SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Data Engineering and Analytics

Learning the Language of Protein Structures

Joaquín Gómez Sánchez

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Data Engineering and Analytics

Learning the Language of Protein Structures

Die Sprache der Proteinstrukturen lernen

Author:	Joaquín Gómez Sánchez
Supervisor:	Prof. Dr. Burkhard Rost
Advisors:	Dr. Michael Heinzinger and M.Sc. Tobias Olenyi
Submission Date:	15th May 2023

I confirm that this master's thesis in data engineering and analytics is my own work and I have documented all sources and material used.

Munich, 15th May 2023 Joaquín Gómez Sánchez

To Èric, who is always there for me,
even on the though days.

A discovery is like falling in love and reaching the top of a mountain after a hard climb all in one, an ecstasy not induced by drugs but by the revelation of a face of nature that no one has seen before and that often turns out to be more subtle and wonderful than anyone had imagined.

—Max F. Perutz,
a structural biology father

Acknowledgments

I would like to start by expressing my sincere gratitude to my advisors, Dr. Michael Heinzinger and M.Sc. Tobias Olenyi, for their guidance at every stage of the project, and their insightful comments and suggestions. I would also like to extend my sincere thanks to Prof. Dr. Burkhard Rost for supervising this work and for leading RostLab, a pleasing place to learn and research.

As for the people indirectly related to the project, I would like to thank Tim Karl for the invaluable help with hardware; Sebastian Franz and Christian Dallago for developing and maintaining biotrainer, without which it would not have been possible to simplify parts of this project; and Michel van Kempen, Dr. Martin Steinegger, and colleagues for conceiving and developing 3Di, the base of this work.

To my life partner, Èric, and my special persons, Anna, Ester, and Lidia, I would like to express my love and infinite gratitude for encouraging and supporting me in my projects, and for being there on the good and tough days.

Last but not least, I would like to thank my parents and my sister for their support and understanding during these two years. One couldn't have better life supports.

Abstract

The AI revolution, which culminated recently in the release of GPT-4, not only impacted how we process natural language but the very same principles could readily be transferred to other sequential data such as protein sequences. Adapting these techniques to computational biology changed the way we process proteins today fundamentally.

By simply feeding single amino acid sequences to large language models trained to parse and understand protein sequences, researchers were able to build state-of-the-art predictors on top that estimate the effect of mutations, the subcellular localization of proteins, their binding regions, or even guide the design of variations with expected functions. Even though, it was shown that these protein language models (pLMs) learn a rudimentary understanding of a protein's 3D structure solely from 1D sequences, the large abundance of high-quality 3D structures available since the release of AlphaFold2 raises the question, whether this wealth of 3D information could be somehow leveraged directly during the pLM pre-training. Towards this end, we map 3D structures to 1D strings using Foldseek which uses a learned mapping from 3D coordinates to a 1D string. The resulting 3Di alphabet was shown to capture structural information well enough to detect extremely remote homologs that could not be detected by other means such as sequence alignments.

Here, we assess whether we could get more informative protein representations compared to pLMs trained solely on amino acids by training multiple pLMs either solely on amino acids (baseline) or 3Di sequences (gain from the structure) or a merger of both (best of both worlds) and benchmark their performance on various downstream tasks. To establish a meaningful comparison, we did extensive hyper-parameter optimizations on our model and scaled up training to UniRef50 - a large protein sequence database.

Our findings show that 3Di sequences are good at predicting structure-related information but not at chemistry-based tasks. Instead, the combination of both types of information seems to be the most promising path.

Kurzfassung

Die AI-Revolution, die kürzlich mit der Veröffentlichung von GPT-4 ihren Höhepunkt erreichte, hat nicht nur die Arbeitsweise der Computerlinguistik grundlegend beeinflusst, sondern auch alle anderen Disziplinen die mit sequentielle Daten arbeiten. Da sich dieselben Prinzipien auch auf die Bioinformatik übertragen lassen, hat sich dadurch auch die Art und Weise wie wir heute Proteinsequenzen verarbeiten grundlegend verändert.

Indem man großen Sprachmodellen auf Aminosäuresequenzen trainiert, konnten diese Modelle lernen, Proteinsequenzen zu analysieren und bis zu einem gewissen Maß zu verstehen. Daraus konnten Forscher modernste Vorhersagemodelle entwickeln, die den Effekt von Mutationen, die subzelluläre Lokalisierung von Proteinen oder ihre Bindungsstellen vorhersagen. Zusätzlich konnten dieselben Methoden dazu genutzt werden um neue Proteinvarianten zu generieren. Obwohl gezeigt wurde, dass diese Protein-Sprachmodelle (pLMs) ein rudimentäres Verständnis der 3D-Struktur eines Proteins allein aus 1D-Sequenzen erlernen, stellt sich angesichts der Fülle hochwertiger 3D-Strukturen seit der Veröffentlichung von AlphaFold2 die Frage, ob diese 3D-Informationen direkt während des pLM-Trainings genutzt werden können. Zu diesem Zweck haben wir 3D-Strukturen mithilfe von Foldseek auf 1D-Strings abgebildet, wobei eine erlernte Zuordnung von 3D-Koordinaten zu einem 1D-String verwendet wird. Es wurde gezeigt, dass das resultierende 3Di-Alphabet strukturelle Informationen gut genug erfasst, um extrem entfernte Homologe zu erkennen, die auf andere Weisen wie Sequenzvergleiche nicht erkannt werden konnten.

Hier bewerten wir, ob wir im Vergleich zu pLMs, die ausschließlich auf Aminosäuren trainiert sind, aussagekräftigere Proteinrepräsentationen erhalten können, indem wir mehrere pLMs entweder ausschließlich auf Aminosäuren (Basislinie) oder auf 3Di-Sequenzen (Gewinn aus der Struktur) oder einer Kombination von beiden (das Beste aus beiden Welten) trainieren. Dazu vergleichen wir ihre Performance auf verschiedenen nachgelagerten Aufgaben. Um einen sinnvollen Vergleich herzustellen, haben wir umfangreiche Hyperparameter-Optimierungen an unserem Modell durchgeführt und das Training auf UniRef50 - einer großen Datenbank für Proteinsequenzen - skaliert.

Unsere Ergebnisse zeigen, dass 3Di-Sequenzen gut geeignet sind, um strukturbezogene Informationen vorherzusagen, aber nicht für chemiebasierte Aufgaben. Stattdessen scheint die Kombination beider Arten von Informationen der vielversprechendste Weg zu sein.

Contents

1. Introduction

1.1. Motivation

Since the '70s, along with the expansion of computing, the field of bioinformatics has been helping to research biology, pharmacology, medicine, and other life science areas. To name some concrete examples, it has been helping with the understanding of DNA and RNA, macro-molecular dynamics, pandemics, and others.

Over the last years, this development has been increasingly accelerated by applying developments of deep learning to biological data. This way, prediction tools became useful for drug discovery or protein design.

To be more specific, for the research on proteins, many advances have been done since the first bio-informatics solutions like the Needleman-Wunsch algorithm [1][1] in 1970. Until the last decade, almost all the steps forward in the understanding of proteins using computers have been mainly done at the sequence level, since it is easier to obtain a protein sequence instead of its three-dimensional structure. This is because the techniques to obtain de novo amino acid sequences, like Mass spectrometry, the Edman degradation, or the translation of entire proteomes from DNA or RNA, can be done in a common laboratory, while for the three-dimensional structure determination, we have to use crystallography techniques, that e.g. work on big facilities like synchrotrons. The use of other techniques like cryo-Electron Microscopy or Nuclear Magnetic Resonance is rising because they are simpler compared to crystallography, but they do not provide the same resolution. This difference between the number of available sequences and structures is known as the sequences-annotation gap, which gets highlighted by the most recent release of UniProt (release 2022_05), i.e. it has more than 230 million sequences, while the number of structures in Protein Data Bank (PDB) is around 200 thousand (by the date of this work).

The large number of available sequences has motivated the apparition of many different sequence-based techniques in order to automatically characterize proteins, i.e. determine their location, mutation effects, transmembrane residues, function, etc. Particularly over the last years, these developments have been increasingly accelerated by applying deep learning to biological data.

Until the explosion of the Natural Language Processing (NLP), the characterization of proteins was mainly achieved via Homology-based Inference (HBI), i.e. given a newly discovered protein, we can try to find similar proteins at residue-level (e.g. a similar protein in another specie that has changed with the evolution) and then characterize the new one.

[1] Dynamic-programming-based algorithm for global sequence alignment. It uses a similarity/scoring/substitution matrix to know how to align, e.g., PAM (mutation-based scoring matrices) [2] or BLOSUM (evolutionary-divergent-protein-sequences-based scoring matrices) [3].

For the structure determination from sequences, homology modeling has been the main, and sometimes unique, solution. It has been used alone or combined with constraining models based on physical and chemical characteristics. Notwithstanding, HBI was not the unique solution, and even if they were not comparable, Machine Learning models were successfully used for certain tasks. For example, these models were used to search for evolutionarily related sequences used later to generate Multiple Sequence Alignments (MSAs). These MSAs were typically used to compute Position-Specific Scoring Matrices (PSSMs) which in turn were used as input for Neural Networks o Support Vector Machines [4].

Two groundbreaking appearances have changed over the last decade how we work with proteins in bio-informatics. The first one is the introduction of Transformers [5] and different Transformer-based models like BERT [6]. Although other models like Recurrent Neural Network (RNN) were previously used for proteins [7], and they allow us to talk about learning the language of proteins, Transformers have made a breakthrough, achieving indeed good performance for many tasks.

The second breakthrough is AlphaFold 2 (AF2) [8], the first model achieving >90% of Global Distance Test (GDT)[2] average score in Critical Assessment of Structure Prediction (CASP) round 14 [9], something comparable with the experimental results obtained with X-ray crystallography.

The introduction of AF2 has allowed access to never-seen-before good predicted structures, deposited in the AlphaFold Protein Structure Database (AFDB) [10], and this has rapidly impacted different research communities like medicine and pharmacology (e.g. helping to find a malaria vaccine [11]) or environmental science and biochemistry (e.g. finding enzymes for plastic-degradation [12]). But it is also being used as a complementary tool to solve structures that are difficult to be obtained through crystallography [13].

With the Transformers revolution, so many protein-Language Models (pLMs) have been proposed (see section 1.2), and also some Language Models (LMs) for genomic data like GenSLMs [14]. The first ones rely on amino acid sequences, while GenSLMs rely on genomic sequences. It has been used to learn the evolutionary landscape of SARS-CoV-2 genomes, allowing researchers to identify variations of concerns, and it could be useful in future pandemics. However, the application of Deep Learning techniques to structures remains an intensive research area, essentially focused on using graphs-related models like Graph Neural Network (GNN), but this approach has shown to be limited due to the complexity of the models and its limitations, like the expressive power. Another problem to face has been the scarcity of experimental structures, notwithstanding, this last statement has changed with AF2.

The aim of this work is to explore the possibilities of using Transformers and the current wide availability of structures, thanks to the recently introduced set of characters to describe them, i.e., 3Di [15], explained in deep in section 2.3. That is, exploit the opportunities that a sequence structure gives when using sequential models.

[2]It measures the similarity between two proteins structures with identical amino acid sequences but different tertiary structures. It is computed over C_α atoms and ranges from 0 to 100, where the higher the score, the closer the structure approximates the other one. It is more accurate than the RMSD.

1.2. Related Work

Since Transformers, and also other types of LM, need sequential or 1D representations of the information, mappings of 3D information, like protein structures, to 1D are required. To the knowledge of the author, besides 3Di states, there are no other direct alternatives, i.e. a correspondence between residues and their 3D descriptions. An approximation could be the use of the three-dimensional spherical polar Fourier representations of 3D-BLAST [16], but a mapping between the numerical representations and 1D states remains unavailable. Some alternatives could be the use of local structure prototypes (named Protein Blocks) to translate residues to sequences of prototypes [3, 17, 18], or the 8-conformational-states DSSP [19] of proteins.

If we move out from the mere representation of the structure, especially for learning protein-ligand complexes, the options are more varied and are particularly focused on the use of graphs. Some examples are DeepFRI [20], GraphBAR [21], PG-GNN [22], or EquiBind [23]; but they only simplify the 3D space without reducing the dimensionality to one.

About the available pLMs, the zoo of models is huge, and there are many alternatives. They can be classified in RNN/LSTM-based pLMs like PARROT [24]; ELMo-based pLMs like SeqVec [7]; or Transformer-based. For this last group, we have encoder-decoder models like ProGen [25]; but also encoder-only models like the ones from ProtTrans (e.g. ProtBERT or ProtT5) [26], ProteinBERT [27], or the ESM family (ESM-1b [28], ESM-1v [29] or ESM-2 [30]); and decoder-only models like DARK [31] or ProtGPT2 [32], which are suitable for the generation of protein sequences. All them are based on amino acid sequences, but although still small, there is a set of alternatives, for example, based on MSAs like ESM-MSA-1b [33] or based on genetic information like GenSLMs [14] and the Nucleotide Transformer [34].

The development of pLMs remains a dynamic research area, adapting all the models proposed from the NLP. In this work, the used models are BERT [6] and RoFormer [35], but the alternatives are very diverse. If one wants to put focus on efficiency ReFormer [36] could be an option, but also ELECTRA [37], ALBERT [38] or DistilBERT [39]; or if the focus is rather on the task, for example text generation, an option could be the GPT family [40, 41]. These mentions are just a small set, and the possibilities still increasing with the introduction of new models.

2. Background

2.1. Language Models and protein-Language Models

In Natural Language Processing (NLP), a Language Model (LM) is understood as the probability distribution over a sequence of words. This distribution allows us to assign, for example, a probability $P(w_1, \ldots, w_n)$ to a whole sequence and use it to decide which are valid sequences or to predict unknown words.

LM are mainly classified into automata-based models, like the n-gram model, based on the Markov property; and neural-based models. These last ones have been effectively used in Machine Translation, Natural Language Generation, or Text Summarization. With the Neural Networks revolution during the previous decade, many Neural LMs have been proposed. Some examples are RNN-based LMs [42, 43], Long Short-Term Memory (LSTM)-based LMs [44] or Transformer-based LMs [5]. The main advantage of these novel architectures over the prior classical approaches, i.e., automata-based models, is that Neural Network-based models rely on word embeddings, i.e., continuous representations for words. These representations allow us to obtain a space of word representations and they can be used for different purposes like word characterization, i.e., for parsing, or part-of-speech tagging.

pLMs consist in using these set of Neural LMs for proteins, which implies some advantages. For example, while in NLP we can have thousands of words, in protein sequences we only have 20 standard amino acids, i.e., there are only 20 possible words, which drastically decreases the complexity; at least theoretically. At the same time, these models allow us to catch through the statistics the intrinsic complexity of amino acids, which goes beyond chemistry. Some examples of pLMs are SeqVec [7], an ELMo-based model[1] [45]; or ProtT5 [26], ProteinBERT [27] and ESM-2 [30], all them Transformer-based models.

For this work, the used models are Transformer-based pLM since they have become by the date of this project the gold standard. Following, they are introduced together with their relevant details for the project.

2.1.1. Attention Mechanism and Transformers

Attention Mechanism, originally introduced by Bahdanau et al. [46], and generalized and extended by Vaswani et al. [5], is the base for Transformers. It consists of three vectors (for each input token): query q, key k, and value v. Each query vector is matched to different keys in order to compute a score value:

[1] LSTM-based bidirectional architecture, called bidirectional-Language Model (biLM). It has one forward layer and one backward layer, both concatenated at the end of the network.

$$e_{q_i,k_j} = q_i^T \cdot k_j \tag{2.1}$$

After softmaxing the score, it is used to weight value vectors. All the computation can be summarized in the following matrix-form expression:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{2.2}$$

where d_k is the dimension of key vectors, and it acts as a scaling factor. With these computations, a model can learn how a word in a sequence is related to others and then scales the importance of every word.

The attention introduced here is known as **Scaled Dot-Product Attention**, but Vaswani et al. [5] also introduced **Multi-head Attention** which consists of several attention mechanisms running in parallel (each one called **attention head**). These independent heads are concatenated and linearly transformed to produce a single attention output. Other attentions have also been proposed, for example, the **Fixed Factorized Attention**[2] or the **Stride Attention**[3], being both proposed as a part of the **Sparse Transformer** architecture [47].

As presented by Vaswani et al. [5], the **Transformer** architecture consists of an **Encoder** and a **Decoder**, being both replicated for N layers. Given an input embedding, which is learned from the tokenization, and the positional encoding of that embedding (see subsubsection 2.1.1), the encoder consists of a Multi-Head Attention layer and a Feed Forward Neural Network (FNN). The decoder adds to the aforementioned layers a Masked Multi-Head Attention, as it can be seen in Figure 2.1. This new attention differs from the previous one because it prevents positions from attending to subsequent ones, i.e., for this attention the prediction for a position i only depends on the previous positions.

