LUNDS UNIVERSITET

Extracting protein flexibility features from a pre-trained protein language model

Vera Karlin

Master's Degree Project in Biochemistry, 60 credits, 2023.

Department: Department of Chemistry

Supervisors: Sofia Andersson and Ingemar André

Abstract

The deep learning revolution has contributed to a great leap in protein structure prediction, but predicting the multiple conformational states that proteins can hold remains an open problem. This thesis approaches this problem through the use of MSA Transformer, a protein language model pre-trained on multiple sequence alignments, that has demonstrated the ability to learn protein properties such as contacts and secondary structure. The aim was to investigate if MSA Transformer has learned to represent flexibility features of proteins during pre-training and if that information can be captured somehow. This mainly took the shape of training neural networks on the outputs of the transformer model to predict the local RMSD flexibility metric of flexible proteins in the PDBFlex database. None of the attempted networks succeeded with classifying flexibility, most likely because of problems with the chosen architectures, evaluation metric and dataset. Much more promising was the discovery of patterns in the attention maps of MSA Transformer which displayed correlation to the local RMSD metric. The findings were expanded upon by the identification of certain attention heads that seemingly correlated with specific types of flexibility, but further investigation is required to draw any conclusions. Prior studies have shown that multiple sequence alignments can be clustered and modified in order to sample multiple conformations with AlphaFold2. While not attempted in this project, the groundwork has been laid for expanding on these approaches with the use of MSA Transformer features to select residues and sequences of specific significance.

Acronyms

AF2	AlphaFold 2
ASA	Accessible Surface Area
CLM	Causal Language Modeling
CNN	Convolutional Neural Network
IDR	Intrinsically Disordered Region
lDDT	Local Distance Difference Test
LSTM	Long Short-Term Memory
ML	Machine Learning
MLM	Masked Language Modeling
MLP	Multilayer Perceptron
MSA	Multiple Sequence Alignment
RMSD	Root Mean Square Deviation
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent

Table of contents

1. Introduction

The field of structural biology has in recent years undergone a large shift as the rapid developments in deep learning have been applied with great success to the protein folding problem; the process of computationally predicting the three-dimensional structure of a protein, solely from its sequence of amino acid residues. While the feats of structure prediction models like AlphaFold2 are without a doubt very impressive and transformative for the field, multiple subproblems with regards to protein folding still remain, perhaps most significantly the inability to capture the conformational flexibility that is inherent to most proteins (Lane, 2023). The truth is that the ability to dynamically assume a range of conformational states is the key behind many important protein functions, and accurately predicting this range or at least multiple significant conformations would be of great benefit to a wide variety of applications in biochemistry and medicine.

1.1 Aims of the Study

In this report, I aim to approach the problem of predicting structural flexibility of proteins from two directions. Firstly, through the training of deep neural networks to directly predict the flexibility of protein residues as described by metrics such as local RMSD. Secondly, I aim to sample multiple structural conformations with the structure prediction model AlphaFold2 by inputting clustered multiple sequence alignments, as has been demonstrated in prior studies (del Alamo et al., 2022; H. K. Wayment-Steele et al., 2022).

The common denominator between the two approaches is the use of the protein language model MSA Transformer, which has been shown to emergently learn properties about proteins such as contacts, secondary structure and intrinsically disordered regions (IDRs) from training on millions of multiple sequence alignments. The study is carried out on the basis of the hypothesis that MSA Transformer might have learnt to identify conformational flexibility in order to better perform its objective, and that it could be possible to extract this information from it.

2. Background

2.1 Coevolution

In spite of a staggering variety of possible protein amino acid sequences, substituting mutations are ultimately constrained by a residue's surroundings – a random mutation often detracts from a protein's stability and function. This has the effect of strongly benefitting simultaneous and complementary mutations of residues that are in close proximity in order to preserve the structure of the protein. These kinds of residue pairs are known as coevolving residue pairs. However, direct interaction by spatial proximity is not the only way in which substitutions are constrained – indirect interactions can be mediated by intermediate molecules or arise from a chain of directly interacting residue pairs (Burger & Nimwegen, 2010; Lockless & Ranganathan, 1999).

A common way to visualize and identify coevolving residues is by studying multiple sequence alignments (MSAs) since conserved amino acids will often be vertically aligned with each other. The implicit information that covariance between residues convey is commonly referred to as coevolutionary signal. This connection between coevolutionary information and structural contacts has long been utilized for structure prediction, commonly by Potts models. Potts models are statistical models that capture coevolutionary information from individual and pairwise frequencies in an input MSA (Ekeberg et al., 2013). From this, the models use average-product correction to promote the direct couplings from the indirect ones and generate contact maps which can be used for structure prediction. Potts models have in recent years met a challenger at the task of capturing information from evolutionary data – deep learning.

2.2 Deep Learning

2.2.1 Deep Neural Networks

Weights and biases

Deep learning is a subfield of machine learning that takes inspiration from biological brains in that it utilizes large networks of neurons with connections of different strengths between them. The strength of the connections are governed by the model's parameters: weights and biases. Weights determine how much the activation of the node at the start of a connection should affect the node at the end of it, while biases determine the threshold that a node will have to pass to give an activation.

In the simplest kind of neural network, a fully connected network, the nodes are arranged in layers where each node in a layer is connected to all of the nodes in the subsequent layer. While networks can consist of just an input and output layer, what characterizes deep neural networks is the presence of one or more hidden layers in between them. Multi-Layer Perceptrons (MLPs) are an example of fully connected deep neural networks (Fig.1a). In MLPs, activations are calculated through performing a matrix multiplication between the input vector **x** and the matrix **w** that contains the weights for that layer. The sum of the resulting vector and the bias vector **b** are then put into an activation function to give an output vector **a** of activations (Fig.1b). There are many different activation functions in use, with ReLU, Sigmoid and TanH being some of the most common ones (Fig.1c).

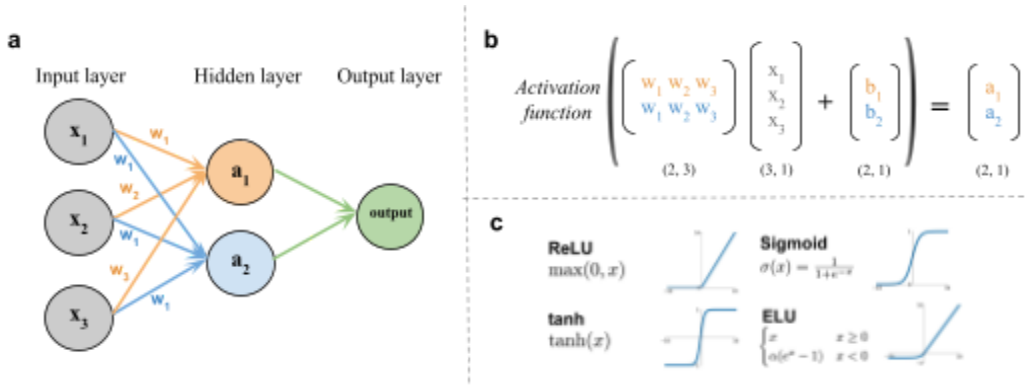

Figure 1. (a) Visualization of a simple MLP with a single hidden layer. (b) Equation for calculating the hidden layer activations from the weight matrix and input and bias vectors. The numbers under each vector/matrix represent its dimensions. (c) Examples of activation functions. Image source: https://images.app.goo.gl/ne28J1ZwJXWjHoHm9

Backpropagation

A way of looking at neural networks is through seeing the weights and biases of a model as implementing an algorithm that transforms some given input to an output. The training process thus becomes an optimization problem of finding the set of parameters which implement the best algorithm for completing a given task. Different learning tasks can be broadly divided into two types – supervised and unsupervised learning. In supervised learning there is a specific desired output label for each input, which can vary greatly based on the task. In for example classification tasks the output is one out of two or more discrete classes, while regression tasks involve the prediction of a continuous value. Unsupervised learning lacks labels and is instead used for finding patterns in unlabeled data, which can for instance help with clustering tasks.

Gradient descent through backpropagation is the standard methodology for training neural networks. It operates through iteratively updating each parameter of a model in a direction so they better implement an algorithm that can predict the training data labels. This starts with a forward pass of the network to give a prediction $\hat{\mathbf{y}}$ based on its input data $\mathbf{x}$. This prediction is compared to the expected output $\mathbf{y}$ with a loss function, which gives a value for how accurate the prediction was.

The goal of the network is to minimize loss, i.e. prediction error, which the backpropagation algorithm does by calculating a gradient for how every parameter in the network should change in order for the loss to be smaller for a given training example. This is done through a backward pass which utilizes the chain rule of calculus to calculate the derivative of each parameter with respect to the full objective function $\mathbf{J}$. The parameters are then updated according to the calculated gradient with a magnitude governed by the learning rate $\boldsymbol{\alpha}$ (Eq.1-2). This process is repeated for many training examples, with each pass over the training dataset being referred to as an epoch, until the network has hopefully learnt to generalize.

$$w := w - \alpha \frac{dJ(w,b)}{dw} \tag{1}$$

$$b := b - \alpha \frac{dJ(w,b)}{db} \tag{2}$$

Hyperparameters

Learning rate is an example of a hyperparameter – a set of parameters that govern the training process and are usually kept the same for an entire training run. They can be everything from the number of layers of a network, to how many epochs training should last or the extent a certain optimization technique should be employed . Hyperparameters are tuned either manually or through algorithms to provide the best possible conditions for learning.

Optimization

A model that is not able to predict training labels is considered to be underfitted. Underfitting can be reduced by training for longer, increasing the amount of trainable parameters, or through utilizing some of the optimization techniques that have been developed for effectivizing the traversal of parameter space.

Some optimization techniques make use of the vector-based nature of deep neural networks to parallelize the training process. In batch gradient descent the input data is parallelised by stacking all training data pairs into a batch in the form of a tensor from which a single gradient can be calculated based on the combined loss of each training example in it. This fully utilizes vectorization and takes large steps along the gradient at the cost of high memory usage and very long epochs, thus making it more suitable for smaller datasets (Fig.2).

The dataset can alternatively be split up into multiple mini-batches, where each mini-batch contains a number of examples equal to the batch size hyperparameter. Stochastic gradient descent (SGD) uses a batch size of one, meaning that the weights will be updated after each example, producing fast iterations of very noisy steps that will never fully converge with a minima (Fig.2). Keeping the mini-batch size somewhere in between usually hits the “sweet spot” by taking advantage of batches while avoiding some of their downsides (Fig.2).

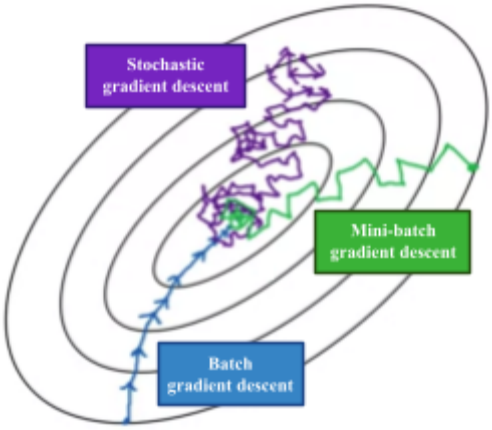

Figure 2. A simplified 2D depiction of how three kinds of optimizers traverse the parameter gradient: Stochastic gradient descent (purple), Mini-batch gradient descent (green) and Batch gradient descent (blue).

Momentum is an optimization technique designed to accelerate learning by accumulating an exponentially decaying moving average of previous gradients and updating the parameters in that direction (Ian Goodfellow et al., 2016). This smooths out the gradient and compensates for the large parameter oscillations which can arise from an increased learning rate. RMSprop is another optimizer which expedites convergence by making the learning rate adaptive. It does this through dividing the update to each parameter by the square root of an accumulated squared gradient. As a result, the learning rate of highly fluctuating parameters gets reduced, while the learning of parameters with smaller updates gets accelerated. The ideas of momentum and RMSprop are combined in the Adam optimization algorithm, which has gained widespread adoption thanks to its outstanding performance.

Overfitting

When training a network, it is essential to keep the training and testing data separated through splitting up the dataset into a training set and a testing set. The goal is to fit the model to the training data by reducing the training error, but the performance on the test set will not always follow. It is common for models to "overfit" to the training data, meaning that it starts picking up idiosyncrasies and features in the training set that are not representative of real-world patterns instead of learning to properly generalize from the data (Maini, 2018). Overfitting occurs when a gap between training and test error arises (Fig.3) and is commonly reduced by increasing the number of training examples or employing some of the many regularization techniques that have been developed to combat overfitting.

Figure 3: Example plots of (a) overfitting and (b) underfitting, showing how training loss and test loss develop over time during training.

Regularization

One of the most common regularization techniques is L^2 regularization, or "weight decay", which is implemented by adding a regularization term $\frac{1}{2}||w||_2^2$ to the objective function J (Ian Goodfellow et al., 2016). This punishes large weights, which incentivizes the network to be simpler. Dropout is another commonly employed technique where random units in a neural network are temporarily removed during training according to a set dropout probability (Srivastava et al., 2014). This makes the networks more robust and less reliant on specific weights, which helps reduce overfitting. Both weight decay and dropout are regulated by hyperparameters.

When trying to reduce overfitting to the test set, it can be possible to make the model overfit to the test data, which is why a third validation (or "dev") set is often employed. This makes the actual testing data less known to the model and thus a better representation of the real world.

Normalization

Despite the assistance of optimization techniques, traversing the search space can remain challenging for cases of multidimensional data with varying scales. This is why input data is often normalized by setting the mean to zero and changing the variance to be equal to one. Alternative normalization techniques include layer normalization, which normalizes the features within each example vector, and batch normalization that normalizes each feature across all examples in a batch collectively.

Batch normalization can also be used to prevent vanishing gradients – a problem which may arise in networks with many layers, where a small decrease in activation per layer leads to an exponentially decreasing gradient. Batch normalization prevents this by ensuring that each layer has consistent means and variances.

