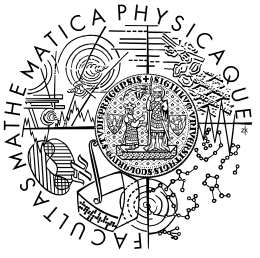


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Hugo Hrbáň

Generation of Protein Sequences With a Given Characteristic

Department of Software Engineering

Supervisor of the bachelor thesis: doc. RNDr. David Hoksza, Ph.D.
Study programme: Computer Science

Prague 2024

I declare that I carried out this bachelor thesis on my own, and only with the cited sources, literature and other professional sources. I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank doc. RNDr. David Hoksza, Ph.D. for his guidance and introducing me to the field of bioinformatics.

I would also like to thank my family and friends for their support throughout my studies.

Title: Generation of Protein Sequences With a Given Characteristic

Author: Hugo Hrbáň

Department: Department of Software Engineering

Supervisor: doc. RNDr. David Hoksza, Ph.D., Department of Software Engineering

Abstract: Proteins are essential for life as they play a fundamental role in many biological processes. Designing novel proteins with a desired function is an important problem in drug development and biological research. Large databases of protein sequences can be used to train large language models adapted from natural language processing on the language of proteins, written in the alphabet of amino acids. In this work, we demonstrate how large language models based on pretrained deep neural networks can be effectively finetuned for controllable generation of protein sequences from several distinct protein families. Using bioinformatic and deep learning-based methods, we show that the model is able to generate high-quality protein sequences that exhibit low similarity to existing proteins.

Keywords: bioinformatics, large language models, protein engineering

Název práce: Generování proteinových sekvencí s danou charakteristikou

Autor: Hugo Hrbáň

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: doc. RNDr. David Hoksza, Ph.D., Katedra softwarového inženýrství

Abstrakt: Proteiny jsou nezbytné pro život, protože hrají zásadní roli v mnoha biologických procesech. Navrhování nových proteinů s požadovanou funkcí je důležitým problémem ve vývoji léků a biologickém výzkumu. Velké databáze proteinových sekvencí lze použít k trénování velkých jazykových modelů převzatých ze zpracování přirozeného jazyka na řeči proteinů zapsané v abecedě aminokyselin. V této práci demonstrujeme, jak lze velké jazykové modely založené na předtrénovaných hlubokých neuronových sítích efektivně vyladit pro kontrolovatelné generování proteinových sekvencí z několika odlišných proteinových rodin. Pomocí bioinformatických metod a metod založených na hlubokém učení ukazujeme, že model je schopen generovat vysoce kvalitní proteinové sekvence, které vykazují nízkou podobnost s existujícími proteiny.

Klíčová slova: bioinformatika, velké jazykové modely, proteinové inženýrství

Contents

Introduction

Proteins are a key component of all living organisms. They serve as the fundamental building blocks for cells, tissues, and organs, playing a crucial role in various biological processes, such as transport of small molecules throughout the body, enzyme catalysis, and immune responses in the form of antibodies. The importance of proteins extends also to DNA, where they function as integral elements of DNA replication and transcription. Another example is structural support of cells, organs, muscle and bone tissue which ultimately allows animals and humans to perform complex movement (Alberts et al., 2015).

The fundamental components of proteins are amino acids, which are linked together to form linear chains that form a protein. These so-called protein sequences can be therefore viewed as a language, where the alphabet consists of 20 proteogenic amino acids instead of letters. Protein sequences ("sentences" in the protein language) are information complete, so they fully determine the protein's structure and therefore also its function. Just as understanding the grammar and vocabulary of a natural language allows us to understand and generate meaningful sentences, learning the "grammar" of proteins can help us predict their function, interactions and localization within the cell. This means that just by knowing the sequence alone, we can predict the protein's function without explicitly having the information about the three-dimensional structure it folds into, especially because predicting protein structure is a hard problem in biochemistry.

In the recent years, language models from natural language processing (NLP), have been shown to exhibit useful capabilities emerging from scaling up the model size, amount of training data and compute. This paradigm has also been adapted to biological sequences (Elnaggar et al., 2021). Protein language models can be used for a wide range of tasks in bioinformatics, like function prediction, binding site prediction from the primary sequence, structure prediction and sequence generation, similarly to the well-known GPT model family (Brown et al., 2020). Using self-supervised learning objective and the vast amounts of available protein sequence data, these models are able to effectively learn the contextualised representation which can be used on numerous tasks in bioinformatics, including *de novo* protein design, which means designing novel proteins completely from scratch (Madani; McCann, et al., 2020).

There are many publicly available pre-trained protein language models based on the transformer decoder architecture trained with the next-token prediction objective which can be used for generation of high-quality sequences. The problem with such models is that they have been trained on a very diverse set of protein sequences which can perform various functions, but unlike instruction models in NLP, they are missing an interface for specifying what kind of proteins we want them to generate. It is only possible to specify a prefix and let the model continue generating rest of the sequence, but we can't specify what function we want the generated protein to perform.

The goal of this work is to fine-tune an open-source pre-trained protein language model ProGen2 (Nijkamp et al., 2022) to make the process of generation more controllable. This model is based on the transformer architecture (Vaswani et al., 2017), which has achieved great popularity in the recent years. We focus on

particular protein families taken from the Pfam (Mistry et al., 2020) database. We achieve this goal by expanding the vocabulary of the language model by several new tokens which are used as conditioning tokens that specify which protein family each sequence belongs to. These special tokens are placed in the beginning of the protein sequence and we continue training the model with this new data. We explore which values of hyperparameters, such as batch size, number of updates, learning rate, etc. achieve the best results for model fine-tuning as measured by the model's loss function on an independent test set. During inference, we choose which family we want to generate from by only specifying a single conditioning token and use the model to generate the rest of the protein sequence in an autoregressive way. We systematically evaluate top-k sampling with temperature to find out how they influence the generated proteins and how concepts from NLP transfer to the realm of proteins. We evaluate the quality of generated proteins by profile hidden Markov models to see if the protein sequence is likely to belong to that family and also look for false positives (e.g. sequences belonging to a different family than specified during generation). We also compute the sequence identity to sequences in the training dataset, to see that the model is not just replicating sequences it has seen during training, but is actually producing novel sequences. Depending on the family, we may also look for motifs, which are conserved sequence patterns in the protein, similar to regular expressions in computer science. To further validate the quality of our model, we use other deep learning models to predict the three-dimensional structure and likelihood of the generated sequences. Additionally, we look into the intermediate learned representations within the model to better understand how the transformer's self-attention mechanism processes the input in the fine-tuned model compared to the base model and how it processes the newly introduced conditioning token.

1 Bio(informatical) background

Firstly, we will introduce the main biological concepts and bioinformatic tools necessary for understanding and executing our goals.

1.1 Proteins

Proteins are macromolecules crucial for all life on Earth. We will briefly describe how they are formed, how they can be represented, how they evolve over time, and motivate our goal by introducing the concept of *de novo* protein design.

1.1.1 Amino acids

Amino acids are the fundamental building blocks of proteins. There are over 500 amino acids in nature, however only 20 of them are the so-called standard amino acids which are encoded in the standard genetic code. Amino acids are characterized by having a central carbon atom (C_α) to which an amino group (–NH2), a carboxyl group (–COOH), a hydrogen atom, and a variable side chain (R group) are attached. The properties of an amino acid are determined by its unique side chain, which can be, for example, polar or nonpolar, acidic or basic, hydrophobic or hydrophilic, etc. These properties affect the amino acid's role in proteins, and the protein's function as a whole. Each amino acid is assigned a unique three-letter and one-letter code. Structures and letter codes of amino acids are in Fig. 1.1

Peptide bonds form when the carboxyl group of one amino acid reacts with the amino group of another, releasing a molecule of water as a byproduct. This process creates chains of amino acids linked by peptide bonds. Depending on their length, these chains are called peptides if their length is less than 50 residues, or polypeptides if they are longer, however this threshold is not strict. A protein consists of one or more peptides or polypeptides. After the peptide bond formation, amino acids are often referred to as residues that are connected to the protein backbone (Alberts et al., 2015).

1.1.2 Levels of protein structure

Structure of proteins can be described at the following four layers (Brändén; Tooze, 1999):

Primary structure (usually referred to as protein sequence) is the linear sequence of amino acids that form a protein. The length of a protein sequence can vary widely, ranging from just a few tens to several thousands of amino acid residues. The direction of the sequence matters, so the sequence of amino acid residues `MGEAK` is not the same as `KAEGM`. By convention, the sequence is written from the N-terminus to the C-teminus, where N-terminus is the end of the polypeptide chain with the free amino group (–NH2) and C-terminus is the free carboxyl group (–COOH). This choice is due to the way proteins are synthesized in cells during the process of translation, although this choice is somewhat arbitrary and not biologically significant.

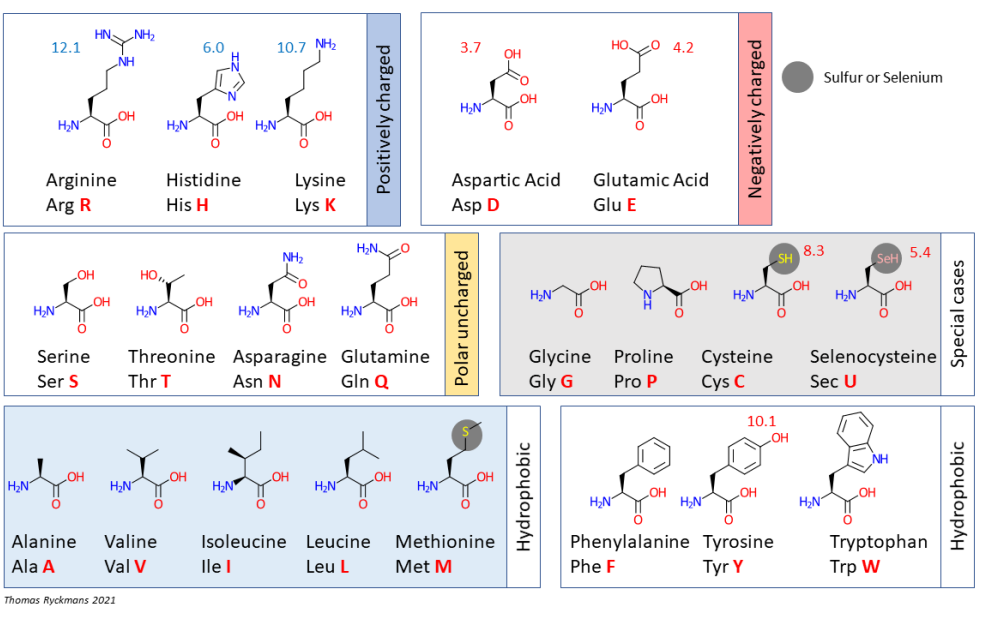


Figure 1.1 Table of 21 proteogenic amino acids (20 standard amino acids, plus selenocysteine, which is encoded in special cases by a stop codon) Ryckmans, 2022

Secondary structure refers to local conformations of the protein backbone. There exist two types of secondary structures:

– *Alpha helix*: backbone is twisted into a right-hand coil shape formed by hydrogen bonds between amide and carbonyl groups four residues apart in the sequence.

– *Beta sheet*: they consist of beta strands laid out next to each other and connected by hydrogen bonds between residues of adjacent strands. They can be parallel or anti-parallel, depending on the orientation of the beta strands.

While the secondary structure describes three-dimensional features of a protein, it can be described using a one dimensional representation, because we can assign each residue as being part of an alpha helix, beta sheet or neither.

Tertiary structure is a three dimensional representation of the whole protein formed by the folding of its polypeptide chain. This structure is very useful for understanding the protein's biological function.

Quaternary structure of a protein describes the arrangement of multiple polypeptide chains into a larger functional complex. Some proteins, such as hemoglobin, which carries oxygen in the blood, occur in nature as an assembly of several polypeptide chains.

Visualizations of all levels of protein structure can be found in Fig. 1.2 and are adapted from Berman et al., 2000 and Bateman, 2013. Note, that when we talk about protein structure, we usually refer to the tertiary structure (3D representation) and when we mention protein sequence we are talking about the primary structure (1D string).

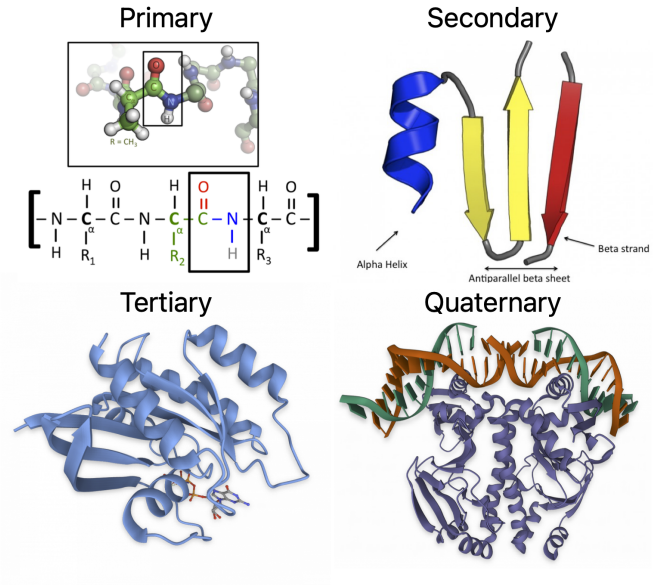


Figure 1.2 Example visualizations of primary, secondary, tertiary and quaternary protein structure

1.1.3 Relationship between sequence, structure and function

Genetic information about the protein sequence is stored in the DNA which is a long chain of nucleobases, where consecutive triplets of nucleobases, so-called *codons*, each encode a specific amino acid. During the process of protein synthesis, this information is used to produce the polypeptide chain of a protein, which then assumes a three-dimensional shape. This process is known as protein folding. Knowing the structure of a protein means knowing the coordinates of every atom in it.

The function of a protein in the organism is closely linked to its structure, which determines how it interacts with other molecules (Alberts et al., 2015). Binding sites, which are parts of a protein where a ligand can bind, are usually highly specific, allowing only particular ligands to bind to it. The ligand can be for example an ion, small molecule, other protein, RNA or even bacteria or viruses in the case of antibodies, which are proteins that play a key part in immune responses.