The Fully-Connected FNN of both, the encoder and the decoder, consists of a bilayer Neural Network with ReLu as an activation function and it is applied point-wise with different parameters for different layers.

Conditioned to the structure of the model and the used attention mask pattern, a taxonomy of three types of Transformers has been proposed [48]:

- **Encoder-Decoder Transformer**. Standard transformer where the encoder uses a fully-visible masking attention pattern and the decoder uses causal masking, i.e., it only attends to previous cells. We only feed the input to the model. This is typically used in sequence-to-sequence modeling, e.g., Neural Machine Translation. Examples of this type of architecture are T5 [49] or its pLM counterpart, ProtT5 [26].

- **Encoder only** (also known as Language Models). In this case, the output of the model is used as a representation of the input, which is appropriate for Natural Language Understanding, i.e., text classification or sequence labeling. Some models also use the

[2] It is an attention pattern, concretely a factorized pattern, where some cells summarize previous locations and propagate these summarizations to all future cells.

[3] For this attention, given a stride s, an attention head attends l previous locations and another head attends to every l-th location.

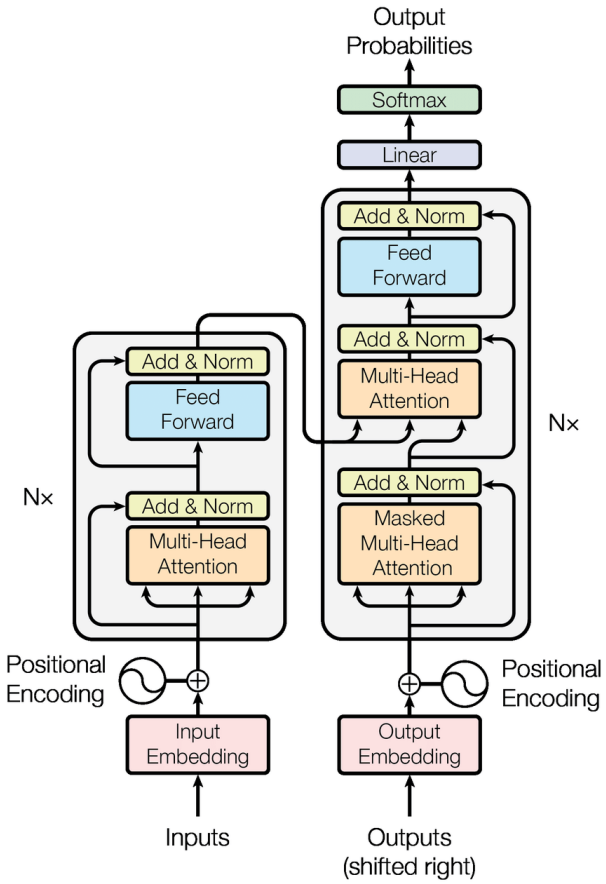


Figure 2.1.: Transformer architecture. Figure from [5].

expected output as input, using a causal mask. Another possibility is the so-called Prefix LM, for which only a randomly selected part of the input is visible (the prefix). Examples of pLM of this type are BERT [6], or the pLMs ESM-1b [28] and ESM-2 [30], both BERT-style encoder-only transformers.

- **Decoder only**. This type of model relies on attending only to tokens positioned before a given word and it is often used for sequence generation. GPT-2 [40] or its pLM counterpart ProtGPT2 [32] are examples.

Transformers imply some important advantages over previously used methods. For example, one of the main problems using RNNs is that they are sensitive to vanishing gradients when feeding long sequences. Another advantage over RNNs is the memory, because they are typically not good at long-term memory, even for LSTMs. Finally, the most important aspect of Transformers is parallelization, because they do not need to process the entire sequence sequentially, which is the source of disadvantages in RNNs.

Learning Objectives

Since Transformers learn in an unsupervised way, as they are focused on modeling the language, and then there is no sense of labels, learning objectives play an important role in the training and different options have been proposed.

Based on the most used models that have been trained using these objectives, the most common ones are:

- **Masked Language Modeling (MLM)**. Mainly used by auto-encoding models[4] like BERT, it consists in masking some input words at random and training the model to guess these missings [6]. Using this objective it is expected that the model may learn sufficiently informative representations for every word from the vocabulary. This while leaving to the model architecture the responsibility of which words should be taken into account, i.e., the attention mask pattern. Despite the fact that it is one of the most widespread training objectives, it has been hypothesized that the corruption of the inputs neglects dependencies between masked words [50], and also that the conventionally used randomness, i.e. 15%, might not be the best option [51].
- **Causal Language Modeling (CLM)**. Used in LMs like GPT-2 [40], instead of corrupting the input sequences, for this objective the sentences are partially truncated at some point and the model must be able to reconstruct the missing part. It has been shown that the models that use this objective, i.e., auto-regressive models like the GPT family, are good at generating text.

However, there are other less relevant options available. For example, **Next Sentence Prediction (NSP)**, also introduced with BERT, is used to allow the model to learn inter-sentence relationships, which is not possible with language modeling but it is useful for some tasks like question-answering.

Positional Encoding

Since the architecture of Transformers is not positionally aware, the concept of **Positional Encoding** was incorporated. It consists in performing some operation over the input embeddings in order to add to them relevant information about the position of every word in the sequences, i.e., the position is indirectly encoded together with every word encoding.

Several alternatives have been proposed for textual inputs:

- **Absolute Positional Encoding**. Introduced with the original Transformer [5], it directly adds the position encoding to the input encoding. The original implementation proposes the use of the cosine and the sine functions:

$$\mathrm{PE}(pos, 2i) = \sin\left(pos / 10000^{2i/d_{model}}\right) \tag{2.3}$$

[4] Alternatively to the model structure or the used attention mask pattern, Transformers can also be classified depending on the training objective or task: **auto-encoders**, for those models that learn an encoded representation of the input; **auto-regressive** when the model works fully sequentially, producing each output based on all the previous ones; and, **Seq2Seq** for those models that receive one sentence as input and produce a transformed one, for example, a translation.

$$\mathrm{PE}(pos, 2i+1) = \cos\left(pos/10000^{2i/d_{model}}\right) \tag{2.4}$$

being pos the position and i the dimension. It was hypothesized that they would allow the model to easily learn to attend by relative positions since every offset position can be represented as a linear function of the current processed position. The complexity of this positional encoding is $\mathcal{O}(nd)$ (where n is the maximum sequence length and d is the dimension of the arrays).

- **Relative Positional Encoding**. Originally proposed by Shaw et al. [52], and improved by Huang et al. [53, 54], this encoding attempts to include positional information to the input embeddings of the Transformer but computing and adding the positional information on the fly while computing the attention. For this purpose, it is fit into the architecture an additional component to the key computation[5]:

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}} \tag{2.5}$$

remaining the softmax computation as given in the original formulation. Shaw et al. [52] also proposed an additional supply for value, i.e., $z_i = \sum_{j=1}^{n} \alpha_{ij} \left(x_j W^V + a_{ij}^V\right)$, but this was dropped by Huan et al. [53, 54] in favor of a better approach that combines positional information for key and query:

$$e_{ij} = \frac{\left(x_i W^Q + a_{ij}\right)\left(x_j W^K + a_{ij}\right)^T - \langle a_{ij}, a_{ij} \rangle}{\sqrt{d_z}} \tag{2.6}$$

It relates the interaction between key and query, like in absolute encoding; key and relative position; and also query and relative position. This last version has been shown to be the best relative option while keeping the same complexity as Shaw's one, i.e., $\mathcal{O}\left(mhn^r d\right)$ (where m is the number of layers, h is the number of attention heads, n is the maximum sequence length, and d is the dimension of the arrays).

- **Rotary Positional Encoding**. Introduced with RoFormer [35], it aims to encode the absolute position with a rotation matrix and to incorporate the explicit relative position in the self-attention formulation. It defines key and value functions as

$$f_{\{q,k\}}(x_m, m) = R_{\Theta,m}^d W_{\{q,k\}} x_m \tag{2.7}$$

where $R_{\Theta,m}^d$ is a rotation matrix with pre-defined angle $\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \ldots, d/2]\}$. Then, the attention score Equation 2.1 becomes

$$q_i^T \cdot k_j = \left(R_{\Theta,i}^d W^Q x_i\right)^T \left(R_{\Theta,j}^d W^K x_j\right)^T = x^T W^Q R_{\Theta,j-i}^d W^K x_j \tag{2.8}$$

[5]The original formulation computes the weight coefficient of the model as $\alpha_{ij} = \frac{\exp e_{ij}}{\sum_{k=1}^{n} \exp e_{ik}}$, where $e_{ij} = \frac{\left(x_i W^Q\right)\left(x_j W^K\right)^T}{\sqrt{d_z}}$

It can be seen that while absolute and relative encodings are additive, rotary is multiplicative. As the authors state, it incorporates relative position information through the rotation matrix, without altering self-attention terms. The main advantage of this encoding is that it provides a long-term decay.

2.1.2. BERT and RoFormer

As previously stated in this chapter, BERT [6] has an encoder-only architecture, and it was one of the first Transformer-based models. It was also one of the first models that deeply explored the Transfer Learning possibilities of Transformers, opening not only a wide range of research possibilities inside the NLP area but also in so many others like our concern here, proteins.

It was introduced with a dual framework, pre-training and fine-tuning. Firstly, the model is trained in an unsupervised way thanks to the training objectives MLM and NSP, having at the end a general model that can be fine-tuned for downstream tasks, being possible to adapt the architecture for these tasks.

Since its architecture is identical to the original Transformer, the difference lies in the special tokens. These are `[MASK]`, for masking the input sequences; `[UNK]`, for the unknown inputs; `[SEP]`, for the separation of two different sequences; `[EOS]`, to indicate the end of a sequence; and `[CLS]`, used to represent sentence-level classifications.

These special tokens are used, for example, when we want to classify an entire sequence in a Sentence Tagging Task. We should feed the entire sequence in BERT, being the first input a `[CLS]` token. Then, in the same position of the output, we would find the classification result. On the other hand, if we want to classify two sentences at the same time, we can feed them into BERT, but including a `[SEP]` token between both.

Another important aspect of BERT, as its B in the name states, is its use of bidirectional attention, i.e., it attends the entire sequence.

Different BERT variations have been proposed, namely, ALBERT [38], which shares parameters between layers to make the model more efficient; RoBERTa [55], which modifies the training masking strategy to use a dynamic one; or ELECTRA [37], which instead of using masking, it replace input tokens and the model must detect the replacements.

However, it has been decided to use BERT for this work, since it is the most well-established (encoder-only) Transformer model; and also RoFormer, since it is a simple modification of the original BERT in order to include Rotary Positional Encoding.

Details on the implementation and usage of both models can be found on section A.2.

2.2. AlphaFold

Until the apparition of AlphaFold 2 (AF2) only $\sim$ 200.000 protein foldings were resolved as mentioned in chapter 1, this is due to the complexity of resolving structures through experiments. Fortunately, thanks to AF2 this has changed.

Before AF2, and over the years, thanks to CASP, several methods to resolve protein structures were introduced, but all of them without achieving results similar to the ones that experiment-based methods provide to us; i.e., a GDT score above 90, which is the standard equivalent to experimentally determined structures score. These methods mainly consist of free modeling (i.e., ab initio algorithms) or templated-based modeling (i.e., folding recognition and homology) algorithms [56]. Some examples are QUARK [57], an ab initio method; or SWISS-MODEL [58] and Modeller [59], both homology-based models.

Since the introduction of the first AlphaFold model, and especially with the second version, considered nowadays the gold standard, other methods have been proposed. Namely, RoseTTAFold [60] or ESMFold [30], both being also non-homology- and deep learning-based models. Nevertheless, AF2 remains the most used one, together with its variations like ColabFold [61], which uses MMseqs2 [62] instead of jackhammer [63] to generate the MSAs used by the model. This change allows it to predict fivefold faster on average. Another variant is OpenFold [64], which has PyTorch-based implementation instead of using JAX like AF2. For this project, it has been decided to work using ColabFold, due to its efficiency.

Without the aim of being exhaustive, because AlphaFold has been already detailed in several works, and since the extensive explanation of its functioning is beyond the scope of this work, following there is a sneak peek about how it works.

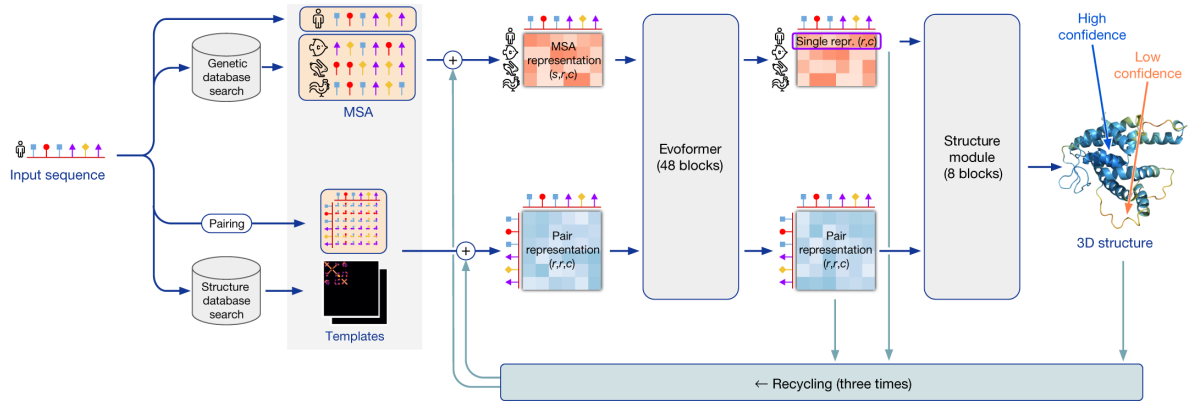


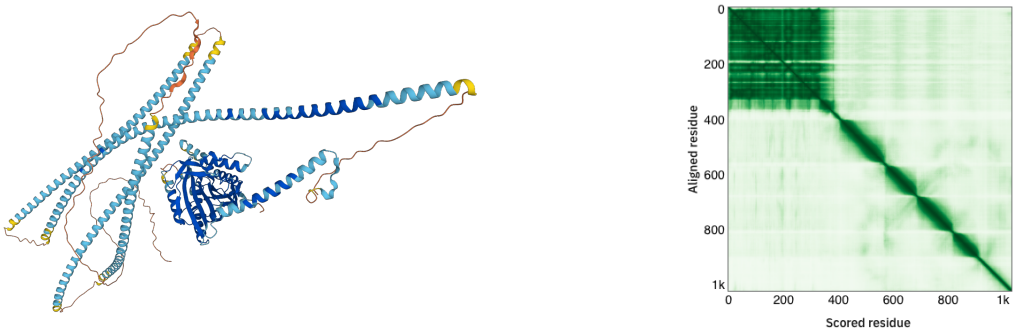
Figure 2.2.: AlphaFold 2 architecture. Figure from [8].

In terms of architecture (summarized in Figure 2.2), given an input sequence, AF2 (and its variants) produces an MSA of evolutionarily related proteins after looking for them in a given genetic database. It also produces a set of templates (from homologous structures) after looking for them in a given structures database. An MSA representation and a pair representation of the templates are used as input to a novel architecture termed Evoformer. It consists of 48 blocks made up of attention-based layers that alternate the refinement of the MSA and the pair representation. Each block may be seen as a pipeline that establishes communication between both representations in order to construct a new structural hypothesis, this while preserving consistency and importance of evolutionary and geometric information [65]. After the Evoformer, the pair representation and the single representation of the input sequence (extracted from the MSA) are fed into the Structure module, which consists of 8

blocks. Applying translation and rotations, it constructs the final distribution of atoms in space. The outputs of the Evoformer together with the final structure are re-feed three times into the network in order to refine the folding.

A key aspect of AF2 is its ability to self-explain the quality or confidence in the predictions. It uses the Predicted Local-Distance Difference Test (pLDDT), a per-residue estimation based on the lDDT-Cα metric [66], a score on the local distance differences of all atoms in a protein and the stereochemical plausibility. It has been established that a score higher than 90 means that it can be expected that the model is highly accurate; between 70 and 90 that the protein has been well modeled; and below 50 means that the part of the protein with that score should not be interpreted.

