destabilizing its 3D structure.

	Amino Acid	Hydrophobicity
0	A	1.8
1	B	-3.5
2	C	2.5
3	D	-3.5
4	E	-3.5
5	F	2.8
6	G	0.4
7	H	-3.2
8	I	4.5
9	K	-3.9
10	L	3.8
11	M	1.9
12	N	-3.5
13	P	-1.6
14	Q	-3.5
15	R	-4.5
16	S	-0.8
17	T	-0.7
18	V	4.2
19	W	-0.9
20	X	0.0
21	Y	-1.3
22	Z	-3.5

This approach produces realistic sequence variants that are likely to retain similar Tm values and structural characteristics.



As shown in the PCA plot, **augmented sequences have embeddings similar to their original counterparts**, indicating that the augmentation preserves key structural features.



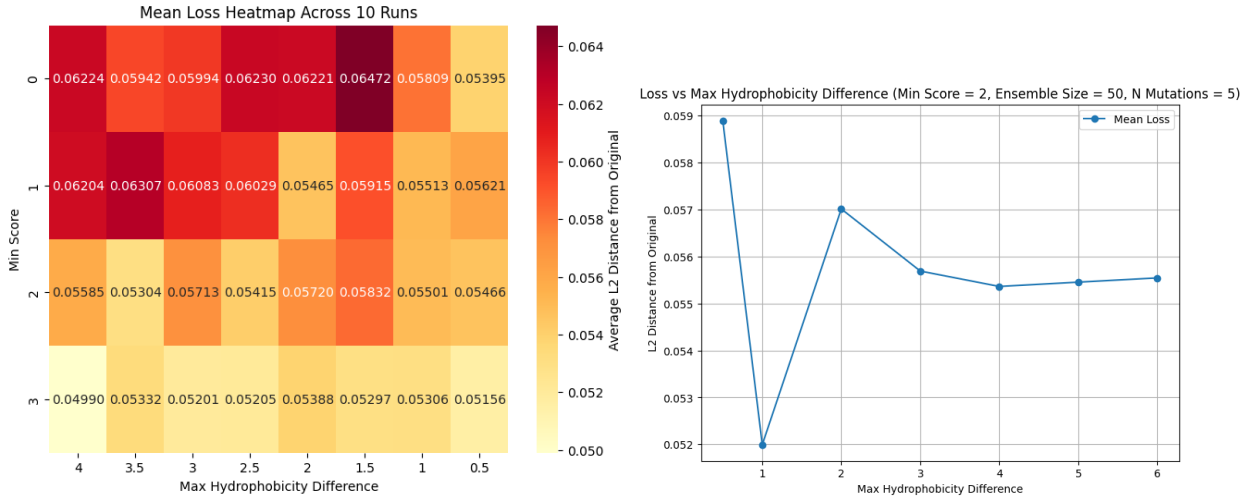
Proteins augmented with stricter constraints — such as higher **BLOSUM** scores and lower **hydrophobicity** differences — have embeddings that are closer to the original sequence. This indicates better preservation of the sequence's structural and semantic meaning.
The **loss** shown in the legend reflects the Euclidean distance between the original and augmented sequence embeddings.

Fine Tuning Constrains

By measuring the **L2 loss** between original and augmented embeddings, we can tune the constraints for optimal similarity.
The lowest loss was achieved with:

- **min_score = 0**
- **max_hydrophobicity_difference = 1.0**

making it the best setting for preserving structural meaning.



Train Test Split

All Features

- Basic features: ASA, pH
- Secondary structure: Coil, Helix, Sheet, Turn
- Stability metric: Tm_(C)
- Protein embeddings: 320 dimensions (embed_0 to embed_319)

Model	X	Y	Train Sample Count	Train Sample Count Augmented
CatBoost	Sec-Str, ASA, pH	T_m	37169	1151904

$$X_i = [ASA\ pH\ Coil\ Helix\ Sheet\ Turn\ 320\ embeddings \ldots]_{\times 326}$$