2.2.2 Convolutional Models

Convolutional network structure

Convolutional neural networks, CNNs for short, are a type of architecture developed for image classification by mimicking the human visual cortex (Raschka et al., 2022). They operate by identifying simple features such as edges or corners in input images and using them to identify more complex features like shapes or even faces in later layers. The convolutional layers are usually followed by a series of fully connected MLP layers and a Softmax layer. Softmax turns the activations, referred to as logits, into a distribution of probabilities over the output classes. In CNNs, the class with the highest probability is picked, but for other architectures the sampling could be more varied.

Convolutional layers

The fundamental operation of a convolutional layer involves the sliding of a filter matrix across an input image matrix. At each step of movement, the pairwise products of the overlapping numbers are summed and added to the corresponding element in the output matrix (Fig.4a). The length of the steps is determined by the stride hyperparameter (Fig.4b); a short stride gives a large output matrix while a long one makes it smaller. The dimensions of the output matrix can also be adjusted through padding, which involves adding a border of pixels, typically set to 0, around the input with a specified thickness (Fig.4c).

Typically, each layer consists of multiple filters that share the same dimensions but have distinct weights. Each filter operates independently from the others and generates individual output matrices called channels, which get concatenated into a tensor. When applying convolution to a multi-channel tensor, the filter has to have the same channel count as the input, where each channel of the filter possesses a unique set of weights. The resulting products for each of the channels are then summed to yield a single output element per position and filter (Fig.4d).

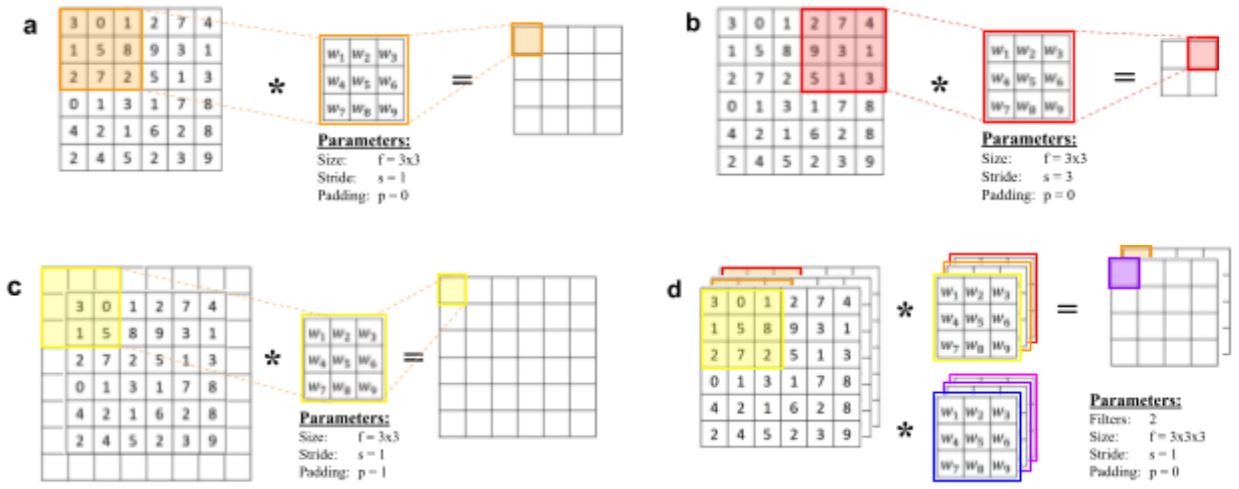

Figure 4. (a) Example of a simple convolutional layer with a single 3x3 filter. (b) Variation of the first setup but with a stride of 3. (c) Variation of the first setup but with a padding of 1. (d) Variation of the first setup but with three input channels and two 3x3x3 filters.

Another staple of CNNs are pooling layers that break the input matrix into segments and output a matrix of either the average or maximum activation for each segment. This reduces the sensitivity to small input variations and increases computational effectiveness by reducing the size of features (Raschka et al., 2022). 1x1 convolutional layers on the other hand are a type of layer that can be used when aiming towards reducing the channel count. These layers slide 1x1 filters across the input and subject the dot products to an activation function, giving them a similar behavior to MLPs.

2.2.3 Sequence Models

Recurrent neural networks

Recurrent neural networks (RNNs) are a type of architecture specifically designed to process sequential data, such as text, sound, or in our case, properties of protein sequences. In the case of data consisting of tokens such as words or amino acids, the tokens will have to be converted into vectors known as embeddings in order for them to be properly processed by the network. Each embedding represents a specific token, so output embeddings can then be translated back to token form. However, it is also possible to let the network learn different ways of embedding and de-embedding the tokens.

In their most basic form, RNNs consist of a series of connected hidden units between an output and input layer. The purpose of these hidden units is to act as a form of memory, where activation values from each time-step are carried over to the next. Each cell of a basic RNN shares the same three parameters W_{hx}, W_{yh} and W_{hh} (Fig.5). It is also possible to stack multiple hidden layers to form a multilayer RNN (Raschka et al., 2022).

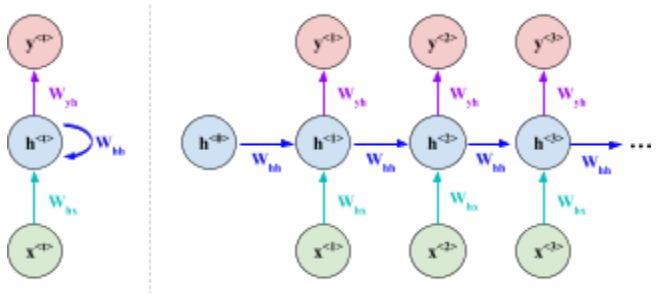

Figure 5. Compressed structure of a single-layer RNN (left) and an expanded structure of the same network (right). $x^{<t>}$ are the input elements, $h^{<t>}$ the hidden units and $y^{<t>}$ the output elements. W_{hx} is the weight matrix between the input and hidden units, W_{hh} the weights between the hidden units and W_{yh} the weights between the hidden units and the output.

LSTM models

RNNs can struggle with vanishing gradients if the number of cells is large and the hidden layer weights $|w_{hh}|$ is smaller than one . Architectures, such as Long Short-Term Memory (LSTM) networks, have managed to help with modeling long-range dependencies through the use of memory cells (Raschka et al., 2022). LSTMs contain both a long-term focused cell state and a shorter term hidden state that can influence each other (Raschka et al., 2022).

The central components of an LSTM network are three gates (Fig.6). The forget gate decides which information should be forgotten from the cell state, while the input gate determines which parts of a candidate cell state should be added to the cell state. The output gate decides how the values of the hidden units should be updated.

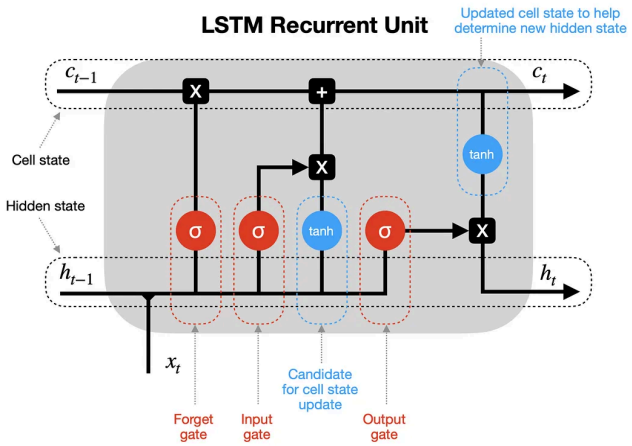

Figure 6. Illustration of a LSTM recurrent unit, showing how the gates (shown in red) decide how the hidden state updates the cell state. Image source: https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e5 4c2ff22e

2.2.4 Transformer Models

While RNNs may still be useful for processing sequential data, they are ultimately limited by their lack of parallelization and ability to prioritize what information is worth focusing on. The invention of the transformer architecture has since enabled networks to excel in those areas, causing a paradigm shift for sequence modeling, especially in tasks related to natural language processing (NLP). The remarkable performance of transformer networks stem from their attention mechanism, which relies on two key ideas: self-attention and multi-headed attention.

Self-attention

Attention mechanisms have been incorporated within RNNs in the past but what sets the transformer architecture apart is the removal of the recurrent process in favor of more self-attention (Vaswani et al., 2017). Self-attention functions by creating a Query (Q), Key (K) and Value (V) matrix by multiplying the input embeddings with three corresponding weight matrices: $\mathbf{W^Q}$, $\mathbf{W^K}$ and $\mathbf{W^V}$ (Eq.1-3, Fig.7a).

$$Q = W^Q \bullet X \tag{1}$$

$$K = W^K \bullet X \tag{2}$$

$$V = W^V \bullet X \tag{3}$$

The query matrix **Q** undergoes a matrix multiplication with the transpose of the key matrix **K**, with the resulting dot product, also referred to as an attention matrix or filter, being a representation of how well the query fits the key for each token pair. The dot product is then scaled through division by the square root of the number of dimensions in **K** (d_k) and undergoes softmax before being multiplied with the value matrix **V** to get the scaled dot product attention (Eq.4, Fig.7b) (Vaswani et al., 2017).

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4}$$

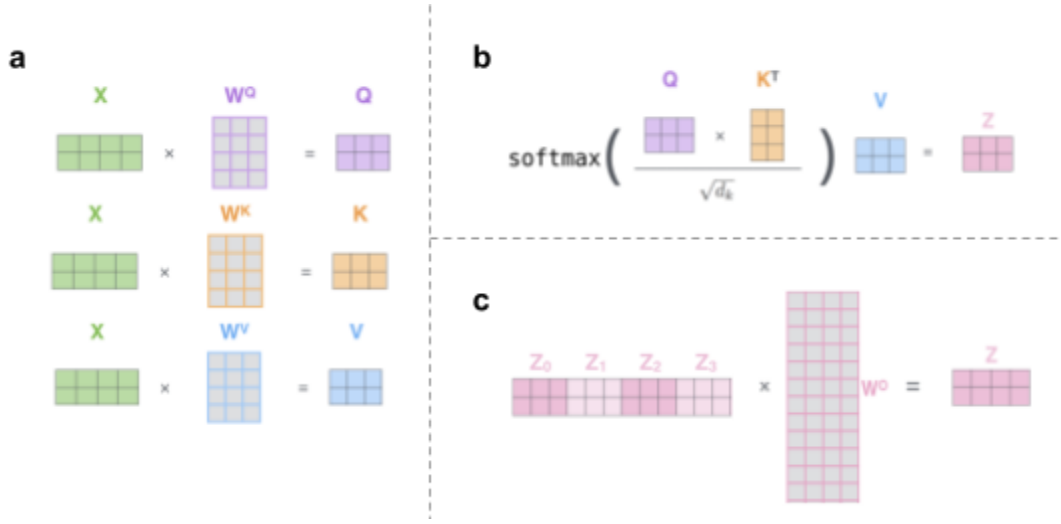

Figure 7. Visual depictions of attention calculations. (a) Creation of query, key and value matrices. (b) Self-attention mechanism. (c) Multi-head attention with four concatenated scaled dot product attention matrices.

Source: https://jalammar.github.io/illustrated-transformer/

Multi-head attention

The self-attention mechanism can be amplified by running multiple instances in parallel. Each iteration of self-attention has its own weights and is referred to as a head, hence the name multi-headed self-attention, or just multi-head attention. The scaled dot product attention of each head is concatenated and undergoes a linear transformation through matrix multiplication with the output weight matrix W^O (Eq.5-6, Fig.7c).

$$MultiHead(Q, K, V) = Concat\left(head_1, \dots, head_h\right)W^O \tag{5}$$

$$\text{where} \quad head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right) \tag{6}$$

Positional encoding

By processing entire sequences in parallel, the inherent order of sequential data gets lost. The information about the positions of each token in a sequence is instead added through positional encoding. The original transformer made use of alternating sin and cos functions to give each position a unique positional embedding, but it is possible to have the model learn the positional embeddings instead.

Transformer architecture

The original transformer model architecture (Fig.8) consisted of two stacks, with one stack composed of encoder blocks and the other of decoder blocks. The goal of the encoder is to map the sequential input data **X** into a continuous representation **Z** that is passed to the decoder. The decoder then uses **Z** to generate an output sequence **Y** through autoregression, where it iteratively adds the previous output token to the input.

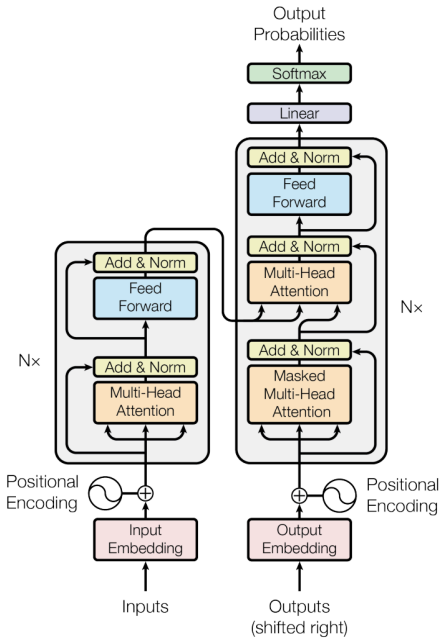

Figure 8. The model architecture for the original transformer model introduced in the Attention is all you need paper. The block to the left is an encoder block and the one to the right is a decoder block. Image source: https://arxiv.org/abs/1706.03762

There have been numerous variations upon this architecture, some which only uses encoder or only decoder blocks (Brown et al., 2020; Devlin et al., 2019). The transformer model at the center of this thesis, MSA Transformer, has an architecture more closely resembling an encoder. Encoder blocks have a relatively simple architecture of a multi-head attention layer followed by

a fully connected feed-forward network. Wrapped around each of the two sublayers are so-called residual connections, followed by layer normalization. Residual connections form the basis of what is referred to as the residual stream in transformers. The sublayers of a transformer can be thought of as reading and writing features to the residual stream, forming complex information processing circuits between layers (Elhage et al., 2021). This makes the embeddings layered with representations from earlier heads, allowing the transformer to detect even more sophisticated features.