Given a protein sequence, it is difficult to accurately determine the structure it will fold into. One such experimental technique is X-ray crystallography. It involves condensing protein into a crystal and shooting X-rays at it to determine the shape. However, structure determination is expensive, both in terms of time and resources, and these techniques are not able to capture the structure of any protein, for example, in case of X-ray crystallography, many proteins cannot be condensed into a crystal. Moreover, there are large classes of proteins, called intrinsically disordered proteins (IDP) or intrinsically disordered regions (IDR) which do not conform to a rigid three dimensional structure, but their shape

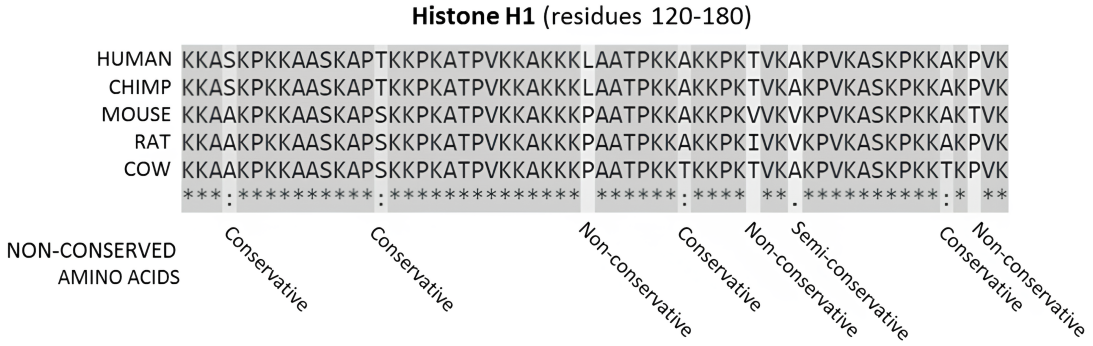


Figure 1.3 Multiple sequence alignment of histone H1 in a few species (Paysan-Lafosse et al., 2022)

is flexible and dynamic, typically in the absence of macromolecule interaction partners, such as other proteins or RNA. Also, the conformation of a protein can change depending on whether it is binding a ligand or not (Trivedi; Nagarajaram, 2022). All of this means that databases of protein structures are limited to a small and biased subset of proteins.

1.1.4 Evolution

The process of evolution is a fundamental aspect of life. Through natural selection and random mutations, it provides organisms the ability to adapt to their environment and yields the diversity of species that we observe in nature today. Just as species evolve over time, so does their DNA and hence proteins it encodes. Proteins that perform an equivalent function in different organisms that share a common ancestor usually have similar sequences. Two amino acid sequences are said to be *homologous* if they share ancestry in their evolutionary history. They differ due to random mutations in their coding DNA sequence. Such mutations may have negative effects, for example, the relatively rare genetic disease *osteogenesis imperfecta* (commonly known as brittle bone syndrome) is caused by a single amino acid substitution (Gly $\rightarrow$ Ala) in collagen protein responsible for the structural integrity of bones (Xiao et al., 2011). However random mutations can also be positive and improve the protein's fitness (the ability to perform its function). Another type of mutation is an indel (insertion or deletion), where a nucleotide or an amino acid residue is inserted or removed from a sequence. Through natural selection, specimen with higher fitness get to spread their genes to their offspring. This process has been going on for billions of years, and as whole species evolve, so do the proteins that carry out functions in them.

Protein families

A protein family is a group of evolutionarily related proteins (Bateman, 2013). This is reflected in similarity of their amino acid sequence, and therefore often also structure and function. These proteins are derived from a common ancestral sequence and have diverged over time through various evolutionary processes such as mutations, gene duplication, and functional specialization. Members of a protein family can be found within the same organism or across different species. Depending on how long ago lived their last common ancestor, protein families

may be more or less conserved. Certain parts of sequences may be well conserved across an entire family, suggesting functional importance. Such regions are called protein sequence motifs and evolve more slowly compared to rest of the sequence, due to their significance for the protein function. Fig. 1.3 shows related histone proteins in different organisms.

1.1.5 Protein engineering, de novo design

While evolution is a reliable tool for creating suitable proteins, it takes millions of years to tailor proteins for specific functions through natural selection. In contrast, protein engineering is a field that accelerates this process by designing proteins with desired properties in a much shorter time. The goal of protein engineering is to design completely new proteins or modify existing proteins to obtain specific, desirable traits that are not found in natural proteins. For example, enhancing enzyme catalytic efficiency, stability under extreme conditions, antibody design, and more ambitiously, creating proteins with entirely new functions.

The term *de novo* protein design describes the process of creating novel proteins from scratch, and aims to discover viable proteins, that have not yet evolved in nature. By exploring parts of the protein sequence and structure space that natural evolution has not reached, protein design has potential to create proteins with unique functions that can be applied to key challenges in biomedicine and biological engineering (Pan; Kortemme, 2021). The combinatorial space of possible protein sequences is large, but only a few protein sequences are able to fold properly and carry out a specific function in the cell. Moreover, it is often difficult to find a suitable *in silico* (computational) metric for determining if a given protein sequence is able to perform a particular function and experiments in a laboratory can be time consuming and expensive. This makes *de novo* protein design a difficult problem.

1.2 Tools

In this section, we describe the bioinformatic tools we will be using in this work, including databases, algorithms, software tools, and lastly discuss available machine learning models related to our work.

1.2.1 Databases

- **Protein Data Bank** (PDB) (Berman et al., 2000) is a public database of three-dimensional structural data of biological macromolecules, mainly proteins. The entries are experimentally determined. It is a key resource for research in structural bioinformatics.

- **UniProt** (Universal Protein Resource) (The UniProt Consortium, 2022) is a database of protein sequence and functional information. It serves as a central hub for the collection of extensive data on the biological functions of proteins, derived from literature curation as well as direct submissions from researchers.

- **Pfam** (Mistry et al., 2020) is a comprehensive collection of protein families, each entry contains, amon others, the sequences belonging to the family, multiple sequence alignment (MSA) and profile hidden Markov models (HMMs) created from these MSAs. The primary aim of Pfam is to classify protein sequences into families based evolutionary relations. Pfam enables researchers to identify homologous sequences and predict the function of uncharacterized proteins based on their membership in a particular protein family. Recently, it has been integrated into the InterPro (Paysan-Lafosse et al., 2022) database as one of its several member databases.

1.2.2 Sequence alignment

Sequence alignment is a way of arranging two or more protein sequences to identify regions of similarity that may imply functional, structural or evolutionary relationship between them. We may insert gaps between residues, so that identical or similar residues can be aligned in successive columns. For two aligned sequences that share a common ancestor, mismatches can be considered point mutations and gaps are indels (insertion or deletion mutations).

Alignment of sequences can be computed using dynamic programming. We distinguish between local alignment, which aligns only select parts of sequences, and global alignment which aligns all residues in the sequences. In practice, protein alignments use a substitution matrix, to assign a score to each possible amino acid substitution, and a gap open and gap extension penalty because fewer longer gaps are more common in nature than many short gaps. This technique can be used to produce pairwise alignments (two sequences) and can be extended to multiple sequence alignment (MSA) consisting of three or more sequences. While guaranteed to find to optimal alignment, computing MSA using dynamic programming is computationally expensive. With the naive approach it takes $O(L^n)$ time, where L is the length of alignment and n is the number of sequences. In practice, various heuristic methods are used to find the MSA (Thompson et al., 2011). Figure 1.3 contains visualization of a multiple sequence alignment. High conservation of a certain region in the MSA suggests that it carries a functional or structural importance.

Sequence identity between two sequences is calculated as the percentage of identical residues in the alignment and is used as a measure of sequence similarity (Pearson, 2013). We will use this metric in some experiments described later.

MMseqs2

MMseqs2 (Many-against-Many sequence searching) is an open-source software suite developed by Steinegger; Söding, 2017 to search and cluster huge protein and nucleotide sequence sets. The software is designed to run on multiple cores and servers and exhibits very good scalability. The clustering of protein sequences works by first performing pairwise sequence alignments and calculating identities between all pairs of sequences in the dataset. Sequences above a set similarity threshold are then grouped together into a single cluster, which is then represented by a single representative sequence. The other common functionality is many-against-many searching. It enables comparing a FASTA file with query sequences, against another FASTA file or an MMseqs2 target database. The output is a

tab-delimited file with columns indicating identifiers of query and target sequences, percentage of sequence identity and length of the alignment, among others. FASTA is a text file format commonly used for storing protein sequences.

1.2.3 Hidden Markov models

Definition 1 (Hidden Markov model)**.** *Let $(X_n)_{n=1}^N, (Y_n)_{n=1}^N$ be sequences of random variables, where $N \geq 1$ and $\forall i : X_i \in H \wedge Y_i \in O$, where H is a finite set of hidden states and O is a finite set of observable states. The pair $((X_n)_{n=1}^N, (Y_n)_{n=1}^N)$ is a (discrete time, discrete space) hidden Markov model if:*

1. X_n is a Markov chain, meaning that $\forall i : P(X_{i+1} = h_{i+1} | X_1 = h_1, ..., X_i = h_i) = P(X_{i+1} = h_{i+1} | X_i = h_i), h_i \in H$. These are called the transition probabilities, and X_n is the sequence of hidden states, which we do not directly observe.

2. $P(Y_i = a | X_1 = h_1, ..., X_i = h_i) = P(Y_i = a | X_i = h_i)$ for every $i \geq 1$ and $h_1, ..., h_i \in H, a \in O$. These are called the emission or output probabilities, defined for each hidden state. We directly observe only the sequence Y_n.

We can think of the hidden Markov model (HMM) as a weighted oriented graph, where each node corresponds to a hidden state and each edge corresponds to a state transition. Each hidden state stores an emission probability distribution over the set of observable states. The HMM can be represented using a transition matrix A, where $A_{ij} = P(X_t = j | X_{t-1} = i)$ for some $t \geq 1$, $i, j \in H$ and emission probabilities matrix $B \in \mathbb{R}^{|H| \times |O|}$, where B_{ij} is the probability of emitting output j at state i. Optionally, we can include an initial probability distribution π over the set of hidden states, but we can work around this by defining a non-emitting starting state as in Fig 1.4.

Profile hidden Markov models

Profile hidden Markov models (Eddy, 1998) are a special type of hidden Markov models (HMM) designed to model multiple sequence alignments. They turn an MSA into a position-specific scoring system, which is suitable for fast and sensitive searches of remotely homologous sequences. For a given protein family, the profile HMM is constructed from an MSA of a few sequences (seed alignment). See Fig. 1.4.

Each edge in the profile HMM transition graph has an assigned transition probability based on the MSA. The hidden states (apart from the start and end states) of a profile HMM can be divided into three types:

- **Match states (M)** are the central element of a profile HMM. They form a linear sequence of states, each emitting a single residue according to the probability distribution of residues in that particular column of the MSA.
- **Insertion states (I)** correspond to insertion of amino acid residues into the sequence and multiple insertions can occur consecutively. Each insertion state stores an emission probability distribution over the 20 residues.
- **Deletion states (D)** correspond to gaps in the alignment and are non-emitting states.

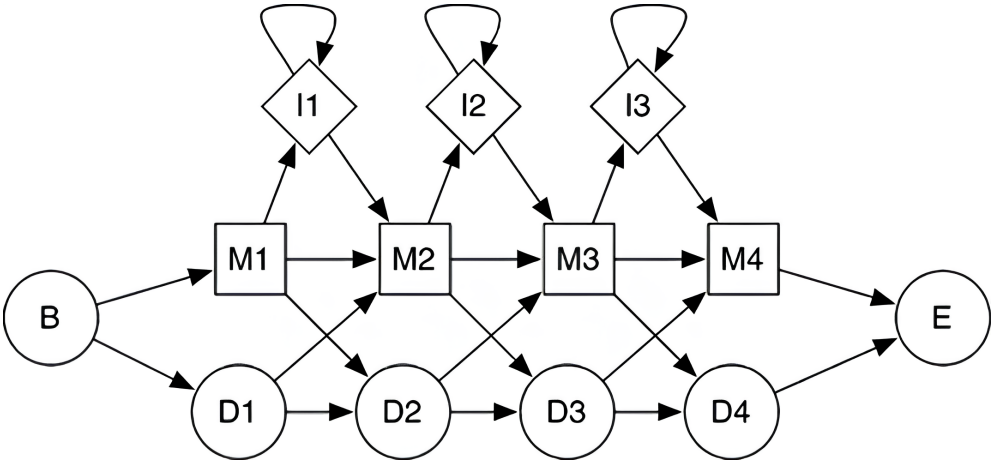


Figure 1.4 Profile hidden Markov model from an alignment of length 4. Transition and emission probabilities are not depicted.

For each family in Pfam database (Mistry et al., 2020), there is a profile HMM built from its MSA. This model can be used to detect if a novel sequence is homologous to sequences in that family.

We use the forward algorithm (Jurafsky; Martin, 2024) to compute the probability of the observed sequence O given the HMM H_A as $P(O|H_A)$. Thanks to the Markov property, we can use dynamic programming to compute this value efficiently. The score of the sequence is computed as log odds against the null hypothesis of non-homology $log\frac{P(O|H_A)}{P(O|H_0)}$. The null hypothesis model H_0 is defined as a HMM with one state, with emission probabilities defined according to frequency of residues in naturally occurring proteins (Johnson et al., 2010).

This gives us a score for a given sequence of observations, which we can use to filter out sequences passing some threshold. After that, we use the Viterbi algorithm (Forney, 1973), which also uses dynamic programming, to find the most likely sequence of hidden states (path through the HMM transition graph), given the observed sequence. This algorithm is also widely used in other fields, where we assume some data can be modeled using a HMM, such as speech recognition, computational linguistics, or in telecommunications for decoding codes from noisy channels. In the case of profile HMMs, the most likely sequence of hidden states corresponds to a sequence alignment.

HMMER

A popular open source tool for protein sequence analysis using profile hidden Markov models is HMMER (pronounced "hammer") (Johnson et al., 2010). It is a software package used for processing nucleotide, or in our case, protein sequences. As the name suggests, it uses profile hidden Markov models to detect homologous proteins. This software package provides several command line programs for building, searching and alignment. For our purposes, we will introduce only the `hmmscan` command.

The `hmmscan` command is used to search protein sequences against a profile HMM database (specifically, in our case, this database will consist of a single profile). Each query sequence is scanned against the profile HMM, which is constructed from an MSA of sequences in a particular protein family, and if it

passes the filter (log odd score above a certain threshold), HMMER produces an alignment with the profile. We consider this as a success, meaning that we generated a sequence belonging to the particular family, for which the profile HMM was built. For each query, HMMER returns the score described above and expectation value (E-value), which denotes the statistical significance of the sequence score. E-value is the number of hits expected to have a score equal to, or better, than the query sequence by chance. We use the default E-value `1e-2` and only report hits below this threshold. This means that on average we can expect one false positive hit in 100 queries.

1.2.4 Machine learning in bioinformatics

Large amounts of collected biological data, such as genomic sequences, protein sequences and structures, but also clinical data, among others, allow for using machine learning tools to analyze and extract important information from this data. Here, we will focus on ML applications to study proteins and their properties and give a brief overview of common tasks relevant for our work and what type of models can be used to solve them.