It also outputs the so-called Predicted Aligned Error (PAE). It consists of a matrix where the value of every cell indicates the expected position error at a given residue if the predicted and true structures were aligned on the other residue. Given two domains, if the PAE is low, it means that the predicted relative positions and orientations are good and can be interpreted. It is a measure independent of the 3D structure, thus it is not going to be used for this work.



(a) Structure and pLDDTs (b) PAEs

Figure 2.3.: Exemplary outputs of AF2 for the sequence protein Q12840 (Kinesin heavy chain isoform 5A) expressed by the Homo sapiens gene KIF5A. Deposited in AFDB with the ID AF-Q12840-F1. For the pLDDTs, the bluer the more confidence; and for the PAEs, the greener the lower the error.

With AF2, the so-called **AlphaFold Protein Structure Database (AFDB)** [10] has been published and increased over time. It includes publicly available pre-commuted protein structures for the entire human proteome, and also for other organisms, reaching by the date of this work over 200 million structure predictions. However, it should be taken into account that the included structures correspond to wild-types or the considered standard sequence for each protein, being sometimes available only protein isoforms[6] for the searched ones. This has been an aspect to deal with during the development of this work as it is discussed in section A.1.

Although its ability to predict de novo protein structures has been demonstrated, it has

[6]A protein isoform is a protein similar to another one but with some modifications result of a genetic difference. An isoform can have the same function or unique functionality.

some disadvantages that should be taken into account. The main one is that it only produces one state, while in vivo proteins acquire different conformations, that is, they are not static. This last aspect relates to the fact that for some applications it could be interesting to predict the position of non-protein components like cofactors, ligands, or DNA. It has also been shown that it does not really work with physical or chemical information since it is a sequence-based method [67], or that it over predicts α-helices and β-strands while working worse for loops with a large number of amino acids [68]. It should also be taken into account that AF2 has been mainly trained on structures obtained from crystallography-based methods, which reduces the information that other experiments like Nuclear Magnetic Resonance Spectroscopy may provide.

Despite all these disadvantages, it is a work in progress and not a final solution, as can be seen in the latterly published multimer version [69].

2.3. The Different Faces of Proteins: from Sequences to Structures

Due to the large amount of available sequences, computer-likely representations for proteins, i.e. embeddings, have mainly relied on amino acid sequences, i.e. the 20 standard amino acids, including sometimes extra characters, for example, to indicate an unresolved residue (typically with an 'X').

In terms of representations at the structure level, the mainly used ones are graphs. Different task-specific models have been proposed, for example for protein-ligand prediction, protein function prediction, or protein design (summarized in [70]), and also for the prediction of residue contact maps [71] or the generation of libraries of motifs [72].

The use of graphs implies some complexities, for example, in terms of memory, we need $\Theta(n)$ for a sequence of amino acids, i.e. the length of the sequence; and for a graph, depending on the representation, $\mathcal{O}(n^2)$ could be needed. In terms of computation, while for Self-Attention the complexity per layer is $\mathcal{O}(N^2D)$ (where n is the sequence length and d is the representation dimension) [5], a forward step in a convolution layer from a Graph Convolutional Network implies $\mathcal{O}(NF^2 + |E|F)$ (where F is the number of features, N is the number of nodes and E the number of edges between nodes) [73].

In order to find a sequential representation of protein structures, van Kempen et al. [15] introduced the **3Di alphabet**. These states are artificial and they have been learned by a vector-quantized variational autoencoder [74] forced to produce twenty 3Di characters, maximizing the evolutionary conservation and using the following information as features to describe the interaction between each residue and its nearest neighbor:

- Seven angles
- Euclidean C_α distance
- Two sequence distance features from the six C_α coordinates of the two backbone fragments

To define this information, the model optimized virtual centers in order to encounter the neighbor residues, and these lie on the plane defined by the atoms N, C_α and C_β, as shown in Figure 2.4.

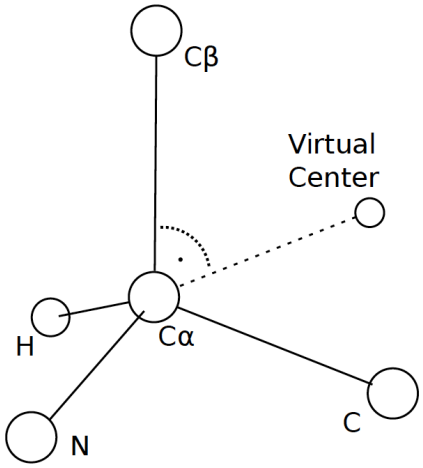


Figure 2.4.: Virtual angles with respect to N, C_α and C_β. Image from [15].

These states were originally created maximizing the search sensitivity that they offer, since the original purpose was to find a sequential representation of the structures in order to look for similar ones in a database. This is due to the fact that it is clearly easier to compare lineal sequences than three-dimensional structures.

It has been shown that these novel states offers similar sensitivity and precision to other more "structural preservative" representations like DALI [75], a distance-matrix alignment-base method; or 3D-BLAST [16], which uses a substitution matrix and it also has the handicap that it discretizes the structure.

Due to its outstanding qualities, it seems that 3Di is a good candidate to describe proteins, in a different way but also providing new or extra information to that given by amino acids.

3. Methods

3.1. Training Data and Benchmarking Downstream Tasks

Given that the purpose of training pLMs is to use their embeddings as features to feed into task-specific models, a set of downstream tasks has been selected in order to benchmark the trained models with 3Di sequences and to compare them with models trained on AA sequences.

Based on the hypotheses that AA models should catch chemical-related information and 3Di models structure-related information, the set includes structure-related and chemical-related tasks.

Here is also described the 3Di version of every task dataset, which has been obtained following the procedure described in section A.1. Sequences with non-standard residues had to be removed because AF2 (and variants) is not able to process them. Those sequences with only regular amino acids have been processed to obtain the structure (and the subsequent 3Di sequence), and also a "counterpart" AA dataset for those finally considered sequences has been produced, in order to enable fair comparisons. That is, for every 3Di set, the AA sequences have been retrieved, and these are the ones used for AA-based models.

Before explaining the different downstream tasks, the data sets used for the training are introduced below.

3.1.1. Training Data

Taking a look at the literature [26, 27, 28, 32] it can be seen that the gold standards for training pLMs are the UniRef sets, concretely UniRef50 and UniRef90. These sets are clustered versions of UniProt, which lowers the bias towards large but highly redundant families. UniRef90 consists of the seed sequences of UniRef100, which is a clustering of the entire UniProt. Subsequently, after clustering UniRef90's seed sequences, UniRef50 is obtained [76], and it is the one among the two training datasets used here.

Concretely, the used UniRef50 set is a derived version from AFDB consisting of 52.327.413 sequences with up to length 2500. The lengths distribution can be seen in Figure 3.1, which is consistent with distributions found in other studies, i.e., a gamma or log-normal distribution [77].

Besides using the widely known UniRef50 (U50), something that is important when working with proteins is redundancy. It is known that in protein sets there are over-represented classes, i.e. redundant protein sequences forming large families, which tend to bias the networks. As a result, performance for smaller families can decrease.

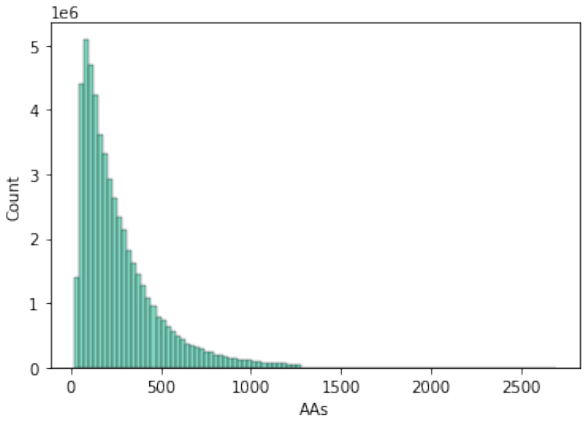


Figure 3.1.: Distribution of length for sequences from UniRef50.

In order to address this problem, a so-called Reduced Redundancy UniRef50 (RR U50) [78] is used. It has been obtained after the following process: first, the entire UniRef50 is clustered by sequence similarity using MMSeqs2 [62], then further clustered by structure using Foldseek [15], providing 2.7M representatives and 17M singleton clusters. After that, the 20 most sequence-diverse members from each cluster (without taking into account the singletons) were extracted. The rest of the sequences, i.e. 34M, were filtered by length larger than 30 residues and pLDDT larger than 70. Finally, 500 clusters were split off for testing and for validation, resulting in around 1.3k sequences for both subsets; and the rest for training. For both datasets, a uniformly distributed subsampling of 100.000 sequences has been performed in order to use them for training small models.

With the aim of exploring the different types of concatenations between amino acids and 3Di states, an extra dataset has been considered. It consists of a combination of amino acids and 3Di states. For every residue instead of having a single state, it consists of an upper-cased amino acid and a lower-cased concatenated 3Di state. For example, if a residue is an alanine (`A`) and its 3Di state is `D`, then its new state is `Ad`. It is an exact mapping of UniRef50 and it is explained and argued in-deep in section 3.2.

3.1.2. Secondary Structure

The secondary structure prediction of proteins has been used as the primary and well-established problem to benchmark amino acid-based pLMs, for example, to determine and compare the quality of models like ProtT5 and ESM.

It is based on the DSSP classification [19], which takes into account the conformations 3_{10} helix or 3-turn helix (named G), α-helix or 4-turn helix (H), π-helix or 5-turn helix (I), hydrogen-bonded turn (T), β-sheet (E), residue in isolated beta-bridge (B), bend (S), and an extra state for coil (C) or loop (L). This is called 8-conformational-states DSSP, but a 3-conformational-states DSSP version is also used, which classifies conformations as α-helixes or H (G, H, and I), β-sheets or E (E and B) and coils or C (S, T, and C).

In order to enable an apples-against-apples comparison, the training and test (called NEW364) sets used for ProtT5 [26] are going to be evaluated in this work. It is a 3-conformational-states DSSP set, where 93% of the residues are resolved and can be used for training/evaluation. Table 3.1 summarizes the characteristics of the dataset.

# Sequences			# Residues	Min. length	Max. length	# Seqs. with non-std. residues
Training	Validation	Testing	Total			
9712	1080	364	2845094	20	1632	77

Table 3.1.: Characteristics of the Secondary Structure task dataset. An extended version of this table can be found in Table B.1 and Table B.2.

Since 77 sequences (0.69%) include non-standard residues, there is a reduction in the number of available residues, which could be inadequate since this is a residue-level task. However, the proportion of 93% of resolved residues is preserved and the high impact is on the training subset (-66 sequences versus -5 on validation and -6 on testing).

Regarding the quality of the structures, in terms of pLDDT, 10835 sequences (97.79%) have a score higher than 70, 166 (1,49%) between 50 and 70, and 78 (0.70%) less than 50.

Unlike other datasets, for this one, all the structures used to obtain 3Di sequences have been computed using ColabFold. For the rest of the cases, on the other hand, part of the structures were obtained from AFDB.

For the evaluation of this task, the Q3 accuracy is going to be used since it consists of 3 possible states and the different classes are balanced.

3.1.3. Subcellular Localization

Subcellular Localization is a downstream task working at the sequence level commonly used for the evaluation of pLM [7, 26]. Concretely, the considered gold standard is the first version of the DeepLoc dataset [79]. The second version is focused on multi-localization [80], which is beyond the scope of this work. Here, two datasets are going to be used, both modifications of DeepLoc created for the evaluation of light attention[1] [81] [81], which to the knowledge of the author it is the State-of-the-Art (SOTA) task-specific model for predicting subcellular localization.

The original DeepLoc set consists of 16.626 sequences with one of the following labels (based on experimental evidence): cell membrane, cytoplasm, endoplasmatic reticulum, Golgi apparatus, lysosome/vacuole, mitochondrion, nucleus, peroxisome, plastid, and extracellular.

Exploring the dataset two disadvantages come to light. We have that the set is unbalanced for some classes (e.g. there are 4043 proteins for the class nucleus while 154 for the class

[1] Instead of directly feeding the embeddings to architectures like FNNs, light attention convolutes the embeddings in order to obtain the attention score and the values, both independents of the sequence length. What is feed to the FNN is a fixed-size representation result of adding the maximum values vector and $x_i = \sum_{j=1}^{L} \alpha_{i,j} v_{i,j}$, where $\alpha_{i,j}$ is the attention score.

peroxisome) and redundant. In order to solve the last problem, it has been decided to use the setDeepLoc set [81], which is a redundancy-reduced version of the testing subset with $\leq$ 30% of Pairwise Sequence Identity (PIDE) and E-values $\leq 10^{-6}$ to any sequence in the training subset. To address the second problem, the test setHARD [81] is used. It is an independent novel set obtained after filtering SwissProt as for the original DeepLoc, but reduced preserving only with a PIDE $\leq$ 20% to the training/validation set. Then, the final sequences were retrieved from the cluster representatives at $\geq$20% PIDE. Table 3.2 summarizes both datasets.

Set	**# Sequences**			**Min. length**	**Max. length**	**# Seqs. with non-std. residues**
	Training	**Validation**	**Testing**			
setDeepLoc	9503	1678	2768	40	13100	34
setHARD			490			31

Table 3.2.: Characteristics of the Subcellular Localization task datasets. An extended version of this table can be found in Table B.1 and Table B.2.

Besides the number of sequences with non-standard residues (34 and 31), other problems have been faced while preparing this dataset. It is seminal to remark that one of the discarded sequences is the protein with 13100 residues, being then 2697 the maximum sequence length.

The first problem is regarding the signal peptide, i.e. the first 16 to 30 amino acids at the beginning of the sequence. It is known that signal peptides prompt the cell to translocate the protein to its working environment, i.e. its subcellular localization, mostly toward the secretory pathway [82, 83].

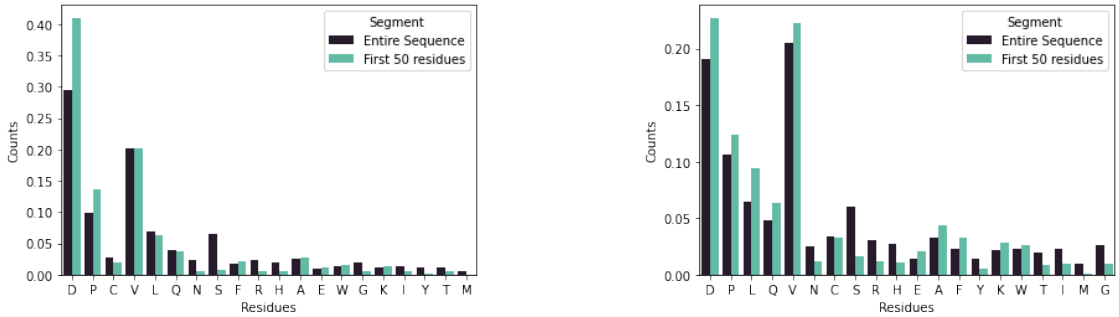
If we compare the distribution of 3Di residues of the entire sequence with the first 50, it can be seen that the signal peptide does not follow the same distribution. We have that 3D structure predictors and also experiments tend to resolve the ends of the sequences with lower resolution because those are on average less constrained.

Thus, the lower resolution and the special characteristics of the signal peptide could be contributing to a different distribution of the first 50 residues. Therefore, these datasets could be problematic.

In Figure 3.2 both distributions for setDeepLoc are plotted, together with the same analysis but for the binding prediction task dataset. Comparing both datasets it can be seen that the subcellular localization dataset has a special difference between both distributions. It is also clear when we compute the KL-Divergence[2] between the distribution of both sets. For setDeepLoc it is 0.1167, while for the binding prediction set it is 0.0889.

It has also been noticed that most of the 3Di sequences start with a large sequence of `D` states, while for the same amino acid sequences the beginning presents more variety.

[2]The Kullback-Leibler (KL) divergence measures how one probability distribution differs from another one. It is expressed as $D_{KL}(P\|Q) = \sum_{x\in\mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$.



(a) Subcellular Localization

(b) Binding Residues

Figure 3.2.: Normalized distribution/counting of residues for entire sequences and only the first 50 states for the datasets Subcellular Localization and Binding Prediction.

The second faced problem is the limitation of AlphaFold/ColabFold to process some sequences. It may be easy to understand that for the largest sequence in the set AF2 may have problems since the MSA could be extremely complex; but this also happens with medium size sequences, all of them above 2000 residues. In order to address the problem without further reducing the set, every problematic sequence has been truncated at 1000 residues. Although it may not seem to be the best solution, the most important information for this task is concentrated at the beginning of the sequence (signal peptide), thus, this solution becomes the simplest and most effective one.