Training transformer models

One of the drawbacks with supervised learning is the difficulty of finding the large quantities of labeled data that is required for training large transformer models. Self-supervised learning has provided a solution to the problem through automatically generating labels by masking tokens in unsupervised data.

Autoregressive models, such as GPT models, are often trained using a causal language modeling (CLM) objective. CLM objectives involve the prediction of the next token in a sequence by looking at the previous tokens in it. This is done by masking the token to be predicted along with all the ones to the left of it before running the model and comparing the prediction to the label.

Models which lack autoregressive decoders, such as BERT or other encoder-based models, don't iteratively predict the next token, but instead add layered representations of features to the input embeddings. Identified features can then for example be used by classification layers downstream of the model. Masked language modeling (MLM) objectives are a standard way to train these types of encoder models. MLM works by randomly masking some percentage of the input tokens and having the model predict the masked tokens (Devlin et al., 2019). The prediction is done by applying softmax over the vocabulary (all possible tokens) on the output embeddings of each position in the sequence.

Although transformers have been predominantly designed for NLP, certain architectures and techniques apply just as effectively to a language not constructed by humans but evolved over time – protein sequences.

2.3 MSA Transformer

2.3.1 Protein Language Models

Self-supervised protein language modeling had its start in LSTM models but were quickly outcompeted by transformer-based language models. A major upside of the latter is the connection between sequence-based attention matrices and contact maps – it turns out that knowing which residues are in close proximity is very useful for predicting their coevolution. This makes attention matrices strongly correlated with contact maps, allowing for the prediction of 3D structures from just an amino acid sequence (Fig.9).

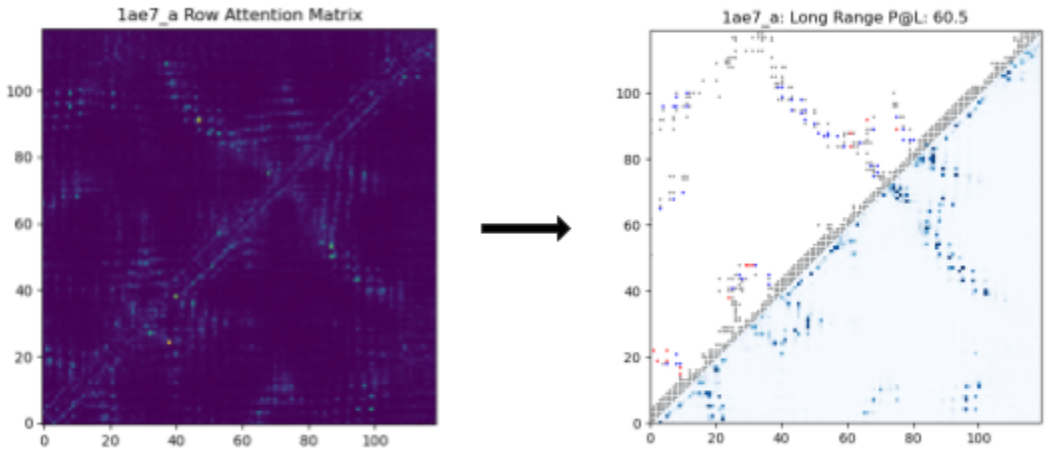

Figure 9. Comparison of a layer 12 MSA Transformer row attention matrix (left) and a predicted contact map (right) for an example protein 1AE7. The x and y axes for both matrices correspond to a residue in the protein. Blue pixels in the predicted contact map's bottom right are residues which are predicted to be in proximity, while the pixels in the top left show how well the predictions correspond to actual contacts.

Amino acid sequences are intuitive inputs for a protein language model as they resemble text input in natural language processing, albeit much less directional. While models using this approach have been proven successful for structure prediction purposes, they are often quite opaque in how they identify relevant features (Lin et al., 2023). The MSA Transformer language model operates on entire MSAs, meaning a lot of coevolutionary information is already stored in the input, potentially making the learned features simpler and thus more approachable for use in downstream tasks. Another perk of this approach is the shared MSA input type with AlphaFold2 that could allow for a straightforward pipeline between them.

2.3.2 The MSA Transformer Architecture

Input and output

After embedding the two-dimensional input MSA, the input is a matrix $x \in R^{M \times L \times d}$, where **M** is the number of sequences per MSA (512), **L** is the number of positions in each sequence and **d** is the hidden dimension of the embedding. MSAs are often much larger than the maximum sequence count (M), which is why the most diverse sequences are "greedily" subsampled from the original MSA in a process that maximizes their hamming distances.

The two-dimensional structure of the data means that the standard one-dimensional embeddings work less well. Instead, learned positional embeddings are added independently to each row to allow the model to distinguish between sequences as well as positions in them.

Axial attention

The attention mechanism uses axial attention to process the two-dimensional input data. Axial attention splits up attention over rows and columns into two separate multi-head attention sublayers. Multi-head self-attention is performed on sequences corresponding either to rows or columns in the input matrix and the resulting features are added to the residual stream like normal.

The axial attention going on in MSA Transformer uses a slight variation of this method. In MSA Transformer, column attention heads operate on positions across sequences and thus become the model's way of explicitly reading coevolutionary signals about the current protein instead of implicitly memorizing with learned weights. The row attention uses a variant called tied row attention. In tied row attention, the row attention matrices are all summed into one single matrix that is then multiplied with the value weights. The rationale behind summing the attention matrices is that the sequences have similar structures and thus attention patterns, which will be strengthened through summation. Tied row attention maps can later be used for contact prediction like in other protein transformer models.

MSA Transformer blocks

MSA transformer blocks have a similar design to the original encoder blocks, except with the tied row attention and column attention being split into two multi-head attention layers, each preceded by layer normalization and added to the residual stream via residual connections (Fig.10). Like in the original transformer, feed-forward layers are placed at the end of each block.

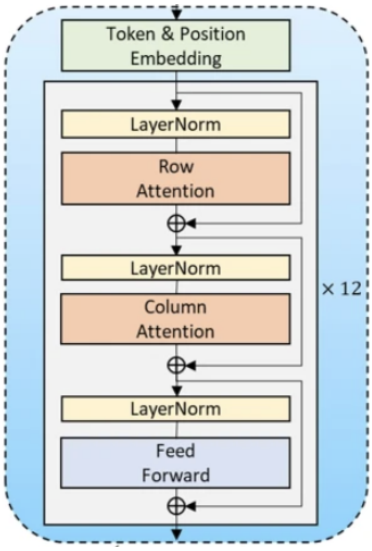

Figure 10. Architecture of the MSA Transformer model. The transformer block is repeated 12 times after creating the embeddings and positional encodings from the input MSA.
Source: https://www.nature.com/articles/s41598-022-18205-9/figures/1

The total amount of parameters in the model is 100M which might seem like a lot, but is on the small side by 2023's standards. The protein language model ESM-2 has up to 15B trainable parameters and NLP models have reached over one trillion (Fedus et al., 2022; Lin et al., 2023).

Pre-training

MSA Transformer was trained on a dataset of 26M MSAs created from the UniClust30 database with HHblits 3 – a sequence alignment tool that makes use of hidden markov models (Steinegger et al., 2019). A MLM objective adapted to the MSA setting was then used to train the model to predict randomly masked out MSA tokens.

2.3.3 MSA Transformer Applications

The MSA Transformer paper showed how the features of the residual stream could be used for downstream classification tasks by training a secondary structure prediction head based on the Netsurf architecture (Klausen et al., 2019).

The activations of the model's attention heads have also been shown to contain useful structural information, which the LSTM-based S-Pred model uses, along with the output embeddings of the residual stream. S-Pred is the architecture for a small group of protein structural property predictors trained to predict one of three different properties: secondary structure, accessible surface area (ASA) and intrinsically disordered regions (Hong et al., 2022). Intrinsically disordered regions (IDRs) are regions of proteins which lack a well-defined 3D structure, often because of a lack of hydrophobic residues that mediate folding.

2.4 Prediction of Multiple Conformational States

AlphaFold2 might be constrained by only being able to predict one conformation, but the fact that it uses MSAs as a source of coevolutionary signal enables manipulation of the MSA to influence the predicted structure. Two approaches that have been shown to make progress in this area involve either sequence clustering or directly modifying residues in the input MSAs.

Sequence clustering attempts to split MSA sequences into multiple clusters, where each cluster contains different coevolutionary signal that can be read by AF2. Clustering methods based on sequence similarity have proven successful at sampling multiple conformations of a couple of fold-switching proteins (H. K. Wayment-Steele et al., 2022). They showed that removing the coevolutionary information about a higher-energy fold-switching state allowed AF2 to predict the ground state as well.

The secondary approach, dubbed "in silico mutagenesis", involves the mutation of aligned residues in specific positions (columns) by changing them to alanin (Stein & Mchaourab, 2022). The positions to mutate are chosen by identifying contact points in the structures generated from the original MSA. Similar to clustering, the idea is to sample different conformations by blocking out structural coevolutionary signal.

In this report I hypothesize that indiscriminately mutating residue positions across sequences or clustering sequences by similarity might be too blunt a tool for selecting specific coevolutionary information. This could potentially be improved by using the activations and features of the MSA Transformer, which have evidently captured useful structural information.

3. Results and Discussion

3.1 Overview of Strategy

The overarching objective of extracting flexibility information from the protein language model, MSA Transformer, was tackled through two distinct sub-projects: Training neural networks on the output data of the language model to predict per-residue protein flexibility, and using the output data for clustering and modifying MSAs in order to sample multiple conformations of the same protein with AlphaFold2.

Section 3.1 describes the preparations made to allow for the first project, the training of networks. Initially, a dataset of flexible proteins to use for training and evaluating the model designs was chosen, along with a metric for per-residue flexibility that could be set as labels for the dataset. Lastly, some filtering and analysis was performed on the dataset to improve the quality of the training data.

The designs and training processes for the three types of model architecture are showcased along with their results in the following sections: MLP networks in **Section 3.2**, LSTMs in **Section 3.3** and CNNs in **Section 3.5**.

Section 3.4 details the development of a framework that enables the use of column attention data by convolutional networks and clustering methods. The section also examines patterns between column attention matrices and flexibility data, as well as across different attention heads.

Evaluating clustering methods for the second sub-project requires, much like the first project, a set of examples and a metric to compare them by. **Section 3.6** describes the process of constructing a benchmark of example proteins and comparing a couple evaluation metrics. Unfortunately, time constraints prevented the development and evaluation of clustering methods.

3.1 Experimental Setup

3.1.1 The PDBFlex Database

PDBFlex is a flexible protein database structured around 38,341 protein clusters of superimposed PDB structures that share at least a 95% sequence identity, created with CD-HIT (Huang et al., 2010). According to the paper behind the database, the natural or engineered substitutions making up the 5% difference in sequence identity should not have a large-scale impact on the structures, except for certain substitutions like glycines and prolines that might change the shapes of alpha-helices (Hrabe et al., 2016). Despite reservations about the impact of differences in sequence identity, the database was selected as a promising candidate for a training dataset due to several factors, including its substantial size, pre-calculated flexibility metrics, and accessible web-based interface for visualizing conformational changes (Fig.11).

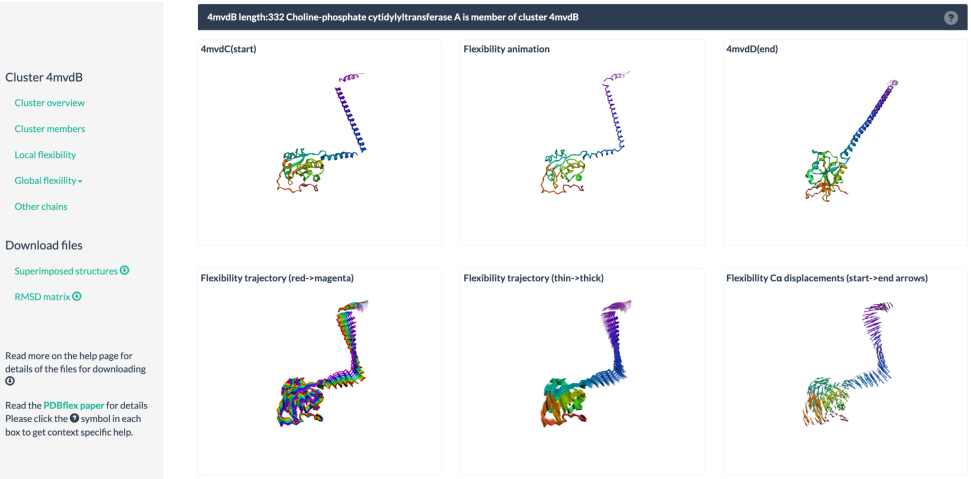

Figure 11. Example of the visualization options available for the PDBFlex cluster 4mvdB. Screenshot taken from the PDBFlex website: https://pdbflex.org/cluster.html#!/4mvdB/23116/4mvdB.

3.1.2 Local RMSD

The PDBFlex database has a couple flexibility metrics, with the main one being local RMSD profiles that are accessible via an Application Programming Interface (API) that enables external programs to retrieve data from the website (Fig.12a). The average local RMSD for each residue was calculated by first identifying a ‘master’ sequence, i.e. the most complete sequence in each cluster. The average RMSD values between the master sequence’s Cα atoms and the

corresponding atoms in the other cluster sequences are then calculated over a 10-residue sliding window. This makes local RMSD a decent metric for finding local flexibility, but also "hinges" where the residues on either side of the hinge change their relative position between the structures in a cluster. Hinge movements tend to produce sharp peaks in the local RMSD profile, while fold-switching seems to be represented by a wide band of high local RMSD (Fig.12b).