Ligand binding site prediction

Predicting parts of protein where a ligand, which can be a small molecule, atom, or another protein among others, is an important question, because proteins execute their functions by binding to a ligand. Binding sites can be viewed as pockets on the surface of the protein and are usually highly specific, only allowing molecules with particular shapes and physicochemical properties to bind to the protein. Identifying binding sites is essential for drug development as they often represent targets for therapeutics. The methods can be broadly divided into two types:

- **Structure-based methods** take into account the tertiary structure of a protein. There have been numerous machine learning algorithms developed for this task, such as decision trees (Krivák; Hoksza, 2018) and neural networks (Zhao et al., 2020).
- **Sequence-based methods** only rely on the protein sequence to predict the binding site. This problem is harder, because as we know, binding is dependent on the three dimensional structure of the pocket. The advantage here is that there is much more available data than when working with structures. Protein language models (pLMs) (Elnaggar et al., 2021) trained only on unannotated sequence data are able to learn high-quality contextualized representations of amino acids (embeddings) which can be used as features for other models to classify which residues are part of the binding site (Littmann et al., 2021). Note, that even though the residues are close in the 3D space, they may not be close in the sequence. Successful application of pLMs to these tasks imply that they are able to capture structural information. There have also been developed hybrid methods which use both structural and sequence paradigms (Gamouh et al., 2023).

Tertiary structure prediction

Since protein function is determined by its structure, but sequence data is far easier to obtain compared to structures, an obvious problem of determining structure computationally emerges. For short sequences, molecular dynamics simulations can be used, but they are very expensive in terms of time and computational resources.

Deep learning methods have achieved state-of-the-art results in this task. The development of AlphaFold (Jumper et al., 2021), which achieved much more accurate structure prediction than previous methods, has had a great impact on the field of bioinformatics. It first produces a multiple sequence alignment with protein sequences from a large database to find similar sequences, and then the MSA is used as input for a deep neural network. Other notable models for structure prediction are OmegaFold (Wu et al., 2022) and ESMFold (Lin et al., 2023), which do not rely on MSA, but use representations from a language model as features for the folding model. This makes them much faster then searching a database and aligning sequences. All of these models output the predicted structure, along with a per-residue confidence of the prediction called pLDDT (predicted local distance difference test). It represents a confidence score, which is widely used as an *in silico* metric for evaluating the quality of artificially generated proteins, or as a filter for deciding which sequences are more likely to work well before synthesizing them in a living cell (Watson et al., 2023).

De novo protein design

Designing new proteins with desired functions is a challenging problem which can also be divided into two paradigms: sequence and structure.

An example of structure design model is RFdiffusion (Watson et al., 2023), which generates the backbone of the protein structure in 3D space. It is trained to iteratively denoise a signal to reconstruct the protein's shape, using the diffusion process for efficient exploration of protein space. The downside of structure based methods is that in order to actually synthesize the protein in a lab, it has to be encoded as a DNA sequence which is then created in a living cell. This means that there has to be an inverse folding component, ProteinMPNN (Dauparas et al., 2022) in case of RFdiffusion, which predicts the sequence from the generated structural backbone, since the backbone is only a shape in 3 dimensions, but does not tell us the actual amino acid residues attached to it.

Because protein sequences can be considered a language, language models can also be used for protein sequence generation. In natural language processing, models like GPT-4 (OpenAI, 2024), LLaMA (Touvron et al., 2023), and countless others have proved that by increasing model size, they are able to perform well on a wide range of complex tasks which they have not been specifically trained for, and with instruction-tuning they can serve as conversational assistants. Same principle can be applied to proteins: models like ESM (Rives et al., 2021), ProtBert, ProtT5 (Elnaggar et al., 2021) provide embeddings that can be used to solve various tasks, as previously mentioned. Generative language models are also available, for example EvoDiff (Alamdari et al., 2023) is a diffusion model for sequence denoising, ProtGPT2 (Ferruz et al., 2022) based on the GPT2 (Brown et al., 2020) architecture, ProGen (Madani; McCann, et al., 2020) which uses conditioning

tokens for specifying desired properties of the protein and ProGen2 (Nijkamp et al., 2022) which is a more general causal language model available in different size checkpoints and which will be the main subject of our work.

2 Machine learning background

In this chapter, we provide a brief overview of language modeling theory, a background of the machine learning tools we are using, including the model architecture, training the model and model inference. Lastly, we discuss some related works, describe the motivation for using these methods on proteins and introduce model we will be working with.

2.1 Language modeling

At its core, the objective of a language model is to estimate the probability distribution over the next tokens in a sentence, given a sequence of previous tokens $P(w_t|w_1, ..., w_{t-1})$ (Jurafsky; Martin, 2024). Tokens, also called sub-words, are fundamental units of input for the language model. In most cases a tokens corresponds to a word, but for rare or longer words one word can be broken down into multiple tokens. This depends on how the tokenizer for a particular language model was constructed. In case of protein language models, every token represents one amino acid residue, or some utility tokens, which we will describe in following chapters. For a sequence of tokens $W = (w_1, ..., w_n)$, we can use the language model to estimate the probability of a sentence $P(W)$ by using the chain rule of probability

$$P(W) = P(w_1, ..., w_n) = P(w_1)P(w_2|w_1)...P(w_n|w_1, ..., w_{n-1}) = \prod_{t=1}^{n} P(w_t|w_{<t})$$

n-gram language models

Historically, there have been various statistical approaches to language modeling. One such straightforward approach are the so-called n-gram language models, which use relative frequencies of n-grams to compute the next-word probability. An n-gram is a contiguous sequence of n words from the training corpus. The downside of n-gram language models is that they can't generalize beyond sequences unseen during training.

Neural language modeling

Neural language models offer a significant improvement over n-gram models in terms of generalization. They rely on neural networks to learn distributed representations of tokens, such that semantic information can be transferred between similar sentences, while simultaneously learning the probability distribution of the next word conditioned on previous tokens (Bengio et al., 2003). In practice, they are also limited to a fixed context size of n previous tokens, but this n can be much larger than in case of n-gram models. These distributed token representations are referred to as *embeddings*. Neural language modes represent words in this prior context by their embeddings, rather than just by their word identity.

The first layer of the neural network is the embedding layer, which is a matrix with $|V|$ rows and E columns, where V is the model's vocabulary and E is the model's hidden dimension. Each token in the model's vocabulary is assigned a

unique index, which corresponds to a row in the embedding matrix, that is the token's contextualized vector representation. In a well-trained model, we expect the embeddings of similar tokens to be close in the embedding space. The output of the embedding layer is a sequence of embeddings, which is used as the input to the neural language model.

The language model outputs logits $\hat{y} = f(x_{t-n+1}, ..., x_{t-1})$, which is used to calculate the next token probabilities as

$$P(w_t|w_{t-n+1}, ..., w_{t-1}) = \frac{exp(\hat{y}_{w_t})}{\sum_i exp(\hat{y}_i)}$$

where for every i, x_i is the embedding of the i-th token in the sequence. The definition of f depends on the network architecture. For example multilayer perceptrons (MLP) or recurrent neural networks (RNN) have been used, but the transformer architecture (Vaswani et al., 2017), which we will describe in section 2.2, has been the most used one in the past years.

2.1.1 Self-supervised training

Language models are trained using self-supervised learning. Unlike in supervised learning, there are no explicit annotations, however since our task is language modeling, also referred to as next token prediction, the labels are actually provided thanks to the structure of the data. This allows us to use large unannotated corpora for training language models.

Training of the language model is an optimization problem of minimizing the cross-entropy (negative log likelihood) loss of the data using the gradient descent algorithm. Cross-entropy measures the difference between two probability distributions over the same set of underlying events and is calculated as $H(P^*, P) = -\sum_x P^*(x) \log P(x)$, where P^* is the target distribution and P is the distribution computed by the model. In case of language modeling, there is only one correct label, so the loss can be calculated as $-\log P(w_t|w_{t-n+1}, ..., w_{t-1})$, which is the negative logarithm of probability that our model assigned to the correct class (Jurafsky; Martin, 2024). One update of model parameters is then performed as

$$\theta \leftarrow \theta + \alpha \frac{\partial \log P(w_t|w_{t-n+1}, ..., w_{t-1})}{\partial \theta}$$

where θ are the model parameters and $\alpha > 0$ is the learning rate. Using the chain rule of derivatives, the derivative of loss is calculated with respect to all model parameters in a process is called *backpropagation*.

2.2 Transformer architecture

In this section, we will describe the transformer architecture, originally introduced by Vaswani et al., 2017 for machine translation, but is now used for many different applications. The original transformer architecture, as shown in 2.1 consists of and an encoder and a decoder, where the encoder is used to process the sentence in the source language and the decoder generates the translation in target language. Our task is language modeling, i.e. predicting the next token, so we use a decoder-only architecture.

Attention

The main part of the transformer architecture is the scaled dot-product attention mechanism. The inputs of attention are queries, keys and values $\mathbf{Q} \in \mathbb{R}^{n \times d_k}, \mathbf{K} \in \mathbb{R}^{n \times d_k}, \mathbf{V} \in \mathbb{R}^{n \times d_v}$ and the output is computed as

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

where the queries, keys and values. Sequence length is denoted as n, d_k is the dimension of keys and queries, d_v is the dimension of values and d is the model's hidden dimension. Since we want the model to predict the next token, the attention has to be causal, meaning that each token can only use the information conveyed by the previous tokens. This is implemented by adding a matrix of shape $(n \times n)$ with zeros on the diagonal and below it, and $-\infty$ above the diagonal to the matrix product $\mathbf{Q}\mathbf{K}^T$ before applying the softmax activation. The softmax function is defined as $\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^{|V|}\exp(x_j)}$ for every index i in vocabulary of size $|V|$.

Multi-head attention

In practice, multi-head attention is used. We split queries, keys and values into several groups and compute the attentions of each group separately in parallel, concatenate their outputs and pass them through a linear projection W_O to dimension d. See figure 2.1 for visualization.

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W_o$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

where $W_i^Q \in \mathbb{R}^{d \times d_k}, W_i^K \in \mathbb{R}^{d \times d_k}, W_i^V \in \mathbb{R}^{d \times d_v}$ and $W_O \in \mathbb{R}^{hd_v \times d}$ are learnable parameters.

Feed-forward network

Following is the feed-forward network, which consists of two fully-connected layers with ReLU activation between them.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

The inner dimension of the feed-forward network is empirically chosen to be 4 times larger than the model's hidden dimension (Nijkamp et al., 2022).

Since the attention mechanism does not explicitly know relative positions of tokens in the text, positional embeddings are used to pass information about the positions to the transformer. There are residual connections bypassing each sublayer which are then added to the sublayer's output and normalized. There are usually several transformer layers in a model, followed by a final linear projection to $|V|$ classes, which are used to calculate the probabilities using softmax.

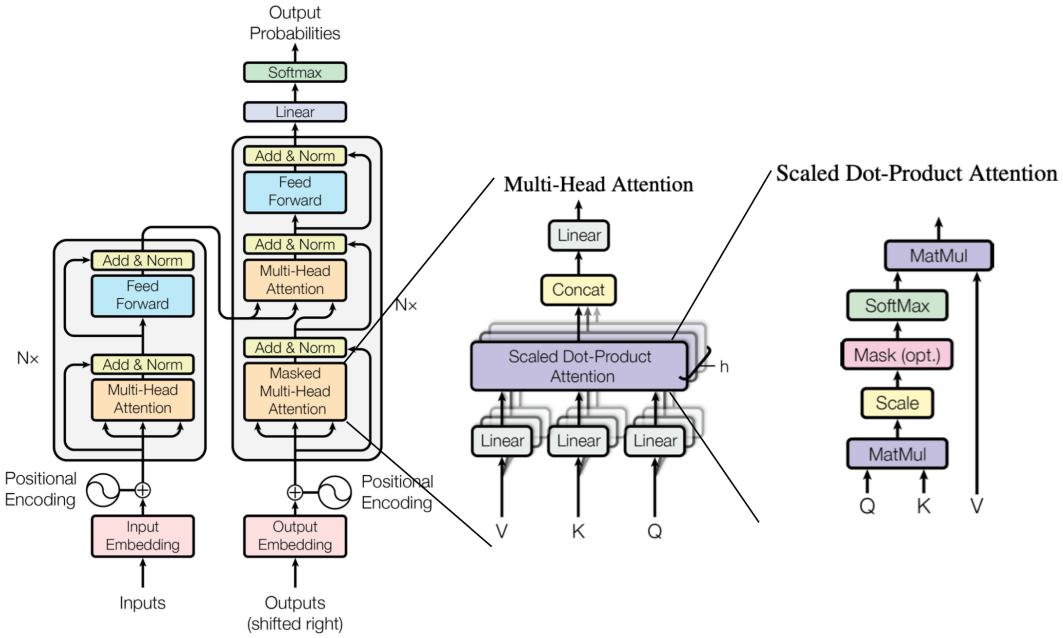


Figure 2.1 Transformer architecture, multi-head attention, scaled dot-product attention diagram. Adapted from Vaswani et al., 2017

2.2.1 Sampling

The process of generating text using a language model is referred to as inference, generation or decoding. The model outputs a probability distribution over tokens in the vocabulary at every step of the autoregressive generation. This means that the token selected at time t is appended to the context given at time t and used as context at time $t + 1$. We need to decide how we use the output probability distribution to select the next token.

The most straightforward way is to always choose the token with the highest assigned probability. This is called the **greedy decoding** and works well in practice, but for a given prompt only generates deterministic outputs and choosing the most probable token does not guarantee to generate the most probable sequence.

An improvement over this is a strategy called **beam-search**, which is a modification of BFS that only expands a fixed number of candidate nodes at each layer. It works better than the previous method, however can be quite computationally expensive and in practice may produce repetitive or otherwise degenerated outputs (Ferruz et al., 2022).

Another alternative is **top-p** sampling, also known as nucleus sampling. It involves choosing a value for $p \in (0, 1]$ and then restricting the sampling to the set of most probable tokens with cumulative probability less than p. Each token in this set is chosen with a probability proportional to the probability it was assigned by the model. The motivation is that we want to avoid sampling from the tail of the distribution, which may be very long, especially in case of models with a large vocabulary.

Top-k sampling has a similar motivation to top-p sampling, but restricts the sampling to a fixed number of $k \in \{0, ..., |V| - 1\}$ tokens with the highest

probability. It is often combined with so-called **temperature** $t > 0$, which is a constant that the model outputs (logits) $\hat{y}$ are divided by before applying the softmax activation function.

