# ProtACon: Exploring Relationships between Attention and Protein Properties

---

Complex Networks & Pattern Recognition

*Alma Mater Studiorum* - Università di Bologna

S. Chiarella, R. Eliasy

December 17, 2024

**Abstract**

The purpose of this study is to find relationships between the attention mechanism of the ProtBert model and the physico-chemical properties of proteins, using both network methods and machine learning algorithms. We found out that some attention heads specialize in targeting specific amino acids, and one attention head is able to identify contacts between the protein residues. Furthermore, we showed that ProtBert is able to capture the evolutionary-based substitution relationships between amino acids. The network representation of the peptide sequence, flanked by the Louvain algorithm, and supported by a deeper biochemical knowledge of the proteins, has the potential to bring to further discoveries.

# Contents

# 1 Introduction

ProtACon is a wide project aimed to explore and interpret the representations generated by transformers when they are applied to proteins. In particular, the main task is to detect possible connections and similarities between the attention weights generated by a transformer, and the physico-chemical properties of the proteins that are fed into it. This project was inspired by the work of Jesse Vig et al. [1]. As they do, we also analyzed the connections between the attention masks from the encoder and the contact between amino acids in the peptide chains—in fact, that is the core of our work. However, while their study focuses on the protein structure, the second part of our work is about the extraction of features from the network representation of the protein, where the nodes are the chain residues and the edge attributes represent the relationships between the residues. We used machine learning algorithms to cluster the edges, to extract information from the transformer representations.

## 1.1 Natural Language Processing and Proteins

A language model is a type of machine learning model designed to understand, generate, and interact with human language. These models are trained on large corpora of text data and can perform a variety of tasks such as language translation, text summarization, question answering, and more. Recently, natural language processing (NLP) has become very popular because of its usage on human languages. Its principles and functioning can be generalized to other kinds of languages though, such as programming languages, or the complex ways developed by nature in living beings. Proteins represent a remarkable example in this field.

The connection between language models and proteins lies in the use of similar machine learning techniques to understand and predict the structures and functions of both language and biological sequences. Natural language is based on sequences of words, their meaning depending on their context within a sentence. Analogously, proteins are made up of smaller building blocks called amino acids. There are 20 types of amino acids commonly found in proteins, each with a unique side chain that gives it specific properties. Amino acids are linked together in long chains through peptide bonds to form protein molecules (see Fig. 1).

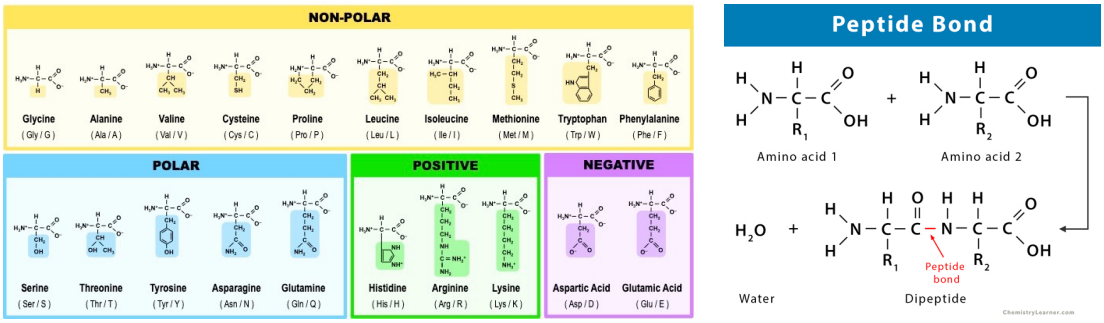


**Figure 1:** Chemical structure of the 20 amino acids (left) and peptide bond (right).

These 1D sequences adopt unique three-dimensional shapes (referred to as protein 3D structure) [2, 1], and the 3D structures perform specific functions. The order, or sequence, of the amino acids in a protein determines its unique shape and function, just like words do in a sentence. The sequence of amino acids in a protein is stored in the genetic code.

NLP methods are replacing the previous methods for the prediction of the protein secondary structures (basically, how a protein chain folds), that combined machine learning and evolutionary information. Those methods involve two steps:

- First, multiple sequence alignment (MSA) is used to identify a group of evolutionarily related proteins. This alignment reveals the evolutionary relationships between the proteins by highlighting similarities and differences in their amino acid sequences. Essentially, it captures the encoded evolutionary history within the protein family.

- Then, the extracted evolutionary information (EI) is fed into a machine learning model using supervised learning. This model is trained to recognize patterns in the EI that correlate with specific protein structures or functions. [3]

NLP allows to predict secondary structures but skip the first step, that has some drawbacks, such as the high computational cost or the lack of EI for some proteins. This is made possible by treating proteins as sentences and amino acids as words. Indeed, protein sequences are constrained to adopt particular 3D structures optimized for accomplishing particular functions. These constraints mirror the rules of grammar and meaning in NLP. [3]

## 1.2 Transformers and Attention

NLPs initially relied on models like recurrent neural networks (RNN) and long short-term memory networks (LSTM). In a nutshell, in these models the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions. This makes it more difficult to learn dependencies between distant positions [4, 5].

Those models have now been overcome by *transformers*, in terms of training efficiency—there is a larger possibility of parallelization, because they do not rely on a recurrent mechanism [6]—and handling long-range dependencies, thanks to the *self-attention* mechanisms.

**Transformer** It consists of an encoder that processes the input sequence, and a decoder that generates the output sequence (see Fig. 2). Each is composed of multiple identical layers. Each layer consists of a multi-head self-attention mechanism, and a feed-forward neural network. Multi-heads allow the model to attend to all positions in the input sequence simultaneously, capturing dependencies between different parts of the input.

**Encoder** Its role is to process the input sequence and encode its information into a set of continuous representations. Before entering the encoder, the input sequence is first broken down into individual units called tokens. Each token is converted into an *embedding*, a high-dimensional vector that captures its meaning in the context of the entire

sequence. Basic embeddings are processed by the layers to build richer representations, which are used by the decoder to generate the output sequence. At the end of the encoder stack, we have a set of vectors, each corresponding to a position in the input sequence. These vectors encapsulate contextual information about the input tokens, incorporating dependencies and relationships among them.

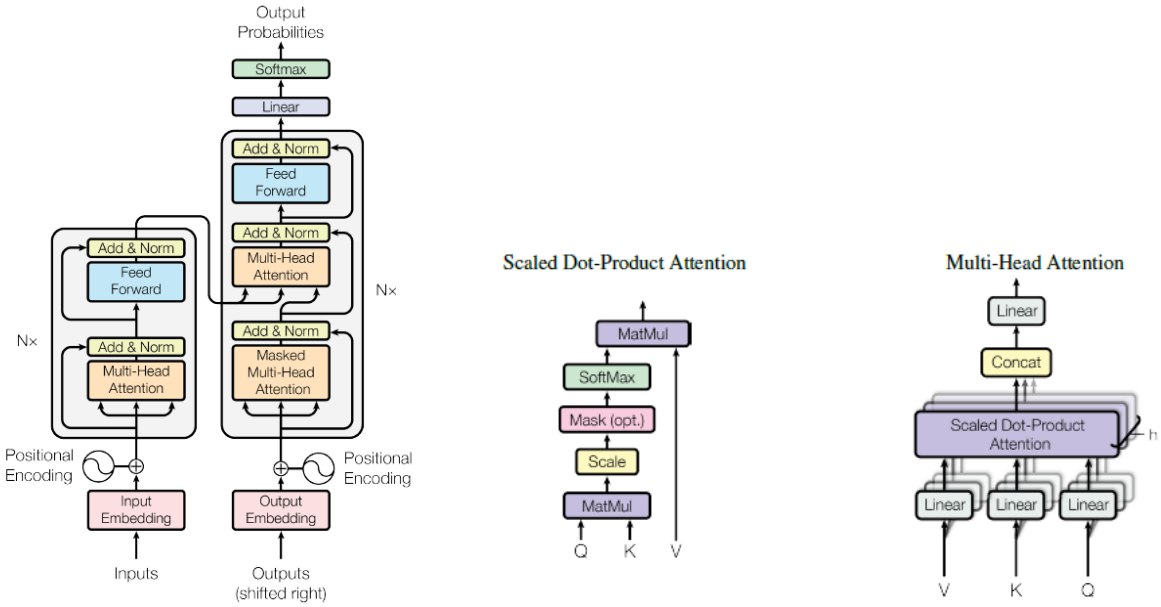


**Figure 2:** Structure of a standard transformer (left) and representation of attention computation as in (1) (right).

**Decoder** In translation tasks—or, more generally, in a sequence generation tasks—the decoder is responsible for generating the output sequence from the encoded input sequence. It receives the final hidden states from the encoder, which contain the contextualized representations of the input sequence. Then, it produces the output sequence step-by-step, attending to both the encoder's outputs and the previously generated tokens in the output sequence.

**Attention** It is the fundamental mechanism of the transformer model. An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

$$\mathit{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{d_{\mathrm{k}}}\right)\mathbf{V} \tag{1}$$

The encoder contains *self-attention* layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder. [5]

**Multi-head** Having multiple heads allows the model to learn multiple representations of the input data by using different sets of queries, keys, and values. Each head operates independently and captures different features or patterns from the input sequences. This is a fundamental point for the present work, because in principle it makes possible to feed a protein and some of its properties (3D structure, bio-chemical properties of its amino acids, etc.) into the transformer, and to find reflected those properties in the attention weights stored in the different heads of the transformer.

In this section, we introduced the transformer and all its parts for the sake of completeness and clarity. Nevertheless, since the present work is about interpreting *attention*—in other words, sequence-to-sequence tasks are not involved—we do not consider at all the decoding part of the transformer. As a consequence, from now on every time that attention will be mentioned, it will refer to the *encoder self-attention*.

## 1.3 ProtBert

ProtBert [7], introduced by Elnaggar and colleagues [3] as a part of the ProtTrans project [8], is based on Bert model, which was pre-trained on a large corpus of protein sequences in a self-supervised fashion. This means it was pre-trained on the raw protein sequences only, with no humans labelling them in any way [7].

Bert stands for Bidirectional Encoder Representations from Transformers. It was developed in 2018 by Devlin and colleagues [9]. Bert works with a fine-tuning approach, i.e., the model is pre-trained on unlabeled data over different tasks; then, in order to apply pre-trained language representations to downstream tasks, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. Each downstream task has separate fine-tuned models, even though they are initialized with the same pre-trained parameters. The advantage of this approach is that few parameters need to be learned from scratch. Pre-training is not carried out by using traditional left-to-right or right-to-left language models. Instead, a masked language model (MLM) is used, which randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked word based only on its context. Unlike left-to-right language model pre-training, the MLM objective enables the representation to fuse the left and the right context, making them bidirectional. [9]

ProtBert employs an architecture with 30 layers, each provided with 16 attention heads.

# 2 Materials and Methods

## 2.1 Attention Extraction

We chose to use ProtBert for our analysis. The model [7] we downloaded was pre-trained on UniRef100 [10], a dataset consisting of 217 million protein sequences. We chose it because it is easy to use, very popular, well documented and trained on a huge dataset. The same analysis could be applied to other transformers and datasets, though.