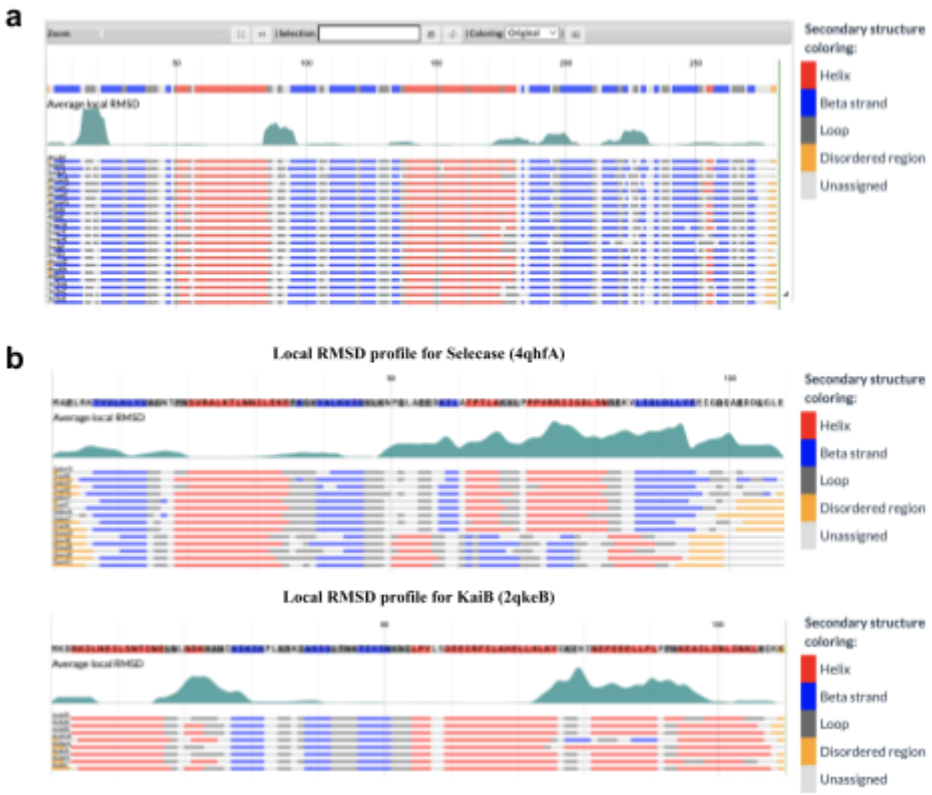

Figure 12. (a) Example of a local RMSD profile for the 4huuA protein cluster in the PDBFlex database. The coloured bands represent secondary structure and the bands below the RMSD profile plot represent the secondary structures of the other cluster members. (b) Local RMSD profiles for two fold-switching proteins: Selecase (PDB: 4qhfA) and KaiB (PDB: 2qkeB). Screenshots taken from the PDBFlex website: https://pdbflex.org/viewer.html#!/4huuA/21499/4huuA.

Other metrics of flexibility were also considered, such as difference distance maps, Local Distance Difference Test (lDDT) and B-factors. B-factors were ruled out for their low resolution and focus on dynamics instead of larger scale conformational changes, but difference contact maps and lDDT could just as well have been chosen. Ultimately, local RMSD was chosen for the sake of convenience and saving time, since local RMSD labels had already been created for the entire PDBFlex database. Examining other metrics would have been interesting, but was deemed out of scope for the project.

3.1.3 Filtering the Dataset

Having a dataset of over 38,000 clusters provided the luxury of liberally filtering to select only the most desirable clusters in terms of size and flexibility. The histogram of the sequence lengths of each cluster (Fig.13a) shows the majority of the distribution laying between 50 to 400 residues. 400 was used as an upper limit to save computational time, and the lower limit of 50 residues was put in place to avoid the issues that could come about with very short sequences, such as when generating MSAs. Since the objective of the project was to detect larger conformational changes, the clusters with a maximum local RMSD value of less than 1Å were filtered out. The low-flexibility clusters made up a large portion of the total dataset (Fig.13b), but a sizable 13,042 clusters remained after the two aforementioned reductions.

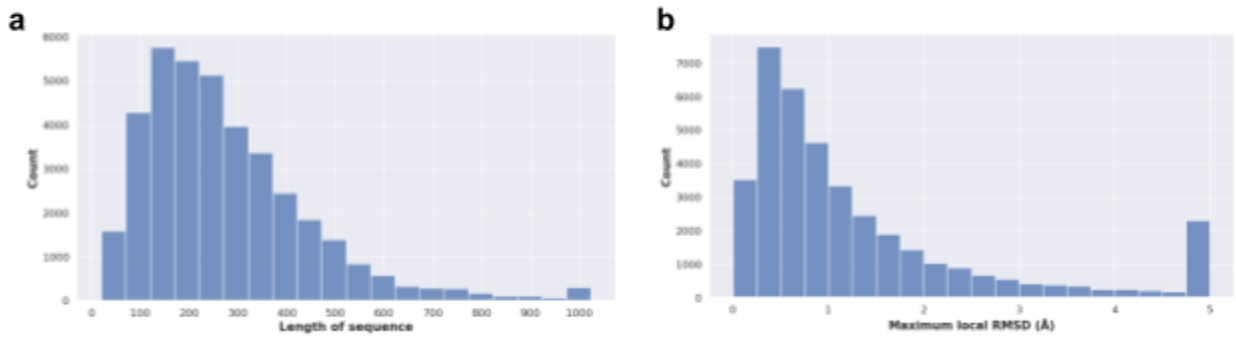

Figure 13. Histograms of the database showing the distribution of (a) sequence lengths or (b) maximum local RMSD.

3.2 Training MLP classifiers

As an introductory task, a few MLP networks were created to increase familiarity with deep learning and try out some techniques. The designs of the networks were architecturally simple variations of a series of fully connected layers as exemplified in Figure 14.

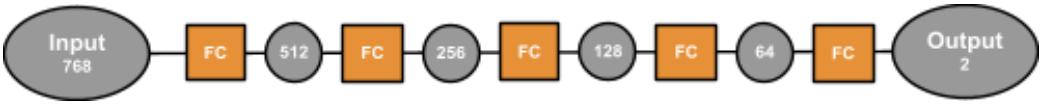

Figure 14. Architecture of a simple fully connected per-residue flexibility classifier. The input dimensions are 768, which through five fully connected layers (orange) gets reduced to 2 classes: local RMSD above or below a certain threshold.

The binary classification task

It seemed too complicated for such a simple network to predict the flexibility of an entire sequence at once, so the models were instead trained to simply predict whether each residue had a local RMSD above or below a certain threshold. The reason for using a binary classification task instead of multiclass classification or regression was partly because it is simpler, but also because the heights of local RMSD peaks appeared to be relatively arbitrary and difficult to predict. The idea was that it would be easier for the model to just separate flexible and non-flexible residues from each other instead of having to get the exact level of flexibility right.

Generating embedding input data

The input data for the model was originally intended to include the entire output embedding for each residue, but the dimensions of this would have been far too large, with tensor dimensions up to 768 x 512. Instead, only the embeddings for the residues in the query sequence were used, as inspired by the S-Pred model input. To speed up the process across multiple training runs, the MSA Transformer was run in advance, saving only the parts of the embeddings to be used as inputs.

The network was trained on the embeddings for single residues. These were created by first separating the input protein embeddings across training, testing and validation datasets, and then splitting each of them into residue embeddings. This was done to prevent residues from the same protein to be present in both the training and testing dataset.

MLP model results

Unsurprisingly, none of the simple models trained with MLP networks managed to accomplish more than overfitting on the training data, reaching a per-residue accuracy barely above the one obtained from predicting the local RMSD value to be under the threshold every time.

Throughout the project, four types of plots were created to evaluate the performance of the trained models. Loss curve plots (Fig.15a) compare the training and validation loss per epoch and are a useful tool for noticing when the model overfits or gets stuck. Similarly, accuracy curve plots (Fig.15b) were used to evaluate the average accuracy for each epoch, providing another dimension to study overfitting. Accuracy is simply calculated by dividing the number of correct predictions with the total number of predictions. Accuracy plots also allow for comparisons with

a baseline accuracy as a means to determine whether a model has achieved any relevant predictive ability. The baseline in this case is referred to as the zero guess line, and is the accuracy achieved from predicting each local RMSD value to be below the selected threshold. The ratio between residues with local RMSD below the threshold of 0.2 Å to ones above were about 4:1, which is why the zero guess baseline and initial predictions had accuracies so close to 0.8.

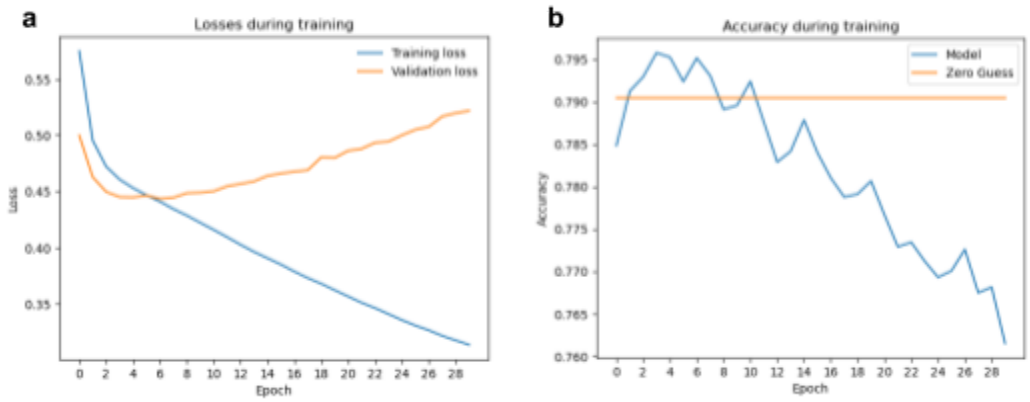

Figure 15. (a) Loss curve displaying the training loss (blue) and validation loss (orange) across 30 epochs. (b) Accuracy plot comparing the testing accuracy (blue) with the baseline zero guess accuracy (orange) across 30 epochs.

Confusion matrices are another staple among model evaluation metrics (Fig.16a). They are used to put a number on the distribution of true and false positives and negatives. From these, other metrics such as precision, recall and F1 score can be calculated for further analysis. Lastly, I wanted to visualize the results on a per-example basis to better understand what the model predicts for each residue. A new type of plot was created for this purpose, where the local RMSD profile and predictions are shown alongside each other (Fig.16b).

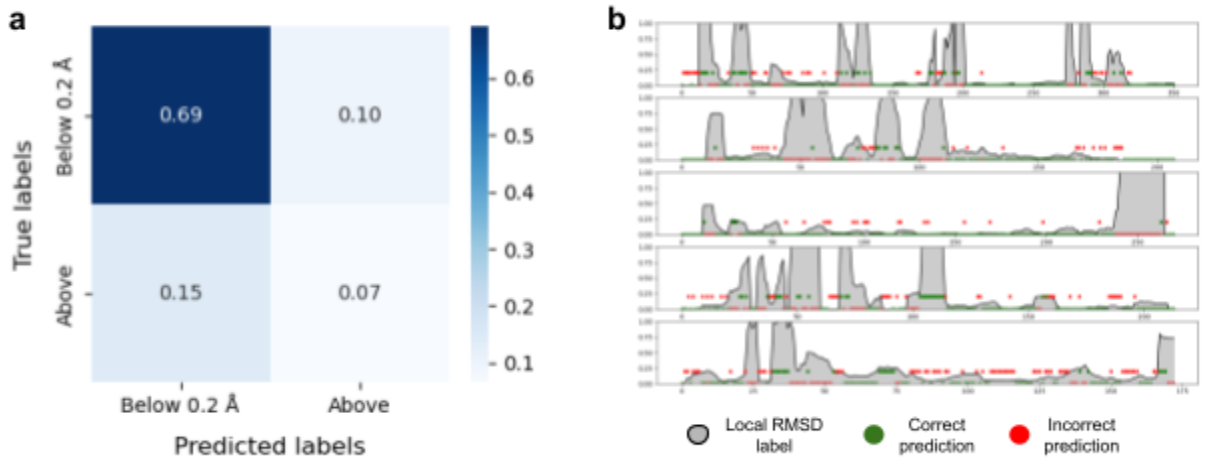

Figure 16. (a) Confusion matrix displaying what share of the predictions were true negatives (top left), false negatives (bottom left), false positives (top right) and true positives (bottom right). (b) Plot comparing the local RMSD labels (gray) with the predictions for 5 test set proteins. Green dots signify residues for which the model correctly predicted whether or not the local RMSD was above the 0.2 Å threshold, while red dots signify incorrect predictions.

3.3 Training S-Pred Inspired LSTM Classifiers

Following the success of the S-Pred model's LSTM architecture on classification and regression tasks related to protein properties, it seemed reasonable to attempt using a similar network for classifying residue flexibility. The model was created by simply using the secondary structure classification version of the S-Pred model and changing the output classes from 8 to 2. The model was also modified to allow it to be trained and allow for use of pre-generated embeddings as inputs.

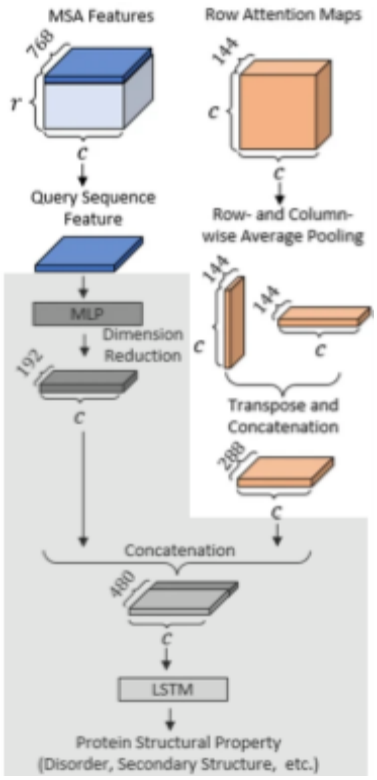

Figure 17. Architecture of the S-Pred model. The non-grayed out parts were pre-generated when creating the dataset, while the grayed out parts were performed when running the model. Image source: https://www.nature.com/articles/s41598-022-18205-9/figures/1

Generating input data

Like for the MLP models, input data consisted of embeddings generated by MSA Transformer beforehand to save time. Query sequence embeddings and average-pooled row attention matrices were concatenated and saved as tensors with dimensions [c x 1056]. The non-greyed out parts of Figure 17 correspond to the data that was generated as the inputs in the dataset.