The temperature parameter keeps the relative order of probabilities the same, but influences the shape of the probability distribution. Sampling with a high temperature ($t > 1$) "flattens" the probability distributions, and causes the model to pick less likely tokens more often and therefore its responses are more diverse and creative. However when the temperature is too high, the outputs become too nonsensical and gibberish. A low sampling temperature ($t < 1$) makes the distribution "sharper" and makes picking the high-probability tokens even more likely, which causes the outputs to be more confident and conservative, but similar to the sequences in training data. Sampling with a very low temperature approximates greedy decoding.

2.3 Related models

CTRL (Conditional Transformer Language model) is a language model developed by Keskar et al., 2019. Is is a natural language model trained on a large corpus of text data collected from the internet, with special *control tokens* that are placed in the beginning of each document in the training data and denote its specific properties, such as style, content and task-specific behavior. These control tokens were usually determined by the website the data was downloaded from. Some examples of control tokens include `Wikipedia` for Wikipedia articles, `Review` for Amazon reviews, etc. During generation, users can prompt the model by only providing a control token and use the model to generate text that conforms to the style and content that particular control token is associated with.

Following this model, **ProGen** was developed by Madani; McCann, et al., 2020, which is trained on protein sequences prepended by special control tokens (which were called *conditioning tags* or *conditioning tokens* in this paper). These conditioning tokens specify various properties of the whole sequences, such as molecular function, cellular component and biological process. Overall, they used over 1100 different conditioning tokens and each training sequence contained several conditioning tokens, up to 14. By finetuning this model on lysosome families Madani; Krause, et al., 2023 have shown that the model is able to generate high-quality protein sequences and experimentally validated that these artificial sequences show similar catalytic efficiency to natural ones, with a very low sequence identity to the natural sequences. Unfortunately, this model is not open source.

Subsequently, its successor **ProGen2** (Nijkamp et al., 2022) was released. It is also a protein language model based on the transformer decoder-only architecture, and is available in various size checkpoints, showing that by increasing the number of trainable parameters, the model's language modeling capabilities improve. Contrary to its predecessor, its vocabulary does not contain special conditioning tokens, which makes it hard for users to generate protein sequences with desired properties.

3 Implementation

Our goal is to finetune the ProGen2 (Nijkamp et al., 2022) language model to be able to generate sequences from selected protein families. We achieve this by prepending each sequence in a given family by a unique family token and continue training the model on the next-token prediction task using these new sequences. We use the cross-entropy loss function, same as during pretraining of the base model. Then during inference, we want to generate valid protein sequences, which belong to the correct family, only by specifying the special family token and a terminus token as the prompt.

In this chapter, we will describe the code required for downloading and preprocessing the data, finetuning the model, and generating protein sequences with it. The code is available on GitHub: `https://github.com/hugohrban/ProGen2-finetuning/tree/main`.

3.1 Data preparation

Each protein family in the Pfam database (Mistry et al., 2020), is characterized by a unique code, consisting of the letters `PF` followed by five digits.

We chose the following seven families to finetune the model on:

- **Family 2 of the G-protein coupled receptors (GPCRs)** (*PF00002*) are cell surface receptors that detect molecules outside the cell and activate cellular responses. They contain seven alpha helices which pass through the cell membrane.
- **Globins** (*PF00042*) are heme-containing proteins involved in binding and transporting of oxygen molecules. They belong to a very large and well studied family that is widely distributed in many organisms. Major types include hemoglobin, myoglobin and neuroglobin.
- **Core histones** (*PF00125*) act as spools around which DNA winds to create structural units called nucleosomes.
- **Copper binding proteins** (*PF00127*) are found in plants and bacteria and facilitate electron transfer between cells via a copper ion.
- **Dehydrins** (*PF00257*) contribute to freezing stress tolerance in plants.
- **Calreticulins** (*PF00262*) are a family of calcium-binding proteins.
- **P-loop ATPase protein family** (*PF03668*). Members of this family contain an ATP-binding site, mostly in form of the P-loop sequence motif. Adenosine triphosphate (ATP) is a source of energy for the cell.

Using MMseqs2 (Steinegger; Söding, 2017) command `easy-cluster`, we clustered sequences in each family at 90% sequence identity, in order to get rid of redundancy in training data. In the table 3.1, we show information about the individual protein families (after clustering):

Pfam code	Num. sequences	Avg. length	% of data
PF00002	19942	545	22.10
PF00042	17491	273	19.39
PF00125	16997	195	18.84
PF00127	18085	249	20.05
PF00257	203	228	0.23
PF00262	4433	433	4.91
PF03668	13065	287	14.48
Total	90216	324	100.00

Table 3.1 Summary of training families

Download

We download the individual families using the InterPro API (Paysan-Lafosse et al., 2022). The python script `download_pfam.py` is a modification of the automatically generated code from the InterPro website. Users can specify which Pfam entries they want to download by specifying the Pfam codes as command line arguments when running the script. Downloaded files are saved in FASTA format.

Preprocess

For each family, we introduce a special token, based on its Pfam code, e.g. `<|pf00002|>`. Same as the base model, we use tokens `1` and `2` to denote the N and C terminus respectively, and each amino acid residue corresponds to one token. Each sequence is stored on a separate line, for example: `<|pf00262|>1MGEAKRV...LKLKMKML2`. The `prepare_data.py` script takes as input a list of files in FASTA format, converts the sequences to the described format, splits the sequences according to the train-test split ratio and writes them in two separate train and test files. Users can also specify whether they want the data to also be stored in reverse, for training a bidirectional model, for example: `<|pf00262|>2LMKMKLKL...VRKAEGM1`.

3.2 Finetuning & finding right hyperparameters

Model architecture and configuration

We chose to finetune the smallest available ProGen2 (Nijkamp et al., 2022) checkpoint called ProGen2-small, containing 151 million parameters, because we can finetune it in a relatively short amount of time on a single GPU. This model consists of 12 layers, 16 attention heads, hidden dimension of 1024 and context window size of 1024 tokens. The original model's vocabulary contains 30 tokens, out of which 25 represent amino acids and 5 special tokens, namely `1, 2, <|pad|>, <|bos|>, <|eos|>`. Note, that apart from the 20 standard amino acids, there are also tokens for special amino acids, for example `X` denotes unknown amino acid, but these appear quite rarely in the data.

Since we are not training the model from scratch, but only finetuning it, we did not need to make any changes to the model's architecture. The model archi-

tecture and forward pass implementation are defined in file `modeling_progen.py` and the configuration is defined in `configuration_progen.py`, both in the `models/progen/` directory. Both files in this directory are adapted from the original ProGen2 implementation by Nijkamp et al., 2022. We slightly modified the configuration to allow us to use different vocabulary dimension for the initial embedding layer and the language modeling head, which is the model's final layer. The model architecture is broken down into several classes, defining the attention module, the MLP module, the transformer block, and finally the whole `ProGenForCausalLM` model class. The `ProGenForCausalLM` model class inherits from the `PreTrainedModel` class implemented in the Hugging Face transformers library (Wolf et al., 2019), which allows us to easily push our model weights and the modeling and configuration files to the Hugging Face Hub. This makes it easy for users to download and use the model for sequence generation or other downstream tasks. The original ProGen2 model is implemented in the PyTorch framework (Paszke et al., 2019), and therefore we are also using PyTorch for finetuning and generating sequences using the model.

New embeddings

We added new tokens to the tokenizer's vocabulary, therefore we need to train new embeddings for them. The embedding layer is a matrix with shape (V, D), where V is vocabulary size and D is the model's embedding dimension. Note, that although it is common in language models for the embedding matrix and the language modeling head layer to share their weights in order to to reduce the number of trainable parameters, ProGen2 does not do this, most likely due to its small vocabulary size. This means that we only need to add new rows to the embeddings matrix. We noticed, that although the model's tokenizer only contains 30 tokens, the embedding matrix has 32 rows so the last two rows have not been trained since their initialization. We initialized the embeddings for the new special tokens with random numbers from normal distribution with the same mean and variance as the raw untrained token embeddings.

Finetuning

During finetuning, we use the Adam optimizer (Kingma; Ba, 2017) to update the model weights. We use cosine learning rate decay with linearly increasing warmup steps, in order to not get stuck in local optima and improve the rate of convergence. For the data we described, we experimented with several combinations of values for the hyperparameters, most influential ones being the batch size, initial learning rate and number of training epochs. Our search for the right hyperparameters was not really systematic, due to limited time and compute resources. For reference, one training epoch on the described dataset took around 45 minutes on the faculty's cluster, running on one A100 80GB GPU, using a batch of 64 sequences. Fitting a model of such large size (151 M parameters), with Adam optimizer for its weights and training data onto a GPU requires a significant amount of memory. In our case, we were only able to fit 16 sequences in a batch for one backward pass. In order to use a larger effective batch size for the parameter update, we had to use a technique called gradient accumulation, which involves not calling `optimizer.step()` after every backward pass, but only after

a set number of batches, as shown in code listing 3.1. Using gradient accumulation allows us to train the model with an arbitrary batch size, although each training update takes longer real time.

Listing 3.1 Trinaing loop with gradient accumulation

```
for i, batch in enumerate(dataloader):
    batch = batch.to(model.device)
    loss = model(batch, labels=batch).loss
    loss = loss / args.accumulation_steps
    loss.backward()
    if (i + 1) % args.accumulation_steps == 0:
        optimizer.step()
        scheduler.step()
        optimizer.zero_grad()
```

We decided to train the model with a batch size of 64. Using a smaller batch caused the model to reach a local optimum and start overfitting after just a few epochs. On the other hand, using a larger batch resulted in much slower convergence rate.

For the initial value of learning rate, we observed the best results for value `1e-4`. A larger learning rate caused the model to have a lower loss after the first few hundred iterations, but then get stuck in a local optimum and not reach low enough loss, even with using learning rate warmup. A lower learning rate led to not having enough time to reach an optimum and underfitted the data. We used 400 warmup steps for the lreaning rate scheduler, which was roughly 1/3 of the updates per one finetuning epoch.

The final hyperparameters we settled on were `batch_size=64` (4 accumulation steps, 16 training examples per step), `epochs=5, lr=1e-4, warmup_steps=400`.

By running the `finetune.py` script, users can finetune the ProGen2 model, which gets automatically downloaded from Hugging Face repository containing a mirror of the original model [1]. For the inputs, users specify the train and test files containing preprocessed sequences, as described in the previous step. Before actually finetuning the model parameters, the program automatically infers the special family tokens from the provided train and test datasets and extends the tokenizer's vocabulary and the embedding matrix. Users can specify values of hyperparameters using command line arguments, and can also specify the checkpoint rate, meaning how often to save the model checkpoint during the finetuning process, and optionally also save the optimizer and scheduler.

3.3 Inference

We implement autoregressive generation of sequences using top-k sampling with temperature. Listing 3.2 shows an implementation of autoregressive sequence generation employing top-k sampling with temperature and caching of previous attention keys and values, which we describe below, in PyTorch (Paszke et al., 2019). When launching `sample.py`, users specify the model name from Hugging Face [2], or optionally a path to local checkpoint, device on which to run the program (CPU or CUDA), the batch size (how many sequences to generate in parallel),

[1] `https://huggingface.co/hugohrban/progen2-small`

[2] `https://huggingface.co/hugohrban/progen2-small-mix7`

how many batches to generate, and maximum length of sequences. Users can also specify the prompt, which the model uses as initial context for the autoregressive generation. For sampling from the output probability distribution, users can specify values of k and temperature t. There is also the `--bidirectional` flag, which causes the model to first generate half of the sequence in one direction, then reverse it, and generate the second half in the opposite direction, ideally when using a bidirectional model.

Listing 3.2 Implementation of top-k sampling with temperature

```
x = torch.tensor(tokenizer.encode(prompt).ids)
generated = x
past_key_values = None
while generated.shape[-1] < max_length:
    output = model(x, past_key_values=past_key_values)
    past_key_values = output.past_key_values
    logits = output.logits
    logits = logits[:, -1, :]
    logits = logits / temperature
    if k is not None:
        v, _ = torch.topk(logits, k, dim=-1)
        logits[logits < v[:, -1].unsqueeze(-1)] = -1e9
    probs = torch.softmax(logits, dim=-1)
    x = torch.multinomial(probs, num_samples=1)
    generated = torch.cat([generated, x], dim=-1)
```

At each iteration of generation, we cache the attention keys and values for every transformer layer, so at every step of generation, we compute attention where the query is only from the token generated at the previous step and the keys are projections of the embeddings of all previous tokens, which decreases the computational complexity of the attention mechanism from quadratic to linear with respect to sequence length. In practice, this results in roughly a 10-fold speed improvement over a naive implementation for the generation of a whole sequence.

After the generation, padding and non-residue tokens are removed and all generated sequences are saved in a `.fasta` file, whose name contains the prompt and values of sampling parameters k and t.

4 Results

4.1 Metrics

In this section, we will briefly describe the metrics we use to evaluate the models and the sequences they generate.

- **Cross-entropy** We can use cross-entropy (same as the loss function during training), denoted as $H(p, q)$, to assess the model's ability to to predict residues in sequences unseen during training. It is usually reported as the mean across all tokens in the sequence. It is important to distinguish whether cross-entropy is calculated on padded or non-padded sequences.

- **Perplexity** is defined as $PPL = exp(H(p, q))$. Perplexity is a monotonous function of cross-entropy and is often reported when evaluating language models. It has a more intuitive interpretation than cross-entropy: If a model has perplexity x, on average it predicts the correct label with probability $1/x$. So a simple model predicting one of the 20 residues with uniform probability would have $PPL = exp(-log(1/20)) = 20$.

- **HMMER matches** – We run the `hmmscan` command (Johnson et al., 2010), scanning all generated sequences against the profile HMM and measure the percentage of sequences that have an E-value below the default threshold of `1e-2`.

- **Sequence identity to training data** – We use the MMseqs2 `easy-search` command (Steinegger; Söding, 2017) to align all generated sequences against all training sequences. For a given query (generated) sequence, we find the target sequence with highest percent identity of aligned residues. We aim to achieve low sequence identity to the training data to show that the model is generating new sequences. Not all generated sequences will produce an alignment, MMseqs2 only reports those with at least 40% sequence coverage, which is the ratio between alignment length and length of the longer sequence.

- **pLDDT** – Predicted local distance difference test is the confidence score from models which predict the tertiary structure of a protein, in our case ESMFold (Lin et al., 2023). The model outputs pLDDT per-residue, but we report the mean across all amino acid residues in the sequence.

4.2 Finetuning progress

We finetuned the ProGen2-small model (Nijkamp et al., 2022) on a dataset consisting of 7 protein families. Each training example consisted of a special token specifying the family followed by the `"1"` token denoting the N-terminus, then the sequence itself, and lastly a `"2"` token for the C-terminus. We used 80% of the data for training and remaining 20% for testing. Here we visualize the loss on train and test set with respect to number of finetuning iterations and epochs (in

this case one epoch was 1114 iterations). We investigate how the performance on our task is influenced by the number of training updates and how long it takes to overfit in the following sections.