$$y = T_m$$

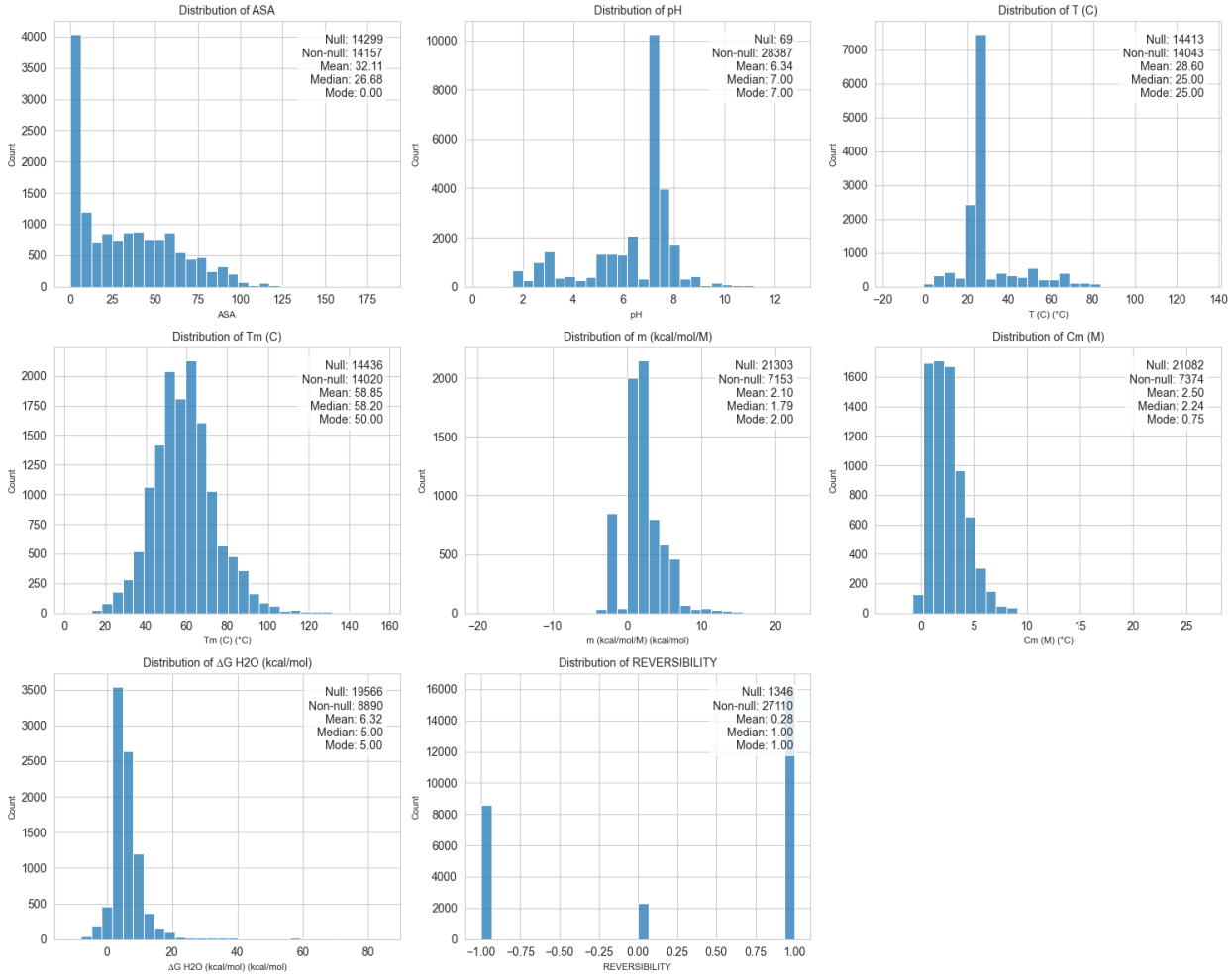
Split ratio

- Training set: 68%
- Val set: 17%
- Test set: 15%

Basic Statistical

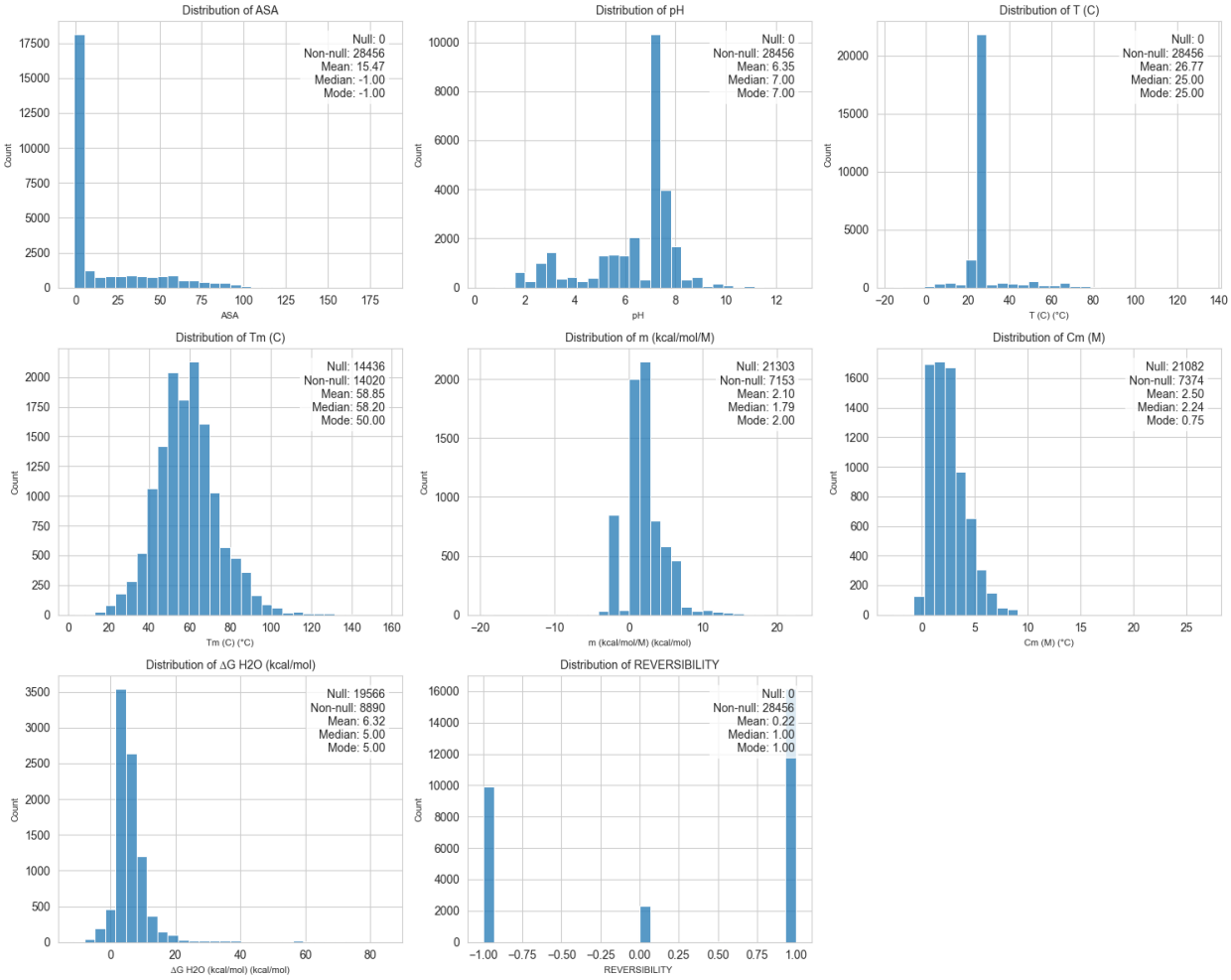
Histograms

Histograms of features are shown before and after null handling:



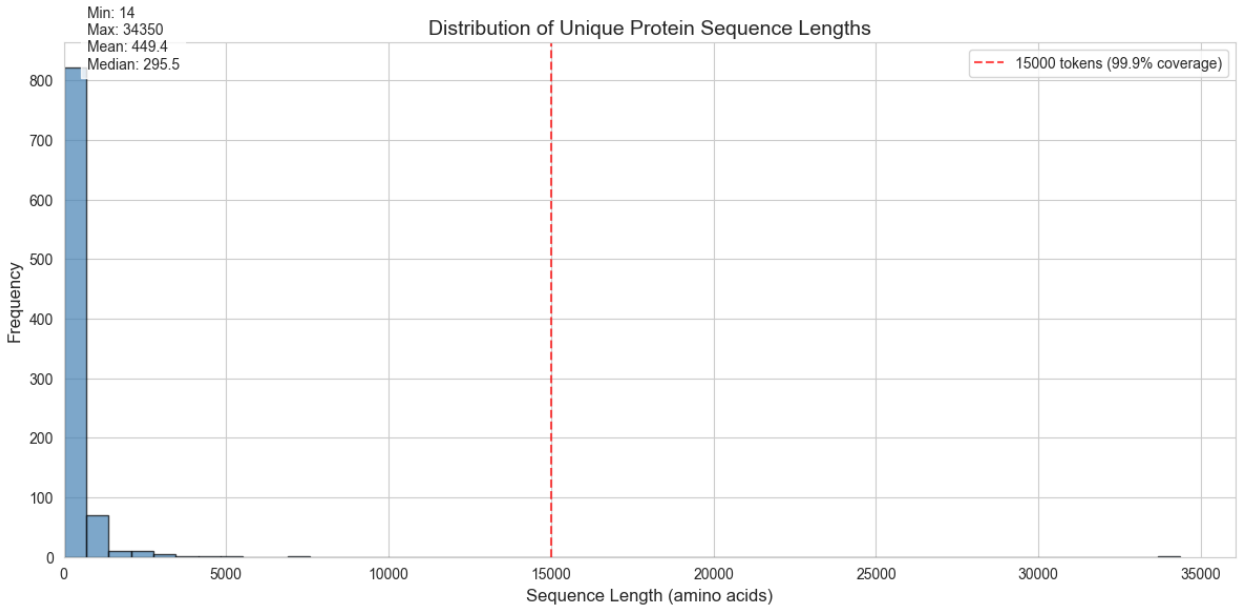
features before null handling

PH and T values are mostly neutral and room temperature.

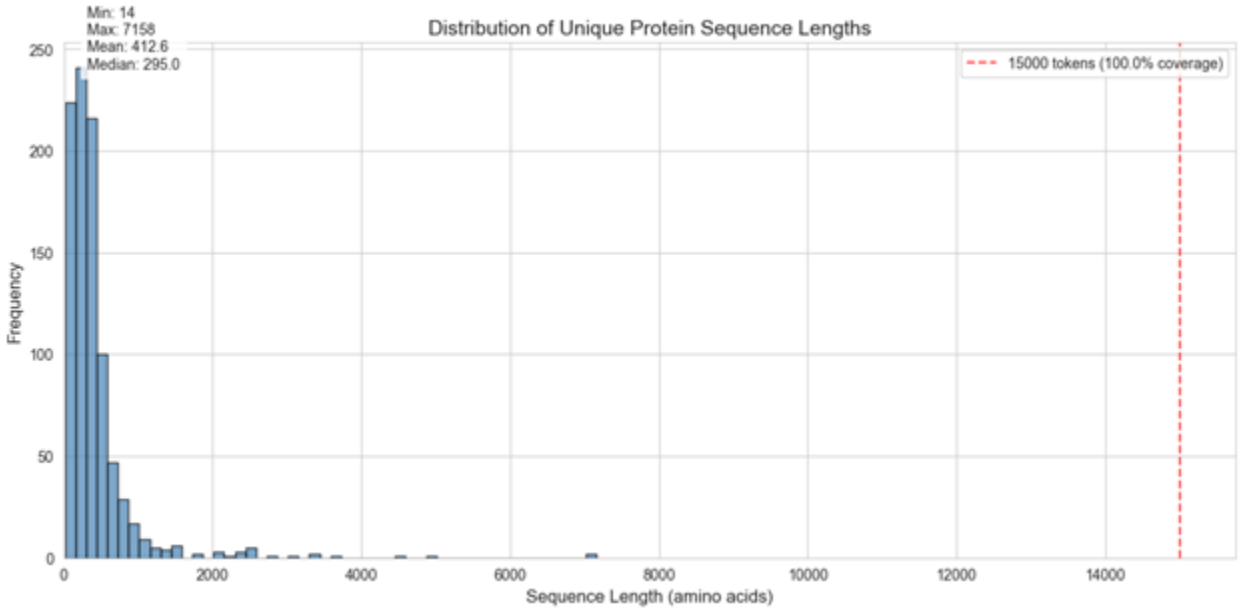


features after null handling

Sequence length distribution before and after removing the long sequence:



length distribution before removing.