Results of using a LSTM model

The trained models tended to almost always predict a local RMSD below the threshold, even after changing some hyperparameters. A likely reason for this could be mistakes in implementing the S-Pred model, so to validate the methodology, an attempt was made to reproduce the original S-Pred model, including using the original weights. The reproduction attempts gave similar results to the altered model, so it seems likely that further tweaking of the model is required if using the S-Pred architecture. There was not enough time to do this so the focus was instead moved to the second part of the project: clustering sequences to sample multiple conformations with AlphaFold2.

3.4 A Deep Dive Into Column Attentions

Finding features with meaningful differences between sequences corresponding to different conformations was the primary goal of the second part of the project. This rests on the hope that these features could then be used when clustering MSA sequences, so that the clusters contain even more relevant coevolutionary information than they would from just clustering by sequence similarity.

Row attention matrices were briefly considered for this, but their summation into tied row attention matrices meant that information that separated sequences from each other was lost. Modifying the MSA Transformer code could provide a way of accessing the untied matrices, but the plan was discontinued in favor of pursuing column attention maps instead since their very purpose is to compare coevolutionary information across multiple sequences.

Each run of MSA Transformer creates 144 column attention matrices for each residue (column), where each column attention matrix has a width and height equal to the number of sequences in the input MSA. This totals hundreds of millions of data points; way more than can easily be clustered, so the dimensions had to be reduced in some way. One way to accomplish this was through summing the attention matrices along one axis and stacking them together to form a matrix of the same shape as the original MSA for each attention head. These matrices were then summed across the heads to form a single matrix per input MSA. While the matrices showed some potential, the noise from summing across all column attentions was hard to manage (Fig.18).

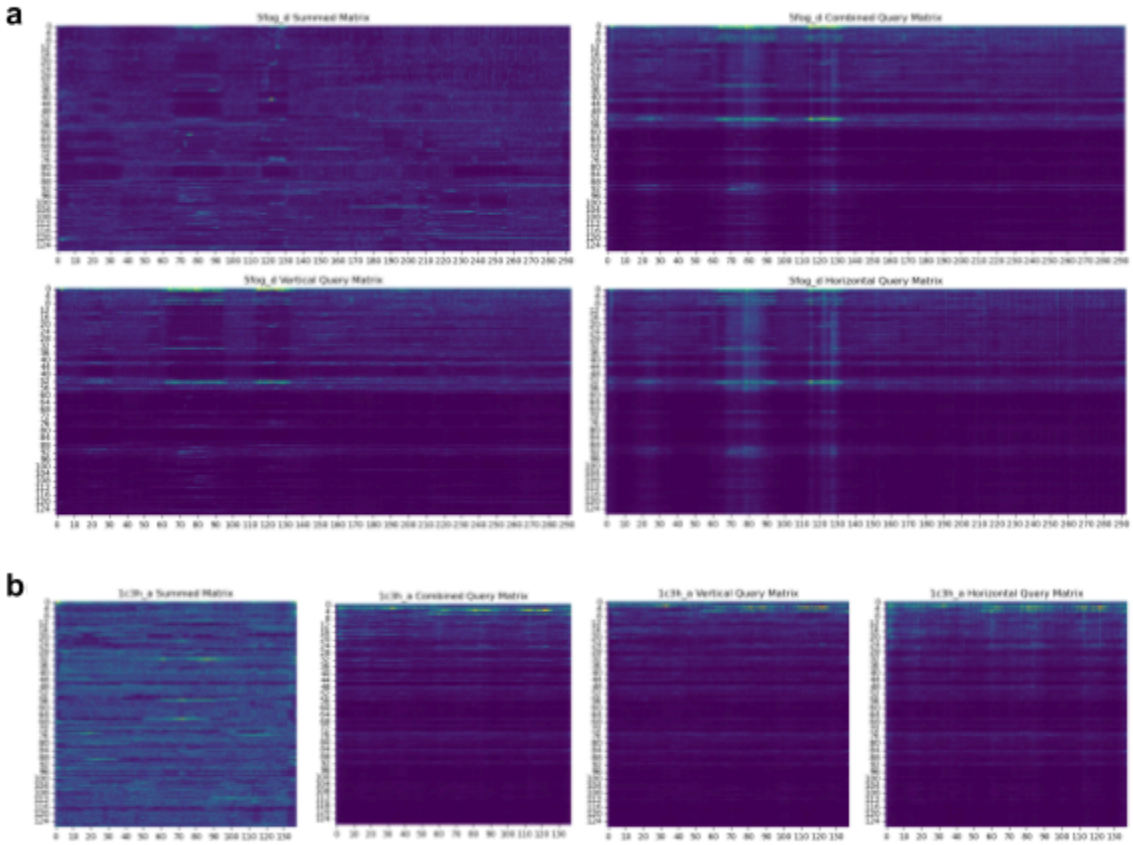

Figure 18. (a) Comparison between the stacked summations of column attention matrices, the combined query matrix, and the vertical and horizontal query matrices for protein cluster 5fog_d. (b) Comparison between the same matrices as a, but for protein cluster 1c3h_a.

Query matrices

Another type of matrix was created by stacking the row / column of the column attention matrices that corresponds to the query sequence, i.e. the original sequence used for creating the MSA (Fig.18). The rationale behind the creation of these "query matrices" was built on the assumption that attention related to the query sequences would be especially rich in useful information about the sequence of interest. The idea is that the vertical query matrix (matrix 0) describes which sequences the model is taking information from when updating the query sequence's embeddings, while the horizontal query matrix (matrix 1) describes how much attention is paid to the query sequence when updating each of the other sequences' embeddings. Both of these pieces of information seemed highly related to coevolutionary signal and thus of

interest for clustering. The query matrices showed some promising differences between sequences, but also contained some unexpected patterns.

Query matrices and local RMSD

Closer inspection of the query matrices revealed a surprising correlation between bands of activation and local RMSD profiles (Fig.19). While not present in all proteins, this correlation pointed towards the fact that MSA Transformer had learnt to detect features that correlate with protein flexibility. The secondary structures of the MSA sequences begin to tell the story of how this correlation might come about: As can be observed in Figure 19a, the peaks of high activation appear to correlate with IDRs and flexible linkers between more rigid secondary structures. This is in accordance with the findings from downstream ML models like S-Pred that show that features from MSA Transformer can correspond to IDRs and secondary structures. It is however striking that these patterns are so clearly visible by simply observing the model's column attention activations.

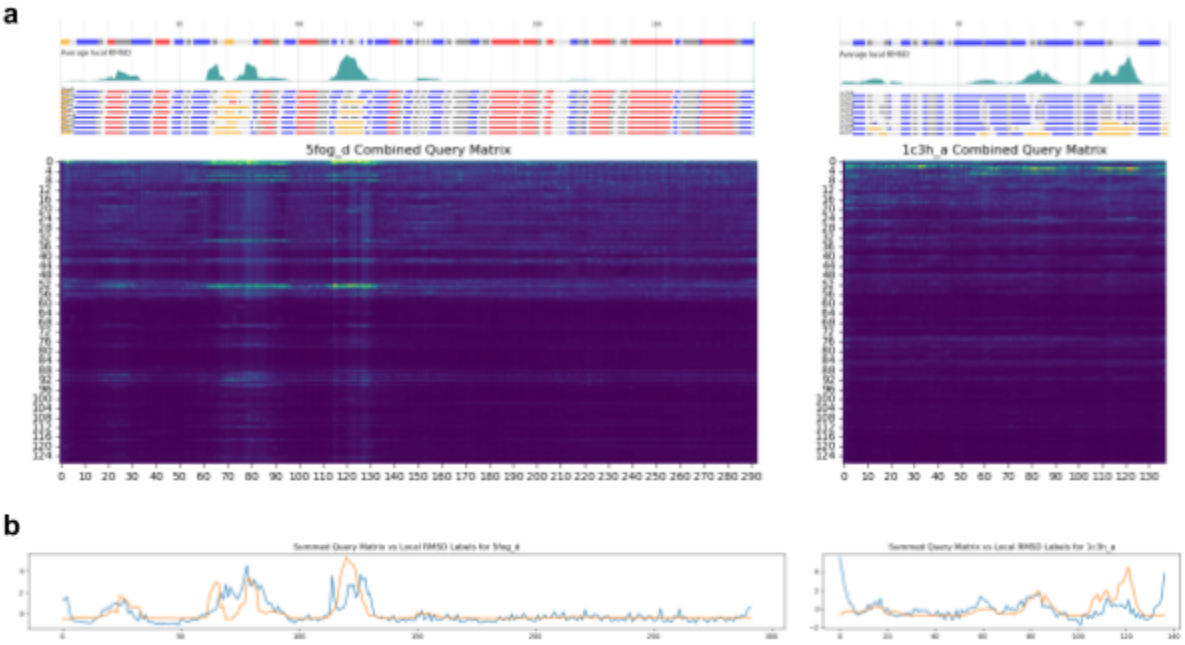

Figure 19. (a) Comparison of query matrices and their local RMSD profiles for protein clusters 5fog_d (left) and 1c3h_a (right). (b) Line plots comparing the normalized sums of the query matrices (blue line) and the normalized local RMSD profiles (orange line) for the two protein clusters.

Figure 19a also suggests some redundancy among sequences, in that many of them contain similar information. A comparison between using 128, 64 or 32 sequences (Fig.20) showed that

there was only a minimal difference between using 128 and 64 sequences. Future experiments thus use only 64 sequences in order to save time and memory, but a more thorough analysis could definitely prove interesting.

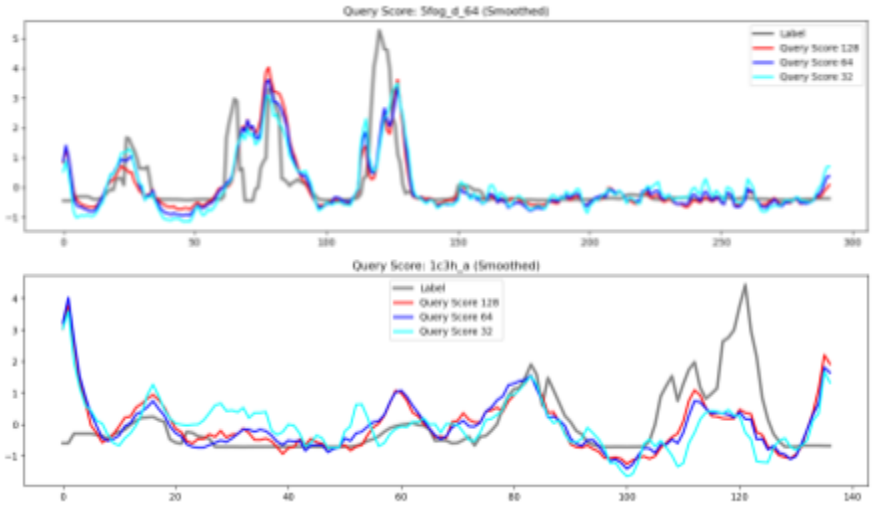

Figure 20. Comparison of smoothened normalized summed query matrix values from query matrices with 128 (red), 64 (blue) and 32 (cyan) sequences for the protein clusters 5fog_d and 1c3h_a. The gray line represents the local RMSD values for each residue.

Evaluating correlation

The ability to more quantitatively measure the correlation between the query attention activations and local RMSD values at a larger scale would provide a way to evaluate the correlation better than a handful of cherry-picked examples could. Summation across attention heads and sequences might drown out correlating patterns with weaker activations, for example the large peaks in query matrix activation that is commonly found near the terminals. Separately calculating the similarity between the RMSD labels and query activations for each combination of attention head and MSA sequence allowed for a more precise measurement of correlation. Observing the most correlated sequences can give a clue as to what features the attention heads of the model might have learnt to pay attention to. Moreover, the data can be presented in a more readable way by assembling the similarity scores of each combination of attention head and MSA sequence into a "similarity score matrix".

As an example of utilizing these techniques, I chose to study a highly flexible PDBFlex cluster 5bpd_a, which contains the multiple conformations of the DNA-binding archaeal chromatin

protein TrnBL2. The summed query matrix of the protein displayed an interesting pattern in that it showed multiple large peaks at residues connecting secondary structures, of which two coincided with major local RMSD peaks (Fig.21a). When examining the structural conformations of TrnBL2, the first major peak appears to represent a hinge-like movement of two subunits while the second one seems to correlate to a smaller-scale flexibility within a subunit (Fig.21b). The matrix of similarity scores for each head/sequence combination showed a band of sequences for which some early heads produced activations with high similarity to the local RMSD label (Fig.21c). Closer examination showed that the similarity came from overlap with the first, hinge-related peak (Fig.21d), prompting the question of whether different types of flexibility are recognized by different attention heads.