Figure 4.1a shows the graph of model's cross-entropy loss during finetuning. We measure loss on training data for every batch and plot the moving average with window size 10 to make the graph appear less noisy. We also measure loss on test dataset, which comes from the same distribution as training data. Both train and test loss are computed on sequences padded up to 1024 tokens. This allows us to put several sequences in a batch and compute the loss efficiently in parallel on a GPU. However, since loss for a sequence is computed as the average across sequence length, this means that padded sequences have lower loss compared to non-padded sequences, because the padding token is trivial for the model to predict accurately. This implementation is useful, for example, when we want to compare models trained and tested on the same data but with different hyperparameters, or to compare the same model during the finetuning process.

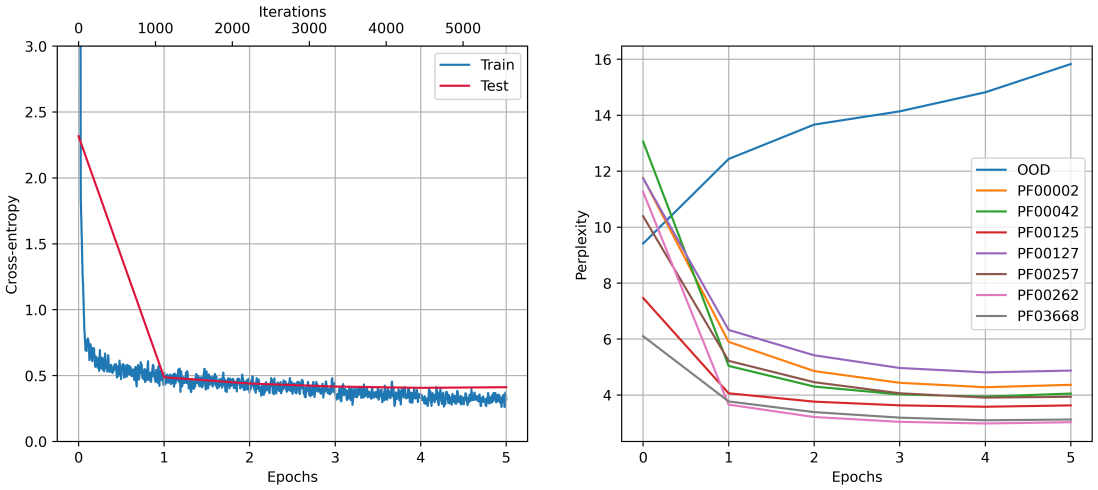
Figure 4.1b, we visualize the perplexity on test data for each family separately. We see that perplexity varies for the different families, but for most of them, we achieve lowest perplexity after 5 finetuning epochs, mainly thanks to decaying the learning rate during finetuning. We also plot the loss on out-of-distribution (OOD) data, which we obtain by taking a random subset of 250 sequences from the UniProt database (The UniProt Consortium, 2022). This shows us how the model starts to "forget" the original data it was trained on. For the purposes of our task, this is not detrimental, because we want the finetuned model to only be able to generate from the selected families. During pretraining of the base model ProGen2 (Nijkamp et al., 2022), the authors concatenated sequences to fill the context widow size of 1024 tokens, so the padding token was not seen during pretraining and the loss of the base model on padded sequences was extremely high. Contrary to the left part of the figure, here we do not use padding, so we can directly compare with the base model, which has very high loss on padded sequences, because it did not learn the special padding token embedding, and also we can directly compare the families regardless of the lengths of their sequences.

4.3 Experiments

In this section we will present experiments we have conducted with the model. We will investigate how to find suitable parameters for sequence generation and analyze the properties of the generated protein sequences.

4.3.1 Sampling parameters

As we discussed in section 2.2.1, there are various kinds of algorithms for decoding outputs of language models. Here, we are using top-k sampling with temperature t. We want to find out how the generation behaves depending on these two parameters. Let $K = \{3, 5, 10, 15, 20, 32\}$ and $T = \{0.1, 0.5, 1.0, 1.5, 2.0, 5.0\}$. For each pair $(k, t) \in K \times T$ and every family, we generate 128 sequences. For example when generating from family PF03668, we set the prompt as `<|pf03668|>1` to specify the family special token and that we want to generate from the N-terminus. For each generated set of sequences, we run HMMER's `hmmscan` command and



(a) Cross-entropy loss on train and test data

(b) Perplexity per family and on OOD sequences

Figure 4.1 Finetuning hyperparameters: `batch_size=64, epochs=5, lr=1e-4, decay='cosine'`.

from its output determine the percentage of sequences that belong to the desired family according to the profile HMM. We use MMseqs2 to calculate the average sequence identity to training data. We show the results for the generated sequences from the model finetuned for 5 epochs. In figure 4.1 we show the tables of results for 3 arbitrarily chosen families out of 7. The tables on the left show the percentage of sequences belonging to the family according to HMMER. The tables on the right show the average sequence identity to the training data. In cases when no HMMER hits or no alignments from MMSeqs2 were obtained, the cell is left blank. We chose not to perform greedy decoding ($k = 1$), because it generates sequences in a deterministic fashion, and since our prompt is only the family token and `1`, we could always only produce one sequence per family.

For low values of temperature the sequences have very high sequence identity to training data. Conversely, for higher values of temperature, sequences have much lower percentage of sequence identity to training data, but also a much lower percentage of HMMER hits, meaning that very few of them belong to the family. This goes along with the intuition that with higher temperature, we are able to produce more "creative" sequences that are not similar to the ones in training data and with low temperature, the model is more "conservative" with its responses and therefore sequences are close to the training data and hence definitely belong to the specified family.

Depending on the family and t, changing values of k can have various effects. In most cases, low values of k cause higher HMMER match rate, but overall the best value for k is 10 or 15. In all cases, high values of both t and k cause the generated sequences to become "gibberish" and none of them produce a significant score with the profile HMM or alignment to training data using MMseqs2. This is because at each step of generation, the model is very likely to sample from the tail of the probability distribution.

In case of some families, sampling with a low temperature $t = 0.1$, causes the model to sometimes produce many repetitive and degenerated sequences, for

PF00002 - GPCR

k \ temperature	0.1	0.5	1.0	1.5	2.0	5.0
3	10.94	30.47	19.53	17.19	10.94	4.69
5	9.38	32.03	30.47	15.62	14.84	3.12
10	10.94	39.84	57.81	30.47	7.81	
15	8.59	39.06	71.88	20.31	2.34	
20	8.59	39.84	63.28	10.16	1.56	
32	7.03	37.5	68.75	9.38	0.78	

k \ temperature	0.1	0.5	1.0	1.5	2.0	5.0
3	91.37	85.76	72.65	66.59	53.43	48.08
5	91.78	81.73	62.95	56.19	49.9	31.72
10	90.6	81.9	57.17	46.96	42.02	
15	89.75	77.94	54.97	46.53	44.52	
20	90.51	77.83	52.57	50.2	51.12	
32	92.86	76.13	54.94	50.09	45.93	

PF00125 - Core histone

k \ temperature	0.1	0.5	1.0	1.5	2.0	5.0
3	100.0	96.88	93.75	85.16	78.91	32.81
5	100.0	94.53	92.19	90.62	69.53	16.41
10	100.0	94.53	89.84	57.03	29.69	
15	100.0	89.84	83.59	29.69	0.78	
20	100.0	91.41	72.66	7.03	1.56	
32	100.0	88.28	67.19	9.38		

k \ temperature	0.1	0.5	1.0	1.5	2.0	5.0
3	100.0	99.26	95.36	84.99	74.67	54.34
5	100.0	99.13	91.88	77.59	67.46	51.18
10	100.0	98.69	86.62	72.66	61.81	
15	100.0	98.46	82.98	69.44	45.58	
20	100.0	98.76	86.61	62.49	78.23	
32	100.0	98.35	85.93	73.26	59.2	

PF03668 - P-loop ATPase

k \ temperature	0.1	0.5	1.0	1.5	2.0	5.0
3	49.22	73.44	82.81	90.62	87.5	47.66
5	47.66	84.38	95.31	89.06	78.12	13.28
10	50.78	95.31	96.88	76.56	42.97	1.56
15	53.91	95.31	89.06	53.91	25.0	
20	46.09	91.41	87.5	45.31	10.94	
32	53.12	95.31	84.38	42.19	10.94	

k \ temperature	0.1	0.5	1.0	1.5	2.0	5.0
3	80.1	78.74	71.87	63.04	57.21	46.32
5	78.88	78.59	67.56	56.73	48.4	44.1
10	80.35	76.65	62.86	52.8	51.37	43.0
15	79.64	77.49	61.46	51.2	54.97	
20	81.84	75.99	61.2	57.48	54.35	
32	79.93	75.9	62.38	54.77	71.51	

Table 4.1 HMMER matches and sequence identity for various values of top-k and temperature

example `"<|pf00002|>1MSLSLSLSLSLSLS..."`, which are obviously not members of the desired families. Sampling with such low temperature is very close to greedy decoding, which means that the model outputs the token with the highest probability almost always. This approach often results in the repetitive nature of the sequences, although the reasons for this pattern remain unclear.

In the following experiments, all sequences will be generated with parameters $k = 15$ and $t = 1.0$, unless specified otherwise.

4.3.2 HMMER matches

In the previous section, we compared generated sequences for each family with the profile HMM built for that particular family in order to find appropriate values of k and t. Now we want to see how capable our model is in generating sequences from the remaining families and if the model also produces false positives, meaning that it generates sequences which belong to a different family than the one specified by the prompt.

In figure 4.2 we compare sequences generated for each of the 7 families against a profile HMM from each family. Each cell in the table contains percentage of the generated sequences that produce a significant score with that profile HMM, and therefore belong to that family. We see that for most families, the percentage of sequences belonging to the correct family is above 70%, but for families PF00257 and PF00262 the values lowest. For the first one mentioned, it may be because it is the lowest resource family, comprising only around 0.23% of the training data. In case of the second low-performing family, we hypothesize that the diversity of the sequences in this family was too high and therefore the model could not effectively able to learn the features of this particular family. For most cases we do not get many false positive hits. The largest amounts of false positives are

Generated for family \ Profile HMM for family	PF00002	PF00042	PF00125	PF00127	PF00257	PF00262	PF03668
PF00002	71.88	0.0	1.56	0.78	0.78	0.0	0.0
PF00042	0.0	79.88	0.0	0.2	0.0	0.0	1.17
PF00125	0.0	0.0	83.59	0.0	0.0	1.56	0.0
PF00127	0.39	0.2	0.0	92.58	0.0	0.0	0.0
PF00257	0.59	0.0	9.96	0.78	60.94	0.39	0.39
PF00262	1.56	0.78	14.84	4.69	0.78	40.62	0.0
PF03668	0.0	0.78	0.0	0.0	0.0	0.0	89.06

Table 4.2 Comparison of every set of generated sequences against every profile hidden Markov model

reported by the profile HMM for family PF00125 suggesting that perhaps this hidden Markov model's parameters are more prone to reporting false positives from the set of sequences generated for family PF00262, especially because in the opposite direction we do not get as many false positive hits.

4.3.3 Diversity of generated sequences

We want to investigate how similar are the sequences our model generates to the sequences in the training data to make sure that it is not just copying the sequences seen during finetuning, but is it can actually generate novel sequences. We measure sequence identity using MMseqs2 (Steinegger; Söding, 2017) command `easy-search`, which outputs sequence identity of every generated sequence compared to every sequence in the training data. Sequence identity is calculated as the percentage of identical residues in the alignment of the two sequences. For each generated sequence we consider the maximum identity to any sequence in finetuning data as its sequence identity to the whole finetuning dataset.

In figure 4.2 we plot the cumulative density function of the sequence identity for each family of generated sequences separately. Same as the previous experiments, the initial prompt is set as only the family token and the N-terminus token, e.g. `<|pf00002|>1`. We see that there are substantial differences between the distributions for individual families. In case of families PF00125 and PF00257 the sequences exhibit highest sequence similarity to the training data, but for example when generating for family PF00002, the sequences have much lower sequence identity to finetuning data. However, in all cases, there is a decent number of sequences below 80% sequence similarity threshold, showing that our model is not too overfitted on any of the seven families, and these differences likely originate from the diversity of the individual families we finetuned on. For clarity, we show the average sequence identity for each family in table 4.3.

4.3.4 Single-family vs multi-family model

So far, we finetuned the model on several distinct families simultaneously. The advantage of this approach is that we can prompt the model to generate sequences from these families by just specifying the family token and as we have shown earlier, the model can generate valid sequences from the selected family. We want to see how the performance of our model compared to a model finetuned

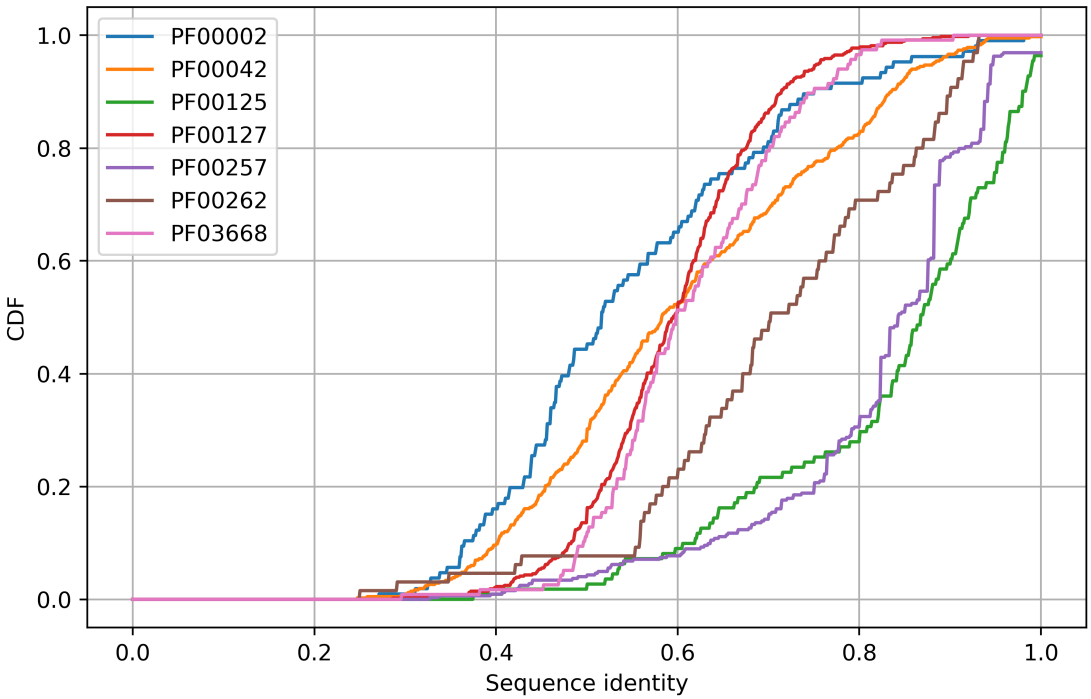
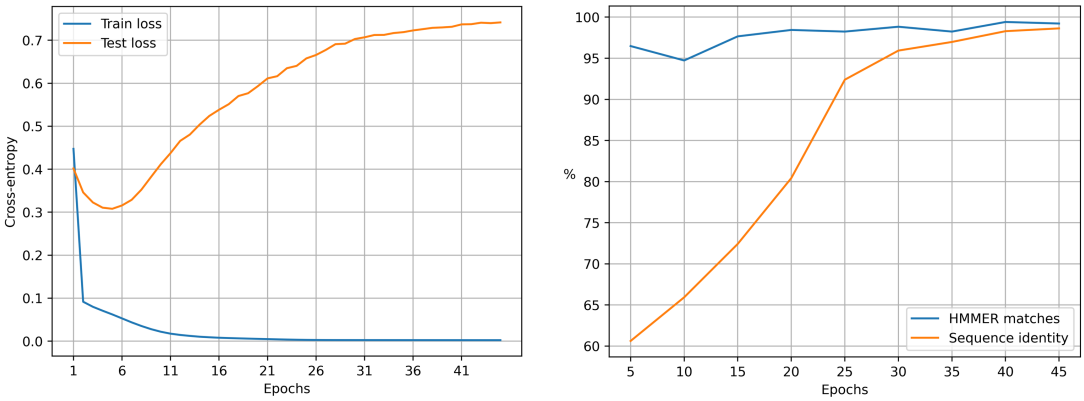