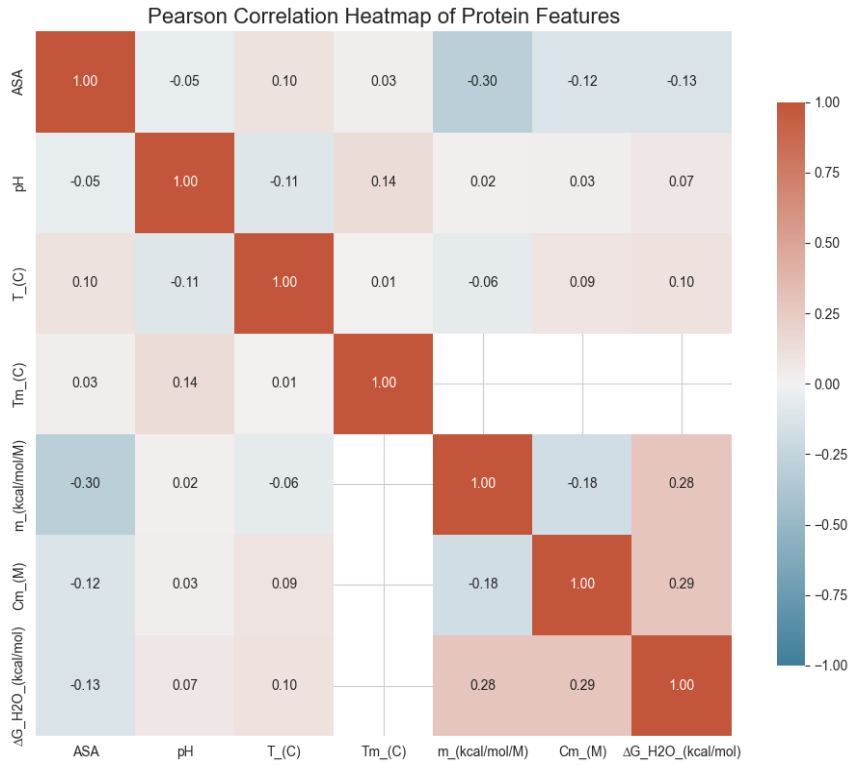


Length distribution after removing.

The removal beside helping runtime in embedding process, Also made distribution more standard.

Correlation Functions

Pearson correlation is used to measure **linear** relationships between variables, while **Spearman** correlation is used to assess **monotonic** (rank-based) relationships.



Pearson heatmap

Strongest Correlations:

- m and ΔG_H2O: **0.28**
- Cm and ΔG_H2O: **0.29**

➢These are both moderately positive correlations and make sense physically: free energy of unfolding in water is often positively correlated with denaturant resistance (m-value and Cm).

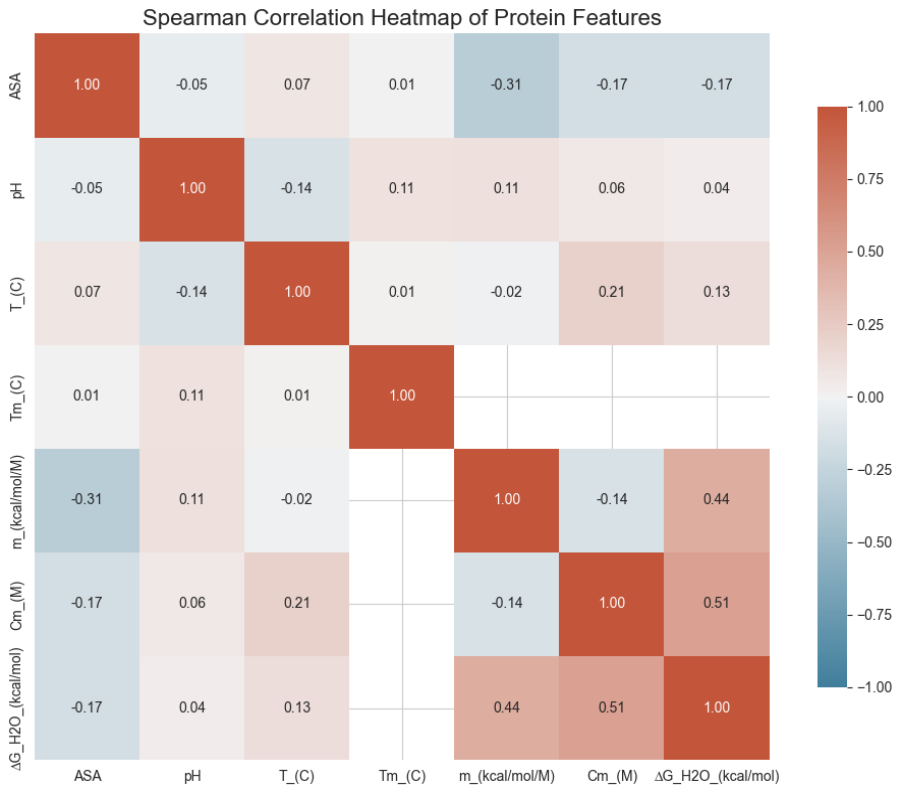
Notable Negative Correlations:

- ASA and m: **-0.30**

➤Suggests that proteins with larger surface areas tend to have lower m-values (possibly related to compactness or stability).

Minimal or No Correlations:

- pH shows weak correlation with everything.
- T_m and most other features are very weakly correlated (almost all values near 0).



Spearman correlation

Same correlations but stronger can be seen in Spearman heatmap. The correlations are probably **monotonic**.

Traditional Techniques

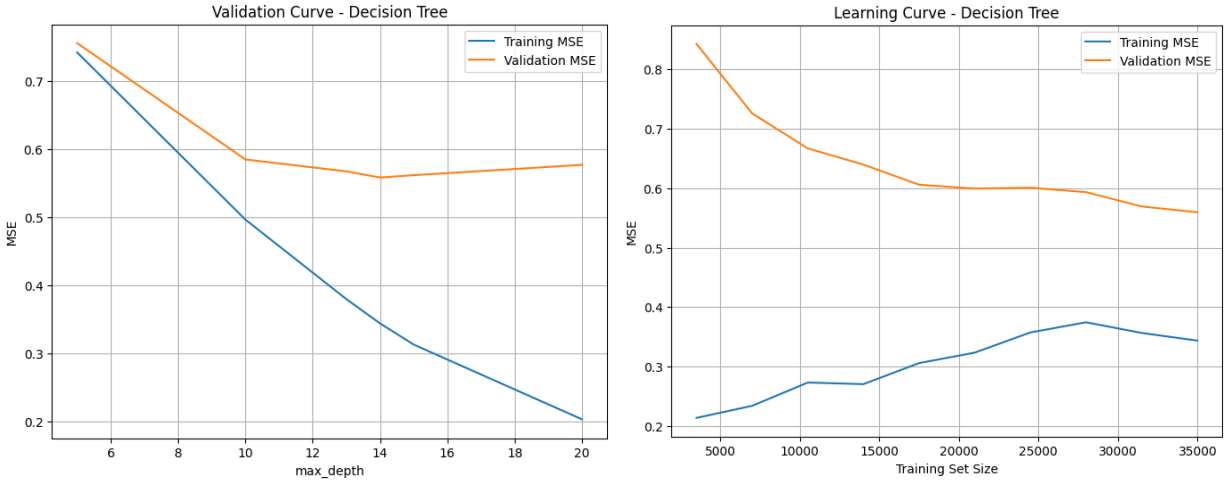
After processing the data, we applied several traditional regression models to the dataset, including **Decision Tree Regressor, Random Forest Regressor, Support Vector Regressor (SVR),Gradient Boosting and Cat Boost (optimized GBR)**. To further evaluate

and compare their performance, we plotted both **learning curves** and **validation curves** for each model.