Proteins are fed into the model one by one. Each protein in the dataset is represented by a unique code in RCSB Protein Data Bank (PDB) [11]. We used the `Bio` package from `Biopython` [12] to extract the amino acid sequences, which are fed into the model and tokenized in form of upper-cased letters, according to the single-letter amino acid code convention [13]. Then, we get the output from ProtBert, and we store the attention returned from each head in form of a tensor. Hence, for each protein we get 30 tensors having shape (16, $l_{\text{prot}}$+2, $l_{\text{prot}}$+2), where 30 is the number of layers, 16 the number of heads in a single layer, while $l_{\text{prot}}$ is the length of the protein fed into the model. The addend 2 is due to the additional tokens [CLS] and [SEP], the first enabling the model to perform classification on textual data, and the second representing the separation between sentences. Attention given to those tokens is excluded, because at this phase we are only interested in token-level attention.

In other words, we get $30 \times 16 = 480$ square matrices, each entry being the attention given by one token in the sequence to another. The i-th row and the i-th column in a given matrix refers to the amino acid being in the i-th position along the peptide chain. The square matrices are not symmetrical. Indeed the sum over all the entries of the i-th *column* in a given matrix, represents the total attention *given to* the i-th token. Conversely, the sum over all the entries of the i-th *row* in a given matrix, represents the attention *given by* the i-th token to all the other tokens. The second sum is always 1, while the first sum is not—some tokens get more attention than others.

In order to better understand the power of self-attention mechanism, and its ability to detect patterns in the input, six experiments were carried out. Each experiment was performed on a set of peptide chains randomly extracted. To assure the repeatability of the experiments, we have always set the same random seed in every extraction. The experiments performed vary in sample size, maximum allowed length of the chains, and presence of a threshold on the attention. The thresholding here consists in setting to zero the attention values that are smaller than a given cutoff. To have a proper idea of how high a cutoff is in comparison with the attention values, one has to keep in mind that each row of any attention matrix returned by the model—considering also the attention from the tokens [CLS] and [SEP], which is actually excluded from the analysis—sums 1. Therefore, considering a threshold of 0.1, if a token gives to another token less than the 10% of its total attention, then that portion of given attention is suppressed. The threshold is set in order to suppress the "random background" in the attention that may hide some patterns. An example of a thresholded matrix is shown in Fig. 3.

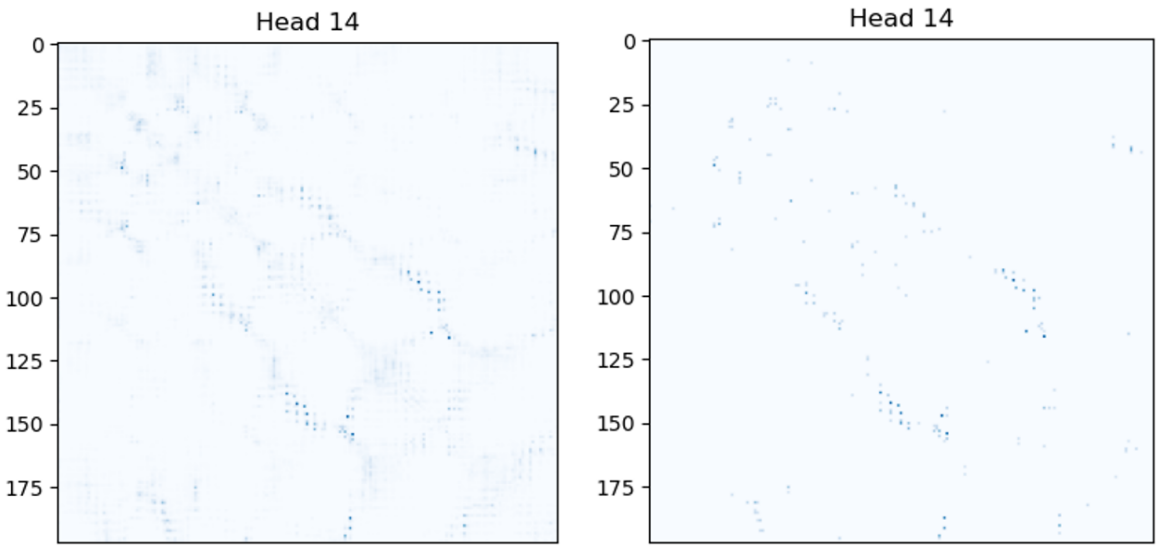


**Figure 3:** PDB ID: 6NJC. Attention matrix of head 14 in layer 30 without any threshold (left) and with a threshold of 0.1 (right).

The threshold of 0.1 is the same used by Vig et al. [1, p. 3]. This value produces very good results in their study, although it is not justified. Furthermore, a fixed cutoff applied to chains that can vary a lot in number of residues, may "penalize" the attention in the longer chains. The sum over each row of an attention matrix is indeed always 1. Thus, if a token gives attention to a lot of other tokens—as it happens in the longer chains—a large part of the attention it gives will be below the threshold and, as a consequence, suppressed. On the other hand, a token in a short chain gives its attention to few tokens, so the attention will be more concentrated and will not fall below the threshold. To deal with this distortion, we introduced another threshold, which depends on the length of each chain. Supposing we are in a random scenario, then each token gives a value of attention of $\frac{1}{\text{chain length}}$ to any other token. Hence, to be sure to suppress this "noise", we set the cutoff to the double of the random background, that is $\frac{2}{\text{chain length}}$. The experiments with the different features are resumed in Tab. 1.

| Threshold / Size / Max len | 0 | $\frac{2}{\text{chain length}}$ | 0.1 |
|---|---|---|---|
| 200 / 100 | Exp. 1 | Exp. 3 | Exp. 5 |
| 2000 / 500 | Exp. 2 | Exp. 4 | Exp. 6 |

**Table 1:** Experiment characteristics. **Size** is the number of peptide chains in the set, **Max len** is the maximum number of residues each chain can have, and **Threshold** is the value below which the attention is ignored.

The ID codes representing the proteins of our experimental sets have been fetched using the RCSB PDB Data API [14] through the package `rcsbsearchapi` [15]. Since a protein can have more than one chain, each represented with a letter, for each protein we kept the first chain in alphabetical order. We built the query for the search based on the following attributes:

- The polymer entity type is exclusively a protein: avoid to fetch molecules that are not proteins, e.g., nucleic acids.
- The minimum number of residues in the shortest chain of the proteins is 20: the computation of some of the properties that will be presented further requires a minimum of residues to be feasible and/or to make sense.
- The maximum number of residues in the longest chain of the protein is 100 or 500 (depending on the experiment): limit the use of computational resources.
- The protein has a PDB file available, from which many information are taken—i.e., the amino acids and the coordinates of the residues, etc.

Actually, setting a minimum length in the query is not enough to guarantee that all the chains really have that minimum number of residues. This happens because many chains contain molecules that do not really belong to it and that are not residues. Those molecules are called ligands [16], and the atoms they are made of are known as heteroatoms [17]. Ligands cannot be part of our analysis, so each chain is checked after its PDB file is been downloaded and all the heteroatoms are just discarded. Nevertheless, after this process, some chains may end with a number of valid residues smaller than the minimum set in the query. To deal with this issue, we set another cutoff to discard all the chains that do not reach a minimum number of valid residues. This cutoff is set to 15, as a compromise between the minimum number required to make feasible the computation of all the quantities that will be presented later on, and the attempt to keep even the smaller chains in, in order not to bias the analysis.

## 2.2 Attention Alignment

In order to verify the presence of a particular property, or feature, in a set $X$ of peptide chains, we compute the "alignment" of that property with the attention patterns. This attention alignment is computed as [1]:

$$p_\alpha(f) = \frac{\sum_{x \in X} \sum_{j=1}^{|x|} \sum_{i=1}^{|x|} f(i,j) \cdot \alpha_{i,j}(x)}{\sum_{x \in X} \sum_{j=1}^{|x|} \sum_{i=1}^{|x|} f(i,j)} \tag{2}$$

where, for a given attention head,

$\alpha_{i,j}$ represents the attention from token $i$ to token $j$ for the sequence $x$ in the dataset $X$.

$f(i, j)$ is an indicator function, i.e., a function that returns 1 if the property is present between tokens $i$ and $j$, and zero otherwise.

$p_\alpha(f)$ quantifies how much the property matches the attention distribution, averaged over all the attention matrices and peptide chains in the dataset.

## 2.3 Feature Collection

To handle the information about the features we used two main structures: the first one is a class created ad hoc to store physical and chemical features such as coordinates, pH, etc., named as `CA_Atom`. The other is the `pandas.DataFrame`, where we reorganize the data stored in the list of residues in a tabular structure. Given one residue, the information stored in the corresponding `CA_Atom` object is:

**name**: the amino acid name in single-letter code convention.

**idx**: the position of the residue in the sequence of the protein.

**coords**: the coordinates in the format $(x, y, z)$ of the $\alpha$-carbon, i.e., the central carbon atom of the residue.

**hydropathy**: the value of hydrophobicity of the residue according to the Kyte-Doolittle scale [18].

**charge**: the electric charge of the residue.

**volume**: the volume in Angstroms to the cube ($Å^3$).

In the data frame, in addition to this information, there are also:

**Charge Density**: the ratio between the charge of an amino acid and its volume.

**R-Charge Density**: the charge density considering only the volume of the R-group, obtained by subtracting the glycine volume, given that its R group is negligible.

**Local Charge**: the total electric charge in a triplet of residues.

**Isoelectric Point**: the pH at which a molecule carries no net electrical charge, for a single residue.

**Local Isoelectric Point**: the isoelectric point computed on a triplet of adjacent residues.

**Hydrophilicity**: according to the Hopp–Woods hydrophilicity scale [19].

**Surface Accessibility**: the Emini Surface fractional probability, representing the likelihood of a certain region of the protein to be exposed to interact with the surroundings. We used the data reported in `Bio.SeqUtils.ProtParamData` [20] (lines 270-274).

**Transfer Energy Scale**: the Janin interior-to-surface transfer energy scale, measuring the free energy of transfer of amino acid residues from the interior to the surface of a globular protein. This scale helps in understanding the hydrophobicity of the amino acids, which is crucial for protein folding and stability. We used the data reported in `Bio.SeqUtils.ProtParamData` [20] (lines 276-280).

**Flexibility**: the normalized flexibility parameters, known as B-value. We used the data reported in `Bio.SeqUtils.ProtParamData` [20] (lines 250-257).

**Local Flexibility**: variation of the corresponding method of `Biopython`, due to a bug (see Appendix A).

**Secondary Structure**: based on the occurrences of specific amino acids that tend to form some secondary structures more than others. For instance, the amino acids Glu, Met, Ala, Leu and Lys tend to the helix structure, Asn, Pro, Gly, Ser and Asp to the turn structure, etc. We used `secondary_structure_fraction()` from `Bio.SeqUtils.ProtParam` [21] (lines 326-348).

**Aromaticity**: boolean, depending on the presence of an aromatic ring in the amino acid.

**Essentiality**: boolean, depending on whether the amino acid is essential for a human organism or not.

**Amino Acid Classification**: side chain-based classification in four groups, depending on whether the amino acid R-group is non-polar, polar but uncharged, negatively charged, or positively charged.

For the network-based analysis, we created the class `Protein_id` to store information about the features of each protein. Hence, for each protein the information is stored in a `pandas.DataFrame` with the following categories:

**Protein_id**: the unique PDB code of the protein.

**first_ten_attention_scores**: given one property (contact, instability, Louvain communities, or $k$-means, see Section 2.4) of the protein, the ten highest attention alignment scores for that property are stored, together with the corresponding attention head where those scores are achieved. Because of this setting, for one protein we get as many data frames as the number of properties analyzed.