Figure 4.2 CDF of sequence identity to finetuning dataset for each family

Family	Avg. seq. identity (%)
PF00002	54.9
PF00042	60.6
PF00125	82.9
PF00127	59.6
PF00257	81.6
PF00262	70.8
PF03668	61.5

Table 4.3 Average sequence identity to finetuning dataset for each family



(a) Cross-entropy loss **(b)** HMMER match rate

Figure 4.3 Overfitting on the P-loop ATPase family (PF03668)

only on one protein family. We choose the family PF00262 which had the lowest HMMER match rate in the previous experiments to see if a model finetuned specifically on this family can generate more sequences that belong to it. We used the same finetuning hyperparameters as before: `batch-size=64, lr=1e-4, decay="cosine", epochs=5, warmup-steps=30`. The training dataset contains 3565 sequences, so one epoch takes only 55 iterations. The following table compares model perplexity on a held-out subset of the PF00262 family, HMMER match rate and average sequence identity to training data of two sets of sequences – one generated by the original 7-family model and model finetuned only on PF00262.

	7-family model	PF00262-only model
Perplexity	**3.03**	3.61
HMMER matches	40.62	**55.27**
Mean seq. identity	70.76	**62.33**

We see that despite having higher perplexity, the family-specific model is able to generate more sequences that belong to the correct family, and also have lower sequence identity to training data.

What does it look like when the model overfits the training data? This means that the model memorizes the training sequences and during generation just reproduces the sequences it has seen during training. The number of epochs it takes to overfit depends on the choice of hyperparameters, most notably the learning rate, and the choice of protein families. An overfitted model would have a very high HMMER match rate, but also a very high average sequence identity to training data. To illustrate why the HMMER match rate metric in itself is not sufficient to asses the quality of the finetuned model, we purposefully overfit another single-family model on family PF03668 – P-loop ATPase for 45 epochs, starting from a learning rate 0.001, using cosine decay and batch size 64. Figure 4.3a shows the graph of train and test loss. The model achieves the lowest test loss 0.301 after just 5 epochs. Fig. 4.3b shows the percentage of HMMER matches and average sequence identity for sequences generated after every 5 finetuning epochs. We see that HMMER match rate slightly increases, but is above 95% already after 5 epochs. The average sequence identity keeps increasing almost up

to 100%, meaning that towards the end, the model is only repeating its training data.

As we showed earlier, despite clustering, the average sequence identity can vary widely among different families depending on their level of conservation, so before finetuning, we do not know what value of sequence identity we can expect. This goes to show that cross-entropy or perplexity on test data is a more reliable metric for evaluating the generative abilities of protein language models, but it may take several training runs to find the right hyperparameters for our objective.

4.3.5 Directionality

The models that we trained thus far were only trained on sequences written in the N→C direction and therefore only learned to predict the next token in one way. The original ProGen2 models were pretrained on sequences in both directions and the N and C termini were labeled using tokens `"1"` and `"2"` respectively. Therefore we want to investigate if a model trained on sequences in both directions can learn the protein language more effectively and hence generate sequences that perform better on our evaluation metrics. This way of training the model is also potentially more useful to end users, because they can specify a prompt which is in the middle of the sequence, for example a conserved motif, and then continue the generation in one direction and then the other direction.

We finetuned the model from the ProGen2-small checkpoint with the same hyperparameters as the previous one-directional model. The finetuning in this case took twice as many weight updates per epoch, since we doubled the amount of finetuning data. From each checkpoint, we generated 512 sequences for each prompt using sampling parameters `k=15, t=1.0`. For the prompts, we used the special family token followed by token `1` in case of the original one-directional model and either `1` or `2` token in case of the bidirectional model to specify the direction of generation. After the generation, sequences generated in C→N direction, with token `2` in the beginning, were reversed so that they could be compared with the profile HMM. Table 4.4 shows the percentage of generated sequences that belong to the specified family.

For 5 out of 7 families, we higher HMMER hit rate performance by generating the bidirectional model. In case of the lowest resource family PF00257, the success rate improves by almost 20% and in the case of the worst-performing family in the one-directional model – PF00262 – the HMMER hit rate improves by nearly 18% using the bidirectional model. When we compare the two directions of generation within the bidirectional model, we see that in the majority of cases the difference between them is just a few percent. On average across all families, the bidirectional model generates more valid sequence in either direction than the one-directional model, suggesting that seeing both directions of each sequence during finetuning improves the model's overall generation capabilities. The average sequence identity did not differ much across the three ways of generation, the largest difference within a family was less than 3%, which can be attributed to random noise in the generated data.

Protein family	One-directional model	Bidirectional model	
	$N \rightarrow C$	$N \rightarrow C$	$C \rightarrow N$
PF00002	71.88	**74.22**	69.92
PF00042	**79.88**	77.73	75.39
PF00125	83.59	**91.02**	82.81
PF00127	**92.58**	91.41	91.41
PF00257	60.94	**80.86**	75.78
PF00262	40.62	55.08	**58.59**
PF03668	89.06	**93.36**	92.97
Average	74.08	**80.53**	78.12

Table 4.4 Percentage of sequences belonging to the specified family according to HMMER. Comparison between one-directional and bidirectional model.

4.3.6 Unconditional generation

We would like to see what the model generates when we do not provide any special family token, but only use `"1"` as the prompt, or also `"2"` in case of the bidirectional model from the previous section, and if it is able to generate sequences that are part of any of the seven families it was finetuned on. In this way we generate 1000 sequences from the one-directional model and also 1000 sequences in both ways from the bidirectional model. The following table shows percentage of sequences that belong to at least one of the 7 families, and if so, which ones. We include only the families which had at least 10% of matches in any of the three generated sets.

	One-directional model	Bidirectional model	
	$N \rightarrow C$	$N \rightarrow C$	$C \rightarrow N$
Any family	22.32	45.14	36.58
PF00125	22.07	3.12	2.73
PF00127	0.00	37.11	19.14
PF00002	0.00	2.34	13.48

Surprisingly, the one-directional model generates predominantly sequences from family PF00125 when no family tag is specified, although overall it generates mostly sequences that do not belong to any family. Note, that only around 19% of training sequences are belong to this family.

The bidirectional model is overall more successful in its generation, presumably due to more weight updates during finetuning. When we use `"1"` as prompt, meaning that it generates in N→C direction, most of the sequences belong to family PF00127, and compared to the previous model, much fewer sequences belong to PF00125. For the opposite direction of generation, the highest represented family is still PF00127, but family PF00002 has substantially more members when we generate in this direction.

4.3.7 P-loop motif

Sequences in the P-loop ATPase protein family PF03668 share a common motif called the P-loop, or Walker A motif, wihch can be described by a Python regular

expression `r"G.{4}GK[TS]"`. This motif is known to have a highly-conserved three-dimensional structure and is responsible for binding phosphate, such as adenosine triphosphate (ATP). There is a relatively high probability, that this pattern occurs by chance. A random sequence of 500 residues, contains the P-loop with probability around 1.2%. Despite its name, only 61.91% of sequences in the family actually contain this pattern and others were probably included in the family due to having a significant match with the profile HMM constructed from the hand-picked seed alignment. Out of the generated sequences from the one-directional model, 58.14% of them contain the P-loop pattern. We can expect the P-loop content to get only as high as the P-loop content in the training data, but the fact that our P-loop content is so high, shows that the model is able to learn more subtle features of the protein families, such as this sequence motif.

4.3.8 ESMFold visualizations

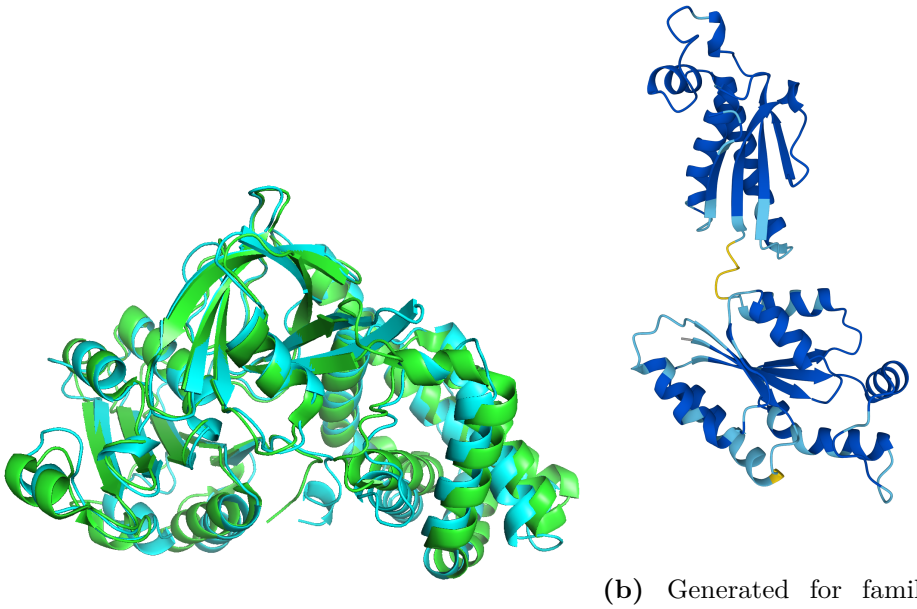
As we mentioned earlier, the confidence score from models predicting protein's three dimensional structure, called pLDDT, is often used as an *in silico* metric to assess the quality of generated protein sequences (Watson et al., 2023). Values of pLDDT range from 0 to 100, with 100 being the highest confidence. We used ESMFold (Lin et al., 2023) to predict structures of several of the sequences, which we have generated using our protein language model. We visualize a couple of cherry-picked examples of structure predictions, for which ESMFold returns high pLDDT, and at the same time have have relatively low sequence identity to the training set.

Figure 4.4a contains the predictied structure of a sequence belonging to the globins family (PF00042) shown in blue overlayed with the experimentally determined structure of protein 1GVH from the PDB (Berman et al., 2000) shown in green. Globins are a family of heme-contatining proteins involved in oxygen binding and transport. Our generated protein has relatively low sequence identity to other known proteins (71.4%) and is only 41.7% identical to 1GVH despite its predicted fold being very similar to it. Figure 4.4b shows the prediction of an ATP-binding protein containing the P-loop motif. Just like the previous example, this structure is also predicted with high pLDDT, but an even lower sequence identity to the training data.

The fact that our model is able to generate sequences, which can be predicted with a high pLDDT score, suggests that it is indeed capable of generating novel protein sequences yet unexplored by nature, which are likely to be well-folded into structures that allow them to carry out their respective functions.

4.4 Learned representations

In this final section, we will not examine the generated protein sequences from our finetuned model, but rather we will investigate some of the internal properties of the model itself. We explore its learned token representations and how it processes its inputs via the self-attention mechanism.



(a) Generated for family PF00042 (blue) and 1GVH (green). pLDDT: 90.4, seq. id.: 71.4%

(b) Generated for family PF03668. pLDDT: 90.9, seq. id.: 58.6%

Figure 4.4 ESMFold structure predictions

4.4.1 New token embeddings

The embedding layer of ProGen2 (Nijkamp et al., 2022) was initialized randomly without any prior knowledge of physiochemical properties of individual residues. Following this, we also initialized the special family token embeddings with random values from the normal distribution with zero mean and small variance. In Figure 4.5, we visualize principal component analysis (PCA) to two dimensions of all learned embeddings from our one-directional finetuned model, including amino acid residue embeddings, as well as embeddings of special tokens. Each amino acid in the protein has unique properties determined by its chemical composition and the structure of its side chain. We see that using self-supervised learning, the model is able to learn the properties of amino acids. We color the individual residues depending on whether they are aromatic, polar, small, or aliphatic. Note, that in reality, these properties are more complex, and one amino acid may belong to several of these groups. We notice that the special tokens `1, 2` are close together as they denote the beginning or end of a sequence. The special family tokens are all clustered together, near point $(0, 0)$, suggesting that their vector representations have not shifted much since their initialization.