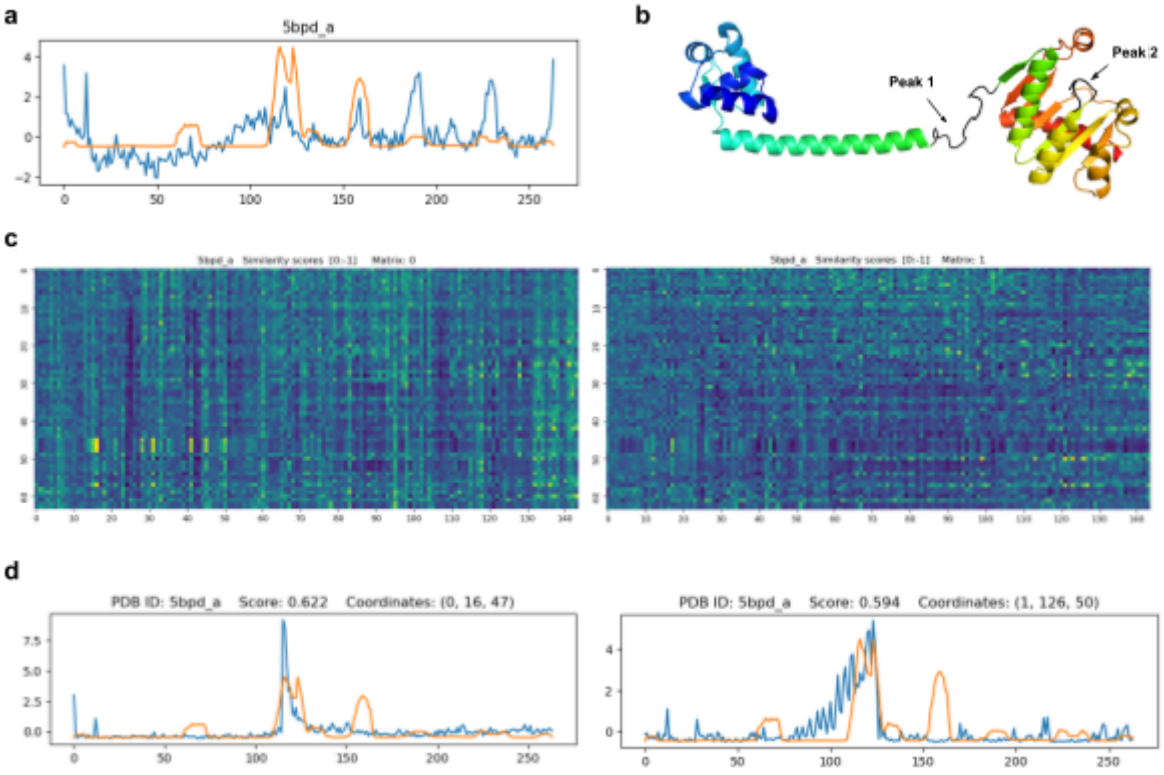

Figure 21. (a) Normalized line plot of summed query matrices (blue) and local RMSD (orange). (b) Protein structure of 5bpd_a, with arrows pointing to the two major peaks of the local RMSD profile. (c) Similarity score matrices constructed from the vertical query matrix (left) and horizontal query matrix (right) for 5bpd_a. (d) Normalized line plots of the two combinations of query matrix type, attention head and MSA sequence (blue) with the highest similarity to the local RMSD profile (orange) for the 5bpd_a cluster.

In an attempt to answer this question, two sets of similarity score matrices were created, with each of them operating on roughly half of the query matrix so that the two peaks would be

isolated from each other (Fig.22). When comparing the score matrices, two types of similarity become apparent. For the slice containing the first peak, the results resemble previous results in that there is a band of high-scoring combinations (Fig.22a, horizontal red arrow), while the second slice shows more vertical lines (Fig.22b, vertical red arrows) that indicate that information about the flexible region is identified by specific attention heads more than from certain sequences.

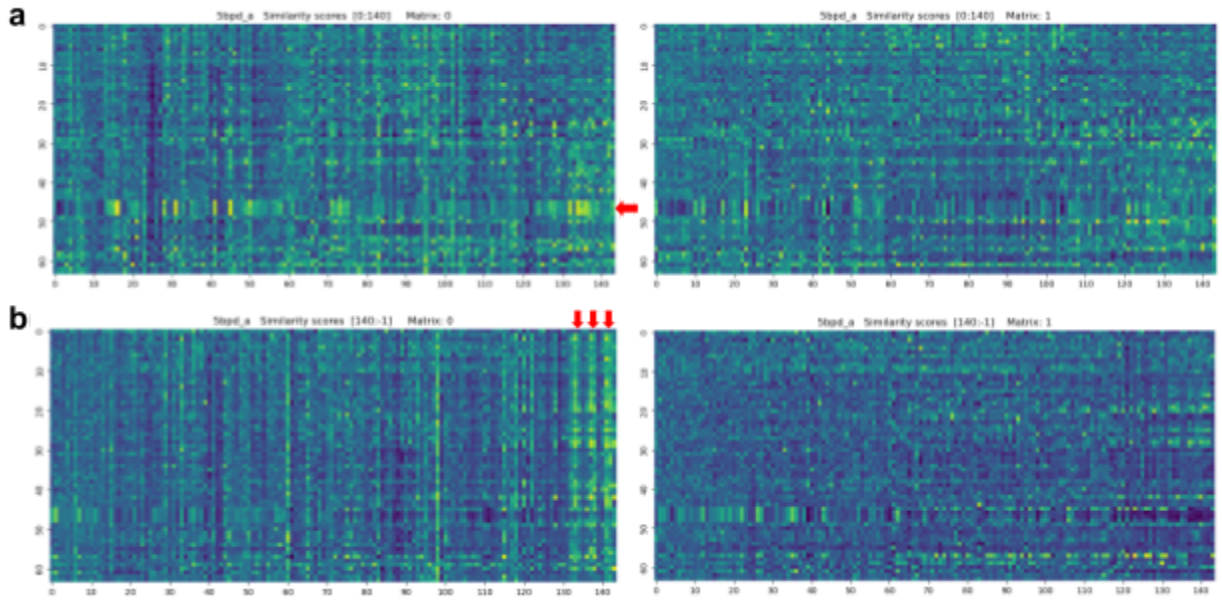

Figure 22. Comparison between two sets of similarity score matrices (vertical on the left, horizontal on the right) created from the 5bpd_a protein cluster. (a) Matrices created from the first 140 residues. (b) Matrices created from the residues after the 140th. The red arrows point at rows and columns which have a clear difference between the similarity score matrices of the two slices.

To verify the results of the first slice, a query matrix was created from the activations of head 16, a head which showed a strong correlation with the first peak (Fig.23a). The lack of signal strength for certain sequences was adjusted for by normalizing the values for each sequence to produce a new, normalized matrix (Fig.23b). Amplifying the signal gave rise to a distinct band at the area of the peak for sequences 45, 46, 47 and 47. Further examination of the sequences contributing to the band reveals a striking similarity between the activations of head 16 and the gaps present in the input MSA, with the bands most likely being the result of the sequences ending at the same spot as the peak in local RMSD begins (Fig.23c). This explanation also could be the reason as to why early attention heads gave the best similarity scores for the first peak; identifying when sequences end seems like a simple task which could easily develop in early

layers.

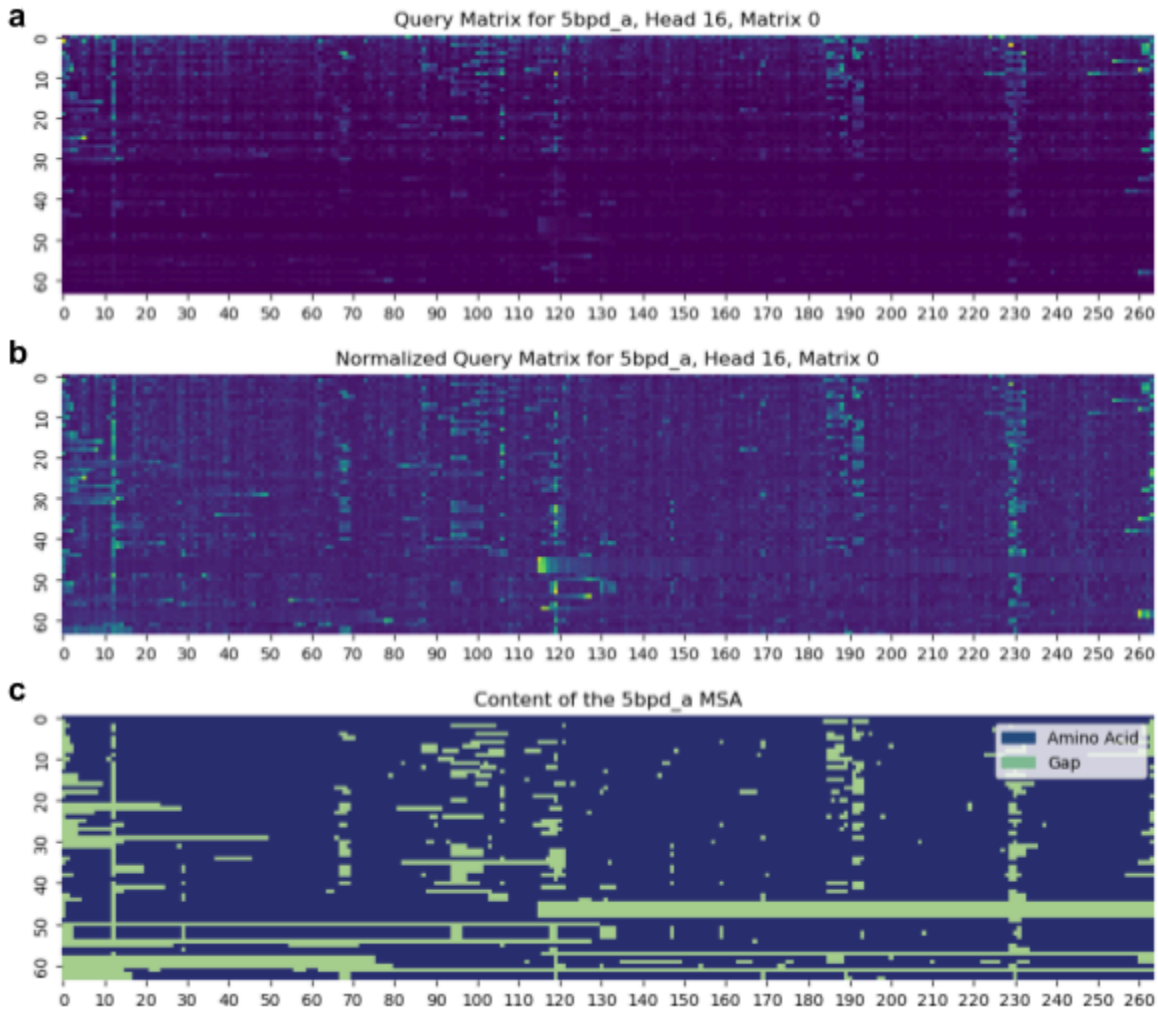

Figure 23. (a) Vertical query matrix constructed from attention head 16 for the 5bpd_a cluster. (b) The same query matrix as in the previous figure, but with each row normalized. (c) Content of the 64-sequence 5bpd_a MSA, with positions containing amino acids shown in blue and positions containing gaps shown in green.

The correlation between gaps in the MSA and activations in the model could potentially help explain some of the identified correlations with local RMSD. Residues missing from MSAs could conceivably be related to disordered regions and finding sequences that are cut off might help the model identify where different subunits end or begin. These theories are however highly speculative and would require a much more systematic approach to verify. One way of investigating the reliance on gaps could be to compare the performance and correlations on input MSAs where sequences above a certain gap content threshold are filtered out. Similarly, the theories about different heads learning different forms of flexibility would have to be

substantiated by more thorough evidence before any meaningful conclusions could be drawn. The findings do however point towards interesting properties about the internal workings of MSA Transformer that further research could investigate more thoroughly and systematically.

Comparing correlation across multiple proteins

In an effort to make more general statements about the model internals, similarity score matrices for multiple proteins were summed. The purpose of this was to give a general picture of which sequences and attention heads produced the activation patterns with the highest similarity to the local RMSD labels (Fig.24). As is to be expected, the similarity scores were fairly distributed across sequences, but the difference was noticeable between attention heads, with some heads having much stronger correlation than others. The pattern showed a clear similarity to the lines in the similarity score matrix for the second half of the 5bpd_a sequence, suggesting that the peak in this section could be the result of a type of flexibility that those attention heads correlate with. Summing the scores across sequences also allowed for comparisons between vertical and horizontal query sums, and between the 500 proteins with the highest or lowest average local RMSD (Fig.24).

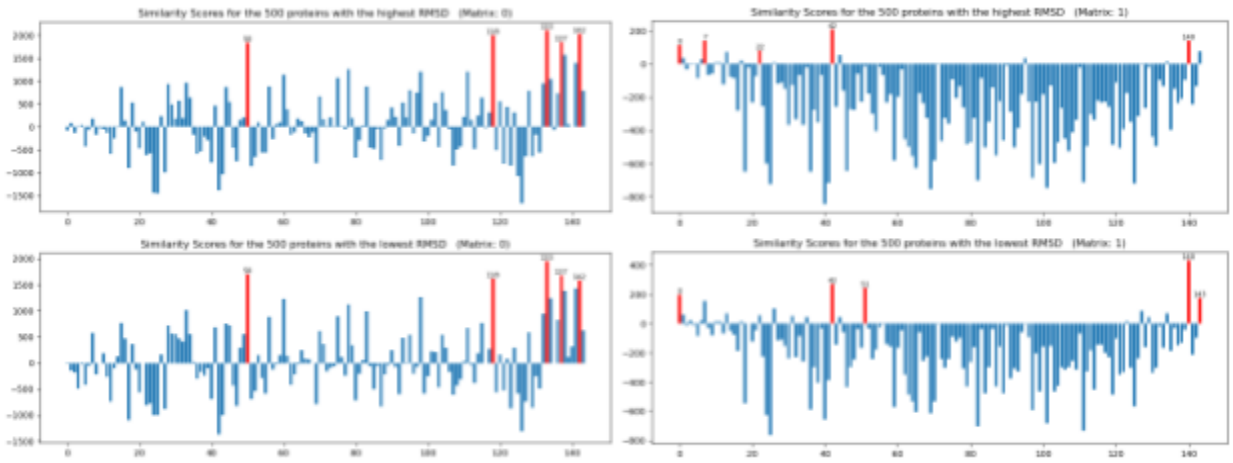

Figure 24: Bar plots comparing the attention heads' total level of similarity to local RMSD labels, between the 500 clusters with highest RMSD (top) and the 500 clusters with lowest RMSD. The plots on the left are made from vertical query matrices while the plots on the right are made from horizontal ones. The 5 highest scoring heads of each plot are shown in red. The unit on the y-axis is not that relevant since the aim of the figure is to give a general picture of the differences between attention heads.