The number of truncated sequences is 69 and 56, for setDeepLoc and setHARD respectively. In terms of structural quality, from the truncated sequences, only 14% have a pLDDT smaller than 50, while for the entire set 73% of the sequences have pLDDT larger than 70, 24% between 50 and 70, and only the 3% less than 50. For the setHARD set the distribution of structural quality is preserved.

For the evaluation of this task, the Q10 accuracy is going to be used because there are 10 possible locations and the focus will be on the so-called Hard set since it presents better qualities for a correct evaluation, but the results for the Soft set will also be included.

3.1.4. Binding Residues

At the interface between chemical-related and structure-related tasks, an interesting one is the prediction of binding residues. It is a widely studied problem, without clear solutions at the moment of this work, but with promising solutions using geometric deep learning, especially for molecular docking or drug binding, like EquiBind [23].

Another approach has been the combination of MSAs with complementary information, and solutions like bindEmbed21 use information from protein embeddings [84]. The deep learning version of it, named bindEmbed21DL, is the architecture that is going to be used in this work, together with the provided datasets: DevSet1014 for cross-validation development and TestSetNew300 for testing. Table 3.3 summarizes the characteristics of the set.

# Sequences			# Residues	Min. length	Max. length	# Seqs. with non-std. residues
Training	Validation	Testing	Total			
1014†		300	233375	31	813	4

Table 3.3.: Characteristics of the Binding Prediction task datasets. An extended version of this table can be found in Table B.1 and Table B.2.
†Training and validation set, i.e. the development set, is used in a cross-validation manner.

For this case, only 4 sequences have some non-standard residues (all of them from the testing subset), which only supposes a reduction of 505 residues, i.e., 0.2% of the original ones. This should not be a problem, because even if for this task all of them are of interest, the relative reduction is minimal. In terms of structure quality, 1267 sequences (96,71%) have a pLDDT score higher than 70, 40 sequences (3%) between 50 and 70, and only 3 sequences (0,22%) lower than 50.

The code used for the evaluation of this task [84] offers an exhaustive analysis with different measures overall and per class, but in order to summarize the results and make easier the extraction of conclusions, the overall results are going to be used. The metric will be F1 since it is a suitable measure in order to compare models when the dataset is unbalanced (the proportion of binding residues is small compared to the total number of residues), like in this case.

3.2. Models and Training

With the aim of evaluating the possibilities of 3Di states for the previously introduced downstream task, and in order to find the best combination of architecture and hyperparameters, different models have been trained.

Besides the direct use of 3Di states, two types of concatenations have been considered. On the one hand, the **embedding concatenation** (named `AA+SS`), i.e., after producing embeddings for a protein using an amino acids-based model and a 3Di-based model, both are concatenated at the residue level. In other words, the amino acid per-residue embedding is extended with the embedding of its corresponding 3Di state. On the other hand, the **direct concatenation** (named `AAss`), i.e., instead of the existing 20 possible inputs (amino acids) for the model, in this case, there are 400 possibilities corresponding to all the possible combinations between amino acids and 3Di states (e.g., `Ad` which corresponds to an alanine residue with 3Di state `D`).

The first idea is the most common one when we want to combine features from different sources. However, given an entire embedding per residue of size $n_{AA+SS} = n_{AA} + n_{SS}$, the task-specific model should be capable of discovering that the first n_{AA} embedding entries are related to the next n_{SS}. In order to simplify this learning, the second type of concatenation is preferred.

Taking both types of concatenations into account, apart from the 3Di-based models, AA-based models and models based on direct concatenation have been trained. Table B.3 summarizes the different trained models together with their training schemas, architectures, and hyperparameters. In order to reference latter each model, each one has an assigned name that summarizes it.

Names follow the pattern: `Type_ModelSize_Training_Dataset_Batch_DropOut`, where

- `Type` can be `AA`, `SS`, or `AAss` (SS states for 3Di or structure sequences, and AAss for the direct concatenation);
- `Model` can be `Abs` for BERT with Absolute Positional Encoding, `Rel` for BERT with Relative Positional Encoding, or `Rot` for RoFormer with Rotary Positional Encoding;
- `Size` specifies the model size following the original BERT sizes [6], e.g. `Tiny` or `Mini 8L` for a Mini model with 8 layers;
- `Training` specifies the type of training: `Const` when a constant upper bound on the length is used or `Prog` when a multi-phase training is performed. For the last one the input length increases, e.g., the model is trained on sequences up to a length of 256 and after a while switch to training on input length up to 512;
- `Dataset`, as specified in subsection 3.1.1, could be `U50` if the training set is UniRef50 (or a subsampling), or `RRU50` if the set is the reduced redundancy version of UniRef50;
- `Batch` indicates the used batch size or `Varied` if different batch sizes have been used in a progressive training schema;
- and `DropOut` can be `DO` if the original BERT dropout is used, or `NODO` if there is no dropout.

In order to train these models, three different GPU systems have been used, and they are summarized in Table 3.4. The RTX models have been used for training small models, and the DGX has been used exclusively for training the largest models described in Table B.3.

Manufacturer	**Model**	**Memory**
NVIDIA	Quadro RTX 8000	46GB
	RTX A6000	48GB
	DGX A100	80GB

Table 3.5.: GPU systems for model training.

Following, there are clarifications about some explored architectures and hyperparameters. Due to the proof-of-concept nature of this work and the complexity associated with training Transformer models (or big language models in general), the grid of explored hyperparameters is limited, but it has been tried to cover as many possibilities as feasible in order to gain a wide view of the capacities of 3Di.

3.2.1. Model Size

Since training a Transformer model implies complexity in terms of memory and time, and due to the proof-of-concept nature of this work, it has been decided not to work with unnecessarily big models.

BERT established a set of common model sizes based on the number of layers L, the hidden size H, the intermediate size I, and the number of self-attention heads A. This set of models is summarized in Table 3.6. The name of the different BERT models follows the nomeclature BERT Tiny, BERT Large, etc.

Name	L	H	I	$A^{\dagger}$	**#Parameters***
Tiny	2	128	512	2	4M
Mini	4	256	1024	4	11M
Small	4	512	2048	8	29M
Medium	8	512	2048	8	41M
Base	12	768	3072	12	110M
Large	24	1024	4096	16	340M

Table 3.6.: Standard BERT models. L stands for the number of layers, H for the hidden size, I for the intermediate size, and A for the number of attention heads.
†The number of attention heads follows the rule $H/A = 64$.
*The number of parameters is an approximation and uses the original configurations, without any modification.

In order to rapidly explore the capacities of 3Di, the Tiny model without any modification except the vocabulary size has been used. Then, following the efficient scalation idea proposed by Tay et al. [85], large models have been trained.

The authors of the study propose that instead of scaling models in two directions, i.e. the number of layers and the size of the layers, the same (or close) accuracy as for big models can be achieved by increasing only the number of layers without increasing the size per layer. This while not exploiting the number of parameters. This scaling strategy allows for increasing the capacity of models just by increasing the complexity in terms of training time and memory. The original study uses T5 [49], but it is extensible to any type of Transformer-based model.

To illustrate the way of working, a T5 Base model has 223M parameters, while using T5 Small with 16 layers (named Small 16L) a surprisingly closed to the Base performance can be achieved, but only with 134M parameters and 7.2 TFlops (T5 Base needs 11 TFlops).

Regarding RoFormer, due to the Rotary Positional Encoding, the number of parameters has a slight variation. For example, while BERT Base, with vocabulary size 25, relative positional encoding, and 512 maximum position embeddings, has $\sim$87M parameters, RoFormer with the same configuration has $\sim$86M, which is not a relevant difference. What is important to remark is that the vocabulary size, the maximum number of position embeddings, and the type of positional encoding indeed impact the number of parameters and the model complexity. It can be seen if we compare BERT Tiny with vocabulary size 25, which has $\sim$0.6M parameters, and the original BERT Tiny with vocabulary size 30517 which has $\sim$4M

parameters.

The exact model sizes and their corresponding number of parameters, computed depending on the type of sequence, are summarized in Table 3.7.

Sequence Type	Model Name	Positional Encoding	*L*	*H*	*I*	*A*	#Parameters
AA/SS	Tiny	Relative	2	128	512	2	0.6M
		Rotary					0.4M
	Mini 8L		8	256	1024	4	6.3M
AAss	Tiny	Rotary	2	128	512	2	0.65M
	Small 16L		16	512	2048	8	50.7M

Table 3.7.: Characteristics and number of parameters for the different used model sizes. The number of parameters also depends on the vocabulary size (25 if `AA` or `SS`, and 405 for `AAss`, corresponding the 5 extra vocabs in both cases to the reserved words of BERT).

3.2.2. Training Schema

As can be seen on the distribution of sequence lengths of UniRef50 (see Figure 3.1), 60.5% of the sequences have less than 250 amino acids, and 87.6% less than 500. Taking into account that BERT's authors proposed to use 512 as the maximum number of position embeddings due to the quadratic complexity of the number of inputs, it is important to constrain them in order to have efficient training, especially when the distribution of lengths is the given one.

Typically, models are trained using a fixed input size, but thanks to relative positional encoding and rotary positional encoding, Transformers can learn from small sequences and then increase the allowed input size after training. ProtT5 [26] is an example since it was increased to a maximum of 5000 input amino acids after training.

Besides the positional encoding, it can also be hypothesized that this works due to the type of sequences because it is common to have repeated parts along the sequence. This is something expected to happen on multimers, especially on the homotypic ones because even if they are big proteins, they can be divided into (almost) identical smaller parts that could potentially fit on the model.

With the aim of decreasing the complexity of some models and turning the training easier, for the small models, continuous training is performed, while for big models progressive training is chosen whenever possible. That is, a large bunch of epochs consist in training the model on sequences of (or truncated at) 256 amino acids, and then a small number of final epochs are used for training, for example, on 512 and 1024 residues. With this schema, the model can learn from a small sequence the direct relation between contiguous amino acids, while learning at the same time small protein constituents. Later, it learns extra information about how these small constituents are distributed along the entire sequence.

3.2.3. Batch Size

It has been shown that Transformers often perform and learn better when the batch size is large [86], but this hyperparameter, like others, is extremely problem-specific, and this conclusion should be considered with caution.

Due to the proof-of-concept nature of this work and also considering the direct impact of the batch size on the memory complexity associated with the training, a large batch size will be established whenever possible. The batch sizes for big models will be found for every training environment using the Batch Size Finder[3] of PyTorch Lightning.

Since for bigger models, trained on the entire previously described sets, it is hard to reach a large batch size, the **gradient accumulation** technique is used. It consists in accumulating gradients for K small batch before doing a backward pass, allowing to simulate a large effective batch size. For example, if the batch size is 256 and the $K = 2$, the effective batch size becomes 512.

The common equation for updating the model parameters

$$W_{n+1} = W_n - \alpha \nabla L(W_n) \tag{3.1}$$

becomes

$$W_{n+1} = W_n - \alpha \left(\sum_{i=0}^{K} (\nabla L(W_n))_i \right) \tag{3.2}$$

3.2.4. DropOut

The regularization technique DropOut [87], together with others, has been widely used in order to avoid overfitting. It consists in randomly setting to zero weights of visible and hidden neurons of the weights matrix while training, and it has been shown to be a really good regularization technique for a variety of neural network types.

For Transformers, it plays a role in two different parts. On the one hand, it is used for the self/cross multi-head attention layers, and on the other hand, it is used for the point-wise feedforward networks, being in BERT 0.1 the probability of both dropouts.

Recently, for models like T5v1.1 or T5X of T5 [49], it has been proposed to disable the dropout while training and enable it for fine-tuning, since the goal of pre-training is the absorption of as much information as possible. Also, it has been shown that disabling it the impact is positive and there is a quality win.

From another point of view, this also makes sense if the complexity of the languages and models is taken into account because overfitting a Transformer is something hard compared to other types of neural networks.

Then, it seems reasonable to analyze the impact of using dropout and consider the possibility of not using it.

[3] It is used exclusively on the *power* mode and it starts from batch size 1 and progresses doubling the size until reaching an out-of-memory. This allows finding the largest batch size for a given specific model, data, and training environment

3.2.5. Optimizer, Learning Rate, and Training Strategy

For other types of Deep Learning models like FNNs and Convolutional Neural Networks (CNNs), Adam optimizer [88] has become the gold standard because it allows the model to learn faster since the algorithm boosts the optimization when the gradient is constant, and it reduces the step-length over the optimization space when the gradient changes constantly.

This way of working is a result of the L_2 regularization, or weight decay, which can be understood as a reduction of the weight per step. It is based on the idea that models perform better and avoid overfitting when their weights are small. It has been shown that Adam generalizes worse than other optimizers like Stochastic Gradient Descent, because this weight decay factor is added to the model cost function which is derived to calculate the gradient, causing the optimizer to track the regularization as well as the model loss. This perturbates the optimization and does not work as it was intended.

With the aim of solving this issue, AdamW [89] was introduced. Instead of adding the weight decay to the cost function, it is added after computing the step size of the next iteration.

Since AdamW seems more consistent, it is going to be used in this work for training small models. For large models ($n \times m$ parameters), the memory requirement of Adam or AdamW is $\mathcal{O}(nm)$, which could turn the training unfeasible when the resources are limited. Several optimizers have been introduced to address this memory problem, for example, SM3 [90] or Adafactor [91]. For this project, Adafactor has been chosen since it is based on Adam, is well established, and is available in the used libraries. It reduces the memory complexity to $\mathcal{O}(n + m)$.

Regarding optimizer-related hyperparameters, they have been set to their default values defined in PyTorch Lightning. Tuning and finding the best configuration can become a complex task. The gain could also be insufficient beyond some small boosting on the training, so it has been decided not to pursue any complicated tunning of the optimizers' hyperparameters.

In order to increase the resources available for the training, i.e. the number of GPUs and subsequently, the available memory, the DeepSpeed Strategy [92] is going to be used when training big models on large datasets. It provides the possibility of training models in a distributed system while optimizing communication. It also allows to work with mixed precision and it is code and model-agnostic, avoiding unnecessary refactorizations. As an example, using 16 V100 GPUs and DeepSpeed, the training of BERT Large takes more than one day. This can be reduced to less than an hour with 1024 V100 GPUs and DeepSpeed.

DeepSpeed provides the so-called ZeRO Stages. For ZeRO Stage 1, it shards optimizer states; ZeRO Stage 2 shards optimizer states and gradients; and, ZeRO Stage 3 shards optimizer states, gradients, and parameters. For this work, ZeRO Stage 2 has been chosen since it reduces the memory impact without affecting communication, which is important since the resources used for this project are shared with other people.

3.3. Task-specific Models

In order to test the capacities of our models for the downstream tasks described in section 3.1, a set of suitable models have been chosen. They are summarized in Table 3.9.

Task	Model	Optimizer	Loss	Learning Rate	Batch Size	Epochs	Source
Secondary Structure	2-layers CNN with kernel size 7, padding 3 and bottleneck dimension 32	Adam	Cross-entropy	1e-3	8	20 (best one is chosen)	Elnaggar et al. [26]
Subcellular Localization	Light Attention + 1-layer FNN with hidden size 32 and batch normalization				128		Stärk et al. [81]
Binding Prediction	2-layers CNN with kernel size 5 and bottleneck dimension 256	Adamax	Weighted Cross-entropy	1e-2	406	200 (until convergence)	Littmann et al. [84]

Table 3.9.: Task-specific models used for the downstream tasks described in section 3.1.

.

For Secondary Structure and Subcellular Localization tasks the training and the evaluation of the models have been done using biotrainer[4]. It is an automatic framework that simplifies the analysis of protein language models combined with tasks-specific models. The chosen architectures for these tasks have been widely tested on their respective datasets, which are the same ones used in this work.

Regarding the Binding Prediction task, it has been decided to use the original implementation, adapting the data and the used embeddings. This has been done in order to obtain directly comparable results with the original ones for the used dataset. This is not possible with biotrainer since it does not have the same model in its options, and due to the necessity of having output channels for every ligand type, a thing that is not available in the pipeline by the date of this work.

[4]Available in `https://github.com/sacdallago/biotrainer`.

4. Results

4.1. Training

After analyzing all the possibilities mentioned in section 3.2 through the different trained models (summarized in Table B.3), a clear conclusion comes out: the used hyperparameters and configurations provide stable training, or at least expected or common training behaviors. This conclusion has been extracted by taking a look at the different training curves which asymptotically decrease (see example in Figure 4.1). To be precise, they rapidly decrease for the first epochs and then get stabilized for the last epochs, this while having the learning gap[1] always small.

