

NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Analysis of AlphaFold 2 algorithm

Vasiliki L. Pitsilou

Supervisors: **Ioannis Emiris**, Professor
Andreas Zamanos, MSc
George-Alexis Ioannakis, PhD

ATHENS

JULY 2023



ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάλυση αλγορίθμου AlphaFold 2

Βασιλική Λ. Πιτσιλού

Επιβλέποντες: **Ιωάννης Εμίρης**, Καθηγητής
Ανδρέας Ζαμάνος, MSc
Γεώργιος-Αλέξης Ιωαννάκης, PhD

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2023

BSc THESIS

Analysis of AlphaFold 2 algorithm

Vasiliki L. Pitsilou

S.N.: 1115201800160

Supervisors: **Ioannis Emiris**, Professor
Andreas Zamanos, MSc
Georgios-Alexis Ioannakis, PhD

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ανάλυση αλγορίθμου AlphaFold 2

Βασιλική Λ. Πιτσιλού

Α.Μ.: 1115201800160

Επιβλέποντες: **Ιωάννης Εμίρης**, Καθηγητής
Ανδρέας Ζαμάνος, MSc
Γεώργιος-Αλέξης Ιωαννάκης, PhD

ABSTRACT

AlphaFold 2 [1] has been a revolutionary algorithm in the field of computational biology. It made a huge leap in predicting the three-dimensional structure of a protein. Predicting the structure of a protein is a problem that has been challenging scientists for decades, and its solution is to date a very desired one, as protein structures play a crucial role in understanding the function of proteins. And subsequently, understanding the function of proteins plays a crucial role in understanding the functions of the human body and its diseases.

AlphaFold 2, an AI-model built by Google DeepMind [2], is able to accurately predict protein structures for a big number of proteins, far outperforming all other methods developed before it. But how did it manage such an impressive performance in such a difficult problem? In this thesis, we will present AlphaFold 2 algorithm after giving some context on proteins and the protein folding problem. We will also include an experimental part, where we execute the algorithm and examine some technical details.

SUBJECT AREA: computational biology

KEYWORDS: protein folding prediction, AlphaFold 2 algorithm, protein folding problem, machine learning, deep learning

ΠΕΡΙΛΗΨΗ

Ο αλγόριθμος AlphaFold 2 [1] υπήρξε ένας επαναστατικός αλγόριθμος στον τομέα της υπολογιστικής βιολογίας. Έκανε ένα τεράστιο άλμα στην πρόβλεψη της τρισδιάστατης δομής μιας πρωτεΐνης. Η πρόβλεψη αυτή αποτελεί ένα πρόβλημα το οποίο επιστήμονες προσπαθούν να λύσουν εδώ και δεκαετίες. Η λύση του είναι ιδιαίτερα σημαντική, μιας και η δομή μιας πρωτεΐνης παίζει πολύ σημαντικό ρόλο στην κατανόηση της λειτουργίας της. Ακολούθως, η κατανόηση λειτουργίας των πρωτεϊνών βοηθάει στην κατανόηση των λειτουργιών του ανθρώπινου σώματος και των ασθενειών του.

Το AlphaFold 2, αλγόριθμος τεχνητής νοημοσύνης που δημιουργήθηκε από την εταιρεία Google DeepMind [2], μπορεί να προβλέψει με υψηλή ακρίβεια την αναδίπλωση μια πρωτεΐνης, για έναν μεγάλο αριθμό πρωτεϊνών. Πως κατάφερε όμως να έχει τόσο καλή απόδοση σε ένα τόσο δύσκολο πρόβλημα; Στην παρούσα πτυχιακή εργασία θα παρουσιάσουμε τον αλγόριθμο, αφού πρώτα δώσουμε κάποιες πληροφορίες για τις πρωτεΐνες και το πρόβλημα αναδίπλωσης πρωτεϊνών. Θα παρουσιάσουμε επίσης ένα πειραματικό σκέλος, όπου θα εκτελέσουμε τον αλγόριθμο και θα εξετάσουμε κάποιες τεχνικές του λεπτομέρειες.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: υπολογιστική βιολογία

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: προσδιορισμός πρωτεϊνικής δομής, αλγόριθμος AlphaFold 2, πρόβλημα αναδίπλωσης πρωτεϊνών, μηχανική μάθηση, βαθιά μάθηση

Στους γονείς μου

ΕΥΧΑΡΙΣΤΙΕΣ

Για τη διεκπεραίωση της παρούσας Πτυχιακής Εργασίας θα ήθελα να ευχαριστήσω τους επιβλέποντες, Ιωάννη Εμίρη, καθηγητή, Γεώργιο-Αλέξη Ιωαννάκη, PhD, ερευνητή του Ε.Κ. ”Αθηνά” και Ανδρέα Ζαμάνο, MSc, ερευνητή του Ε.Κ. ”Αθηνά” για τη συνεργασία και την πολύτιμη συμβολή τους στην ολοκλήρωσή της.