We use **MSE** as loss and validation metric, because we want to avoid few big misses than shaving off many tiny ones.

Note that in this section we just train one model on **Tm** and not other 3 models on other properties.

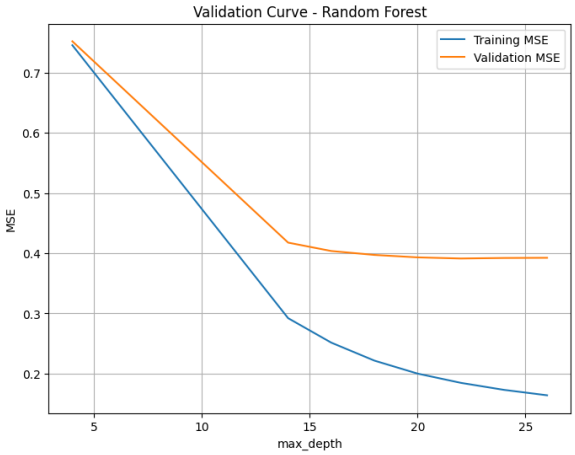
Decision Tree

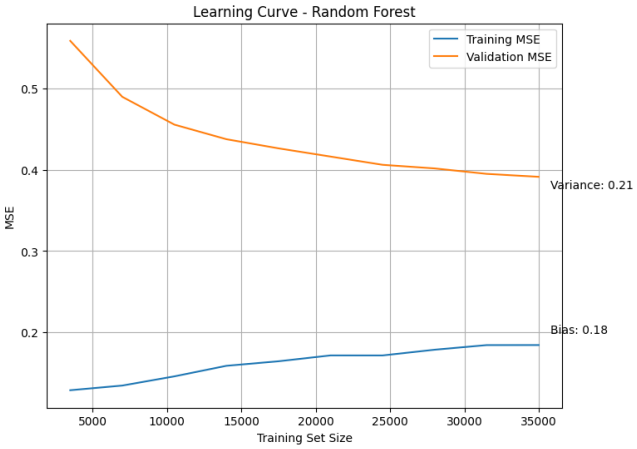


With validation curve we can finetune the model. There is some sign of overfitting after **depth 14**.