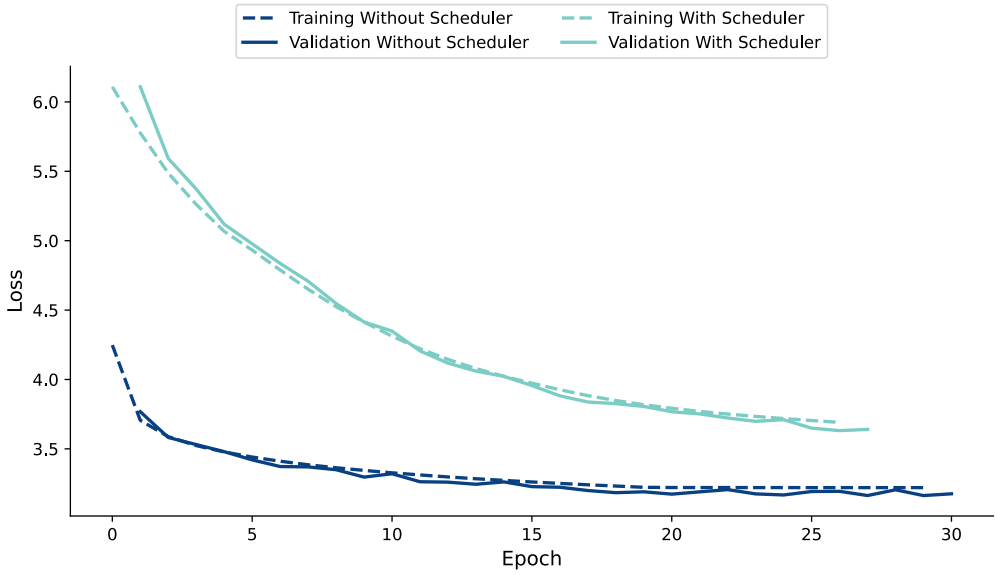


Figure 4.1.: Training curves of the Small 16L model for `AAss` sequences trained with and without a linear scheduler and learning rate (after warm-up) 1e-3 and 1e-5 respectively.

Besides the choice of training configuration itself, the most complex problem faced while training the different architectures has been the learning rate used for `AAss` models. Using the standard learning rate, 1e-3, that worked well for other models, for the Small 16L architecture

[1] It is known as the learning gap the distance between the validation curve and the training curve at each epoch. A small gap is a guarantee that we are probably not underfitting or overfitting, but to extract clear conclusions a test or downstream set must be evaluated.

and AAss sequences the model rapidly converged at a loss not significantly different from the one at the beginning. In order to solve this problem two alternatives were explored. First, a model using a learning rate of 1e-5 was trained; and secondly, given that the previous model had small steps in terms of loss per epoch, a model using a linear warm-up scheduler was trained.

A scheduler typically allows the optimizer to be more precise at some epochs, going faster or slower depending on the epoch or the step. In this case, a linear warm-up scheduler which starts the training using a zero learning rate and linearly increases it to 1e-3 during the first four epochs has been used. However, both trainings converged at the same point, as can be seen in Figure 4.1 and there is no clear advantage to using the scheduler. To extract this conclusion besides taking a look at the loss, both models were evaluated by comparing the performance of the checkpoints at epoch 25 for the different downstream tasks. The observed difference between both does not justify the use of a scheduler, and it was decided to abruptly interrupt the training of the model training with the scheduler. Following, when mentioning results for the AAss model with architecture Small 16L, they will be for the one without the scheduler.

Taking a look at Figure 4.1, included here as an example of the obtained training curves, we can conclude that at least the implemented training pipeline performs well and actually trains, but training curves do not explain the real capacities of the models, which will be evaluated on downstream tasks through the next section. Notice also that except for the AAss models, the other ones have been trained for 40 epochs instead of 30 due to the resources in terms of time and hardware required to train a Small 16L model on this type of sequences.

4.2. Evaluation

Although it is considered that some hyperparameters that work for small models could not be suitable for big ones, the complexity of Transformers forces us to start exploring how the different hyperparameters perform using a small model. Then, use those hyperparameters while training a big one, modifying only the ones that do not seem to work as expected. Following this idea, in this section different perspectives of the training using BERT and RoFormer Tiny models (see Table 3.6 for a description of the different standard BERT architectures) are studied.

After analyzing the different hyperparameters, the small models will be compared to one-hot encoding (baseline) and ProtT5 in order to establish references about how much we gain with respect to the baseline and how much we need to improve or not to reach or surpass the SOTA model. Thus, as a final step big model's results are dissected and compared to ProtT5.

Notice that, in order to analyze the results for the Secondary Structure Task, the Q3 accuracy is used, while for the Subcellular Localization, the Q10 accuracy is used. For the binding residues task instead, the F1 score has been chosen. See section 3.1 for more details and explanations about the different datasets and metrics used here.

Since the soft version of the Subcellular Localization Task has been shown that it is not

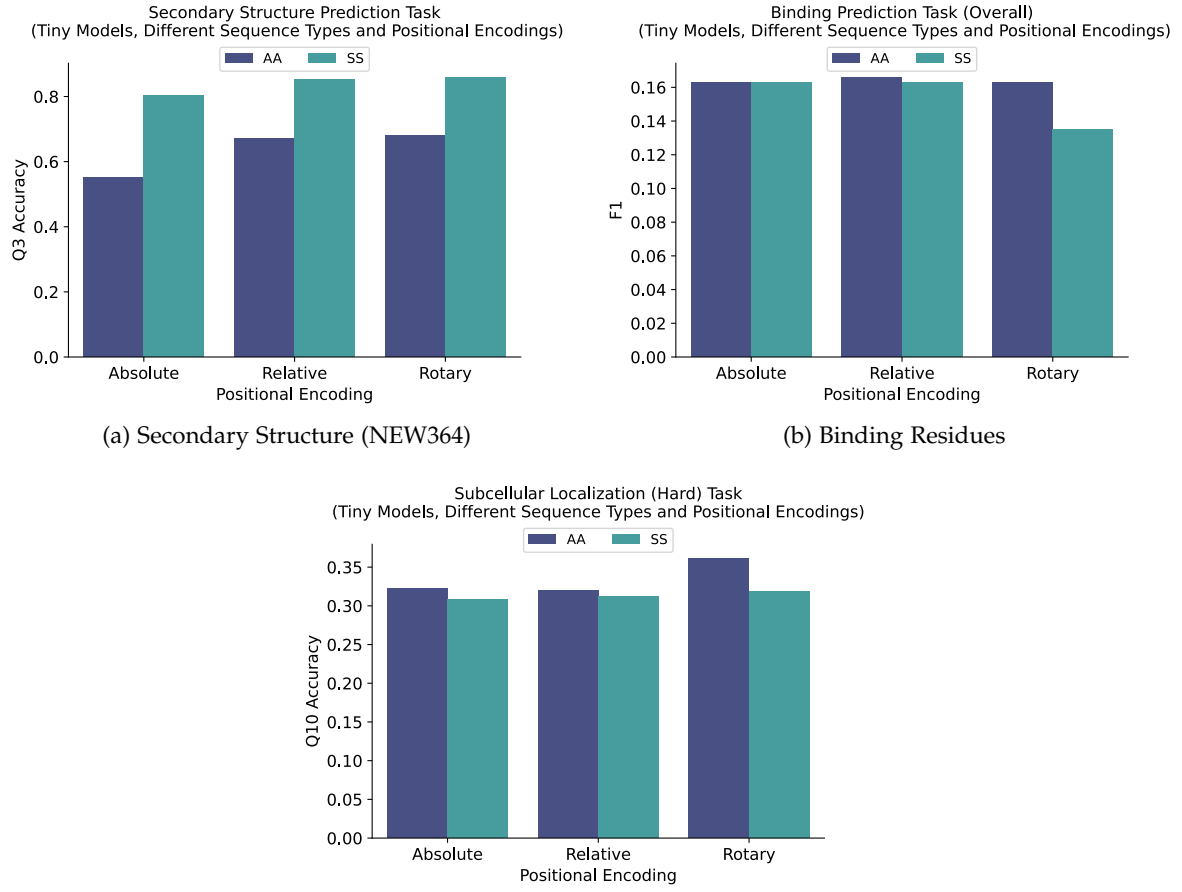
suitable for correct analysis, the results related to this set have been listed in for completeness because they are still used in different publications and could give extra information.

4.2.1. Evaluation of the Effect of Hyperparameters

Positional Encoding and Sequence Types

In the first place, an apples-against-apples comparison between amino acids-based models and 3Di-based models has been performed, exploring at the same time the behavior of the three possible positional encodings, i.e., absolute, relative, and rotary.

The results are shown in Figure 4.2, and the used models from Table B.3 are `AA_{AbsTiny, RelTiny, RotTiny}_Const_U50_32_D0` for the `AA`-based models, and `SS_{AbsTiny, RelTiny, RotTiny}_Const_U50_32_D0`, for the `SS`-based models.



(a) Secondary Structure (NEW364)

(b) Binding Residues

(c) Subcellular Localization (Hard)

Figure 4.2.: Results for the different downstream tasks using Tiny models trained on `AA` and `SS` sequences, comparing the capacities of the sequences types and the effect of different positional encodings (absolute, relative, and rotary).

From the Secondary Structure task performance, we can conclude that `SS`-based models

always perform better than AA ones. An explanation for this could be that 3Di is based solely on the structure, and these results confirm that these states correctly describe the secondary structure of proteins.

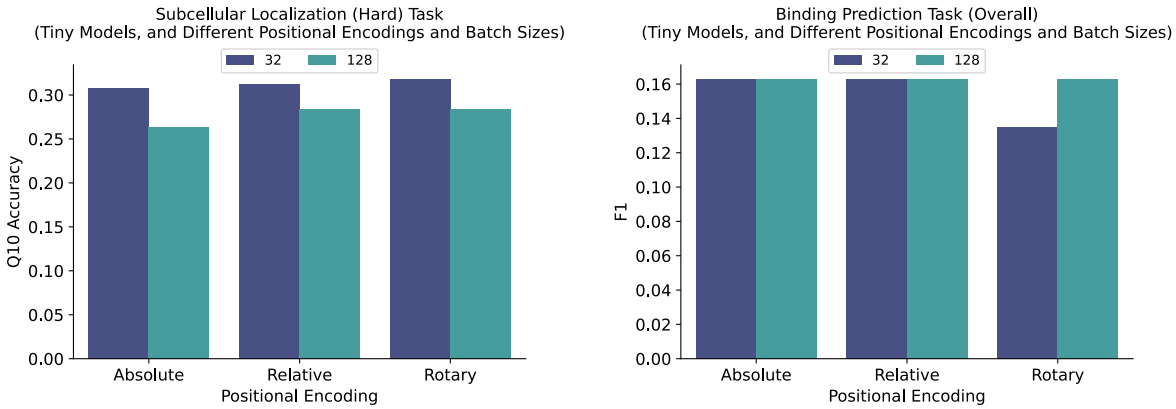
As is going to be explained later in this section, this performance is comparable to the one from ProtT5 [26], a model significantly much larger (3B parameters) than Tiny. Because of this, this task is going to be partially ignored in the analysis from now on.

On the other hand, while for the Secondary Structure task, the difference between relative and rotary posit1ional encodings is negligible, this is not the case for the Subcellular Localization tasks. We can see that the three encodings perform similarly for SS sequences, but in the case of AA rotary positional encoding performs significantly better. The same conclusions cannot be extracted from the Binding Prediction task's results, for which both AA and SS models perform identically except for the rotary encoding combined with SS sequences. For the latter case rotary performs numerically worse, but it is not substantial.

Batch Size

Leaving aside the models based on AA and focusing on the SS-based ones, following the influence of the batch size is analyzed, because it has been stated that Transformers typically perform better with large batch sizes, for example by Popel and Bojar [86].

The models used for this analysis are `SS_{AbsTiny, RelTiny, RotTiny}_Const_U50_{32, 128}_DO`.



(a) Subcellular Localization (Hard)

(b) Binding Residues

Figure 4.3.: Results for the different downstream tasks using Tiny models trained on SS sequences, comparing the capacities of different positional encodings (absolute, relative, and rotary) trained with different batch sizes (32 and 128).

From the results shown in Figure 4.3 we could conclude that the use of a large batch size does not improve the performance, but this might need further evaluation in future work due to the small model sizes (as indicated in Table 3.7, they have 0.6M and 0.4M parameters) that might not generalize to large ones. The used models are comparatively small compared to the ones evaluated for example by Popel and Bojar [86]. Since the exhaustive analysis of

this hyperparameter for big models surpasses the objectives of this work and the available resources, it has been decided not to extract a conclusion, and continue with the idea that a large batch size works better when training LM.

Regarding the Binding Prediction task, considering the models exposed until this point, it can be concluded that the Tiny architecture may not offer the capacity to achieve relevant or good results. Moreover, we can consider that it is a stagnant architecture, since no modification implies any change. Due to this, it is not going to be used for the remaining analysis using the Tiny architecture.

DropOut

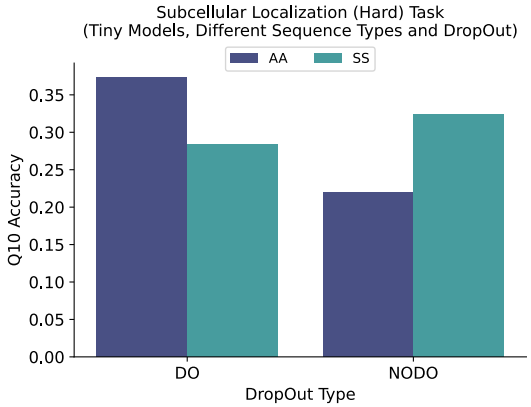


Figure 4.4.: Results of the AA and SS models {AA, SS}_RelTiny_Const_U50_128_{DO, NODO} for the Subcellular Localization (Hard) tasks using and without using dropout.

Newly released models are advocating not to use dropout (see discussion in subsection 3.2.4), and this is something that could be interesting to analyze, at least with small models. This has been done using the AA and SS models AA_RelTiny_Const_U50_128_{DO, NODO} and SS_RelTiny_Const_U50_128_{DO, NODO} described in Table B.3.

As it may be seen in Figure 4.4, when the dropout is not used, for AA the performance worsens, while for SS it improves.

These divergent results have not been observed for the other datasets for which the performance remains the same or improves when not using dropout. Thus, it can be concluded that no clear information can be extracted from this analysis and the idea stated by other studies is going to be used, i.e., the not use of dropout.

Other Hyperparameters

Regarding the training strategy, i.e. constant training or incremental size after bunches of epochs, analyzed using AA_RelTiny_Const_U50_32_DO and AA_RelTiny_Prog_U50_32_DO, no difference has been observed, nor using UniRef50 and Redundancy Reduced UniRef50 (see subsection 3.1.1 for a description about the sets) (SS_RelTiny_Const_U50_32_DO vs.

SS_RelTiny_Const_RRU50_32_DO). These results could be expected since the used dataset for Tiny models only has 100k sequences, and the effect of both aspects could not be relevant at this level.

Overview

Analyzing the results using Tiny models, the following conclusions can be extracted:

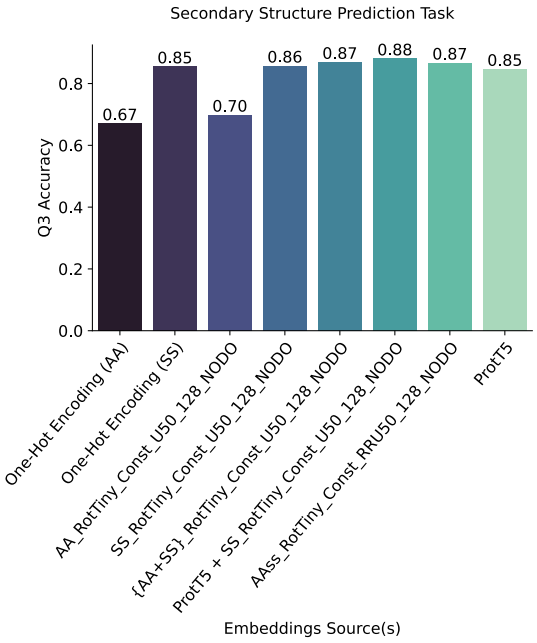
- Tiny SS models clearly outperform Tiny AA models for the Secondary Structure task, which proves the solidity of 3Di states for being informative about the structure of proteins.
- Tiny models do not offer sufficient capacity for the Binding Residues task for any sequence type.
- In most cases Rotary Positional Encoding performs just as well or better than Relative Positional Encoding, and is almost always better than Absolute Positional Encoding.
- In general, for the Subcellular Localization task, SS models perform worse than AA models.
- There is no clear improvement in using large batch sizes while training Transformer models, which contradicts the widespread idea explored by Popel and Bojar [86]. But this result could be a rash conclusion due to the small dataset used and the size of the model.
- Disabling DropOut, some apparent gain can be made regardless of whether AA or SS is used, but this does not seem to be a strict result at this level and for all cases.