4.4.2 Attention head visualization

Key part of the transformer architecture is the self-attention mechanism. It allows the model to handle dependencies and relationships between input tokens regardless of their distance within the context window. In our case, each of the 12 layers in the model contains 16 attention heads, which process the input in parallel, independently of each other, and after that their individual outputs are concatenated and passed to the next layer. This allows each attention head to

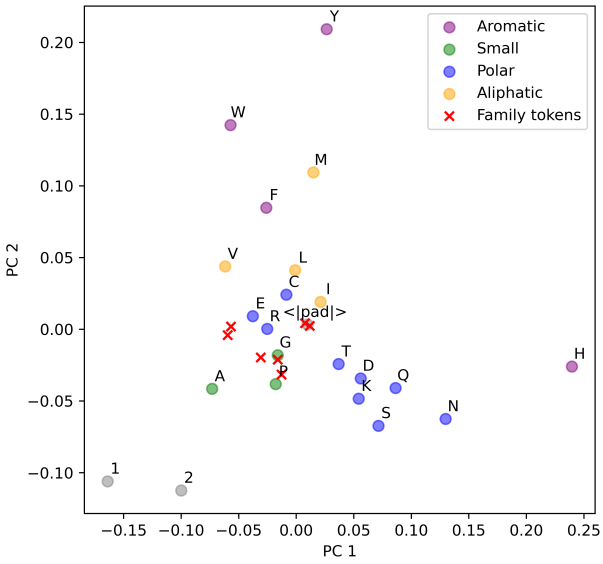
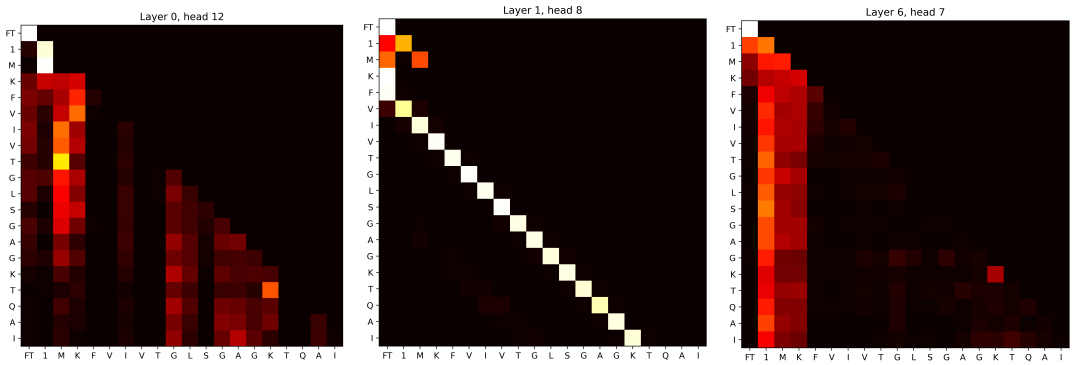


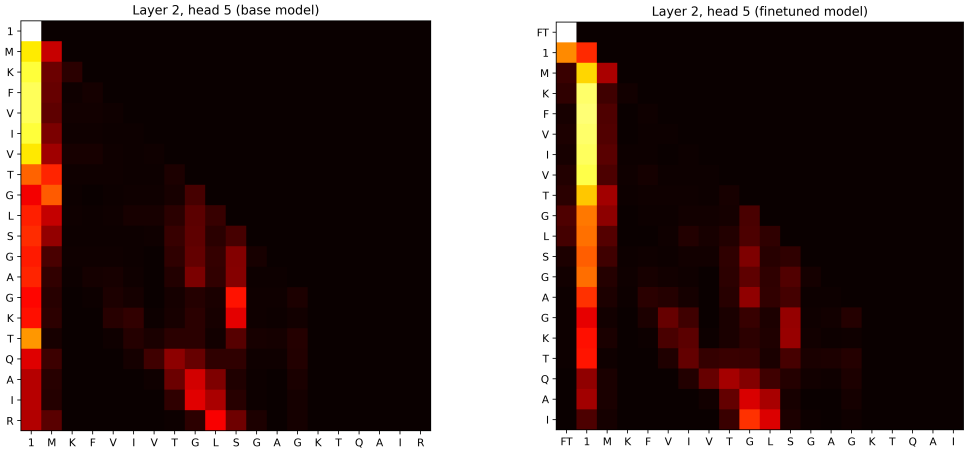
Figure 4.5 PCA of token embeddings

focus on different properties of the input sequence. Generally, it may be hard to interpret the specific function of a given attention head within the model. Since our model follows the decoder-only architecture, the attention is causal, meaning that each token only attends to the preceding tokens, since during autoregressive generation, we can only see the past tokens, not the future ones. Here, we highlight a few selected attention heads to better understand how input data is processed inside the model. In this case, the input was a randomly chosen protein sequence from the training dataset. We visualize the attention weights, which is the matrix output of $\text{softmax}(QK^T/\sqrt{d_k})$, so each row is a probability distribution over the previous tokens. Figure 4.6a shows a visualization of three different attention heads, each with significantly different attention weights on the same input sequence (we only show attention over the first 20 tokens). For example, we see that in most cases, head 8 in layer 1 attends to the token 4 positions earlier, which may suggest that it plays a role in processing alpha helices, because in an alpha helix, a hydrogen bond is formed between two amino acids four positions apart in the sequence. Interestingly, in layer 6, head 7, very little attention is paid to the special family token, which we denote as `FT` in the figure, for simplicity. It is the case in the majority of attention heads, that the family token is usually not attended to, especially when longer context is available. To highlight this, in Figure 4.6b, we feed the same input sequence into the base model and the finetuned model. Overall, the patterns look very similar, except that in almost every level, except that the family token is mostly ignored. Despite this, as we have shown earlier, the model is able to perform its task well, and more importantly, is able to distinguish between the respective families. This implies, that the special token is only important during early parts of generation, and later the model is able to rely on the residue tokens to understand what family is being generated.

Finally, in figure 4.7, we show two attention heads on a longer context window



(a) Attention weights from three different attention heads of the finetuned model



(b) Comparison between attention of base and finetuned model

Figure 4.6 Visualization of attention heads

of size 150 tokens, taken from the same sequence as before. In figure 4.7a, we depict attention weights from head 8 in layer 4. We notice several vertical lines, usually highlighting part of the input sequence around 25 tokens in the past. These could represent a significant part of the protein sequence, such as a sequence or structural motif, an active or biding site, etc. Figure 4.7b shows another attention head, which mostly focuses on the last few tokens at every step, but also the very beginning of the sequence, again, except for the family token.

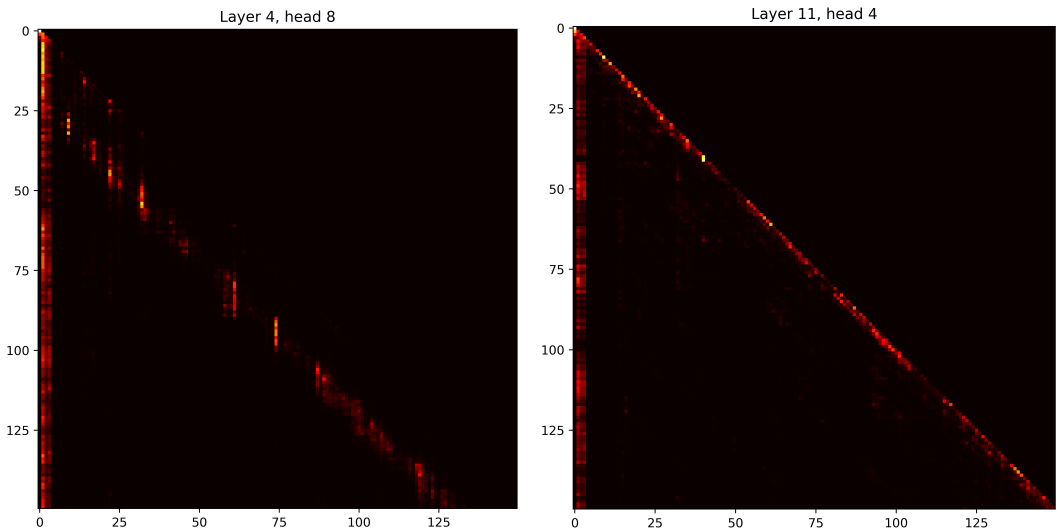


Figure 4.7 Attention over a larger context length

5 Discussion

We have finetuned a generative protein language model on a set of protein families and shown that users can effectively prompt the model by only specifying a single family token. There are limited *in silico* benchmarks for assessing quality of protein sequences, since their actual functions and properties can be observed only in a living cell, so it would be very interesting to see if someone can use this model to generate sequences, synthesize them and experimentally validate their properties.

We have invested quite a lot of time into finding the right finetuning hyperparameters and try to provide a general guideline based on our empirical observations about how each hyperparameter influences the finetuning process.

5.1 Future ideas

We have developed a bidirectional model trained on both directions of the protein sequence and our provided sampling script can perform bidirectional generation, where the original prompt ends up in the middle of the final generated sequence. We have, however, chosen not to perform such bidirectional prompting experiments, because due to lack of deeper biological knowledge we were not sure how to choose prompts that would work well in practice, such as conserved regions or motifs.

An obvious extension of this work would be to increase number of families we are finetuning the model on to see how far we can push the limits of how many new control tokens the model can learn. Another idea is to use special tokens not only for protein families, but also for specific properties. This would give the users even more control over the generation, because they could specify multiple special property tokens per sequence and create proteins tailored specifically to their needs, similarly to how the original ProGen model (Madani; McCann, et al., 2020) functioned. We see this lack of controllability as a major drawback for *de novo* protein sequence design using autoregressive language models. Large language models in NLP have seen tremendous success by instruction tuning to make them useful as conversational assistants (OpenAI, 2024) and hopefully this can be someday adapted in protein engineering. Structure design of proteins has become very popular recently (Watson et al., 2023), but the sequence space is the ultimate design space for proteins, since every protein is fully determined by the sequence of residues.

There exist several techniques for more efficient finetuning of large language models, such as quantization (Zafrir et al., 2019), which uses lower precision for storing the parameters and therefore makes the model much more light-weight and would allow users to work with much larger models. Another approach we would like to try is prompt tuning (Lester et al., 2021), which involves only updating the new token embeddings, instead of all model weights and would greatly decrease the memory requirements for finetuning a large model. Another similar technique called low rank adaptation (LoRA Hu et al., 2021) introduces new trainable intermediate layers, while keeping rest of the model freezed, which also greatly decreases compute requirements for finetuning. Although the smallest ProGen2

checkpoint has been shown to work well in case of 7 families, larger checkpoints may need to be used in order to finetune on more families or other special property tokens, which would require using some of the previously mentioned methods for parameter efficient finetuning.

Conclusion

Designing new proteins with desired functions is an important problem in drug design and therapeutics development. Deep neural networks and large language models have proven useful for many tasks, including *de novo* protein design, which could accelerate the drug discovery process. Exploring properties and abilities of these machine learning models is an interesting area of research.

The goal of this work was to provide a more controllable generation interface for a pre-trained protein language model (Nijkamp et al., 2022). We have finetuned the model on seven distinct protein families and examined the properties of sequences generated by the model. We have shown that the sequences our models generate belong to the families specified by the prompt and can bear low similarity to already existing proteins. This is usually sufficient for biologists who aim to synthesise proteins in a laboratory, because commonly tens of thousands of sequences can be generated by the autoregressive model cheaply and in a relatively short amount of time, and afterwards they can select top candidate proteins to be assembled in a living organism based on their respective objectives, which is comparatively much more expensive.

Furthermore, our code and models are publicly available, and give users the ability to finetune the model and generate sequences from protein families of their choice, or use the finetuned models described in this work for generation of novel protein sequences.

Bibliography

Alamdari, Sarah; Thakkar, Nitya; Berg, Rianne van den; Lu, Alex X.; Fusi, Nicolo; Amini, Ava P.; Yang, Kevin K., 2023. Protein generation with evolutionary diffusion: sequence is all you need. *bioRxiv*. Available from doi: 10.1101/2023.09.11.556673.

Alberts, B.; Bray, D.; Hopkin, K.; Johnson, A.D.; Lewis, J.; Raff, M.; Roberts, K.; Walter, P., 2015. *Essential Cell Biology*. CRC Press. isbn 9781317806271. Available also from: https://books.google.sk/books?id=Cg4WAgAAQBAJ.

Bateman, Alex, 2013. Sequence Classification of Protein Families: Pfam and other Resources. In: *Protein Families*. John Wiley & Sons, Ltd, chap. 2, pp. 25–36. isbn 9781118743089. Available from doi: https://doi.org/10.1002/9781118743089.ch2.

Bengio, Yoshua; Ducharme, Réjean; Vincent, Pascal; Jauvin, Christian, 2003. A neural probabilistic language model. *Journal of machine learning research*. Vol. 3, no. Feb, pp. 1137–1155.

Berman, Helen M.; Westbrook, John; Feng, Zukang; Gilliland, Gary; Bhat, T. N.; Weissig, Helge; Shindyalov, Ilya N.; Bourne, Philip E., 2000. The Protein Data Bank. *Nucleic Acids Research*. Vol. 28, no. 1, pp. 235–242. issn 0305-1048. Available from doi: 10.1093/nar/28.1.235.

Brändén, C.I.; Tooze, J., 1999. *Introduction to Protein Structure*. Garland Pub. isbn 9780815323051. Available also from: https://books.google.sk/books?id=zsWcpqrgG74C.

Brown, Tom B.; Mann, Benjamin; Ryder, Nick; Subbiah, Melanie; Kaplan, Jared; Dhariwal, Prafulla; Neelakantan, Arvind; Shyam, Pranav; Sastry, Girish; Askell, Amanda; Agarwal, Sandhini; Herbert-Voss, Ariel; Krueger, Gretchen; Henighan, Tom; Child, Rewon; Ramesh, Aditya; Ziegler, Daniel M.; Wu, Jeffrey; Winter, Clemens; Hesse, Christopher; Chen, Mark; Sigler, Eric; Litwin, Mateusz; Gray, Scott; Chess, Benjamin; Clark, Jack; Berner, Christopher; McCandlish, Sam; Radford, Alec; Sutskever, Ilya; Amodei, Dario, 2020. *Language Models are Few-Shot Learners*. Available from arXiv: 2005.14165 [cs.CL].

Dauparas, J.; Anishchenko, I.; Bennett, N.; Bai, H.; Ragotte, R. J.; Milles, L. F.; Wicky, B. I. M.; Courbet, A.; Haas, R. J. de; Bethel, N.; Leung, P. J. Y.; Huddy, T. F.; Pellock, S.; Tischer, D.; Chan, F.; Koepnick, B.; Nguyen, H.; Kang, A.; Sankaran, B.; Bera, A. K.; King, N. P.; Baker, D., 2022. Robust deep learning based protein sequence design using ProteinMPNN. *bioRxiv*. Available from doi: 10.1101/2022.06.03.494563.

Eddy, S R, 1998. Profile hidden Markov models. *Bioinformatics*. Vol. 14, no. 9, pp. 755–763. issn 1367-4803. Available from doi: 10.1093/bioinformatics/14.9.755.

Elnaggar, Ahmed; Heinzinger, Michael; Dallago, Christian; Rehawi, Ghalia; Wang, Yu; Jones, Llion; Gibbs, Tom; Feher, Tamas; Angerer, Christoph; Steinegger, Martin; Bhowmik, Debsindhu; Rost, Burkhard, 2021. ProtTrans: Towards Cracking the Language of Life's Code Through Self-Supervised Learning. *bioRxiv*. Available from doi: 10.1101/2020.07.12.199554.