**molecular_weight**: the molecular mass of the peptide chain.

**aromaticity**: the relative frequency of the residues in a peptide chain, whose R-groups have an aromatic ring (Phenylalanine, Tryptophan, Tyrosine). We used `aromaticity()` from `Bio.SeqUtils.ProtParam` [21] (lines 138-149).

**flexibility**: the sum over the peptide chain of the array elements obtained using the function in Appendix A.

**isoelectric_point**: the pH of the protein surroundings at which the protein carries no net electrical charge. We used `isoelectric_point()` from `Bio.SeqUtils.ProtParam` [21] (lines 310-318).

**gravy**: the GRAVY, Grand Average of Hydropathy, according to Kyte and Doolittle [18]. It is computed as the sum of the hydrophobicity values of the residues in the protein chain, divided by the number of residues. We used `gravy()` from `Bio.SeqUtils.ProtParam` [21] (lines 199-218).

**secondary_structure_inclination**: based on the occurrences of specific amino acids that tend to form some secondary structures more than others. For instance, the amino acids Glu, Met, Ala, Leu and Lys tend to the helix structure, Asn, Pro, Gly, Ser and Asp to the turn structure, etc. We used `secondary_structure_fraction()` from `Bio.SeqUtils.ProtParam` [21] (lines 326-348).

## 2.4 Feature Maps

From the features described in the previous section, we built four types of binary feature maps. Examples of them can be seen in Appendix B, together with details about how they are computed.

**Contact Map** If the distance between two residues $i$ and $j$ is lower than 8 Å and there are at least 6 residues between them in the protein sequence, set $f(i, j) = 1$, and 0 otherwise. Those values of reference are of common use in literature [22, 1].

***k*-means Cluster Map** Established a number of four clusters—corresponding to the groups in the amino acid side chain classification—set $f(i, j) = 1$ if the residues $i$ and $j$ belong to the same cluster, and 0 otherwise. We exclude the coordinates of the residues when we use this method.

**Louvain Community Map** In the network representation of the data with NetworkX [23], having set the distances between residues as the edge weights for the modularity, and the residues as the nodes, set $f(i, j) = 1$ if the nodes $i$ and $j$ belong to the same community, and 0 otherwise.

# 3 Results and Discussion

## 3.1 Distribution of Attention

The percentage of attention given by the model to each amino acid changes considerably. Nevertheless, it must be taken into account that some amino acids occur more often than others. The attention received by the amino acids, indeed, is strongly correlated (0.98 in $[0, 1]$) with their relative occurrences, as shown in Fig. 4. Such result indicates that the amino acids receive attention exclusively proportionally to their frequency of occurrence along the chain.

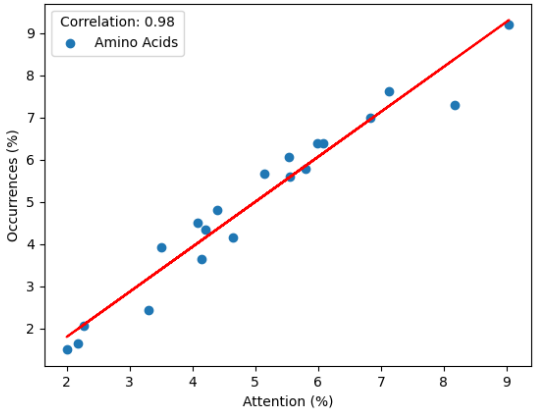


**Figure 4:** Percentage of attention given to each amino acid respecting the percentage of relative occurrences of each amino acid, for the experiment n. 2. The correlation coefficient of 0.98 suggests that no amino acid receives more attention than the others a priori. The data relative to the plot are resumed in Tab. 4 of Appendix C.

Speaking of how the model assigns attention to the amino acids with each one of its heads, it must be noticed that the threshold of 0.1 may be excessive in this case. In the majority of cases, it highlights the heads that already concentrate a high attention to a specific amino acids. In some cases, though, it introduces distortions. For instance, in the case of leucine (Leu), in the experiment n. 6 we found a head that gives the 100% of its attention to this amino acids. Another distortion is the one introduced in the case of glutamic acid (Glu). This is the only amino acid which a whole layer seems to specialize in. In this case, a threshold set to 0.1 "hides" this feature, that is preserved in case of a threshold proportional to the chain length instead. In Fig. 5 are shown the heatmaps of the two cases reported.

Generally speaking, the model does not specialize a single head in targeting any specific amino acids. Each amino acid tends to have at least two or three heads that specialize in it. In the experiment n. 2—no threshold—6 amino acids have at least one head that gives them more than the 20% of its attention. In the experiments n. 4 and 6 the number raises respectively to 16 and 20 amino acids.

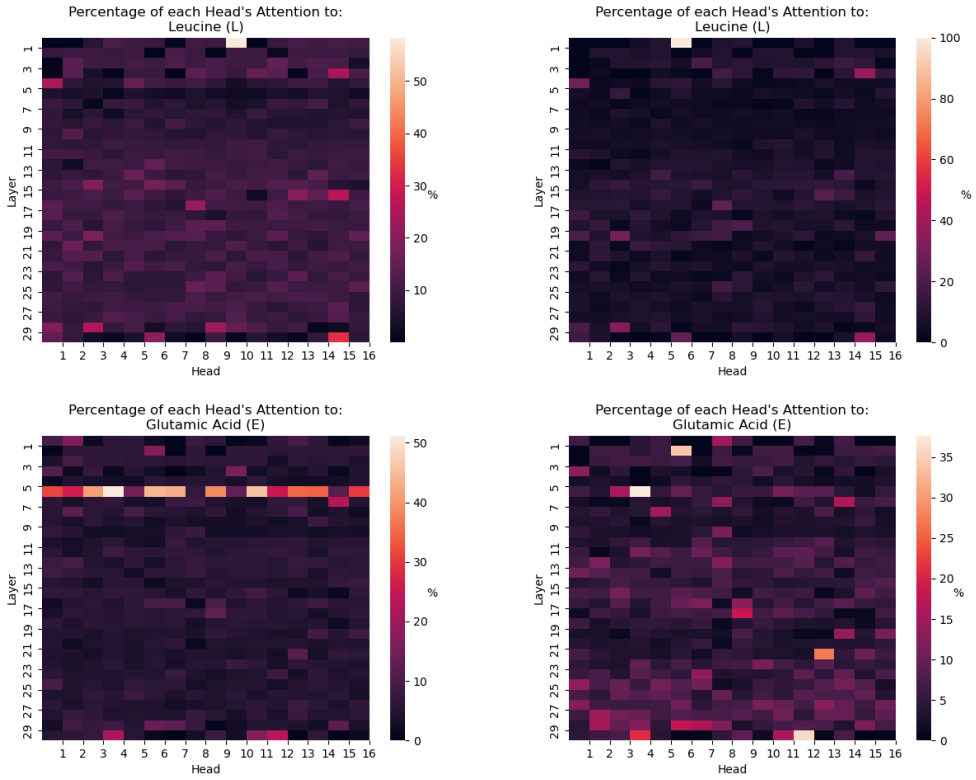


**Figure 5:** Average percentage of the attention of each head to the amino acids leucine (top) and glutamic acid (bottom), in the experiments n. 4 (left) and n. 6 (right). The heatmaps show that a threshold of 0.1 introduces distortions, such as returning the 100% of the attention of one head to one amino acid (leucine), or hiding the layer that specializes in glutamic acid.

To check whether the percentage of attention given to each amino acid depends on the set of chains that is analyzed, we computed the Pearson correlation coefficients between the results that we got from the experiments 1 and 2, 3 and 4, and 5 and 6—keeping in mind that odd numbers refer to less and shorter chains, while even numbers refer to more and longer chains. The experiments are paired using the same attention threshold. The resulting correlation coefficients are resumed in Tab. 2. Since the coefficients indicate a high correlation, and given that the two sets of chains are almost non-overlapping—only 2 chains are in both sets—we can conclude that the way heads give attention to the amino acids is neither random nor dependent on the set analyzed.

It remains to check whether the distribution of attention across the amino acids is compatible with the evolutionary information of the proteins. From this information, it's possible to infer the probability that a given amino acid in a specific position along a chain can be replaced by a different amino acid. A substitution matrix collects those probabilities for each couple of amino acids. If ProtBert is able to learn the underlying relationships between

| Exp. Pairs | PCC |
|---|---|
| 1-2 | 0.86 |
| 3-4 | 0.80 |
| 5-6 | 0.72 |

**Table 2:** Pearson correlation coefficients (PCC) between the percentage of attention of the heads to each amino acid, across different experiments. The results indicate a good correlation, meaning that the quantity is independent of the set of chains analyzed.

amino acids, we should be able to see those relationships reflected in the attention similarity between pairs of amino acids. The attention similarity is defined as the Pearson correlation coefficients between pairs of matrices like the ones shown in Fig. 5. We can use again the Pearson correlation to compare the attention similarity with the substitution matrix BLOSUM62, which is the de facto standard for protein alignment [24].* Both attention similarity and BLOSUM62 are shown in Fig. 6. They turn out to be highly consistent, as evidenced by the PCCs, which are higher than 0.74 in five of the six experiments. We conclude that:

- The distribution of attention across the amino acids does reflect the evolutionary-based substitution relationships between amino acids.
- It is able to do it whatever the threshold on the attention, the length of the peptide chains, or the number of analyzed chains—in reference to the combinations we studied.

## 3.2 Attention-Contact Alignment

### 3.2.1 Experiment Comparison

In all the experiments, ProtBert was able to specialize one single head to align with the contact maps. The best head always turns out to be 14-30. The heatmaps showing the values of attention alignment in the six experiments are shown in Fig. 7.

The values of attention alignment of the head 14-30 are plotted in Fig. 8. We can see that thresholding is fundamental to highlight the attention head that better specializes in identifying contacts between residues. The performances of the two thresholds, from this point of view, are similar. We figured out by comparing attention head 14-30 with and without thresholding, that for longer chains—more than 200 residues—a fix cutoff of 0.1 often cuts off attention that actually aligns with the respective contact map. Nevertheless, in order to identify the most aligning head, cutting off attention not contact-related seems to be more important than preserving attention that actually aligns with contact maps.

*BLOSUM (BLOcks SUbstitution Matrix) is a class of substitution matrices used for sequence alignment of proteins. [25] BLOSUM62 is the matrix built using sequences with less than 62% similarity—that is, sequences with more than 62% identity were clustered. [26]

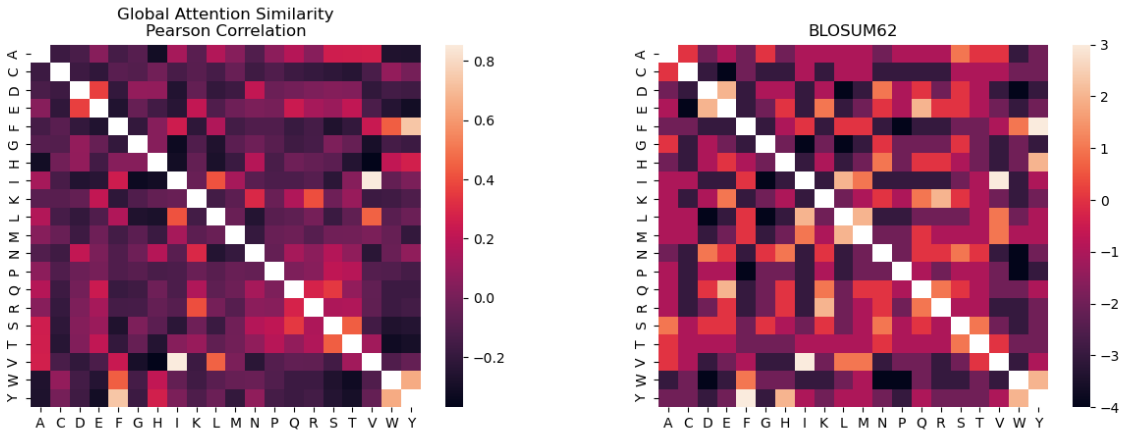


**Figure 6:** Pairwise attention similarity from exp. 6 (left) and BLOSUM62 substitution matrix (right). Both diagonals store no values, because they would be misleading in the computation of the Pearson correlation coefficient, which is equal to 0.77.