Best model: max_depth = 14

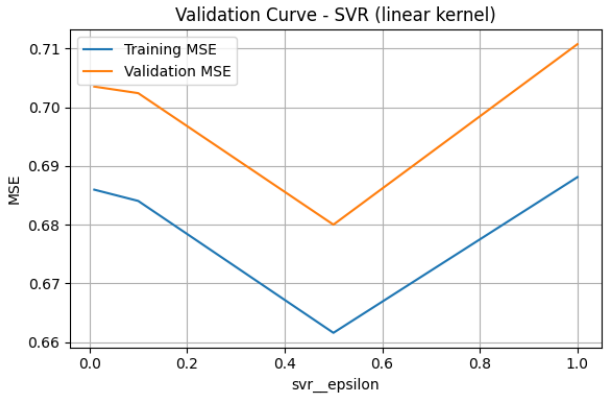
Random Forest

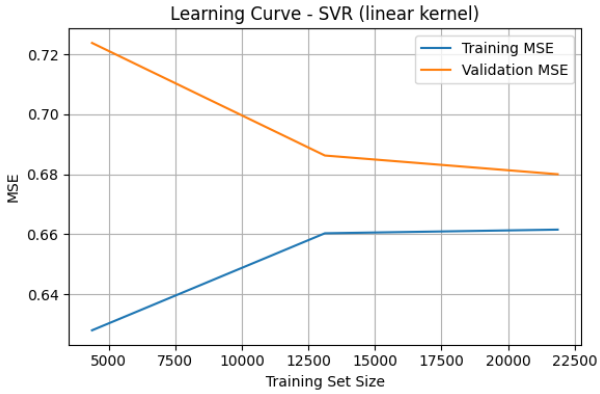




Best model: max_depth = 22

SVR

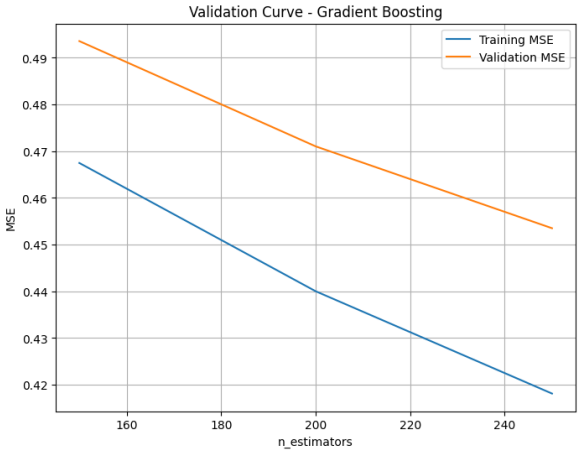


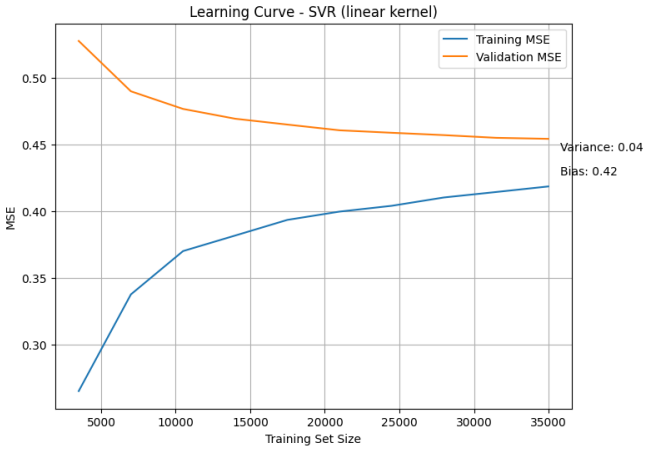


Bias Estimate (Final Training MSE): 0.66

Variance Estimate (Gap): 0.02

Gradient Boosting



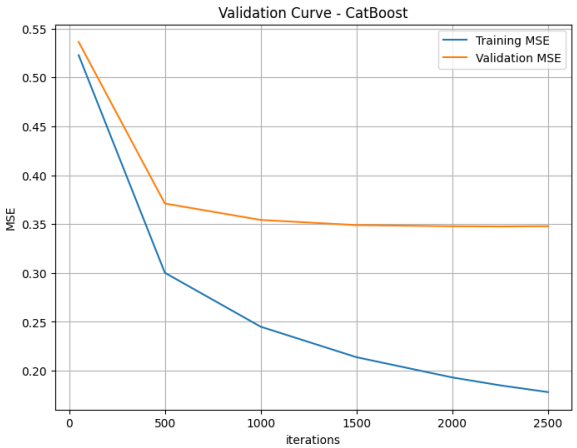


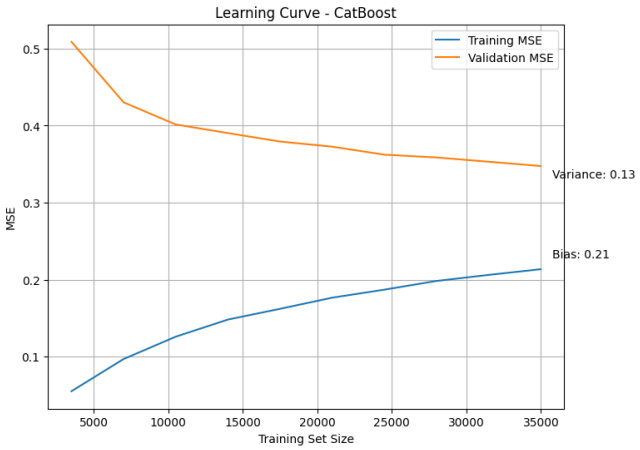
The Gradient Boosting Regressor (GBR) demonstrated slightly better performance and **lower variance** compared to the other models. However, we were unable to fine-tune it extensively due to its **high computational cost** and long run time.

Best model: n_estimators = 250

Cat Boosting (optimized GBR)

Since Gradient Boosting performed better than other models, but we were limited by computational cost, we decided to use a more optimized version of Gradient Boosting, Cat Boost.





Best model: iterations = 1500

As we can see, most models are performing poorly and struggling to bring down the loss and closing the learning curve gap due to **lack of data**.

The chart below summarizes and compares the performance of fine-tuned models.

Model Name	Loss (MSE)	Bias	Variance	Training Time (s)	Prediction Time (s)
Decision Tree	0.3910	0.34	0.21	4.3135	0.0077
Random Forest	0.3245	0.18	0.21	34.1808	0.0138
SVR	0.6985	0.66	0.02	40.7070	6.6785
Gradient Boosting	0.3472	0.42	0.04	414.6182	0.0248
CatBoost	0.3045	0.21	0.13	12.3143	0.0180

Among the evaluated models, **CatBoost** achieved the lowest MSE (0.3045), indicating the highest predictive accuracy, with a balanced bias-variance profile and moderate training time. **Gradient Boosting** also performed well but required significantly longer training (414.6s). **SVR** showed the highest error and bias despite a low variance, while **Decision Tree** and **Random Forest** offered reasonable trade-offs between performance and computational cost. Overall, **CatBoost** stands out as the most efficient and accurate model.

Neural Network

The model is implemented using a relatively deep neural network with multiple Fully Connected layers, Batch Normalization, Leaky ReLU activations, and Dropout.

Fully Connected layers are used along with Batch Normalization and Dropout.

Leaky ReLU is used as an activation function.

Adam is chosen as the optimizer and MSE Loss is used as the loss function.

Early stopping is implemented to prevent overfitting.

The training loss decreases steadily, indicating good learning on the training data.

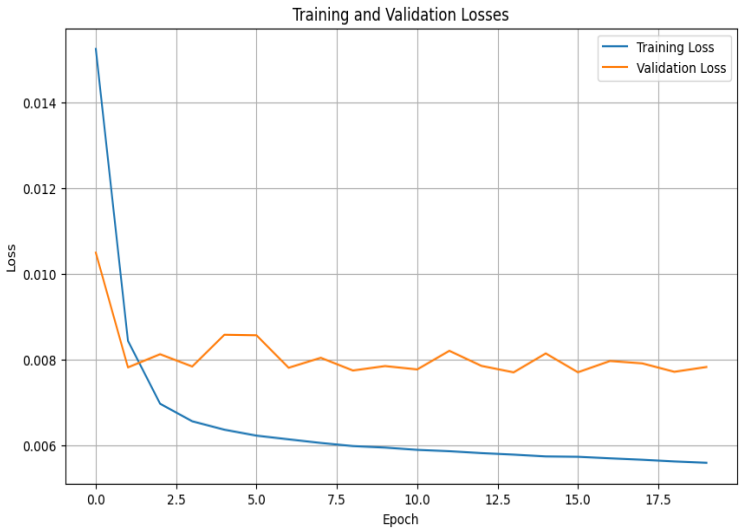
The validation loss initially drops and then remains relatively stable, showing that the model generalizes fairly well to unseen data.

Fluctuations in the validation loss are likely due to the small size of the validation set.