Ferruz, Noelia; Schmidt, Steffen; Höcker, Birte, 2022. ProtGPT2 is a deep unsupervised language model for protein design. *Nature Communications*. Vol. 13, no. 1, p. 4348. issn 2041-1723. Available from doi: 10.1038/s41467-022-32007-7.

Forney, G.D., 1973. The viterbi algorithm. *Proceedings of the IEEE*. Vol. 61, no. 3, pp. 268–278. Available from doi: 10.1109/PROC.1973.9030.

Gamouh, Hamza; Novotny, Marian; Hoksza, David, 2023. Hybrid protein-ligand binding residue prediction with protein language models: Does the structure matter? *bioRxiv*. Available from doi: 10.1101/2023.08.11.553028.

Hu, Edward J.; Shen, Yelong; Wallis, Phillip; Allen-Zhu, Zeyuan; Li, Yuanzhi; Wang, Shean; Chen, Weizhu, 2021. LoRA: Low-Rank Adaptation of Large Language Models. *CoRR*. Vol. abs/2106.09685. Available from arXiv: 2106.09685.

Johnson, L. Steven; Eddy, Sean R.; Portugaly, Elon, 2010. Hidden Markov model speed heuristic and iterative HMM search procedure. *BMC Bioinformatics*. Vol. 11, no. 1, p. 431. issn 1471-2105. Available from doi: 10.1186/1471-2105-11-431.

Jumper, John; Evans, Richard; Pritzel, Alexander; Green, Tim; Figurnov, Michael; Ronneberger, Olaf; Tunyasuvunakool, Kathryn; Bates, Russ; Žídek, Augustin; Potapenko, Anna; Bridgland, Alex; Meyer, Clemens; Kohl, Simon A. A.; Ballard, Andrew J.; Cowie, Andrew; Romera-Paredes, Bernardino; Nikolov, Stanislav; Jain, Rishub; Adler, Jonas; Back, Trevor; Petersen, Stig; Reiman, David; Clancy, Ellen; Zielinski, Michal; Steinegger, Martin; Pacholska, Michalina; Berghammer, Tamas; Bodenstein, Sebastian; Silver, David; Vinyals, Oriol; Senior, Andrew W.; Kavukcuoglu, Koray; Kohli, Pushmeet; Hassabis, Demis, 2021. Highly accurate protein structure prediction with AlphaFold. *Nature*. Vol. 596, no. 7873, pp. 583–589. issn 1476-4687. Available from doi: 10.1038/s41586-021-03819-2.

Jurafsky, Dan; Martin, James H., 2024. *Speech and Language Processing*. 3rd ed. Draft available online. Available also from: https://web.stanford.edu/~jurafsky/slp3/. Draft release.

Keskar, Nitish Shirish; McCann, Bryan; Varshney, Lav R.; Xiong, Caiming; Socher, Richard, 2019. CTRL: A Conditional Transformer Language Model for Controllable Generation. *CoRR*. Vol. abs/1909.05858. Available from arXiv: 1909.05858.

Kingma, Diederik P.; Ba, Jimmy, 2017. *Adam: A Method for Stochastic Optimization*. Available from arXiv: 1412.6980 [cs.LG].

Krivák, Radoslav; Hoksza, David, 2018. P2Rank: machine learning based tool for rapid and accurate prediction of ligand binding sites from protein structure. *Journal of Cheminformatics*. Vol. 10, no. 1, p. 39. ISSN 1758-2946. Available from DOI: 10.1186/s13321-018-0285-8.

Lester, Brian; Al-Rfou, Rami; Constant, Noah, 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. *CoRR*. Vol. abs/2104.08691. Available from arXiv: 2104.08691.

Lin, Zeming; Akin, Halil; Rao, Roshan; Hie, Brian; Zhu, Zhongkai; Lu, Wenting; Smetanin, Nikita; Verkuil, Robert; Kabeli, Ori; Shmueli, Yaniv; Santos Costa, Allan dos; Fazel-Zarandi, Maryam; Sercu, Tom; Candido, Salvatore; Rives, Alexander, 2023. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*. Vol. 379, no. 6637, pp. 1123–1130. Available from DOI: 10.1126/science.ade2574.

Littmann, Maria; Heinzinger, Michael; Dallago, Christian; Weissenow, Konstantin; Rost, Burkhard, 2021. Protein embeddings and deep learning predict binding residues for various ligand classes. *Scientific Reports*. Vol. 11, no. 1, p. 23916. ISSN 2045-2322. Available from DOI: 10.1038/s41598-021-03431-4.

Madani, Ali; Krause, Ben; Greene, Eric R.; Subramanian, Subu; Mohr, Benjamin P.; Holton, James M.; Olmos, Jose Luis; Xiong, Caiming; Sun, Zachary Z.; Socher, Richard; Fraser, James S.; Naik, Nikhil, 2023. Large language models generate functional protein sequences across diverse families. *Nature Biotechnology*. Vol. 41, no. 8, pp. 1099–1106. ISSN 1546-1696. Available from DOI: 10.1038/s41587-022-01618-2.

Madani, Ali; McCann, Bryan; Naik, Nikhil; Keskar, Nitish Shirish; Anand, Namrata; Eguchi, Raphael R.; Huang, Po-Ssu; Socher, Richard, 2020. *ProGen: Language Modeling for Protein Generation*. Available from arXiv: 2004.03497 [q-bio.BM].

Mistry, Jaina; Chuguransky, Sara; Williams, Lowri; Qureshi, Matloob; Salazar, Gustavo A; Sonnhammer, Erik L L; Tosatto, Silvio C E; Paladin, Lisanna; Raj, Shriya; Richardson, Lorna J; Finn, Robert D; Bateman, Alex, 2020. Pfam: The protein families database in 2021. *Nucleic Acids Research*. Vol. 49, no. D1, pp. D412–D419. ISSN 0305-1048. Available from DOI: 10.1093/nar/gkaa913.

Nijkamp, Erik; Ruffolo, Jeffrey; Weinstein, Eli N.; Naik, Nikhil; Madani, Ali, 2022. ProGen2: Exploring the Boundaries of Protein Language Models. *arXiv*.

OpenAI, 2024. *GPT-4 Technical Report*. Available from arXiv: 2303.08774 [cs.CL].

Pan, Xingjie; Kortemme, Tanja, 2021. Recent advances in de novo protein design: Principles, methods, and applications. *Journal of Biological Chemistry*. Vol. 296, p. 100558. ISSN 0021-9258. Available from DOI: https://doi.org/10.1016/j.jbc.2021.100558.

Paszke, Adam; Gross, Sam; Massa, Francisco; Lerer, Adam; Bradbury, James; Chanan, Gregory; Killeen, Trevor; Lin, Zeming; Gimelshein, Natalia; Antiga, Luca; Desmaison, Alban; Köpf, Andreas; Yang, Edward Z.; DeVito, Zach; Raison, Martin; Tejani, Alykhan; Chilamkurthy, Sasank; Steiner, Benoit; Fang, Lu; Bai, Junjie; Chintala, Soumith, 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *CoRR*. Vol. abs/1912.01703. Available from arXiv: 1912.01703.

Paysan-Lafosse, Typhaine; Blum, Matthias; Chuguransky, Sara; Grego, Tiago; Pinto, Beatriz Lázaro; Salazar, Gustavo A; Bileschi, Maxwell L; Bork, Peer; Bridge, Alan; Colwell, Lucy; Gough, Julian; Haft, Daniel H; Letunić, Ivica; Marchler-Bauer, Aron; Mi, Huaiyu; Natale, Darren A; Orengo, Christine A; Pandurangan, Arun P; Rivoire, Catherine; Sigrist, Christian J A; Sillitoe, Ian; Thanki, Narmada; Thomas, Paul D; Tosatto, Silvio C E; Wu, Cathy H; Bateman, Alex, 2022. InterPro in 2022. *Nucleic Acids Research*. Vol. 51, no. D1, pp. D418–D427. ISSN 0305-1048. Available from DOI: 10.1093/nar/gkac993.

Pearson, William R., 2013. An Introduction to Sequence Similarity ("Homology") Searching. *Current Protocols in Bioinformatics*. Vol. 42, no. 1, pp. 3.1.1–3.1.8. Available from DOI: https://doi.org/10.1002/0471250953.bi0301s42.

Rives, Alexander; Meier, Joshua; Sercu, Tom; Goyal, Siddharth; Lin, Zeming; Liu, Jason; Guo, Demi; Ott, Myle; Zitnick, C. Lawrence; Ma, Jerry; Fergus, Rob, 2021. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*. Vol. 118, no. 15, e2016239118. Available from DOI: 10.1073/pnas.2016239118.

Ryckmans, Thomas, 2022. *File:Proteinogenic Amino Acid Table.png — Wikimedia Commons, the free media repository*. Available also from: https://commons.wikimedia.org/w/index.php?title=File:Proteinogenic_Amino_Acid_Table.png&oldid=654943691. [Online; accessed 8-May-2024].

Steinegger, Martin; Söding, Johannes, 2017. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature Biotechnology*. Vol. 35, no. 11, pp. 1026–1028. ISSN 1546-1696. Available from DOI: 10.1038/nbt.3988.

The UniProt Consortium, 2022. UniProt: the Universal Protein Knowledgebase in 2023. *Nucleic Acids Research*. Vol. 51, no. D1, pp. D523–D531. ISSN 0305-1048. Available from DOI: 10.1093/nar/gkac1052.

Thompson, Julie D.; Linard, Benjamin; Lecompte, Odile; Poch, Olivier, 2011. A Comprehensive Benchmark Study of Multiple Sequence Alignment Methods: Current Challenges and Future Perspectives. *PLOS ONE*. Vol. 6, no. 3, pp. 1–14. Available from DOI: 10.1371/journal.pone.0018093.

Touvron, Hugo; Lavril, Thibaut; Izacard, Gautier; Martinet, Xavier; Lachaux, Marie-Anne; Lacroix, Timothée; Rozière, Baptiste; Goyal, Naman; Hambro, Eric; Azhar, Faisal; Rodriguez, Aurelien; Joulin, Armand; Grave, Edouard; Lample, Guillaume, 2023. *LLaMA: Open and Efficient Foundation Language Models*. Available from arXiv: 2302.13971 [cs.CL].

Trivedi, Rakesh; Nagarajaram, Hampapathalu Adimurthy, 2022. Intrinsically Disordered Proteins: An Overview. *International Journal of Molecular Sciences*. Vol. 23, no. 22. ISSN 1422-0067. Available from DOI: 10.3390/ijms232214050.

Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N.; Kaiser, Lukasz; Polosukhin, Illia, 2017. Attention Is All You Need. *CoRR*. Vol. abs/1706.03762. Available from arXiv: 1706.03762.

Watson, Joseph L.; Juergens, David; Bennett, Nathaniel R.; Trippe, Brian L.; Yim, Jason; Eisenach, Helen E.; Ahern, Woody; Borst, Andrew J.; Ragotte, Robert J.; Milles, Lukas F.; Wicky, Basile I. M.; Hanikel, Nikita; Pellock, Samuel J.; Courbet, Alexis; Sheffler, William; Wang, Jue; Venkatesh, Preetham; Sappington, Isaac; Torres, Susana Vázquez; Lauko, Anna; De Bortoli, Valentin; Mathieu, Emile; Ovchinnikov, Sergey; Barzilay, Regina; Jaakkola, Tommi S.; DiMaio, Frank; Baek, Minkyung; Baker, David, 2023. De novo design of protein structure and function with RFdiffusion. *Nature*. Vol. 620, no. 7976, pp. 1089–1100. ISSN 1476-4687. Available from DOI: 10.1038/s41586-023-06415-8.

Wolf, Thomas; Debut, Lysandre; Sanh, Victor; Chaumond, Julien; Delangue, Clement; Moi, Anthony; Cistac, Pierric; Rault, Tim; Louf, Rémi; Funtowicz, Morgan; Brew, Jamie, 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *CoRR*. Vol. abs/1910.03771. Available from arXiv: 1910.03771.

Wu, Ruidong; Ding, Fan; Wang, Rui; Shen, Rui; Zhang, Xiwen; Luo, Shitong; Su, Chenpeng; Wu, Zuofan; Xie, Qi; Berger, Bonnie; Ma, Jianzhu; Peng, Jian, 2022. High-resolution de novo structure prediction from primary sequence. *bioRxiv*. Available from DOI: 10.1101/2022.07.21.500999.

Xiao, Jianxi; Cheng, Haiming; Silva, Teresita; Baum, Jean; Brodsky, Barbara, 2011. Osteogenesis Imperfecta Missense Mutations in Collagen: Structural Consequences of a Glycine to Alanine Replacement at a Highly Charged Site. *Biochemistry*. Vol. 50, no. 50, pp. 10771–10780. Available from DOI: 10.1021/bi201476a. PMID: 22054507.

Zafrir, Ofir; Boudoukh, Guy; Izsak, Peter; Wasserblat, Moshe, 2019. Q8BERT: Quantized 8Bit BERT. *CoRR*. Vol. abs/1910.06188. Available from arXiv: 1910.06188.

Zhao, Jingtian; Cao, Yang; Zhang, Le, 2020. Exploring the computational methods for protein-ligand binding site prediction. *Computational and Structural Biotechnology Journal*. Vol. 18, pp. 417–426. ISSN 2001-0370. Available from DOI: https://doi.org/10.1016/j.csbj.2020.02.008.

List of Figures

List of Tables

A Attachments

A.1 Code

Code for this thesis is available at `https://github.com/hugohrban/ProGen2-finetuning`
Description of provided files:

- `README.md` contains a description of provided python scripts illustrated by a simple workflow that finetunes the model on 3 protein families.
- `download_pfam.py` described in section 3.1, downloads Pfam entries specified by user.
- `prepare_data.py` described in section 3.1 preprocesses the downloaded data and adds special family tokens.
- `finetune.py` described in 3.2 finetunes a ProGen2-small model on preprocessed data.
- `sample.py` described in section 3.3 generates sequences using top-k sampling from a finetuned model or the base model.
- Files in the `models/progen` directory, namely `modeling_progen.py` and `configuration_progen.py` are adapted from the original authors Nijkamp et al., 2022 and define the model architecture and configuration.

Users can run all scripts with the `--help` or `-h` argument to display a help message and a list and explanation of all available arguments.

A.2 Models

All models described in this work are available at Hugging Face (Wolf et al., 2019).

- `https://huggingface.co/hugohrban/progen2-small` – Base ProGen2-small model (Nijkamp et al., 2022).
- `https://huggingface.co/hugohrban/progen2-small-mix7` – Model finetuned on 7 families, only in N $\rightarrow$C direction.
- `https://huggingface.co/hugohrban/progen2-small-mix7-bidi` – Contains the model finetuned on 7 families in both directions.