In the second place, experiments performed on more and longer peptide chains allow to better specialize a single attention head in identifying contacts. However:

- This is true only when a threshold on the attention is applied.
- Head 14-30 is able to specialize even with much smaller data sets with shorter chain. The small difference in the performance ($\sim$ 3%) does not compensate for the much longer time required when using a big dataset.

### 3.2.2 Alignment across the Layers

An interesting pattern can be noticed in the attention alignment of the contact maps, when the values of attention alignment of the heads belonging to the same layer are averaged together. We can distinguish in the distribution two peaks. The distinctness of the two peaks is rather independent of the sample size and chain length. On the other hand, the two peaks are more distinguishable when a threshold on the attention is set.

First of all, it must be noticed that, even though the head that better aligns its attention with the contact maps is located in the layer number 30, the layer itself does not align as well as the best performing. This happens because the encoder specializes only one of its heads in the last layer in identifying contacts, while all the other heads are not used for that. Conversely, in our article of reference [1, p. 7] they find that the last layer has the highest attention alignment. This discrepancy may be due to the different model used, although both models are Bert-based. They do not even observe, indeed, the two-peaks pattern as we do.

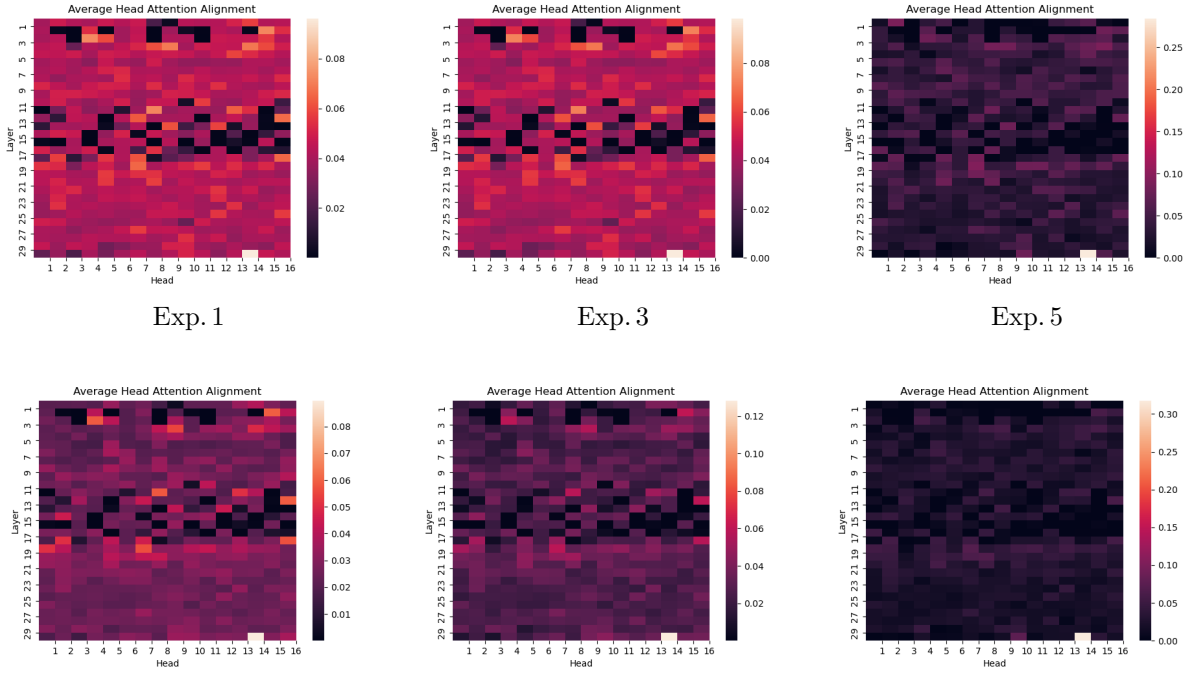


**Figure 7:** Heatmaps of the attention-contact alignment of each head in the different experiments, averaged over the respective chain set. The upper row refers to the experiments performed on 200 peptide chains with a maximum number of residues of 100 each. The lower row refers to the experiments performed on 2000 peptide chains with a maximum number of residues of 500 each.

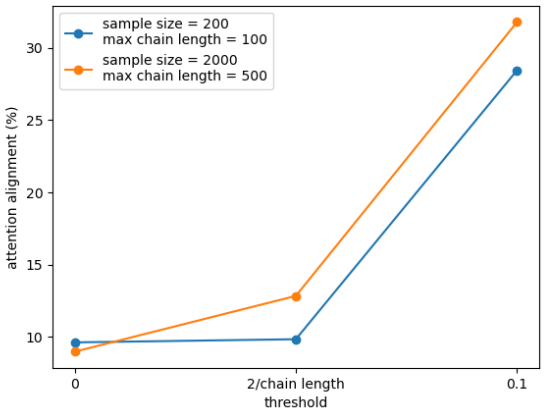


**Figure 8:** Percentage of attention-contact alignment of head 14-30, which is the one that better aligns in all the experiments. The data plotted are resumed in Tab. 5 of Appendix C.

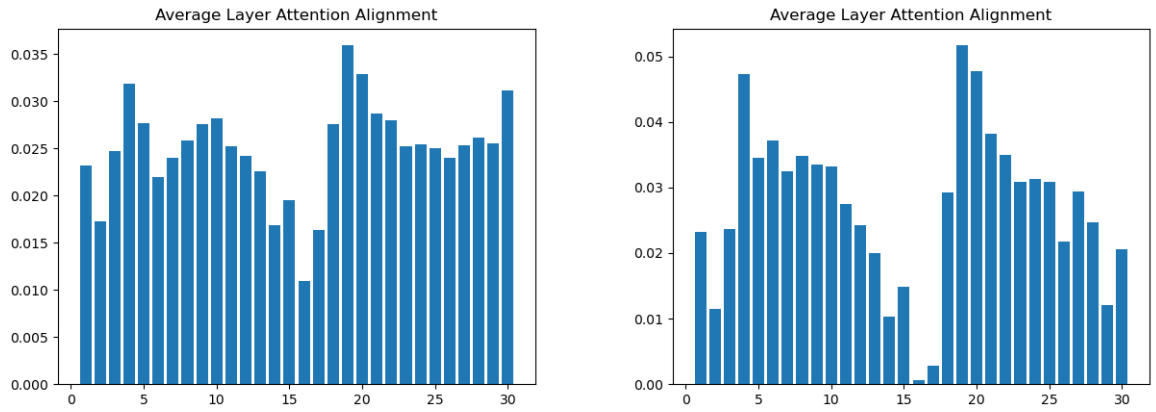


**Figure 9:** Attention-contact alignment across the layers in the experiments n. 2 (left) and n. 6 (right). The two peaks can be distinguished more easily for the experiment where thresholding was performed.

The two peaks that we can see in Fig. 9 can be explained by looking at Fig. 21 and Fig. 22 in Appendix D. Those are the plots of the averages of the attention matrices in each layer, without any threshold and with a threshold of 0.1 applied, respectively. We can't get this kind of plot for the whole experimental set, because the attention matrices from different chains have different sizes and cannot be averaged together. Therefore, we just pick a chain. Even if not representative of the whole set, we can observe some patterns in how the model assigns attention to the tokens in the different layers. For instance, in the first layers—from the $1^{st}$ to the $5^{th}$, but also from the $7^{th}$ to the $9^{th}$—we can observe oblique lines that are not diagonal, which means that every token gives attention to other tokens at a fixed distance along the chain. Another observation is that the layers showing the lowest attention alignment—the $16^{th}$ and the $17^{th}$—are the ones that give most of the attention along the diagonal, that is, each token gives attention to itself and to the tokens adjacent to it. We expect to have such a low attention alignment in those layers, because the diagonal is always excluded from the contact map (see Fig. 18). Generally speaking, vertical lines are present in the average matrices of all the layers. Those lines are often cut off from the thresholding. That's what produces the difference between the two plots in Fig. 9, because thresholding enhances the alignment of those matrices that aligns not because of the vertical lines. At the same time, thresholding causes a drop in the alignment of the layers that aligns just because of those vertical lines. Anyway, in both cases, looking at the attention alignment per layer helps to understand how ProtBert assigns attention in each layer. Conversely, it is not useful to identify neither a specific layer nor a specific head that specializes in detecting contacts between residues.

## 3.3 Attention Alignment with the Feature Maps

In the previous section we thresholded the attention in order to better highlight a specific head, that alings much stronger than the others. Conversely, we have no hints about such a strong alignment in one single head with respect to the other feature maps. For this reason, applying a threshold in this phase could be detrimental, because it may hide precious information. Therefore, from now on the analysis will refer exclusively to the dataset and the results from the experiment n. 2 (large dataset with long chains, and no threshold applied on attention).

### 3.3.1 *k*-means Clusters

The attention alignment with the feature map got from the $k$-means algorithm turns out to be quite high across the vast majority of the attention heads, as shown in Fig. 10. This could be due to the high portion of the feature map that is "turned on" in general—see an example in Fig. 20—and to the very small sizes of the connected regions. Keeping in mind that a link between two residues is established if they are part of the same cluster, this sparsity of the connected regions means that the residues belonging to the same clusters tend not to be grouped in large contiguous segments on the peptide chain.

Generally speaking, the systematically high attention alignment does not necessarily imply a high predictive power of the algorithm for any attention head. However, some heads seem to perform better than the others, and it could be interesting to focus on them.

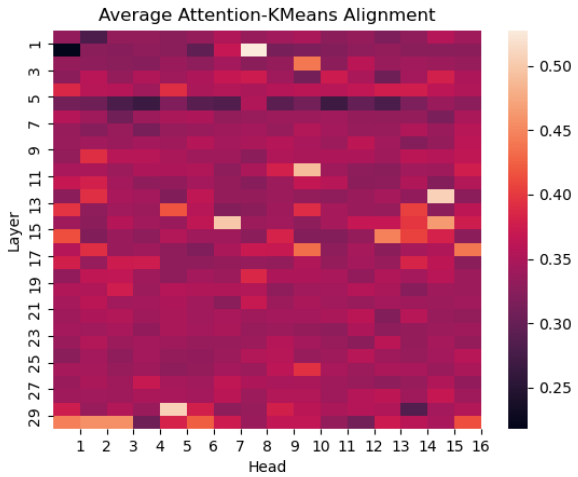


**Figure 10:** Attention alignment with the $k$-means cluster map in the experiment n. 2.

### 3.3.2 Louvain Communities

Fig. 11 shows that some attention heads aligns significantly better than others with the feature maps obtained from the application of the Louvain algorithm. It must be noticed that those heads are located mostly in the initial and central layers. As already mentioned in Section 3.2.2, those layers tend to assign most of the attention along the diagonal. The high attention alignment may then be explained by diagonal patterns in the feature maps of the proteins. Such possibility is indeed confirmed by the example of feature map reported in Fig. 19. It is evident that it shares the same pattern as the binary contact map (see Fig. 18). We expected such a result, given that the only weight we put on the edges of the network is the euclidean distance between the residues. However, we don't observe the head 14-16 to show the highest attention alignment, as it is for the contact maps.