4.2.2. Evaluation of Tiny Architecture w.r.t. the Baseline and the SOTA

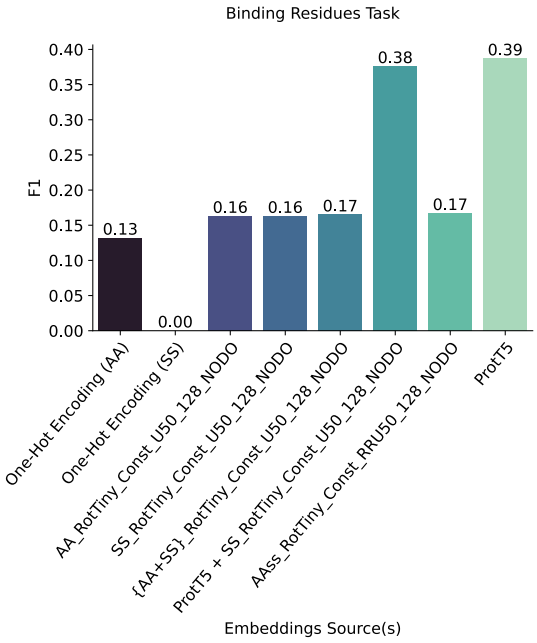
At this point, it is interesting to compare the performance of AA and SS models in agreement with the exposed conclusions extracted from the analysis so far (models with rotary positional encoding, without dropout and trained with batch size 128, i.e. {AA, SS}_RotTiny_Const_U50_128_NODO, not used for the previous analysis) to the baseline (one-hot encoding) and the SOTA model (ProtT5). A comparison to models trained with the direct concatenation of sequences (AAss models) and the concatenation of embeddings (between small models and with ProtT5 embeddings) has been included.

The objective of this comparison is to discern how far the smallest possible SS model and concatenations are from the AA SOTA one, and compared to some baseline, in this case, the one-hot encoding (the simplest possible embedding) for both AA and SS sequences.

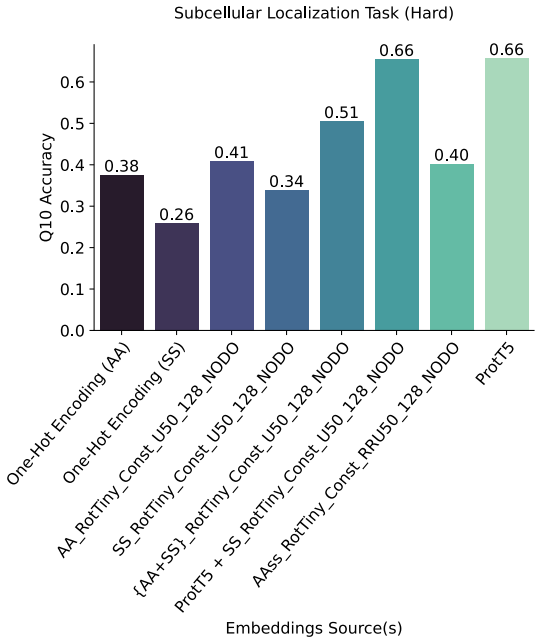
The results are summarized in Figure 4.5. Notice that, for evaluating AA models including ProtT5, the amino acids counterpart from 3Di sets have been used (see section 3.1 for details), and this can cause a difference between the previously published results and the ones here for the same datasets.



(a) Secondary Structure (NEW364)



(b) Binding Residues



(c) Subcellular Localization (Hard)

Figure 4.5.: Results for the different downstream tasks comparing baseline (one-hot encoding) and SOTA (ProtT5) models to Tiny architecture models and different concatenations (sequences (`AAss` models) and embeddings (`AA+SS` and `ProT5+SS`)).

Taking a look at the results for the Secondary Structure task, we can see that any possibility including 3Di states (except for the baseline) outperforms the SOTA model. The best accuracy is given by the concatenation of ProtT5 embeddings and Tiny model trained on SS.

Surprisingly, the direct concatenation of sequences (AAss model) and the embeddings concatenation (AA+SS models) achieve the same accuracy (0.87) even if the models were trained with the same architecture, hyperparameters, and for the same number of epochs, but with a clear higher complexity for AAss because the vocabulary, in this case, is 400, and for AA there are only 20 characters.

Regarding the Binding Residues Task, models using 3Di are not able to reach at least the performance of ProtT5, and they narrowly overpass amino acids' one-hot encoding. In fact, the use of 3Di seems to worsen the sole performance of ProtT5. About the 0.0 accuracy achieved by 3Di's one-hot encoding, there are no clear explanations beyond that it has not exceeded the accuracy of 0.009.

Finally, concerning the Subcellular Localization task, no model has been able to surpass the performance of ProtT5 even if they improved the accuracy of one-hot encoding models.

4.2.3. Evaluation of Large Architectures w.r.t. the Baseline and the SOTA

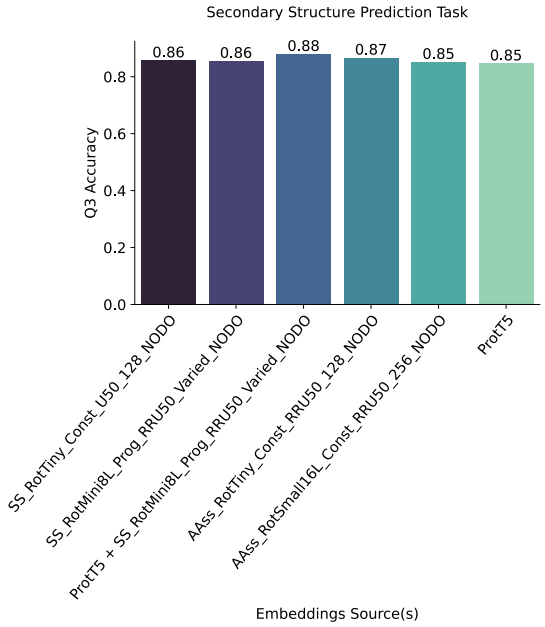
To conclude the analysis, following the comparison between the trained big models (based on the Mini 8L or Small 16L architectures) and the reference (ProtT5) is disclosed including also the performance of the concatenations and AAss. For completeness, the results of the best SS and AAss small models have also been incorporated into the plots.

Note that the idea behind training a Mini 8L model for SS instead of using the Small 16L architecture is due to the clear outperformance of the AAss model. Another cause is that it does not seem likely that an SS model will provide improvements if the architecture is larger. It is not justified to spend resources when one type of sequence is better than another.

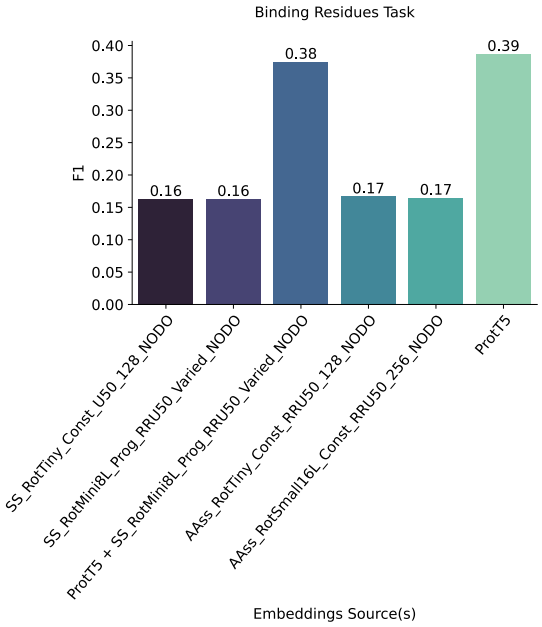
Starting with the Secondary Structure Prediction task, no extra conclusions can be extracted after comparing big models, except that surprisingly, there is no difference between using a Mini 8L or a Tiny architecture for the concatenation of ProtT5 and SS models' embeddings. This can be caused by the size of the embeddings produced by the SS models, but it is not clear. It is also remarkable that for AAss the use of a bigger architecture worsens the performance given by the Tiny model.

Regarding the prediction of binding residues, no model has been able to perform significantly. Taking into account that even if the accuracy achieved by Littmann et al. [84] left room for improvement, it is one of the best ones in the literature and was achieved using ProtT5, a model with 3B parameters. Thus, it does not seem feasible that outstanding results can be obtained with a 50M parameters model. But, at the same time, it is surprising that the use of combined chemical and structural information does not contribute to obtaining superior results using only structural information.

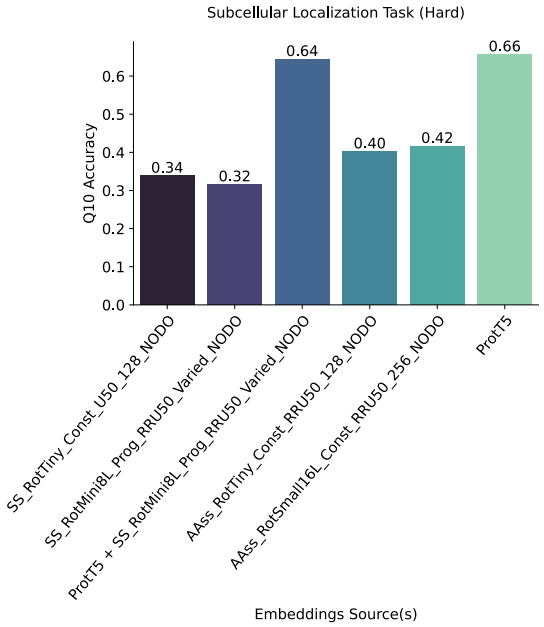
Finally, the results of the Subcellular Localization task look somewhat promising regarding the use of AAss sequences, but for the other models the performance is not improved or it worsens when increasing the size of the architecture.



(a) Secondary Structure (NEW364)



(b) Binding Residues



(c) Subcellular Localization (Hard)

Figure 4.6.: Results for the different downstream tasks comparing the SOTA (ProtT5) model to Small 16L architecture-based models and concatenations (sequences (`AAss` models) and embeddings (`AA+SS` and `ProT5+SS`)).

4.2.4. Results Overview

In order to sum up and simplify the different results disclosed while evaluating big models, below there is a list with the main conclusions:

- For secondary structure prediction, 3Di one-hote encoding performs on par with the SOTA model and the combination of information improves the results of the latter one by simply using a small model for 3Di. The best option is the concatenation of embeddings.
- For the binding residues task, all models perform better than the baselines but without being numerically significant. Moreover, the combination of information through the concatenation of embeddings worsens the performance of the SOTA model. The performance of the concatenation of sequence types shows that there is still room to improve the model and/or its training.
- The performance for the subcellular localization task shows a similar behavior as for the binding residues task but in this case, all models perform significantly. Again, the concatenation of embeddings seems to be the most promising option and the concatenation of sequences shows that there is still room to improve the model and/or its training.

5. Conclusions

The objective of this work has been the analysis of the capacities that a synthetic set of states for protein structures can give us. This has been done through training LMs to later use their information for downstream tasks. In this case, 3Di states have been chosen due to their great foundation and their proven success in protein structure comparison, as well as their simplicity, since it is a 20 characters-based alphabet, which allows us to think in a similar way we do with amino acids.

Through this work, it has been demonstrated that it is possible to train models using 3Di states and also that the learning can be carried out using standard configurations. Nonetheless, this easy training has not been translated into good models for the production of embeddings to help us in the prediction of binding residues or subcellular localizations. The best explanation encountered for this demeanor is the simplicity of the states, since even for a human it could be easy to discern the structure of the protein just by taking a look directly at the 3Di sequence. Moreover, even if it has been established that the models actually predict the secondary structure stunningly well just by using a tiny architecture, the structural information seems not to play an additive information role for the tasks considered here.

In order to offset the scarcities of structural information, an alternative set of states resulting from the combination of amino acid states and 3Di states per residue has been introduced. However, even if the combination of both information sources seems a good theoretical approach, in this work, it has only been possible to establish that it is a promising direction and that future work needs to investigate scaling this approach up.

Nevertheless, some insights found aside from the main results have been exposed. Regarding rotary positional encoding, it has been shown that it may give us an extra gain or at least perform on par with relative positional encoding. It has also been disclosed that without using DropOut, good results can be achieved, leaving the training unhurt, as it has been defended in recently published models. Regarding 3Di states, it has been shown the solidity of the set with respect to the representation of structural information, obtaining results that surpass SOTA models.

5.1. Future Work

From the author's point of view, the results here explained may establish a good base for future research, because even if the results for the analyzed downstream tasks do not look promising, different fronts are still open.

Firstly, it might be interesting to analyze the capacities of 3Di states for tasks more related to the structure. For example, Transmembrane proteins-related tasks could be a great starting

point, since there is still a lot to explore for this type of proteins due to the complexity of analyzing them. When combining structural and chemical information, it may be interesting to also explore other tasks like the prediction of solvent accessibility or the protein-protein interaction.

Subsequently, in order to address tasks that need chemical and structural information combined in some way, it may be interesting to train larger and more powerful models on `AAss` states, since they have been shown promising, but the limited time of this work has prevented this possibility from being explored.

A. Processes and Implementations

In this appendix, the process followed to create the 3Di datasets and the implementation of the different models are explained in order to allow reproducibility and to establish processes for future research.

A.1. From AA to 3Di Datasets and back

Since 3Di datasets for the evaluated downstream tasks are not available due to the novelty of the set of states, the following conversion process has been followed.

1. For every sequence ID, its entry is searched in the AFDB, which is already available with 3Di states (see Foldseek repository (`https://github.com/steineggerlab/foldseek`). For the search, the pattern `AF-ID-F1-model_v3.cif` is used changing `ID` for every case. Isoforms could also be retrieved, but in this case, they have been discarded.

2. A length correspondence between the retrieved 3Di sequences and AA sequences is performed in order to verify that proteins are exactly the same. It has been detected cases for which the same ID is associated with different lengths.

3. For those discarded sequences and those that could not be recovered, a FASTA file is written. This file is then used by ColabFold to compute protein structures. For some cases, there are already available structures in the PDB, but it was detected that for some proteins there are missing atoms and this causes errors in Foldseek.

4. After the computation of structures, Foldseek is used to convert them to 3Di states using the following command sequence (see Foldseek repository for details about other options):
   a) Convert PDF files to a Foldseek database: `foldseek createdb PDB_FILES_PATH OUTPUT_DIR/db`.
   b) Create a symbolic link between the original database and the resulting 3Di database: `foldseek lndb OUTPUT_DIR/db_h OUTPUT_DIR/db_ss_h`.
   c) Convert protein structures to 3Di: `foldseek convert2fasta OUTPUT_DIR/db_ss OUTPUT_DIR/dataset.fasta`.

5. Once the 3Di sequences are computed using the structures given by ColabFold, another length comparison must be performed since if a non-standard residue is in a sequence then Foldseek introduces an extra residue.

6. ColabFold does not preserve the IDs but the order. Therefore, for the final FASTA file, the IDs need to be retrieved and changed.

7. Once the 3Di dataset is ready, due to the possible missing sequences, it is important to build an amino acids counterpart dataset. This is done by simply retrieving amino acid sequences with IDs in the 3Di set.

When sequences are too large to compute their MSA with the given database to ColabFold, it can fail to compute the structures. In this case, the sequences should be discarded or shortened whenever possible, for example, as it has been done for the Subcellular Localization task.

For the training sets the 3Di version for UniRef50 already available in the Foldseek repository has been used.

A.2. BERT and RoFormer

The entire implementation of the models and different pipelines, e.g. for the extraction of embeddings, has been based on PyTorch Lightning (PL), which allows for fast prototyping, and HuggingFace, the current gold standard in the Natural Language Processing area.

For the management of the data, the `LightningDataModule` class from PL combined with the classes `DataLoader` from PyTorch and `DatasetDict` from datasets have been used. The Lightning's data module allows simple creation of pipelines to load and funnel data into different stages of model training. For the tokenization, the `PreTrainedTokenizerFast` from HuggingFace has been used since it is based on Ruby and it is faster than other available options. Previously, tokenizers for the different sets were trained using the module `BertWordPieceTokenizer` from the `tokenizers` library. This module is specialized in BERT (and derived models), including the special tokens described in subsection 2.1.2.

Since RoFormer is a BERT-based model all the different pipelines and implementations used to manage different data aspects can be directly used for RoFormer.

To manage the training, the `LightningModule` from PL has been used. The used model definitions have been `BertForMaskedLM/BertConfig` and `RoFormerForMaskedLM/RoFormerConfig` from HuggingFace. Other architectures are available for different training objectives, but since we only need masked modeling, these have been the chosen ones. Regarding the optimizers used for the training, the implementation of AdamW from PyTorch and the AdaFactor implementation of HuggingFace have been used.