The results in Figure 24 show that the heads with the greatest correlation in matrix 0 were mostly located in the final layer, with a fairly consistent increase in score across heads when increasing

the local RMSD. Matrix 1 showed very different results, with most of the heads having an overall negative correlation and an increase in similarity when using proteins with low flexibility. There is clearly much room for further examination of what the attention heads actually represent and why their correlations change between different levels of flexibility. Even when knowing which heads tend to be the most correlated with flexibility, the problem of knowing which MSA sequences to look at remains.

3.5 Training Convolutional Networks

After the initial discovery that column attention maps demonstrated correlation with local RMSD, a final attempt at training a flexibility classifier was made. The query matrices created in Section 3.4 were essentially two-dimensional images with 144 channels, one for each attention head. This input data seemed suitable for a convolutional architecture due to the task's similarities with image processing.

Creating a dataset of query matrices

Query matrices of shape 400 x 128 x 144 were created by padding the residue length dimension of each query matrix to the maximum residue length, 400. The padding was done to keep the dimensions identical and allow for minibatch training.

Creating the architecture

The general idea was to reduce the number of dimensions from L x M x H (400 x 128 x 144) down to L x 1 x 1 (400 x 1 x 1), i.e. with one output value for each of the L residues. The reasoning behind keeping the L dimension constant throughout the model was so that information about each residue would carry over until the output layer without having to be represented with fewer dimensions. The size of the L dimension was kept the same throughout the model through first employing a series of 2d convolutions with uneven stride and padding, thus reducing only only the M dimension while keeping the L dimension constant and increasing the size of the H dimension. Max pooling is also employed to quickly reduce the M dimension until it reaches 1. At that point, 1d convolutions can reduce the H dimension down to 1.

A couple variations upon this architecture with different numbers of layers were tested. The first model (Fig.25a) was very large, containing 7 layers with parameters and a total of 14M

parameters, making the model very slow to train. Among others, a smaller model (Fig.25b) with 2.5M parameters spread out over 5 layers was created, and trained using Adam to classify flexibility into above or below a set threshold. The model overfitted and failed to generalize beyond the training set, never reaching an accuracy greater than predicting zero every time would have produced.

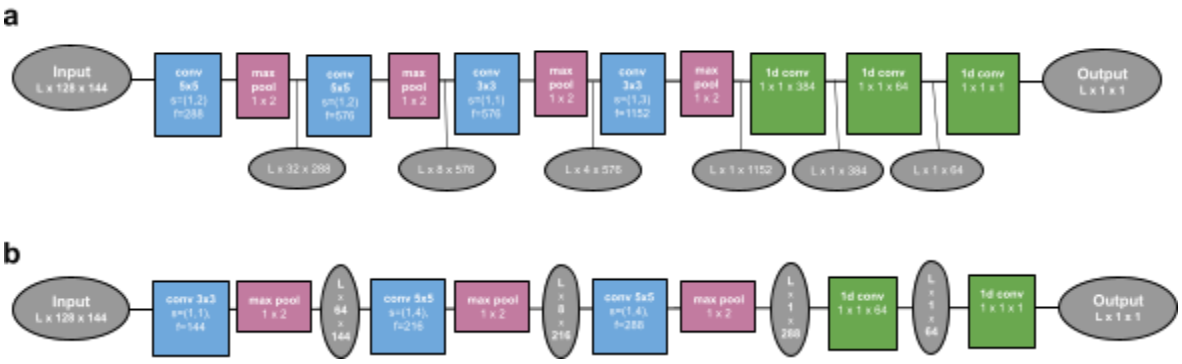

Figure 25. Architectures of two modes. (a) An initial design that first reduces the sequence dimension to 1 and increases the amount of channels through 2d convolutions (blue) and max pooling (purple), followed by 1d convolutions (green) to reduce the channels to 1. (b) A simpler design that uses fewer layers and thus fewer parameters. The channel count is also greatly reduced.

3.6 Setting up a Benchmark for Sampling Conformations

Setting up a benchmark

When developing a new methodology, it is important to set up a benchmark to compare it to other methodologies that are operating on the data, so that the comparison is as fair as possible. To construct such a benchmark, 6 examples of fold-switching proteins were taken from the benchmark used by AF_Cluster (H. K. Wayment-Steele et al., 2022) and 2 examples were taken from the SPEACH_AF benchmark (Stein & Mchaourab, 2022). Two reference structures for each benchmark protein were downloaded, along with their MSAs, as created by both HHBlits and MMSeqs2 using colabfold (*Google Colaboratory*, n.d.). The reference structures are representations of two distinct conformations the protein can take. By comparing the similarity to the reference structures, clustering methods can have their ability to sample naturally occuring conformations tested.

Table 1: List of example proteins used in the benchmark, along with their reference PDB IDs and sources of origin.

Name	Reference PDB 1	Reference PDB 2	Source
KaiB	2qke_e	5jyt_a	AF_Cluster
RfaH	5ond_a	6c6s_d	AF_Cluster
Mad2	1s2h_a	1duj_a	AF_Cluster
Selecase	4qhf_a	4qhh_a	AF_Cluster
Lymphotactin	1j9o_a	2jp1_a	AF_Cluster
CLIC1	1k0n_a	1rk4_b	AF_Cluster
Adenylate Kinase	4ake_a	1ake_a	SPEACH_AF
Ribose Binding Protein	2dri_a	1ba2_b	SPEACH_AF

Choosing an evaluation metric

The two research papers by which the benchmark was inspired each used different metrics for evaluating structural similarities between sampled structures and reference PDBs. The AF_Cluster paper made use of RMSD while SPEACH_AF used TM scores to evaluate the predictions. They both used Principal Component Analysis (PCA), an unsupervised ML technique for reducing the dimensions of data, to compare the contact maps of each predicted structure.

Sampling structure predictions with AlphaFold2 for the sometimes hundreds of clusters generated by the AF_Cluster methodology proved time consuming enough to warrant an investigation into possibly using a proxy metric instead of RMSD. The MSA Transformer paper included methodology for very quickly evaluating the accuracy of its predicted contact map with the one of a reference PDB structure. If this metric correlated with RMSD, would enable quick iteration and development of clustering methods. When comparing the two metrics it became clear that contact prediction accuracy was quite poor of a proxy (Fig.26). While there could still be other, faster evaluation methods, it is likely the case that creating structures with AlphaFold2 is an essential step of evaluation.

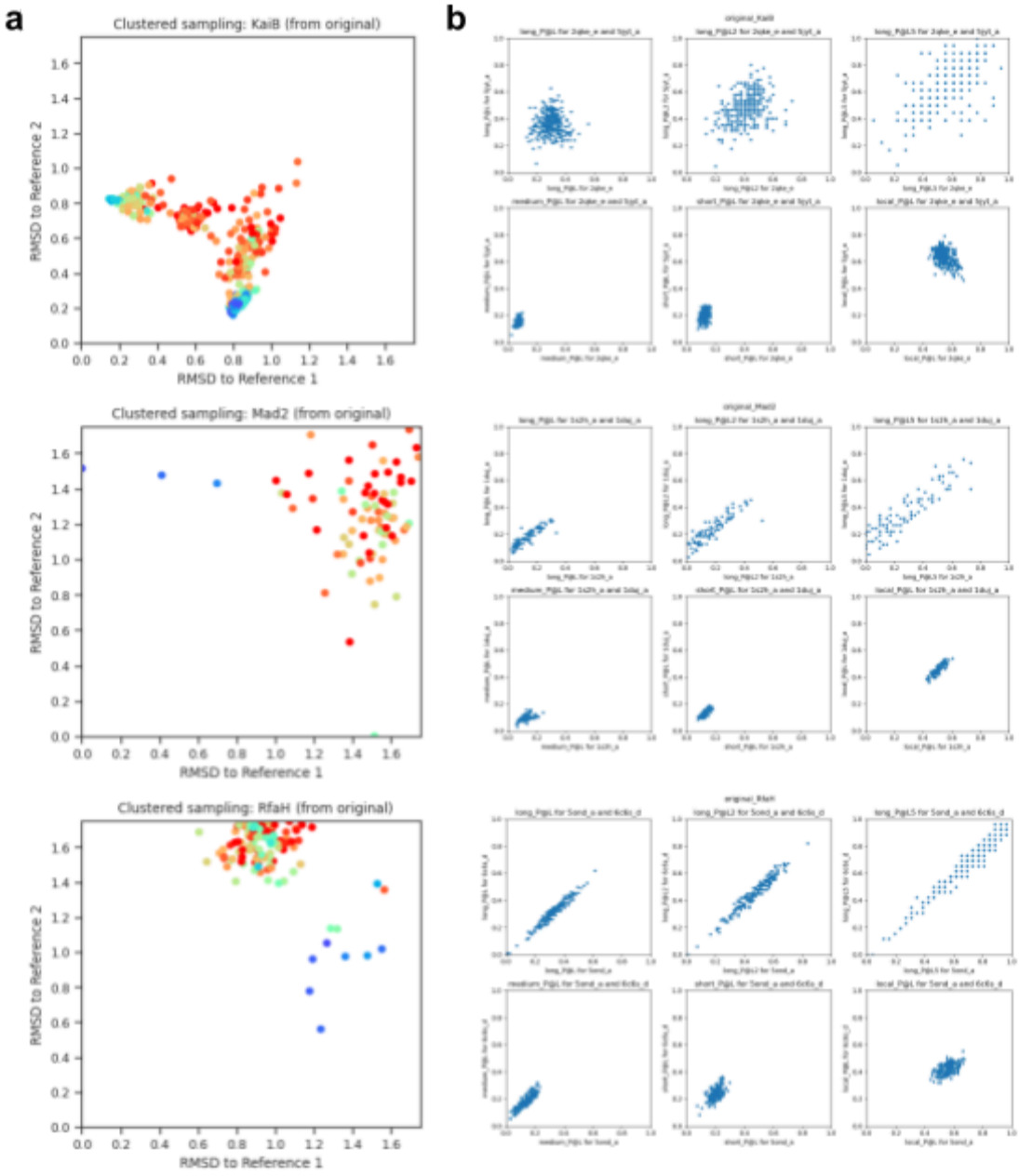

Figure 26. Plots comparing the (a) 2D plots comparing the RMSD to the two reference structures for each structure predicted from the MSA clusters generated by AF_Cluster. (b) 2D plots comparing the predicted contacts of the MSA clusters created by AF_Cluster with the contacts of the two reference structures. Six different metrics were used to evaluate the contacts: long_P@L, long_P@L2, long_P@L5, medium_P@L, short_P@L and local_P@L.

By this point in the project, there was unfortunately no time left to do any of the more thorough clustering experiments that were planned to be performed on the benchmark. If given more time, multiple types of clustering algorithms could have been put to use across different types of MSA

Transformer output data. Some preliminary testing of clustering query matrices was done, but without any reliable results. An example of a proposed methodology was to cluster the sequences in a query matrix since it was hypothesized that attention values for sequences with similarities to an alternate conformation would likely have a different pattern of attention than the rest, at least for the specific residues where the contacts differed the most. Dimensionality reduction through PCA or similar algorithms could perhaps also be employed to aid with the clustering since the data points per sequence could be quite large if every head and sequence is included.

4. Conclusions

Overall, the project has made some progress on its initial aims of understanding how to extract flexibility features from the MSA Transformer language model. The initial goals of training neural networks and clustering sequences through the use of the model's activations were largely unsuccessful for reasons I will delve further into later on.

Core findings of the project

One of the most fundamental parts of any machine learning project is understanding the data one is working with. For this project, the two types of data consisted of local RMSD labels and MSA Transformer outputs. The initial discoveries regarding MSA Transformer's column attention matrices showcased in Section 3.4 were made after already having attempted MLP and LSTM networks, and the latter ones were not made until the very end of the project, at which point they were of little use. The findings do however indicate a variety of interesting correlations that would likely prove valuable for further research into extracting features from MSA Transformer, or similar protein language models.

Since the data to be understood consisted of neural network activations, the research can be seen as a rudimentary form of model interpretability research – the study of opening up models in an attempt at understanding what they have learnt and how they represent this information. Most existing interpretability research has been done with computer vision or NLP models, so furthering the understanding of protein language models would be a novel contribution to the research area.

The most significant finding of the report is most likely the fact that activations for the query sequences in column attention maps often correlate with residue flexibility as described by the local RMSD metric. Furthermore, the construction of query matrices from column attention strips made the correlations easier to visualize and opened up for further analysis. Among this analysis, the discovery of attention heads corresponding to gaps in the MSA, and in particular attention heads displaying varying correlations with flexibility, poses many further questions worthy of investigation. An approach that builds on this project's findings to focus on furthering the understanding of the model's features and internals would likely be able to provide higher quality data for both the training of future networks and clustering of MSAs.

Improving the flexibility classifier architectures

The idea of using neural networks for predicting flexibility from transformer outputs may appear less promising given the absence of positive results, but there are many potential reasons that could explain these unsuccessful attempts. Deep learning can be both difficult and time consuming, often taking entire teams months to develop and refine new models, so much could be attributed to the large scope and limited time. For example, it seems plausible that a model with an S-Pred architecture could succeed, as it is in many ways the closest thing available to the desired flexibility classification model. Similarly with the convolutional models, there are likely other designs that would better be able to take advantage of the information in MSA Transformer's column attention matrices. Much of the process of choosing architectures and hyperparameters could also be automated to save a lot of time, and allow for quicker iteration and ruling out bad designs.