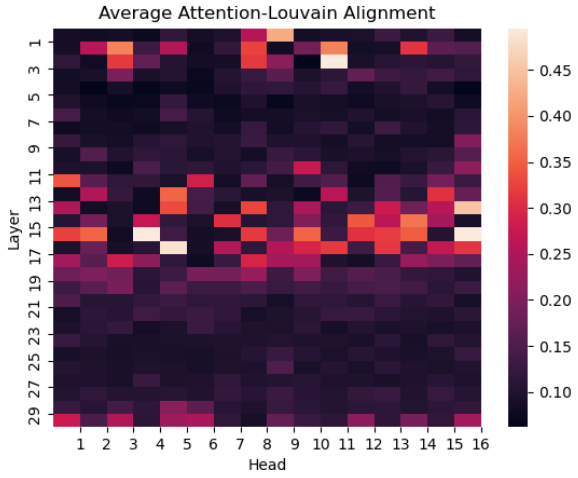


**Figure 11:** Attention alignment with the Louvain community map in the experiment n. 2.

# 4 Attention Alignment per Feature

As reported in the previous sections, the attention of the model does not align with the maps coming from the protein features as good as it does with the contact maps. In particular, the maps that we get using the $k$-means clustering and the Louvain algorithm embrace a lot of features. Thus, it could be interesting to investigate if and how the attention distribution —and, as a consequence, the alignment—depends on the values of those features. In other words, we want to check if the attention alignment gets better or worse when we analyze proteins with different values of the same feature, considering one feature at a time. The focus is shifted from the single residues inside the chain to the protein as an entity itself, considering its chemical and physical properties.

## 4.1 Principal Component Analysis

We start investigating which features in a set of proteins capture the most information of the data. We used the same set we performed the experiments 2, 4 and 6 on, made up of 1953 proteins. We performed a PCA over that set, considering the features reported in Section 2.3. The results shown in Fig. 12 indicates that the chain length—that is, the number of residues—is the main feature in the first principal component, suggesting that it could catch more information than the others.

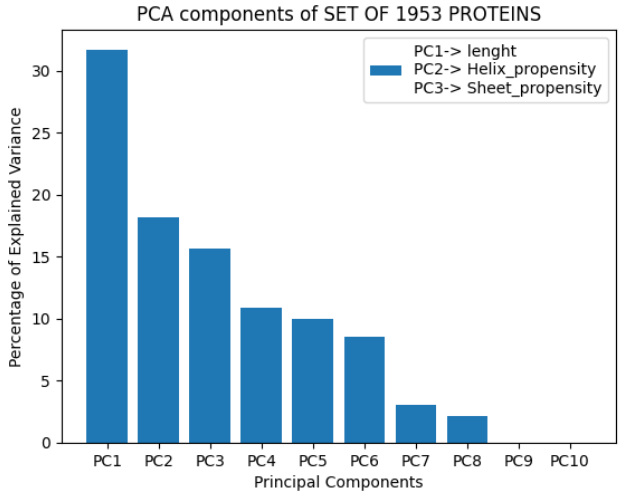


**Figure 12:** Principal component analysis (PCA) performed over a set of 1953 proteins. The first component stands out, with the feature "length" being its main contributor. See Tab. 3 for more details about the PCA results.

| Principal component | Percentage of variance explained | Highest loading feature |
|---|---|---|
| PC1 | 31.7 % | Length |
| PC2 | 18.2 % | $\alpha$-helix tendency |
| PC3 | 15.6 % | $\beta$-sheet tendency |

**Table 3:** Percentage of variance explained by each principal component, with the feature presenting the highest loading for each component.

## 4.2 Length vs. Attention Alignment

Let's see how the alignment of the attention with the feature maps changes when computed on proteins with different lengths.

For consistency of the analysis, we take a homogeneous dataset with respect to the protein length feature, without favoring a specific set of proteins, otherwise the different occurrence density across the bins would be misleading. The final set is made up of 536 proteins, distributed as shown in Fig. 13. For each protein in the uniform set, we took the first ten attention heads that achieved the highest attention alignment, both with the $k$-means and the Louvain maps. The results are shown in the Figures 14 and 15.

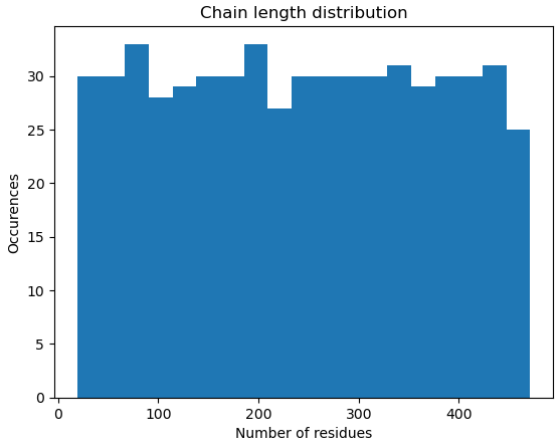


**Figure 13:** Distribution of the proteins in a set of 536, as a function of the number of residues.

The zero slope of the distribution in Fig. 14 means that the attention alignment that we get with the $k$-means feature map does not depend on the chain length. Keeping in mind what already discussed in Section 3.3.1, that leads to conclude that the $k$-means algorithm performs the same way independently of the protein length. Moving to the Louvain algorithm, in Fig. 15 we can observe a direct correlation between the minimum scores of attention alignment and the chain length. The higher average values of attention alignment for longer chains in the selected heads indicate a better behavior of the Louvain algorithm in the formation of the communities when the number of residues in the chains is higher.

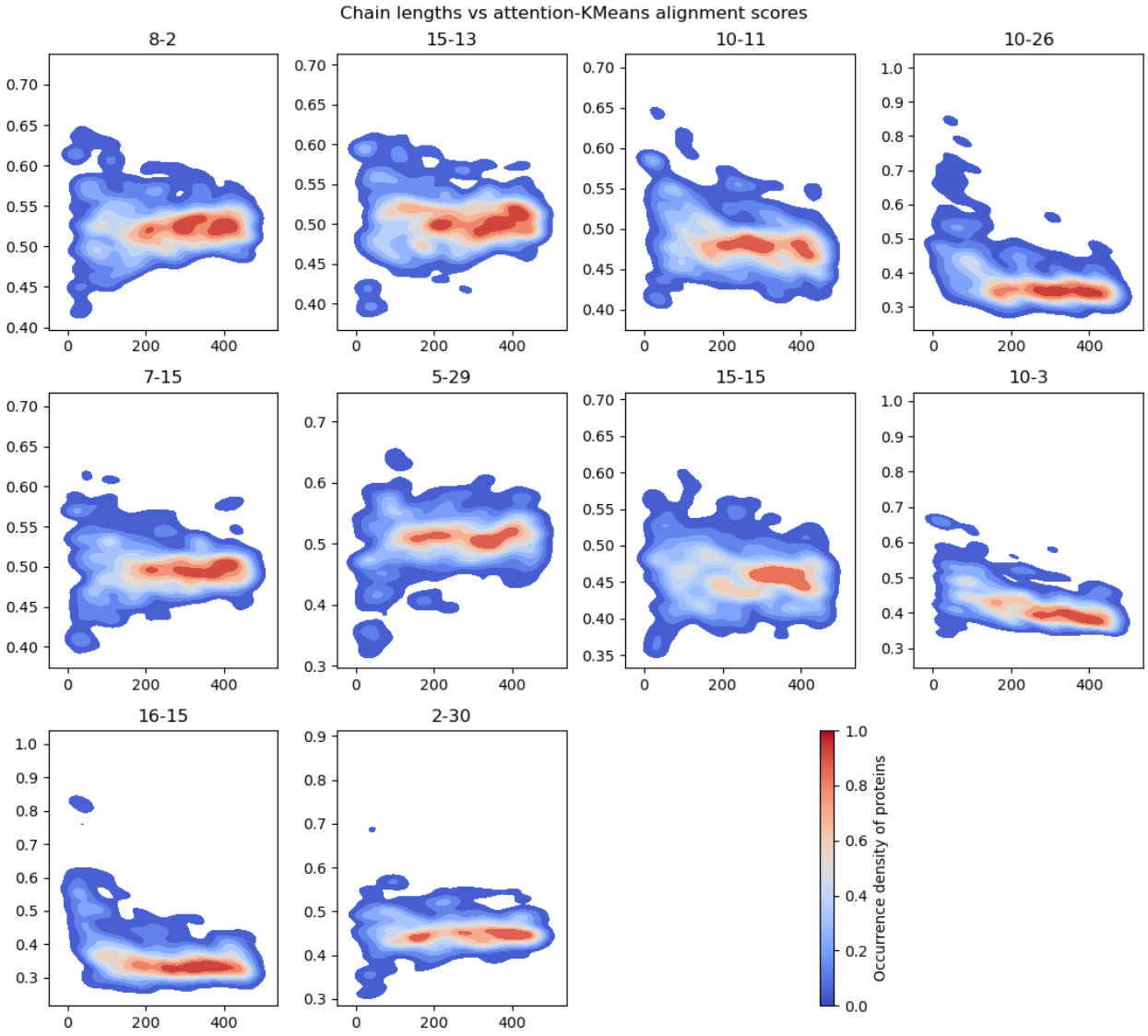


**Figure 14:** Attention alignment scores of a set of proteins as a function of the number of residues, in the attention heads that achieved the ten highest overall alignment scores. The alignment considered is with the feature maps got with the $k$-means clustering. The set is uniformly distributed according to the number of residues.