Regarding the tricks used to boost the training, both the scheduler and the strategy (DeepSeed) have been the ones already available in PyTorch Lightning in order to avoid possible incompatibilities.

B. Datasets and Models

B.1. Amino acid Datasets

Task	Dataset	# Sequences				# Residues					Min. seq. length	Max. seq. length	Non-standard Residues					# Seqs. w. non-std residues
		Train	Val	Test	Total	Train	Val	Test	Total	#Interest or resolved			X	U	B	Z	O	
Secondary Structure	ProtT5 dev set w. NEW364 testing	9712	1080	364	11156	2490021	270532	84541	2845094	2658218 (93%)	20	1632	166	10	-	-	1	77
Subcellular Localization	setDeepLoc	9503	1678	2768	13949	4926536	883439	1494308	7304283	All	40	13100	76	30	2	4	-	34
	setDeepLoc w. setHARD testing			490	11671			258091	258091	All			61	31	1	-	-	31
Binding Residues	DevSet1014 + TestSet300	1014 (CV)		300	1314	170686 (CV)		62689	233375	All	31	813	2	2	-	-	-	4

Table B.1.: Characteristics of the used amino acid datasets.

B.2. 3Di and Amino acid Counterpart Datasets

Task	Dataset	Δ Sequences				Δ Residues					Min. seq. length	Max. seq. length	Source		Quality			Altered seqs.
		Train	Val	Test	Total	Train	Val	Test	Total	Interest or resolved			AFDB	Colab	>70	50-70	<50	
Secondary Structure	ProtT5 dev set w. NEW364 testing	-66	-5	-6	-77	-16635	-509	-1068	-18212	-17554	20	1362	-	11079	10835	166	78	-
Subcellular Localization	setDeepLoc	-26	-3	-3	-32	-116347	-37862	-28448	-182657	0	40	2697	13687	230	10077	3370	469	69
	setDeepLoc w. setHARD testing	-26	-3	-3	-32	-116347	-37862	-9527	-163736	0	40	3214	11440	199	8448	2791	397	56
Binding Residues	DevSet1014 + TestSet300	-4		0	-4	-505		0	-505	0	31	813	1236	75	1267	40	3	-

Table B.2.: Characteristics of the 3Di and amino acid counterpart datasets obtained from the amino acids sets described in Table B.1.

B.3. Trained Language Models

<table>
<tr><th>Model Name</th><th>Sequence Type</th><th>LM Type</th><th>Positional Encoding</th><th>Size</th><th>Training Strategy</th><th>Training Set</th><th>Input size (training)</th><th>Training Batch Size</th><th>With DropOut</th></tr>
<tr><td>AA_AbsTiny_Const_U50_32_DO</td><td rowspan="10">AA</td><td rowspan="6">BERT</td><td>Absolute</td><td rowspan="4">Tiny</td><td rowspan="3">Constant</td><td rowspan="6">U50</td><td rowspan="4">512</td><td rowspan="2">32</td><td rowspan="3">Yes</td></tr>
<tr><td>AA_RelTiny_Const_U50_32_DO</td><td rowspan="4">Relative</td></tr>
<tr><td>AA_RelTiny_Const_U50_128_DO</td><td rowspan="2">128</td></tr>
<tr><td>AA_RelTiny_Const_U50_128_NODO</td><td>No</td></tr>
<tr><td>AA_RelTiny_Const_U50_16_DO†</td><td>16</td><td rowspan="4">Yes</td></tr>
<tr><td>AA_RelTiny8L_Const_U50_32_DO†</td><td>Tiny 8L</td><td rowspan="3">32</td></tr>
<tr><td>AA_RelTiny_Prog_U50_32_DO</td><td rowspan="5">Tiny</td><td>Progressive</td><td>128 / 256 / 512 / 1024</td></tr>
<tr><td>AA_RotTiny_Const_U50_32_DO</td><td rowspan="2">RoFormer</td><td rowspan="2">Rotary</td><td rowspan="6">Constant</td><td rowspan="7">512</td></tr>
<tr><td>AA_RotTiny_Const_U50_128_NODO</td><td>128</td><td>No</td></tr>
<tr><td>SS_AbsTiny_Const_U50_32_DO</td><td rowspan="11">3Di</td><td rowspan="3">BERT</td><td rowspan="2">Absolute</td><td>32</td><td rowspan="4">Yes</td></tr>
<tr><td>SS_AbsTiny_Const_U50_128_DO</td><td>128</td></tr>
<tr><td>SS_RelTiny_Const_U50_32_DO</td><td rowspan="4">Relative</td><td rowspan="2">32</td></tr>
<tr><td>SS_RelTiny_Const_RRU50_32_DO</td><td>RR U50</td></tr>
<tr><td>SS_RelTiny_Const_U50_128_DO</td><td rowspan="2">U50</td><td rowspan="3">128</td></tr>
<tr><td>SS_RelTiny_Const_U50_128_NODO</td><td>No</td></tr>
<tr><td>SS_RelTiny_Const_RRU50_128_DO†</td><td>RR U50</td><td rowspan="3">Yes</td></tr>
<tr><td>SS_RotTiny_Const_U50_32_DO</td><td rowspan="3">RoFormer</td><td rowspan="3">Rotary</td><td rowspan="3">U50</td><td>32</td></tr>
<tr><td>SS_RotTiny_Const_U50_128_DO</td><td rowspan="2">128</td></tr>
<tr><td>SS_RotTiny_Const_U50_128_NODO</td><td rowspan="3">No</td></tr>
<tr><td>SS_RotMini8L_Prog_RRU50_Varied_NODO</td><td>Mini 8L</td><td>Progressive</td><td rowspan="3">RR U50</td><td>256 / 512</td><td>1024 / 256</td></tr>
<tr><td>AAss_RotTiny_Const_RRU50_128_NODO</td><td rowspan="2">AAss</td><td rowspan="2">RoFormer</td><td rowspan="2">Rotary</td><td>Tiny</td><td rowspan="2">Constant</td><td>512</td><td>128</td></tr>
<tr><td>AAss_RotSmall16L_Const_RRU50_256_NODO</td><td>Small 16L</td><td>256</td><td>256*</td></tr>
</table>

Table B.3.: Trained models and their characteristics.

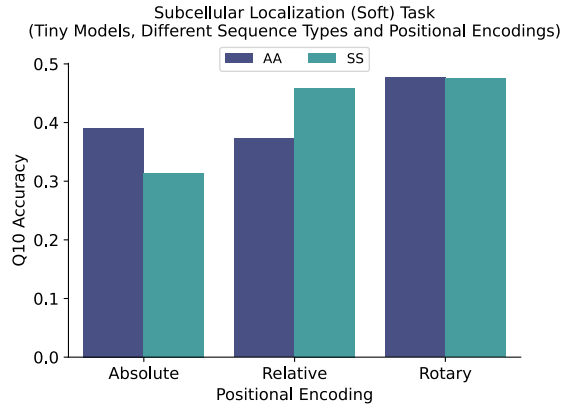
†These models have not finally been used for the analysis.

*Trained with accumulation gradient every 4 epochs, enabling an effective batch size of 1024.

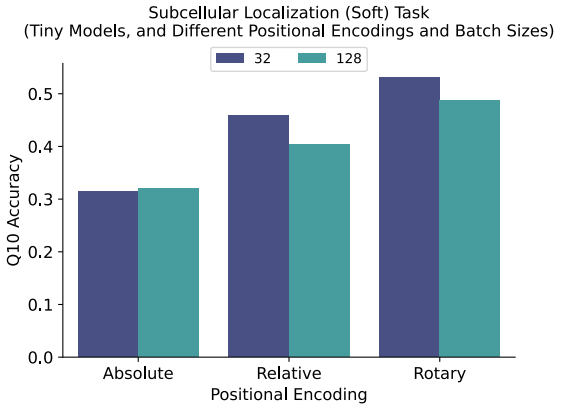
C. Extra Results

C.1. Subcellular Localization Soft

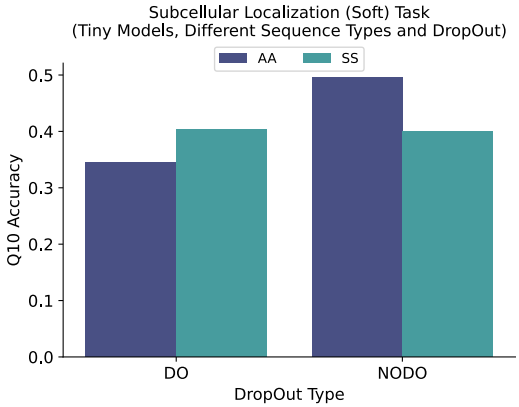
Here are listed using figures the different results obtained for the soft version of the Subcellular Localization task (see section 3.1 for a description of the dataset and its differences with respect to the hard set).



(a) Analysis of the sequence types (`AA` and `SS`) and positional encodings (absolute, relative, and rotary)

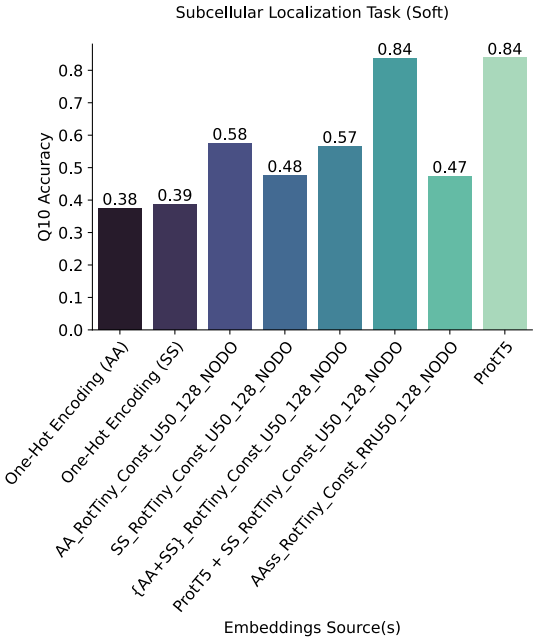


(b) Analysis of the batch sizes (32 and 128) and positional encodings

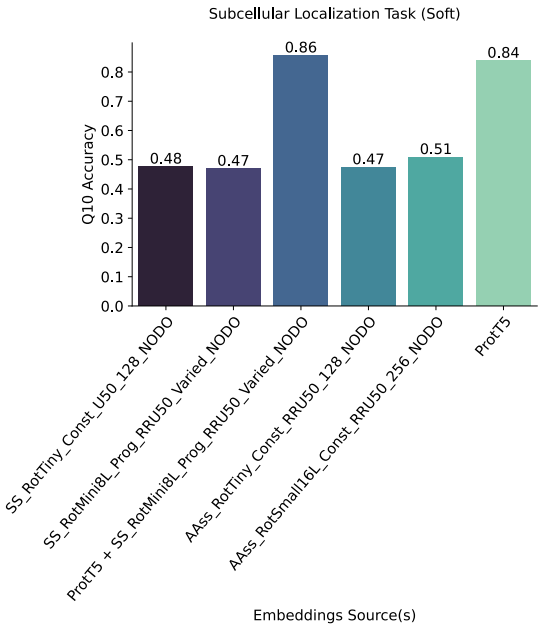


(c) Analysis of the use of DropOut

Figure C.1.: Results for the Subcellular Localization Soft dataset and different hyperparameters using Tiny architecture.



(a) Comparison between one-hot encoding, Tiny models, and ProtT5



(b) Comparison between different big architectures and concatenations

Figure C.2.: Comparison between different architectures and concatenations for the Subcellular Localization Soft dataset.

List of Figures

List of Tables

Glossary

AF2 Artificial Intelligence system for the prediction of protein structures. 49

AFDB Database with protein structures obtained using AlphaFold 2. It is maintained by Deepmind and EMBL-EBI. 49

CASP A bi-yearly competition that aims to find the best folding prediction model. 10, 49

CLM Transformers' training objective based on predicting the next token of a sequence given all the previous ones. 49

CNN Type of neural network mainly based on convolutional layers. 49

FNN It is the simplest network that can be achieved using perceptrons. It does not have cycles. 49

GDT Similarity measure between two protein structures. It is used to compare structure predictions to experimentally determined structures. It is calculated as the largest set of alpha carbon atoms in the model structure falling within a distance cutoff of their experimental structure positions. 49

GNN Type of neural networks devoted to working with graphs. 49

HBI Set of methods that perform inference over new sequences using homology information from other proteins. 49

LM Probability distribution obtained from a corpus and used to assign probabilities to word sequences. 4, 6, 7, 49

LSTM Type of recurrent network able to preserve a short-term memory for long-term, solving one of the problems with RNN. 3, 49

MLM Transformers' training objective based on predicting masked parts of the input sequences. 9, 49

MSA Alignment of three or more sequences (proteins, DNAs, or RNAs). It is used for example to analyze evolutionary relationships. 3, 49

NLP Discipline focused on the interaction between computers and human language, specifically on the understanding, processing, and analysis of natural language data. 3, 49

NSP Transformers' training objective based on predicting the next sentence for a given one. For example, the answer for a given question. 9

PAE Per pair of residues distance error given by AlphaFold. 11, 49

PDB Open bank of protein structures mainly obtained through X-ray Crystallography or Nuclear Magnetic Resonance. The acronym is also used to name the file format used to describe the structures. 49

PIDE Identity between two aligned sequences. Its calculation depends on the alignment and the definition of identity. 49

pLDDT Per-residue confidence estimation produced by AlphaFold for each predicted structure. 11, 16, 19, 49

pLM Language Model for which words are amino acids and the sentences are complete protein sequences. 3–6, 14–16, 49

RMSD Measure of the average distance between atoms of superimposed proteins. It is calculated as $RMSD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\delta_i^2}$, where N is the number of atoms and δ is the distance between a given atom and a reference structure. 2, 50

RNN Type of neural network endowed with memory and devoted to the processing of sequential information. 3, 50

SOTA Adjective for the best solution to a problem or task. 50

Acronyms

AF2 AlphaFold 2. 2, 9–12, 14, 18, 45, 47

AFDB AlphaFold Protein Structure Database. 2, 11, 14, 16, 38, 47

CASP Critical Assessment of Structure Prediction. 2, 10, 47

CLM Causal Languge Modeling. 7, 47

CNN Convolutional Neural Network. 24, 47

FNN Feed Forward Neural Network. 5, 16, 24, 47

GDT Global Distance Test. 2, 10, 47

GNN Graph Neural Network. 2, 47

HBI Homology-based Inference. 1, 2, 47

LM Language Model. 2–4, 6, 30, 36, 47

LSTM Long Short-Term Memory. 3, 4, 6, 47

MLM Masked Language Modeling. 7, 9, 47

MSA Multiple Sequence Alignment. 2, 10, 39, 47

NLP Natural Language Processing. 1, 3, 4, 9, 48

NSP Next Sentence Prediction. 7, 9, 48, 49

PAE Prediction Aligned Error. 11, 48

PDB Protein Data Bank. 1, 38, 48

PIDE Pairwise Sequence Identity. 17, 48

pLDDT Predicted Local-Distance Difference Test. 11, 16, 19, 48

pLM protein-Language Model. 2, 4–6, 16, 48

RMSD Root-Mean-Square Deviation. 2, 48

RNN Recurrent Neural Network. 2–4, 6, 47, 48

SOTA State-of-the-Art. 16, 27, 31–36, 48

Bibliography

[1] S. B. Needleman and C. D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". In: *Journal of molecular biology* 48.3 (1970), pp. 443–453.

[2] M. Dayhoff, R. Schwartz, and B. Orcutt. "A model of evolutionary change in proteins". In: *Atlas of protein sequence and structure* 5 (1978), pp. 345–352.

[3] S. Henikoff and J. G. Henikoff. "Amino acid substitution matrices from protein blocks." In: *Proceedings of the National Academy of Sciences* 89.22 (1992), pp. 10915–10919.

[4] M. Kumar, M. M. Gromiha, and G. P. S. Raghava. "Prediction of RNA binding sites in a protein using SVM and PSSM profile". In: *Proteins: Structure, Function, and Bioinformatics* 71.1 (2008), pp. 189–194.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[7] M. Heinzinger, A. Elnaggar, Y. Wang, C. Dallago, D. Nechaev, F. Matthes, and B. Rost. "Modeling aspects of the language of life through transfer-learning protein sequences". In: *BMC bioinformatics* 20.1 (2019), pp. 1–17.

[8] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žıdek, A. Potapenko, et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589.