The decision to pose the problem as a binary classification task instead of multiclass classification or regression was made for the sake of simplicity, but could likely have hampered the models' performance. The learning signal, an important ML concept that describes the information used to fuel the learning process's feedback loop, might have become too coarse-grained as a result of using binary classification. In reality, flexibility lays more on a spectrum, and reducing the data to being above or below an arbitrarily set threshold likely had some consequences: Either the cutoff point was so high that it excluded many peaks, or so low that it included too much. Too high of a cutoff point also made the balance between the two classes so uneven that the model easily learnt to just predict the flexibility to be below the

threshold every time. Even including different loss weights for the two classes had little effect, likely because of how little learning signal the model got.

Systematic problems with PDBFlex and local RMSD

Apart from the aforementioned shortcomings, there could plausibly exist some more systematic problems with the approach, namely with the chosen dataset and flexibility labels. It might be worth questioning whether the 95% sequence identity within PDBFlex clusters is sufficient to guarantee that a protein is flexible. The database is doubtlessly a useful tool for visualizing protein flexibility, but it might be less suited as a training dataset since the data could be polluted by examples where the different conformations came about as a result of sequence differences.

As previously stated, the local RMSD profiles used during the project might not have been the best choice for dataset labels. As opposed to other labels such as secondary structure, IDRs and accessible surface area used by the S-Pred model, local RMSD has a much less direct connection to physical properties. This is in addition to the fact that local RMSD peaks can correspond to everything from hinge movements to fold-switching to local disordered regions. If MSA Transformer has managed to capture features related to those types of flexibility, it seems unlikely that they would all be represented in the same way. Lumping them all into the same metric likely made the learning process harder for the downstream classifiers. Instead of using just a single metric, a solution could involve training multiple models on different, more specific metrics that each encapsulate certain types of flexibility. This would most likely make the learning problem for each model simpler, and having multiple outputs could be useful for analysis and further use, such as MSA clustering. Difference distance maps (DDMs) are an example of a metric which primarily corresponds to large-scale conformational changes, while lDDT might be better suited to measure local flexibility.

Further studies on sampling multiple conformations

Ultimately, the designated time was not enough to complete the final part of the project, mostly because of the ambitious scope and unexpected setbacks. What can be said about clustering is that using predicted contacts as a proxy for RMSD when evaluating clusters was not feasible. It seems fairly likely that this would extend to all proxies because they would need to capture the way in which AF2 makes structure predictions in order to work. Using the AF_Cluster protocol

to create AF2 predictions from each of the sequence clusters and comparing them to the reference structures currently seems like the best approach, with the potential of speeding up the process by running the predictions on a computer cluster.

Despite not being able to run any clustering experiments, a lot of groundwork for clustering MSAs has been made, including the creation of a benchmark and identification of potential input data and ways in which it can be expanded upon further. Using query matrices, especially from specific attention heads, seems like a very promising approach to both clustering sequences and identifying relevant pieces of coevolutionary information that could be altered through in silico mutagenesis.

5. Method

The code written for the project can be found in the GitHub repository with the following URL: https://github.com/VeraKarlin/master-thesis/

5.1 Creating a dataset

5.1.1 Accessing the PDBFlex database

A list of the master sequences of each cluster in the PDBFlex database was compiled in a csv file using a web crawler, ***pdbflex_crawler.py***, written using selenium and geckodriver. Another script, ***get_rmsd_profiles.py***, created another csv file containing the local RMSD profiles for the first 1024 residues of each cluster in the database by accessing them through the API.

5.1.2 Filtering the clusters

The ***filter_pdbflex_dataset.py*** script was used to create a text file of the master sequence PDB IDs for the clusters that passed the filters imposed on the clusters, namely sequence length and average RMSD.

5.1.3 Generating multiple sequence alignments

In accordance with the MSA tool used in the MSA Transformer paper, HHblits version 3.1.0 (Steinegger et al., 2019) was selected with default settings except for the number of search iterations, which was set to 3. HHblits uses FASTA files as inputs so FASTA files were created with the ***pdb_id_to_fasta.py*** script by turning the SEQRES record over all PDB entry sequences into a dictionary and looking up the FASTA sequence for each of the master sequences.

The ***run_all_batch_alignments.py*** script divided the dataset into clusters and ran the ***batch_alignment.py*** script in parallel, which created the alignments for a list of input PDB IDs by executing the following command:

hhblits -cpu 4 -i <fasta_path> -d <db_directory> -oa3m <output_path> -n 3

MSAs with fewer than 1,000 sequences were filtered out to ensure a diverse selection of sequences in the alignments, leaving a total of 10,381 clusters.

5.1.4 Generating input data from MSA Transformer outputs

MSA Transformer has three forms of output with the following dimensions:

- **Tied row attention:** *layers (12)* × *heads (12)* × *L* × *L*
- **Column attention:** *layers (12)* × *heads (12)* × *L* × *M* × *M*
- **Embeddings:** *layers (12)* × *M* × *L* × *embedding_dim (768)*

Where:

L = Sequence length (50 - 400)

M = Sequence count (usually set to 64, 128 or 512)

Different parts of this output data were used depending on the downstream model architecture. The size of the total output reaches billions of variables, so only selected parts of the outputs were saved during each run. The MSA Transformer outputs used for each type of downstream model were as follows:

Inputs for fully connected (MLP) models

- **Input types:**
  - Layer 12 embeddings for the query sequence (L × 768).
- **Scripts:**
  - *batch_embeddings.py*
  - *run_all_batch_embeddings.py*

Inputs for LSTM models

- **Input types:**
  - Layer 12 embeddings for the query sequence (L × 768).
  - Row attention maps (144 × L × L)
- **Scripts:**
  - *batch_s-pred_features.py*
  - *run_all_batch_s-pred_features.py*

Inputs for convolutional (CNN) models

- **Input types:**
  - Vertical/horizontal query sequences of column attention matrices (144 × L × M × 2).
- **Scripts:**
  - *batch_column_attentions.py*
  - *run_all_batch_column_attentions.py*

5.2 Flexibility classification models

5.2.1 MLP models

Models were designed, trained and tested in the jupyter notebook ***MLP Classifier.ipynb***. Jupyter notebooks were used for many parts of the project because they allowed for quick hands-on experimentation with new concepts and libraries.

What the MLP models had in common was stacks of fully connected layers with ReLU activation functions and decreasing dimensions until they reached a binary output of the local

RMSD being either above or below a set threshold. This threshold was usually set to 0.2 Å since it struck the balance between including enough local RMSD peaks without too much insignificant noise. The Adam optimizer was used with learning rates ranging from between 0.01 to 0.0001, but most commonly 0.001. A mini-batch size ranging from 32 to 2048 was used depending on the size of the model and dataset. In terms of regularization, some models used combinations of batch normalization and various levels of dropout, with 0.2 being a common dropout rate.

5.2.2 LSTM models

The code implementing an LSTM model based on the S-Pred architecture was written in the ***LSTM Classifier.ipynb*** notebook. The model was further expanded upon to allow for minibatch training in the ***LSTM Minibatch Classifier.ipynb*** notebook. The differences in sequence length between the proteins meant that a lot of padding would be necessary when creating batches. As a way to minimize the padding, each dataset was first sorted by sequence length, followed by each batch being padded to the length of the longest protein in that batch, after which the sequences of each batch were shuffled. It was also in this notebook where the S-Pred replication attempt was implemented.

5.2.3 CNN models

Multiple different convolutional model designs were implemented in the ***CNN Classifier.ipynb*** notebook. As with the other classifier models, many of the same optimization and regularization techniques were employed, namely the Adam optimizer, minibatch gradient descent and batch normalization.

5.3 Analysis of column attention matrices

The ***Column Attention Visualization.ipynb*** notebook is where the discoveries showcased in Figures 18, 19 and 20 were made. The analysis shown in Figures 21 to 24 was carried out later on in the ***Query Matrix Analysis.ipynb*** notebook. The headings in the notebooks showcase the code used to produce the plots in the results section.

5.4 Setting up a benchmark

The benchmark was set up so that each example protein had a directory where all relevant information was stored. The directory structure was as follows:

```
/benchmark_data/
        /af_cluster_monomeric/          # KaiB, RafH, Mad2
        /af_cluster_oligomeric/         # Selecase, Lymphotactin, CLIC1
        /classic/                       # Adenylate Kinase, Ribose Binding Protein
                <Name>
                        /af_clusters/   # Clusters from the AF_Cluster GitHub
                        /alignments/    # HHBlits and MMSeqs2 MSAs
                        /fasta/         # The Fasta sequence used to generate the MSAs
                        /reference_1/   # PDB file for the first reference protein
                        /reference_2/   # PDB file for the second reference protein
```

The 2D contact accuracy plots were created using the ***run_benchmark.py*** script, while 2D RMSD plots were created from AF2 structures with the ***CalculateModelFeatures.py*** script taken from the AF_Cluster GitHub (H. Wayment-Steele, 2022/2023).

Bibliography

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020). *Language Models are Few-Shot Learners* (arXiv:2005.14165). arXiv. https://doi.org/10.48550/arXiv.2005.14165

Burger, L., & Nimwegen, E. van. (2010). Disentangling Direct from Indirect Co-Evolution of Residues in Protein Alignments. *PLOS Computational Biology*, *6*(1), e1000633. https://doi.org/10.1371/journal.pcbi.1000633

del Alamo, D., Sala, D., Mchaourab, H. S., & Meiler, J. (2022). Sampling alternative conformational states of transporters and receptors with AlphaFold2. *eLife*, *11*, e75751. https://doi.org/10.7554/eLife.75751

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (arXiv:1810.04805; Version 2). arXiv. https://doi.org/10.48550/arXiv.1810.04805

Ekeberg, M., Lövkvist, C., Lan, Y., Weigt, M., & Aurell, E. (2013). Improved contact prediction in proteins: Using pseudolikelihoods to infer Potts models. *Physical Review E*, *87*(1), 012707. https://doi.org/10.1103/PhysRevE.87.012707

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., DasSarma, N., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., … Olah, C. (2021). A Mathematical Framework for Transformer Circuits. *Transformer Circuits Thread*.

Fedus, W., Zoph, B., & Shazeer, N. (2022). *Switch Transformers: Scaling to Trillion Parameter*

Models with Simple and Efficient Sparsity (arXiv:2101.03961). arXiv. http://arxiv.org/abs/2101.03961

Google Colaboratory. (n.d.). Retrieved January 3, 2024, from https://colab.research.google.com/github/sokrypton/ColabFold/blob/main/AlphaFold2.ipynb

Hong, Y., Song, J., Ko, J., Lee, J., & Shin, W.-H. (2022). S-Pred: Protein structural property prediction using MSA transformer. *Scientific Reports*, *12*(1), Article 1. https://doi.org/10.1038/s41598-022-18205-9

Hrabe, T., Li, Z., Sedova, M., Rotkiewicz, P., Jaroszewski, L., & Godzik, A. (2016). PDBFlex: Exploring flexibility in protein structures. *Nucleic Acids Research*, *44*(D1), D423–D428. https://doi.org/10.1093/nar/gkv1316

Huang, Y., Niu, B., Gao, Y., Fu, L., & Li, W. (2010). CD-HIT Suite: A web server for clustering and comparing biological sequences. *Bioinformatics*, *26*(5), 680–682. https://doi.org/10.1093/bioinformatics/btq003

Ian Goodfellow, Yoshua Bengio, & Aaron Courville. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org

Klausen, M. S., Jespersen, M. C., Nielsen, H., Jensen, K. K., Jurtz, V. I., Sønderby, C. K., Sommer, M. O. A., Winther, O., Nielsen, M., Petersen, B., & Marcatili, P. (2019). NetSurfP-2.0: Improved prediction of protein structural features by integrated deep learning. *Proteins: Structure, Function, and Bioinformatics*, *87*(6), 520–527. https://doi.org/10.1002/prot.25674

Lane, T. J. (2023). Protein structure prediction has reached the single-structure frontier. *Nature Methods*, *20*(2), Article 2. https://doi.org/10.1038/s41592-022-01760-4

Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., Smetanin, N., Verkuil, R., Kabeli, O., Shmueli, Y., dos Santos Costa, A., Fazel-Zarandi, M., Sercu, T., Candido, S., & Rives, A. (2023). Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, *379*(6637), 1123–1130. https://doi.org/10.1126/science.ade2574

Lockless, S. W., & Ranganathan, R. (1999). Evolutionarily Conserved Pathways of Energetic Connectivity in Protein Families. *Science*, *286*(5438), 295–299. https://doi.org/10.1126/science.286.5438.295

Maini, V. (2018, May 28). Machine Learning for Humans, Part 2.1: Supervised Learning. *Machine Learning for Humans*. https://medium.com/machine-learning-for-humans/supervised-learning-740383a2feab

Raschka, S., Liu, Y., Mirjalili, V., & Dzhulgakov, D. (2022). *Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python* (1st edition). Packt Publishing.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958.

Stein, R. A., & Mchaourab, H. S. (2022). SPEACH_AF: Sampling protein ensembles and conformational heterogeneity with Alphafold2. *PLOS Computational Biology*, *18*(8), e1010483. https://doi.org/10.1371/journal.pcbi.1010483

Steinegger, M., Meier, M., Mirdita, M., Vöhringer, H., Haunsberger, S. J., & Söding, J. (2019). HH-suite3 for fast remote homology detection and deep protein annotation. *BMC Bioinformatics*, *20*(1), 473. https://doi.org/10.1186/s12859-019-3019-7

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., &

Polosukhin, I. (2017). Attention is all you need. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6000–6010.

Wayment-Steele, H. (2023). *AF-Cluster* [Jupyter Notebook]. https://github.com/HWaymentSteele/AF_Cluster (Original work published 2022)

Wayment-Steele, H. K., Ovchinnikov, S., Colwell, L., & Kern, D. (2022). *Prediction of multiple conformational states by combining sequence clustering with AlphaFold2* (p. 2022.10.17.512570). bioRxiv. https://doi.org/10.1101/2022.10.17.512570