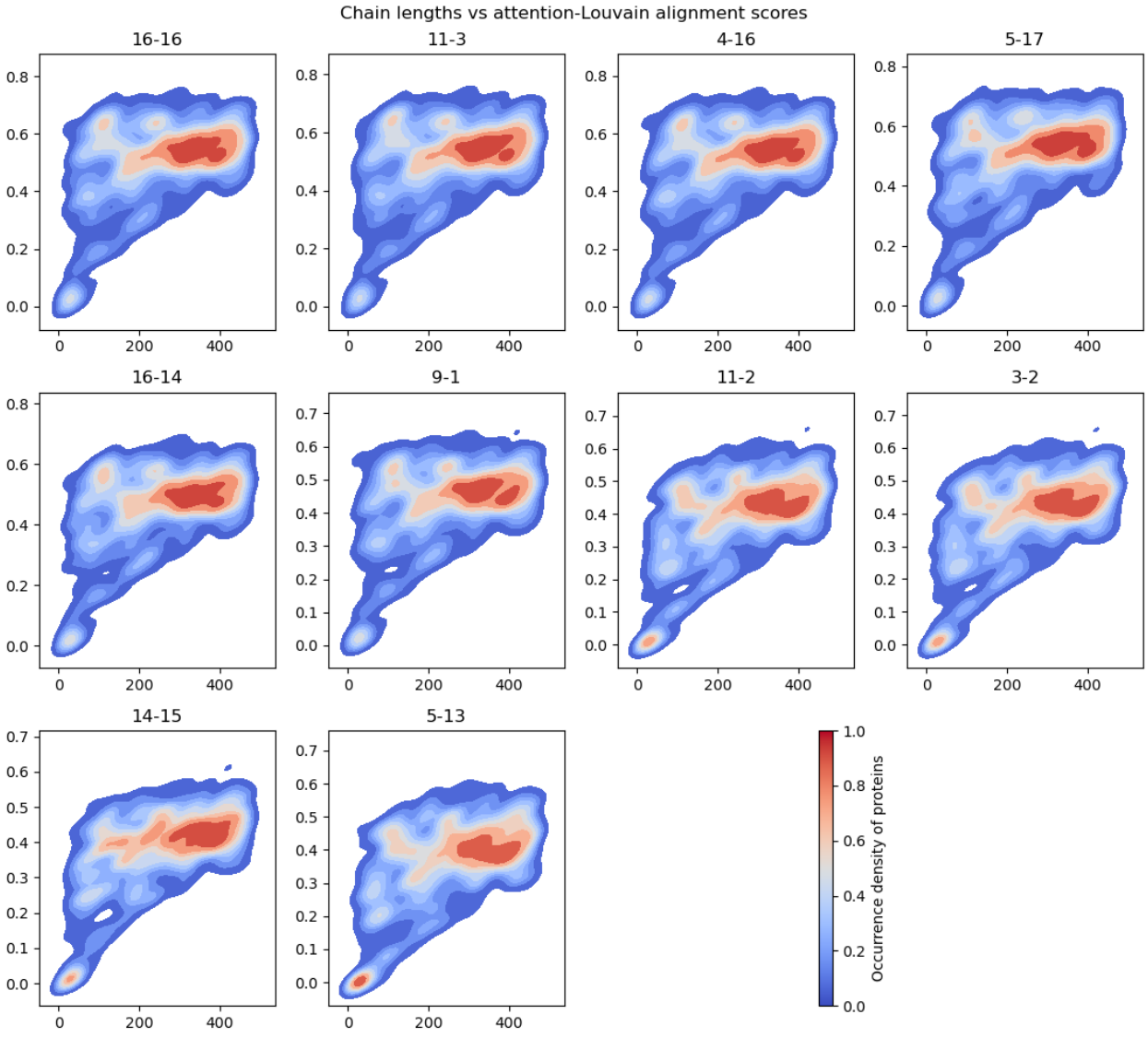


**Figure 15:** Attention alignment scores of a set of proteins as a function of the number of residues, in the attention heads that achieved the ten highest overall alignment scores. The alignment considered is with the feature maps got with the Louvain algorithm. The set is uniformly distributed according to the number of residues.

# 5 Conclusions

The results that we obtained with ProtBert about the attention distribution, and the relationship between attention and contact in sets of proteins, agree with our expectations.

The percentage of attention given to each amino acid depends exclusively on the occurrences of the amino acids themselves. The distribution of attention to the specific amino acids across the attention heads does not reveal a unique head specializing on one amino acid, but rather two or three heads giving more than the 20% of their attention to one amino acid. This behavior is more evident when a threshold is applied on attention, better if proportional to the peptide chain length, so as to not cutting off too much attention. The way the heads give attention to each amino acid turned out to be neither random nor dependent on the set analyzed. Therefore, some attention heads do target specific amino acids. This is further confirmed by the strong correlation of the attention similarity with the substitution matrix BLOSUM62, in agreement with the analysis made by Vig and colleagues. We conclude that ProtBert is able to capture the evolutionary-based substitution relationships between amino acids. What just said happens whatever the threshold on the attention, the length of the peptide chains, or the number of analyzed chains.

What we found out about the attention-contact alignment is also in agreement with the results claimed by Vig and colleagues. It turned out that the attention head 14 of the layer 30 specializes in identifying contacts between residues, with the alignment ranging from 9.0% to 31.8% across the experiments performed, with all the other heads performing significantly worse.

Moving to the attention alignment with the feature maps generated with the $k$-means and Louvain algorithm, we conclude that:

- Even though some heads have higher alignment scores than others, it was not possible to identify one or more heads where the clusters found with the $k$-means algorithm were able to align significantly better than in other heads. We believe that the algorithm is too simple to be able to capture significant dependencies between the features analyzed.

- The alignment scores achieved using the Louvain algorithm indicate that the method is able to find communities of residues in the chains that align with the attention matrices in the middle layers. However, the attention alignment with the head 14-16—the one that better captures the contacts between residues—is not particularly high.

Finally, the PCA indicates the number of residues in each chain as the feature carrying the most information. The attention alignment scores referred to the $k$-means algorithm do not depend on the number of residues in the proteins analyzed. Conversely, the overall attention alignment referred to the Louvain algorithm gets better as the number of residues increases.

# 6 Possible Further Developments

While working on the project, we discovered many potential applications of protein analysis. Most of them are related to the quality and quantities of prior knowledge regarding their biochemical properties. Our visualization tools for both plots and data structures help us to understand and see some analogies between the methods we used.

In Fig. 16 the protein with PDB code 6NJC is represented. That representation was achieved by transposing the information collected in the data frame of the amino acids in a network—using mainly methods from NetworkX [23]—and taking the data for the edges from the binary contact map (Fig. 18). The nodes represents the residues. Their positions are defined according to the projection of the data in the PCA space. On the axes are places the highest loading features in the first and second principal components. The node colors are determined according to the $k$-means clustering.

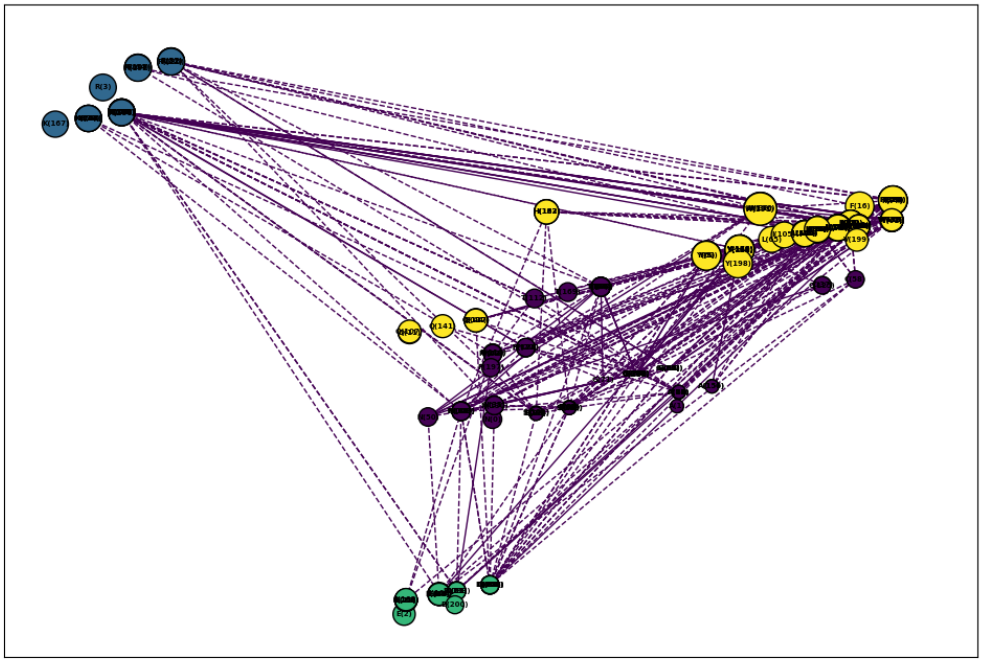

**Figure 16:** PDB ID: 6NJC. Network plot of the protein. The nodes are placed according to their coordinates in a system of reference defined by the first two principal components. The node colors follow a four kernels $k$-means clustering.

In Fig. 17 the same protein is represented according to the Kamada-Kawaii template for

the position estimation. Colors are chosen according to the Louvain partitioning method.

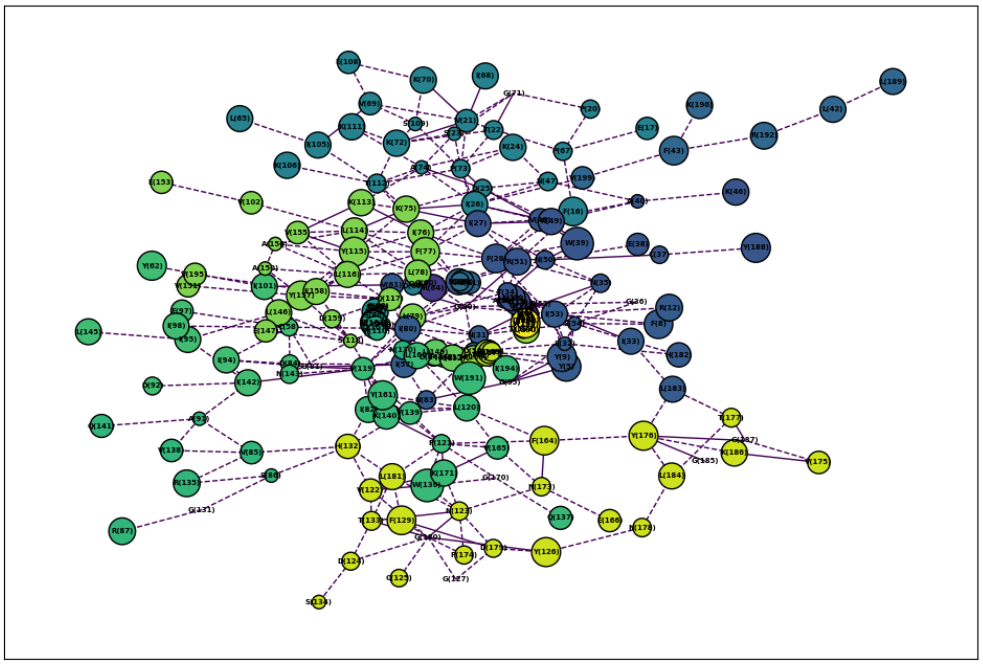

**Figure 17:** network plot of 6NJC protein following kamada-kawaii layout, color respecting louvain partitions

These clusters perform well enough to see a certain similarity between nodes close to each other. There are a lot of similar features to analyze in that way, to enhance the links between nodes through the Louvain partitions, or to target a certain property of the residues to improve the $k$-means clusters. However, such analyses requires a solid and deep knowledge of protein biochemistry, or at least the existence of a ground truth, in order to evaluate the correctness of the processes and results achieved.

# Appendices

## A Flexibility Issue

We found an inaccuracy in the calculation of the flexibility of a sequence of residues in the script `Bio.SeqUtils.ProtParam.py`. In the original code the residue in the center of the window was not included in the computation of the sequence flexibility. Furthermore, the computation of the flexibility for a sequence with the same size as the window was not provided. We fixed the function, adding a method to handle the borders.