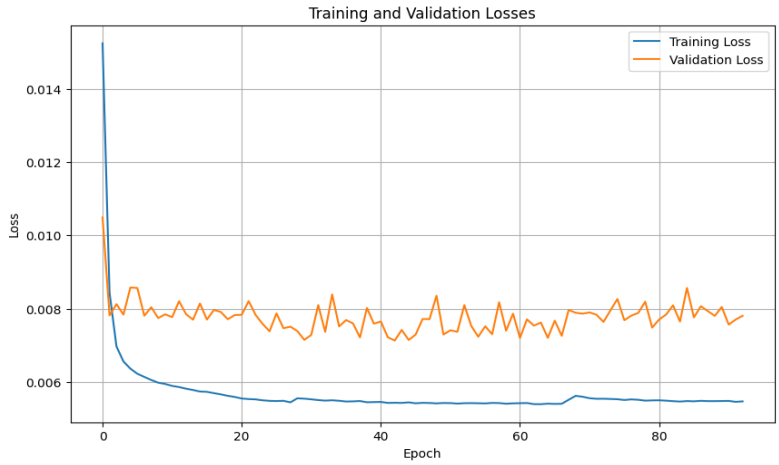
However, there is a noticeable gap between training and validation loss, indicating overfitting.

Techniques like Dropout and Weight Decay have helped control overfitting to some extent.

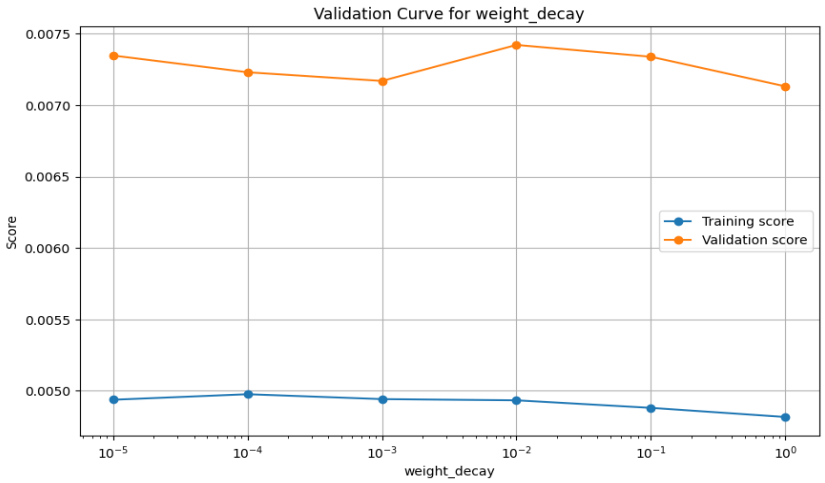
The model's learning curve shows instability, suggesting issues in generalization despite regularization.



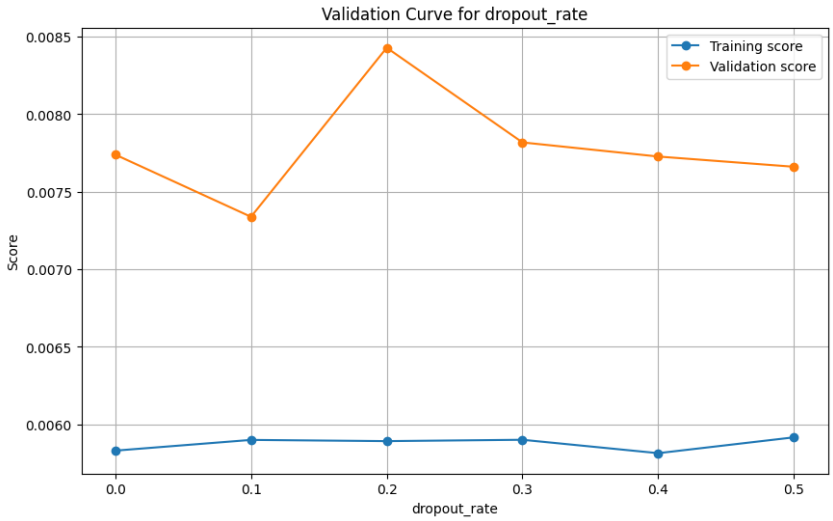
Training loss steadily decreases over time. These fluctuations in the validation loss are due to the small size of the validation dataset.



The blue curve steadily decreases and approaches a constant value around the 20th epoch. This behavior indicates that the model has effectively learned the training data, and the training process has been efficient. The orange curve, representing the validation loss, initially decreases and then oscillates around a relatively constant value. These fluctuations are relatively significant, especially compared to the stability of the training loss. Although there is a difference between the training and validation losses, the validation loss generally remains at a similar level without a sustained increase.



Optimal range: 10^-3 and 10^-2 provides the best balance between learning and generalization.

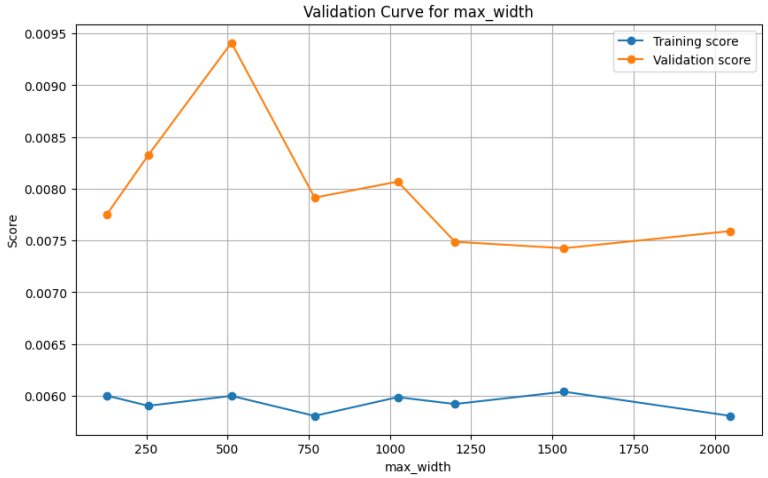


Dropout=0.1 achieves:

- 25% validation error reduction vs no dropout

- Only 18% training error increase

- Effectively balances learning capacity and regularization



The best performance is achieved at max_width = 1250, with:

Training error: ~0.0065

Validation error: ~0.0070

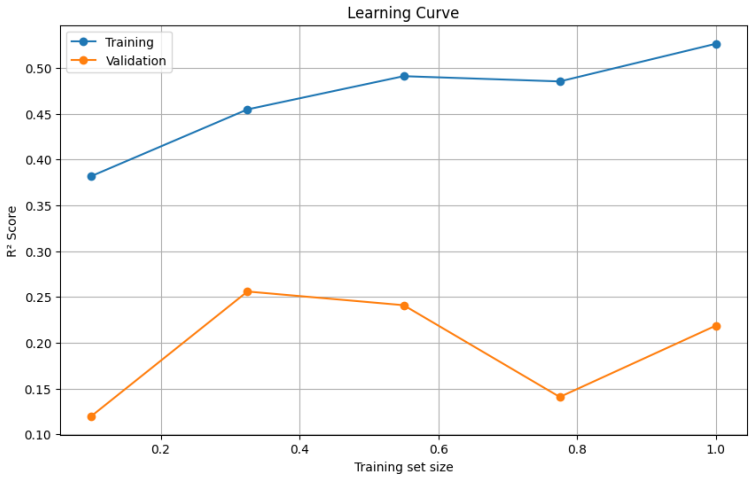
Minimal overfitting (error gap ~0.0005)

Behavior:

< 750: Underfitting (high errors)

1000–1500: Optimal range

> 1750: Slight overfitting, no performance gain

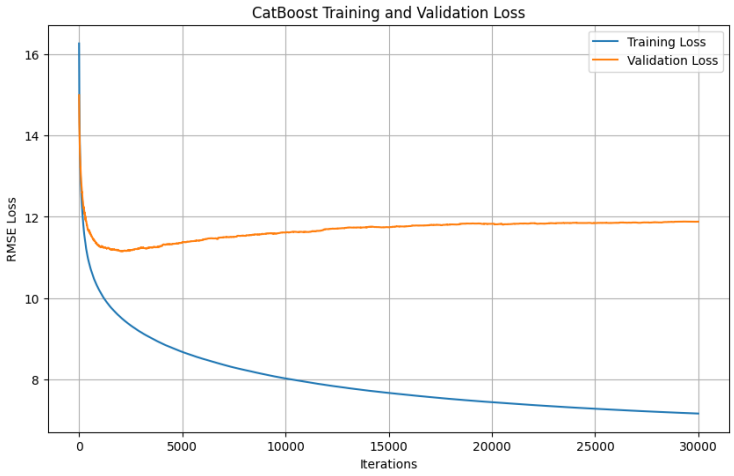


This curve indicates that the current model is not optimal and demonstrates that the model is failing.

Final Model

We used a **fine-tuned CatBoostRegressor** model with the following settings:

- **learning_rate**: 0.1
- **depth**: 3
- **max_bin**: 32
- **l2_leaf_reg**: 10
- **bootstrap_type**: "Bayesian"
- **min_data_in_leaf**: 200
- **random_strength**: 1.0
- **bagging_temperature**: 1.0



Training R2 Score: 0.6653

Validation R2 Score: 0.4549

Test R2 Score: 0.5124

Training Spearman's rho: 0.7483

Validation Spearman's rho: 0.5935

Test Spearman's rho: 0.7232

Although the model achieved only a **moderate R^2 score** on the **test set** (0.5124), the primary evaluation metric in the related competition was **Spearman's rank correlation**, which focuses on the **relative ordering** of Tm values.

The model is **not highly accurate in predicting the exact Tm values**, but it performs well in **ranking sequences by stability**. which is exactly what we need for our goal of identifying and selecting more stable protein variants.

Stability Improvement Tool (Phase 2)

We used the augmentation technique to generate **single-point mutated sequences** based on an input protein. These variants were then passed through our trained model to **predict their stability**. By selecting the sequence with the **highest predicted stability**, we developed a simple tool for **stability improvement**, enabling users to optimize protein sequences for enhanced thermal stability.

Sources

Phase 1: Predicting Thermodynamic Properties

1. Rapid protein stability prediction using deep learning representations

Link

This study introduces the **RASP** method, which uses deep learning embeddings to rapidly predict protein stability. It is relevant to our approach in phase 1, where similar embeddings (like those from ESM2) are used to predict thermodynamic properties.

2. ProSTAGE: Predicting Effects of Mutations on Protein Stability by Fusing Structure and Sequence Embedding

Wiley Online Library

This paper proposes the **ProSTAGE** model that combines structural and sequence-based features for predicting protein stability. The fusion strategy aligns with our embedding + feature-based models.

3. Prediction of protein stability changes for single-site mutations using machine learning

Link

This work applies classical machine learning to stability prediction based on sequence changes. It provides benchmark techniques and evaluation methods useful for validating our phase 1 models.

4. ESM: Evolutionary-scale modeling for protein structure and function

Link

This is the foundational paper for the **ESM** model that we use to create protein embeddings. It demonstrates the power of large-scale language models in protein structure/function understanding.

Phase 2: Designing Protein Sequences

1. ProteinGAN: A generative model of protein sequences

NeurIPS Paper

This early paper introduces **ProteinGAN**, a GAN-based model trained on real protein sequences. Its framework and evaluation are valuable references for our generative setup in phase 2.

2. Generative adversarial networks for de novo protein design

🖇 Link

This article explores how GANs can generate new, **biologically functional** proteins. The emphasis on structure-function alignment mirrors our second-phase objective of designing stable, functional proteins.

3. Conditional Generative Models for Protein Design

🖇 Link

This paper details **conditional generation**, where specific properties are used to guide sequence generation—directly supporting our phase 2 strategy of property-conditioned sequence design.

4. Protein sequence design by conformational landscape optimization

🖇 PNAS Article

This study applies deep learning and optimization techniques to engineer protein sequences with desired **thermodynamic landscapes**. It parallels our goal of optimizing for thermal stability and other properties.