[9] A. Kryshtafovych, T. Schwede, M. Topf, K. Fidelis, and J. Moult. "Critical assessment of methods of protein structure prediction (CASP)—Round XIV". In: *Proteins: Structure, Function, and Bioinformatics* 89.12 (2021), pp. 1607–1617.

[10] M. Varadi, S. Anyango, M. Deshpande, S. Nair, C. Natassia, G. Yordanova, D. Yuan, O. Stroe, G. Wood, A. Laydon, et al. "AlphaFold Protein Structure Database: massively expanding the structural coverage of protein-sequence space with high-accuracy models". In: *Nucleic acids research* 50.D1 (2022), pp. D439–D444.

[11] K.-T. Ko, F. Lennartz, D. Mekhaiel, B. Guloglu, A. Marini, D. J. Deuker, C. A. Long, M. M. Jore, K. Miura, S. Biswas, et al. "Structure of the malaria vaccine candidate Pfs48/45 and its recognition by transmission blocking antibodies". In: *Nature Communications* 13.1 (2022), p. 5603.

[12] W. M. Kincannon, M. Zahn, R. Clare, J. Lusty Beech, A. Romberg, J. Larson, B. Bothner, G. T. Beckham, J. E. McGeehan, and J. L. DuBois. "Biochemical and structural characterization of an aromatic ring–hydroxylating dioxygenase for terephthalic acid catabolism". In: *Proceedings of the National Academy of Sciences* 119.13 (2022), e2121426119.

[13] A. Castellví, A. Medina, G. Petrillo, T. Sagmeister, T. Pavkov-Keller, F. Govantes, K. Diederichs, M. D. Sammito, and I. Usón. "Exploring generality of experimental conformational changes with AlphaFold predictions". In: *bioRxiv* (2022).

[14] M. Zvyagin, A. Brace, K. Hippe, Y. Deng, B. Zhang, C. O. Bohorquez, A. Clyde, B. Kale, D. Perez-Rivera, H. Ma, et al. "GenSLMs: Genome-scale language models reveal SARS-CoV-2 evolutionary dynamics". In: *bioRxiv* (2022), pp. 2022–10.

[15] M. van Kempen, S. Kim, C. Tumescheit, M. Mirdita, J. Söding, and M. Steinegger. "Foldseek: fast and accurate protein structure search". In: *bioRxiv* (2022).

[16] J.-M. Yang and C.-H. Tung. "Protein structure database search and evolutionary classification". In: *Nucleic acids research* 34.13 (2006), pp. 3646–3659.

[17] A. G. de Brevern, C. Etchebest, and S. Hazout. "Bayesian probabilistic approach for predicting backbone structures in terms of protein blocks". In: *Proteins: Structure, Function, and Bioinformatics* 41.3 (2000), pp. 271–287.

[18] A. P. Joseph, G. Agarwal, S. Mahajan, J.-C. Gelly, L. S. Swapna, B. Offmann, F. Cadet, A. Bornot, M. Tyagi, H. Valadié, et al. "A short survey on protein blocks". In: *Biophysical reviews* 2 (2010), pp. 137–145.

[19] L. Pauling and R. B. Corey. "Configurations of polypeptide chains with favored orientations around single bonds: two new pleated sheets". In: *Proceedings of the National Academy of Sciences* 37.11 (1951), pp. 729–740.

[20] V. Gligorijević, P. D. Renfrew, T. Kosciolek, J. K. Leman, D. Berenberg, T. Vatanen, C. Chandler, B. C. Taylor, I. M. Fisk, H. Vlamakis, et al. "Structure-based protein function prediction using graph convolutional networks". In: *Nature communications* 12.1 (2021), p. 3168.

[21] J. Son and D. Kim. "Development of a graph convolutional neural network model for efficient prediction of protein-ligand binding affinities". In: *PloS one* 16.4 (2021).

[22] T. Xia and W.-S. Ku. "Geometric graph representation learning on protein structure prediction". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2021, pp. 1873–1883.

[23] H. Stärk, O. Ganea, L. Pattanaik, R. Barzilay, and T. Jaakkola. "Equibind: Geometric deep learning for drug binding structure prediction". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 20503–20521.

[24] D. Griffith and A. S. Holehouse. "PARROT is a flexible recurrent neural network framework for analysis of large protein datasets". In: *Elife* 10 (2021), e70576.

[25] A. Madani, B. McCann, N. Naik, N. S. Keskar, N. Anand, R. R. Eguchi, P.-S. Huang, and R. Socher. "Progen: Language modeling for protein generation". In: *arXiv preprint arXiv:2004.03497* (2020).

[26] A. Elnaggar, M. Heinzinger, C. Dallago, G. Rihawi, Y. Wang, L. Jones, T. Gibbs, T. Feher, C. Angerer, M. Steinegger, et al. "ProtTrans: towards cracking the language of Life's code through self-supervised deep learning and high performance computing". In: *arXiv preprint arXiv:2007.06225* (2020).

[27] N. Brandes, D. Ofer, Y. Peleg, N. Rappoport, and M. Linial. "ProteinBERT: A universal deep-learning model of protein sequence and function". In: *Bioinformatics* 38.8 (2022), pp. 2102–2110.

[28] A. Rives, J. Meier, T. Sercu, S. Goyal, Z. Lin, J. Liu, D. Guo, M. Ott, C. L. Zitnick, J. Ma, et al. "Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences". In: *Proceedings of the National Academy of Sciences* 118.15 (2021).

[29] J. Meier, R. Rao, R. Verkuil, J. Liu, T. Sercu, and A. Rives. "Language models enable zero-shot prediction of the effects of mutations on protein function". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 29287–29303.

[30] Z. Lin, H. Akin, R. Rao, B. Hie, Z. Zhu, W. Lu, N. Smetanin, R. Verkuil, O. Kabeli, Y. Shmueli, A. dos Santos Costa, M. Fazel-Zarandi, T. Sercu, S. Candido, and A. Rives. "Evolutionary-scale prediction of atomic level protein structure with a language model". In: *bioRxiv* (2022). DOI: 10.1101/2022.07.20.500902.

[31] L. Moffat, S. M. Kandathil, and D. T. Jones. "Design in the DARK: learning deep generative models for de novo protein design". In: *bioRxiv* (2022), pp. 2022–01.

[32] N. Ferruz, S. Schmidt, and B. Höcker. "ProtGPT2 is a deep unsupervised language model for protein design". In: *Nature communications* 13.1 (2022), pp. 1–10.

[33] R. M. Rao, J. Liu, R. Verkuil, J. Meier, J. Canny, P. Abbeel, T. Sercu, and A. Rives. "MSA transformer". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8844–8856.

[34] H. Dalla-Torre, L. Gonzalez, J. Mendoza-Revilla, N. L. Carranza, A. H. Grzywaczewski, F. Oteri, C. Dallago, E. Trop, H. Sirelkhatim, G. Richard, et al. "The Nucleotide Transformer: Building and Evaluating Robust Foundation Models for Human Genomics". In: *bioRxiv* (2023), pp. 2023–01.

[35] J. Su, Y. Lu, S. Pan, B. Wen, and Y. Liu. "Roformer: Enhanced transformer with rotary position embedding". In: *arXiv preprint arXiv:2104.09864* (2021).

[36] N. Kitaev, Ł. Kaiser, and A. Levskaya. "Reformer: The efficient transformer". In: *arXiv preprint arXiv:2001.04451* (2020).

[37] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. "Electra: Pre-training text encoders as discriminators rather than generators". In: *arXiv preprint arXiv:2003.10555* (2020).

[38] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. "Albert: A lite bert for self-supervised learning of language representations". In: *arXiv preprint arXiv:1909.11942* (2019).

[39] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *arXiv preprint arXiv:1910.01108* (2019).

[40] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[41] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[43] M. I. Jordan. "Serial order: A parallel distributed processing approach". In: *Advances in psychology*. Vol. 121. Elsevier, 1997, pp. 471–495.

[44] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[45] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. "Deep contextualized word representations". In: *arXiv preprint arXiv:1802.05365* (2018).

[46] D. Bahdanau, K. Cho, and Y. Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[47] R. Child, S. Gray, A. Radford, and I. Sutskever. "Generating long sequences with sparse transformers". In: *arXiv preprint arXiv:1904.10509* (2019).

[48] T. Lin, Y. Wang, X. Liu, and X. Qiu. "A survey of transformers". In: *AI Open* (2022).

[49] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." In: *J. Mach. Learn. Res.* 21.140 (2020), pp. 1–67.

[50] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. "Xlnet: Generalized autoregressive pretraining for language understanding". In: *Advances in neural information processing systems* 32 (2019).

[51] A. Wettig, T. Gao, Z. Zhong, and D. Chen. "Should You Mask 15% in Masked Language Modeling?" In: *arXiv preprint arXiv:2202.08005* (2022).

[52] P. Shaw, J. Uszkoreit, and A. Vaswani. "Self-attention with relative position representations". In: *arXiv preprint arXiv:1803.02155* (2018).

[53] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck. "Music transformer". In: *arXiv preprint arXiv:1809.04281* (2018).

[54] Z. Huang, D. Liang, P. Xu, and B. Xiang. "Improve transformer models with better relative position embeddings". In: *arXiv preprint arXiv:2009.13658* (2020).

[55] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[56] L. M. Bertoline, A. N. Lima, J. E. Krieger, and S. K. Teixeira. "Before and after AlphaFold2: An overview of protein structure prediction". In: *Frontiers in Bioinformatics* 3 (2023).

[57] D. Xu and Y. Zhang. "Ab initio protein structure assembly using continuous structure fragments and optimized knowledge-based force field". In: *Proteins: Structure, Function, and Bioinformatics* 80.7 (2012), pp. 1715–1735.

[58] N. Guex, M. C. Peitsch, and T. Schwede. "Automated comparative protein structure modeling with SWISS-MODEL and Swiss-PdbViewer: A historical perspective". In: *Electrophoresis* 30.S1 (2009), S162–S173.

[59] A. Šali and T. L. Blundell. "Comparative protein modelling by satisfaction of spatial restraints". In: *Journal of molecular biology* 234.3 (1993), pp. 779–815.

[60] M. Baek, F. DiMaio, I. Anishchenko, J. Dauparas, S. Ovchinnikov, G. R. Lee, J. Wang, Q. Cong, L. N. Kinch, R. D. Schaeffer, et al. "Accurate prediction of protein structures and interactions using a three-track neural network". In: *Science* 373.6557 (2021), pp. 871–876.

[61] M. Mirdita, K. Schütze, Y. Moriwaki, L. Heo, S. Ovchinnikov, and M. Steinegger. "ColabFold: making protein folding accessible to all". In: *Nature methods* 19.6 (2022), pp. 679–682.

[62] M. Steinegger and J. Söding. "MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets". In: *Nature biotechnology* 35.11 (2017), pp. 1026–1028.

[63] L. S. Johnson, S. R. Eddy, and E. Portugaly. "Hidden Markov model speed heuristic and iterative HMM search procedure". In: *BMC bioinformatics* 11 (2010), pp. 1–8.

[64] G. Ahdritz, N. Bouatta, S. Kadyan, Q. Xia, W. Gerecke, T. J. O'Donnell, D. Berenberg, I. Fisk, N. Zanichelli, B. Zhang, et al. "OpenFold: Retraining AlphaFold2 yields new insights into its learning mechanisms and capacity for generalization". In: *bioRxiv* (2022), pp. 2022–11.

[65] D. Li. *Understanding the significance and architecture of AlphaFold*. 2022. URL: `http://therisingsea.org/notes/metauni/notes-li-alphafold.pdf`.

[66] V. Mariani, M. Biasini, A. Barbato, and T. Schwede. "lDDT: a local superposition-free score for comparing protein structures and models using distance difference tests". In: *Bioinformatics* 29.21 (2013), pp. 2722–2728.

[67] C. Outeiral, D. A. Nissley, and C. M. Deane. "Current structure predictors are not learning the physics of protein folding". In: *Bioinformatics* 38.7 (2022), pp. 1881–1887.

[68] A. O. Stevens and Y. He. "Benchmarking the accuracy of AlphaFold 2 in loop structure prediction". In: *Biomolecules* 12.7 (2022), p. 985.

[69] R. Evans, M. O'Neill, A. Pritzel, N. Antropova, A. Senior, T. Green, A. Žıdek, R. Bates, S. Blackwell, J. Yim, et al. "Protein complex prediction with AlphaFold-Multimer". In: *BioRxiv* (2021), pp. 2021–10.

[70] R. Fasoulis, G. Paliouras, and L. E. Kavraki. "Graph representation learning for structural proteomics". In: *Emerging Topics in Life Sciences* 5.6 (2021), pp. 789–802.

[71] K. Liu, R. K. Kalia, X. Liu, A. Nakano, K.-i. Nomura, P. Vashishta, and R. Zamora-Resendizc. "Multiscale Graph Neural Networks for Protein Residue Contact Map Prediction". In: *arXiv preprint arXiv:2212.02251* (2022).

[72] A. Medina, J. Trivino, R. J. Borges, C. Millán, I. Uson, and M. D. Sammito. "ALEPH: a network-oriented approach for the generation of fragment-based libraries and for structure interpretation". In: *Acta Crystallographica Section D: Structural Biology* 76.3 (2020), pp. 193–208.

[73] D. Blakely, J. Lanchantin, and Y. Qi. "Time and Space Complexity of Graph Convolutional Networks". In: *Accessed on: Dec* 31 (2021).

[74] A. Van Den Oord, O. Vinyals, et al. "Neural discrete representation learning". In: *Advances in neural information processing systems* 30 (2017).

[75] L. Holm. "DALI and the persistence of protein shape". In: *Protein Science* 29.1 (2020), pp. 128–140.

[76] B. E. Suzek, Y. Wang, H. Huang, P. B. McGarvey, C. H. Wu, and U. Consortium. "UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches". In: *Bioinformatics* 31.6 (2015), pp. 926–932.

[77] Y. Nevers, N. Glover, C. Dessimoz, and O. Lecompte. "Protein length distribution is remarkably consistent across Life". In: *bioRxiv* (2021), pp. 2021–12.

[78] I. Barrio-Hernandez, J. Yeo, J. Jänes, T. Wein, M. Varadi, S. Velankar, P. Beltrao, and M. Steinegger. "Clustering predicted structures at the scale of the known protein universe". In: *bioRxiv* (2023), pp. 2023–03.

[79] J. J. Almagro Armenteros, C. K. Sønderby, S. K. Sønderby, H. Nielsen, and O. Winther. "DeepLoc: prediction of protein subcellular localization using deep learning". In: *Bioinformatics* 33.21 (2017), pp. 3387–3395.

[80] V. Thumuluri, J. J. Almagro Armenteros, A. R. Johansen, H. Nielsen, and O. Winther. "DeepLoc 2.0: multi-label subcellular localization prediction using protein language models". In: *Nucleic acids research* 50.W1 (2022), W228–W234.

[81] H. Stärk, C. Dallago, M. Heinzinger, and B. Rost. "Light attention predicts protein location from the language of life". In: *Bioinformatics Advances* 1.1 (2021), vbab035.

[82] K. Kapp, S. Schrempf, M. K. Lemberg, and B. Dobberstein. "Post-targeting functions of signal peptides". In: *Protein transport into the endoplasmic reticulum* (2009), pp. 1–16.

[83] G. Blobel and B. Dobberstein. "Transfer of proteins across membranes. I. Presence of proteolytically processed and unprocessed nascent immunoglobulin light chains on membrane-bound ribosomes of murine myeloma." In: *The Journal of cell biology* 67.3 (1975), pp. 835–851.

[84] M. Littmann, M. Heinzinger, C. Dallago, K. Weissenow, and B. Rost. "Protein embeddings and deep learning predict binding residues for various ligand classes". In: *Scientific Reports* 11.1 (2021), p. 23916.

[85] Y. Tay, M. Dehghani, J. Rao, W. Fedus, S. Abnar, H. W. Chung, S. Narang, D. Yogatama, A. Vaswani, and D. Metzler. "Scale efficiently: Insights from pre-training and fine-tuning transformers". In: *arXiv preprint arXiv:2109.10686* (2021).

[86] M. Popel and O. Bojar. "Training tips for the transformer model". In: *arXiv preprint arXiv:1804.00247* (2018).

[87] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[88] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[89] I. Loshchilov and F. Hutter. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017).

[90] R. Anil, V. Gupta, T. Koren, and Y. Singer. "Memory efficient adaptive optimization". In: *Advances in Neural Information Processing Systems* 32 (2019).

[91] N. Shazeer and M. Stern. "Adafactor: Adaptive learning rates with sublinear memory cost". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4596–4604.

[92] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He. "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, pp. 3505–3506.