To support our observation, issues relative to this problem are open on the GitHub page of `Biopython` [27].

```python
def custom_flexibility(protein_sequence_ns) -> list:
    """Calculate the flexibility according to Vihinen, 1994.
    No argument to change window size because parameters are specific for a
    ↪  window size of 9. The parameters used are optimized for determining
    ↪  the flexibility.
    Borders are set to 0 since no flexibility cannot be computed.
    """
    protein_sequence_ns = str(protein_sequence_ns.replace(' ', ''))
    protein_lenght = len(protein_sequence_ns)
    flexibilities = Bio.SeqUtils.ProtParamData.Flex
    # Flex are B-values, normalized flexibility values
    window_size = 9
    weights = [0.25, 0.4375, 0.625, 0.8125, 1]
    scores = []

    for i in range(protein_lenght - window_size + 1):
        subsequence = protein_sequence_ns[i : i + window_size]
        score = 0.0

        for j in range(window_size // 2):
            front = subsequence[j]
            back = subsequence[window_size - j - 1]
            score += (flexibilities[front] + flexibilities[back]) *
            ↪  weights[j]

        middle = subsequence[window_size // 2 + 1]
        score += flexibilities[middle]

        scores.append(score / 5.25)
```

```
        flexibility_border = [0.0, 0.0 , 0.0, 0.0] + scores + [0.0, 0.0, 0.0,
        ↪  0.0]

    return flexibility_border
```

# B Feature Maps

The feature maps that we used for our analysis were built with the following methods:

**Contact** The data on the coordinates of the residues were taken from the PDB-format files of the proteins. Heteroatoms were not considered. The distances between each couple of residues were computed, inverted—in order to get a proximity map—and normalized. The resulting map was thresholded to keep only the interactions that satisfied the given requirements, as shown in Fig. 18.

***k*-means Algorithm** We set the number of kernels to four, just like the number of groups in the amino acid side chain classification. In the application of the algorithm, we excluded from the list of features the side chain classification of the amino acids and the residue coordinates.

**Louvain Algorithm**

```
    def add_louvain_community_attribute(
        G: nx.Graph,
        weight_of_edges: str,
        resolution: float,
    ) -> tuple[nx.Graph, dict]:
        [...]
        partitions = nx.community.louvain_communities(
            G, weight=weight_of_edges, resolution=resolution,
        )
        [...]
```

As shown in the upper box, the function that runs the Louvain algorithm needs the feature data frame to be reorganized as a `networkx.Graph`. In our analysis, the distances between residues were set as parameter `weight_of_edges`. They were used to find the communities through the function `nx.community.louvain_communities`. As resolution we set the ratio of actual groups of amino acids in the protein over the total.

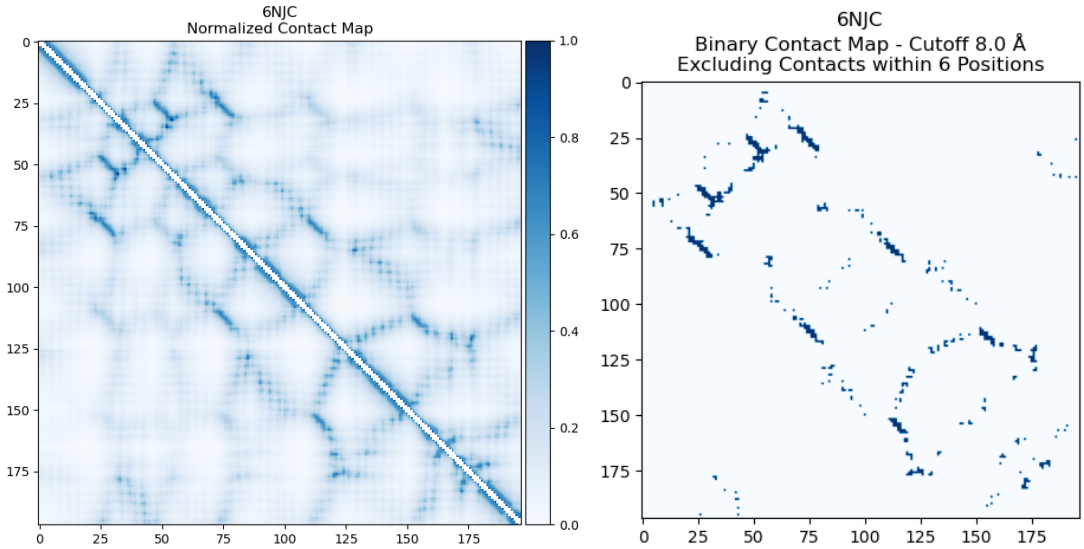


**Figure 18:** PDB ID: 6NJC. Normalized proximity map (left), and contact map (right), used as an indicator function in Eq. 2.

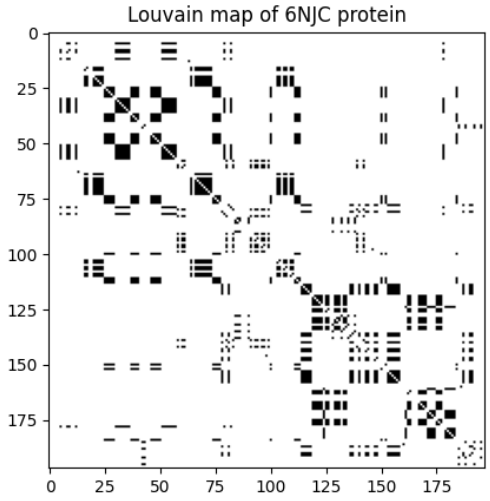


**Figure 19:** PDB ID: 6NJC. Louvain community map, used as an indicator function in Eq. 2. The black regions correspond to 1, the white ones to 0.

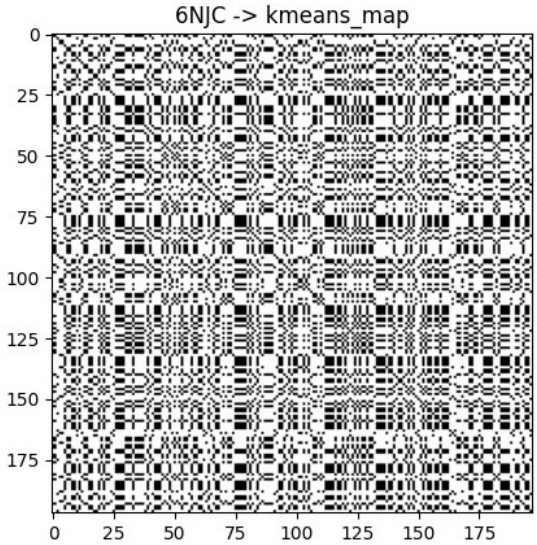


**Figure 20:** PDB ID: 6NJC. $k$-means cluster map, used as an indicator function in Eq. 2. The black regions correspond to 1, the white ones to 0.

# C Tables

| Amino Acid | Attention (%) | Occurrences (%) |
|---|---|---|
| Leu (L) | 9.0 | 9.2 |
| Gly (G) | 8.2 | 7.3 |
| Ala (A) | 7.1 | 7.6 |
| Val (V) | 6.8 | 7.0 |
| Glu (E) | 6.1 | 6.4 |
| Ser (S) | 6.0 | 6.4 |
| Asp (D) | 5.8 | 5.8 |
| Ile (I) | 5.5 | 5.6 |
| Lys (K) | 5.5 | 6.1 |
| Thr (T) | 5.2 | 5.7 |
| Phe (F) | 4.6 | 4.2 |
| Arg (R) | 4.4 | 4.8 |
| Asn (N) | 4.2 | 4.3 |
| Tyr (Y) | 4.1 | 3.6 |
| Pro (P) | 4.1 | 4.5 |
| Gln (Q) | 3.5 | 3.9 |
| His (H) | 3.3 | 2.4 |
| Met (M) | 2.3 | 2.1 |
| Cys (C) | 2.2 | 1.6 |
| Trp (W) | 2.0 | 1.5 |

**Table 4:** Percentage of attention given to each amino acid vs. percentage of relative occurrences of each amino acid, for the experiment n. 2. Experiments n. 4 and n. 6 provide similar data. Rows are sorted by the percentage of attention received. The data are plotted in Fig. 4.

| Highest att. alignment (%) | Experiment |
|---|---|
| 31.8 | 6 |
| 28.5 | 5 |
| 12.8 | 4 |
| 9.8 | 3 |
| 9.6 | 1 |
| 9.0 | 2 |

**Table 5:** Percentage of attention alignment of the heads with the highest attention alignment for each experiment. All the values are referred to the head 14-30, which is the one that better aligns in all the experiments. The data are plotted in Fig. 8.

# D Averages of the Attention per Layer

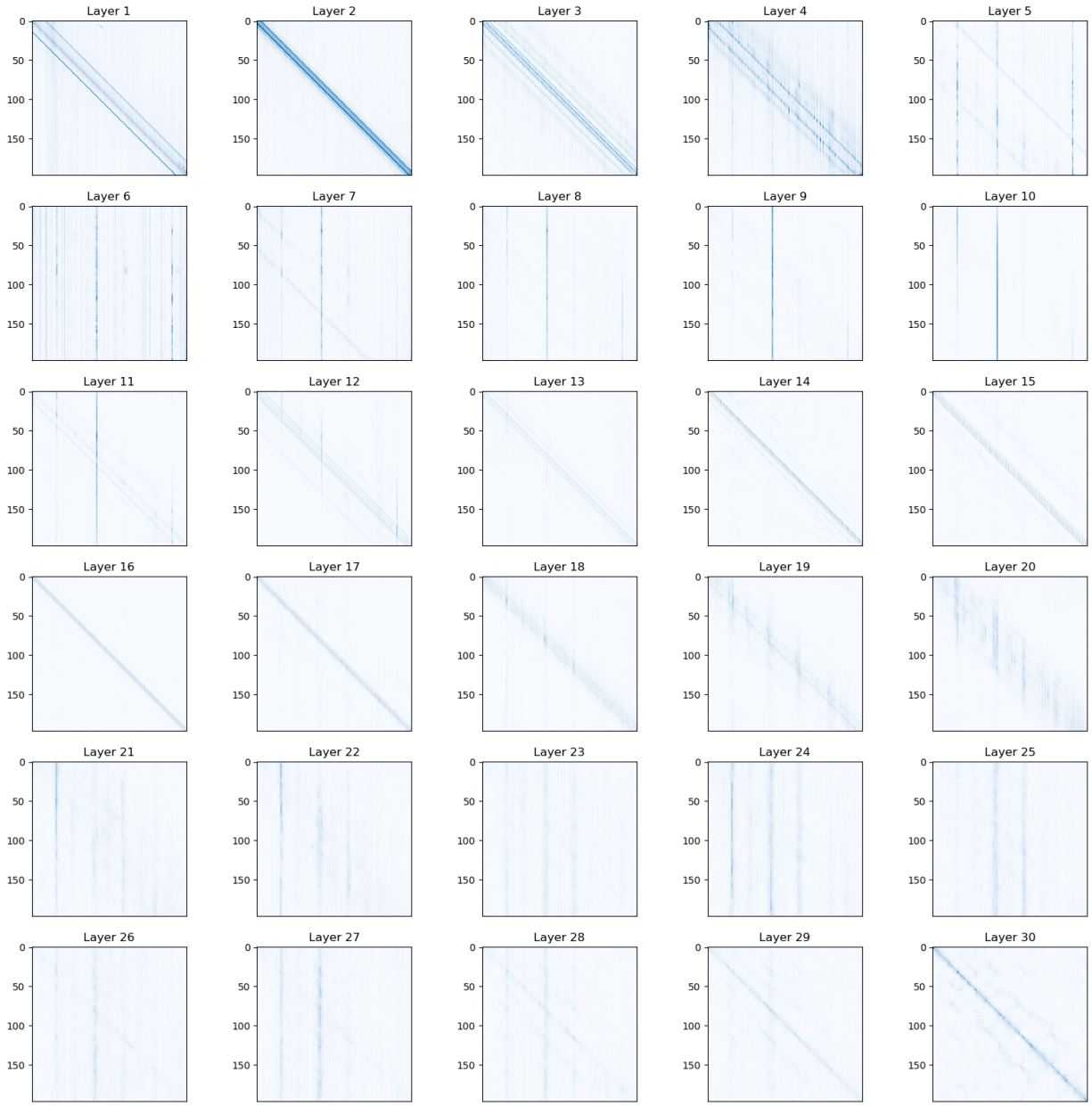


**Figure 21:** PDB ID: 6NJC. Averages of the sixteen attention matrices of each layer, with no threshold applied.

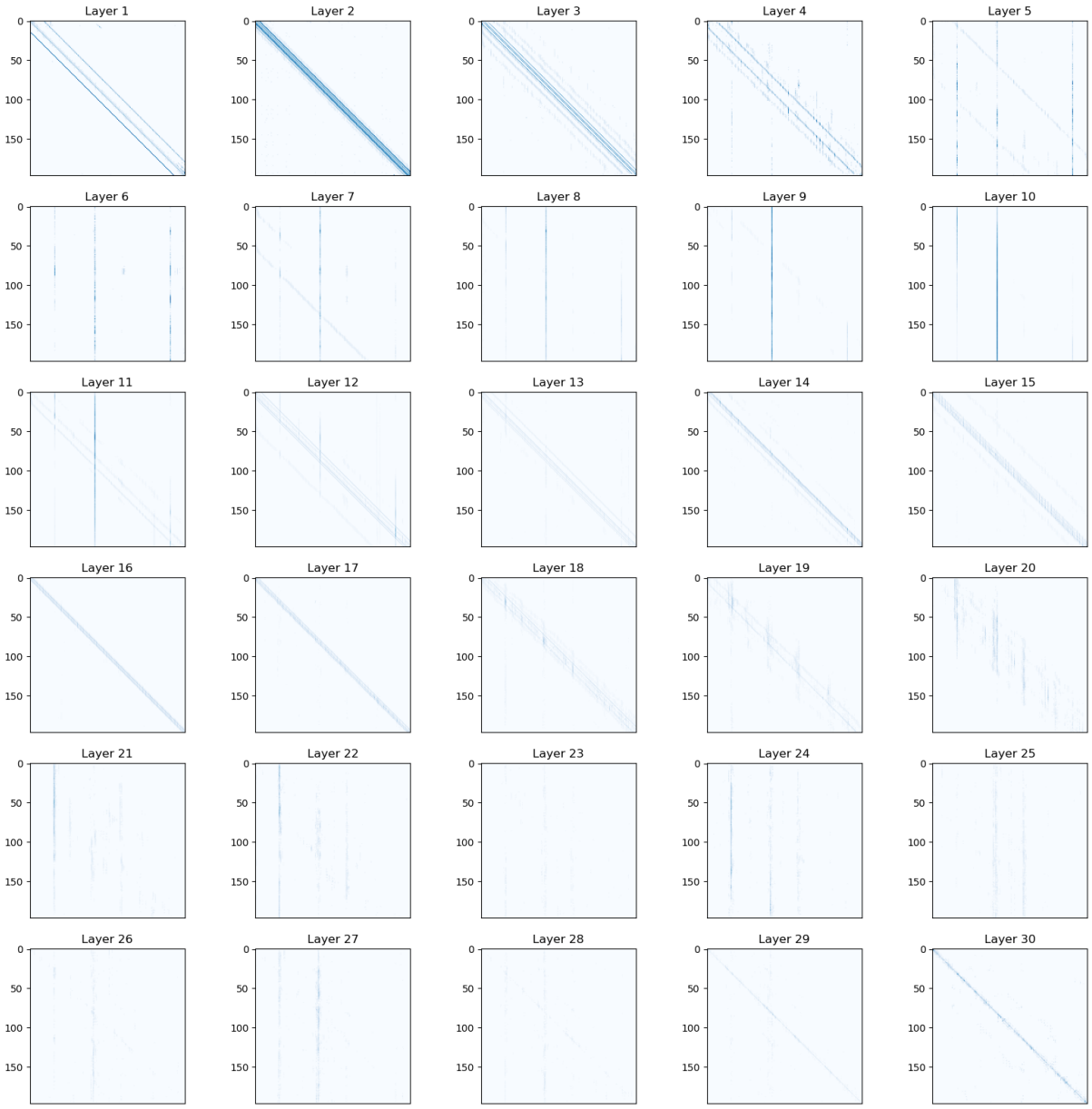


**Figure 22:** PDB ID: 6NJC. Averages of the sixteen attention matrices of each layer, with a threshold on attention of 0.1.

# E Further Network Images

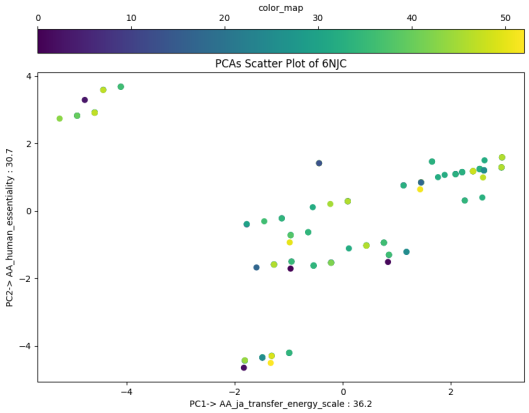


**Figure 23:** PDB ID: 6NJC. 2D plot of PCA-ordered Louvain partitions.

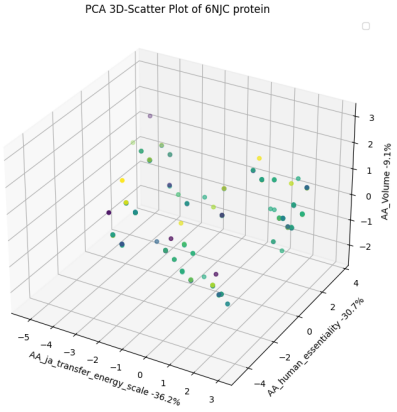


**Figure 24:** PDB ID: 6NJC. 3D plot of PCA-ordered Louvain partitions.

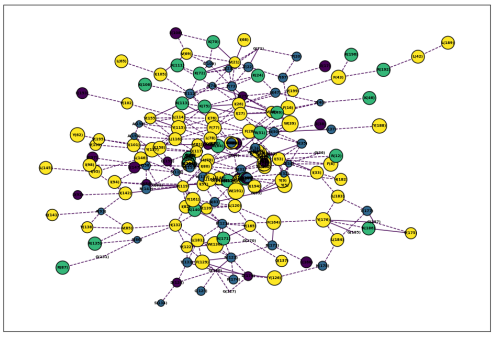

**Figure 25:** PDB ID: 6NJC. 2D network of Kamada-Kawaii-ordered $k$-means partitions.

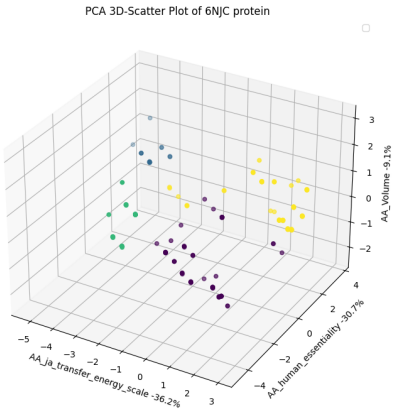


**Figure 26:** PDB ID: 6NJC. 3D plot of PCA-ordered $k$-means partitions.

# References


[1] J. Vig, A. Madani, L. R. Varshney, et al., "BERTology Meets Biology: Interpreting Attention in Protein Language Models", 10.48550/arXiv.2006.15222 (2021).

[2] C. B. Anfinsen and E. Haber, "Studies on the reduction and re-formation of protein disulfide bonds", The Journal of Biological Chemistry **236**, 1361–1363 (1961).

[3] A. Elnaggar, M. Heinzinger, C. Dallago, et al., "ProtTrans: Towards Cracking the Language of Life's Code Through Self-Supervised Learning", IEEE Transactions on Pattern Analysis and Machine Intelligence **44**, 7112–7127 (2021).

[4] S. Hochreiter, Y. Bengio, P. Frasconi, et al., "Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies", A Field Guide to Dynamical Recurrent Neural Networks (2003).

[5] A. Vaswani, N. Shazeer, N. Parmar, et al., "Attention Is All You Need", 10.48550/arXiv.1706.03762 (2023).

[6] Transformers-based Encoder-Decoder Models, (Oct. 10, 2020) https://huggingface.co/blog/encoder-decoder.

[7] ProtBert, https://huggingface.co/Rostlab/prot_bert (visited on 07/14/2024).

[8] ProtTrans, https://github.com/agemagician/ProtTrans (visited on 07/14/2024).

[9] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", 1810.04805v2 (2019).

[10] UniProt, https://www.uniprot.org/help/uniref (visited on 07/14/2024).

[11] RCSB Protein Data Bank, https://www.rcsb.org (visited on 07/14/2024).

[12] Bio package - Biopython, https://biopython.org/docs/1.76/api/Bio.html (visited on 07/14/2024).

[13] Amino Acid Codes, https://www.ddbj.nig.ac.jp/ddbj/code-e.html#amino-1 (visited on 07/14/2024).

[14] RCSB PDB Data API, https://data.rcsb.org/#data-api (visited on 10/09/2024).

[15] py-rcsbsearchapi on GitHub, https://github.com/rcsb/py-rcsbsearchapi (visited on 10/09/2024).

[16] Ligands, https://en.wikipedia.org/wiki/Ligand_(biochemistry) (visited on 10/09/2024).

[17] Heteroatom, https://en.wikipedia.org/wiki/Heteroatom#cite_note-pdbformat-4 (visited on 10/09/2024).

[18] Hydrophobicity scales, https://en.wikipedia.org/wiki/Hydrophobicity_scales (visited on 10/09/2024).

[19] Hopp–Woods scale, https://en.wikipedia.org/wiki/Hopp%E2%80%93Woods_scale (visited on 10/09/2024).

[20] Bio.SeqUtils.ProtParam module, https://github.com/biopython/biopython/blob/master/Bio/SeqUtils/ProtParamData.py (visited on 12/10/2024).

[21] Bio.SeqUtils.ProtParam module, https://github.com/biopython/biopython/blob/master/Bio/SeqUtils/ProtParam.py (visited on 12/10/2024).

[22] R. Rao, N. Bhattacharya, N. Thomas, et al., "Evaluating Protein Transfer Learning with TAPE", Advances in neural information processing systems **32**, 9689–9701 (2019).

[23] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX", in Proceedings of the 7th Python in Science Conference (SciPy2008), edited by G. Varoquaux, T. Vaught, and J. Millman (Aug. 2008), pp. 11–15.

[24] S. R. Eddy, "Where did the BLOSUM62 alignment score matrix come from?", Nature Biotechnology **22**, 1035–1036 (2004).

[25] S. Henikoff and J. G. Henikoff, "Amino acid substitution matrices from protein blocks", Proceedings of the National Academy of Sciences **89**, 10915–10919 (1992).

[26] BLOSUM, https://en.wikipedia.org/wiki/BLOSUM (visited on 10/31/2024).

[27] Biopython on GitHub, https://github.com/biopython/biopython (visited on 10/15/2024).