CONTENTS

1. INTRODUCTION **15**

1.1 Proteins . . . 15

1.1.1 Protein creation . . . 15

1.1.2 Protein misfolding . . . 17

1.2 The problem . . . 18

1.3 Computational solution . . . 18

1.4 Neural networks . . . 18

1.5 CASP competition . . . 19

2. The AlphaFold 2 algorithm **21**

2.1 An overview . . . 21

2.2 Data . . . 21

2.3 Network inference . . . 22

2.3.1 First part of network . . . 23

2.3.1.1 Input embeddings . . . 24

2.3.1.2 Evoformer . . . 26

2.3.2 Second part of the network . . . 28

2.3.2.1 Structure module . . . 28

2.3.3 PDB file as an output of AlphaFold 2 . . . 31

2.4 Additional operations in training mode . . . 32

3. Running the algorithm **33**

3.1 Running locally . . . 33

3.1.1 Installing OpenFold and dependencies . . . 33

3.1.2 Downloading databases . . . 33

3.1.3 Running a simple example . . . 34

3.2 Running OpenFold on Google Colab . . . 36

4. Results **37**

5. Conclusions and future work **39**

ABBREVIATIONS - ACRONYMS **40**

LIST OF FIGURES

1.1. DNA 15
1.2. DNA to protein 16
1.3. Protein folding 17
1.4. Protein misfolding 17
1.5. Protein folding problem 18
1.6. Attention mechanism 19
1.7. All-atom RMSD in CASP14 competition 20

2.1. How sequence aligning gives us information about the structure 21
2.2. Multiple Sequence Alignment 22
2.3. Overview of AlphaFold 2 inference 23
2.4. MSA and pair representation 23
2.5. First MSA and pair representations 24
2.6. Injection of previous iterations information 24
2.7. Injection of templates information 25
2.8. Full first part of model 26
2.9. Row-wise self-attention with bias from pair representation 27
2.10. Column-wise self-attention 27
2.11. Pair representation update using MSA representation 27
2.12. Overview of Evoformer module 28
2.13. Structure module 29
2.14. Invariant Point Attention algorithm 30
2.15. Torsion angles phi,psi,omega 31
2.16. PDB file 32

3.1. FASTA file: MCHU Calmodulin 34
3.2. Protein structure of MCHU Calmodulin as returned by OpenFold 35
3.3. Output of running OpenFold on Google Colab 36

4.1. Alignment procedure by OpenFold 37

LIST OF TABLES

1.1. The 20 amino acids 16

LIST OF ALGORITHMS

1. Structure module: inference mode 29
2. BackboneUpdate 30
3. predictPerResidueLDDT_Ca: inference mode 31
4. predictPerResidueLDDT_Ca: training mode 32

PREFACE

This thesis was done as a part of the Bachelor of Science (B.Sc.) Program of the Department of Informatics and Telecommunications, School of Science, National and Kapodistrian University of Athens.

1. INTRODUCTION

1.1 Proteins

Proteins are macromolecules that take part in nearly every process in our cells. They are constructed inside the cell and perform a large number of different tasks like catalyzing chemical reactions (e.g. digestion), transport nutritional components throughout our body, attack foreign objects entering our body (antibodies), build cell and tissue structures, regulate processes in our body (protein hormones) and others. The building blocks of proteins are smaller molecules called **amino acids**. Among amino acids found in nature, around 20 are commonly found in proteins and are listed in Table 1.1. Each amino acid has unique chemical properties and contributions to protein structure and function. To form a protein, amino acids attach to each other forming a chain, which then folds to shape a 3-D structure (Figure 1.3). This structure allows a protein to perform the task it is responsible for.

1.1.1 Protein creation

Protein structure starts from the DNA. In our DNA, we have four types of nitrogenous bases: Adenine, Guanine, Cytosine and Thymine (A,G,C,T). The DNA looks like a double helix:

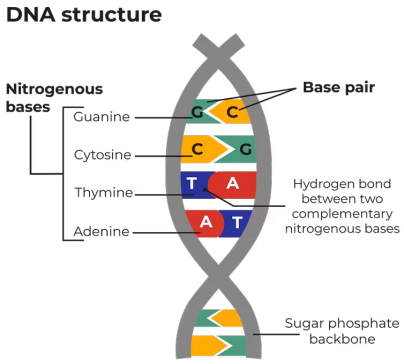


Figure 1.1: DNA. Figure taken from [3].

A gene, a segment of DNA consisting of a sequence of nitrogenous bases, leads to the production of proteins through a multi-step process. Initially, the gene's DNA sequence is transcribed into mRNA, serving as an intermediary. This mRNA molecule subsequently acts as a template for the translation of the genetic code, resulting in the synthesis of a specific sequence of amino acids (Figure 1.2). The precise arrangement of these amino acids within the protein sequence ultimately determines its three-dimensional structure (Figure 1.3).

Table 1.1: The 20 amino acids

Amino acid	Abbreviation	One-letter code
Alanine	Ala	A
Arginine	Arg	R
Asparagine	Asn	N
Aspartic acid	Asp	D
Cysteine	Cys	C
Glutamine	Gln	Q
Glutamic acid	Glu	E
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrosine	Tyr	Y
Valine	Val	V

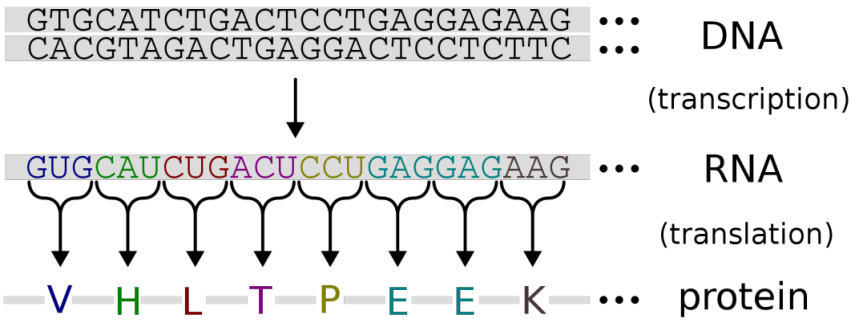


Figure 1.2: How a gene encodes a protein. Figure taken from [4].

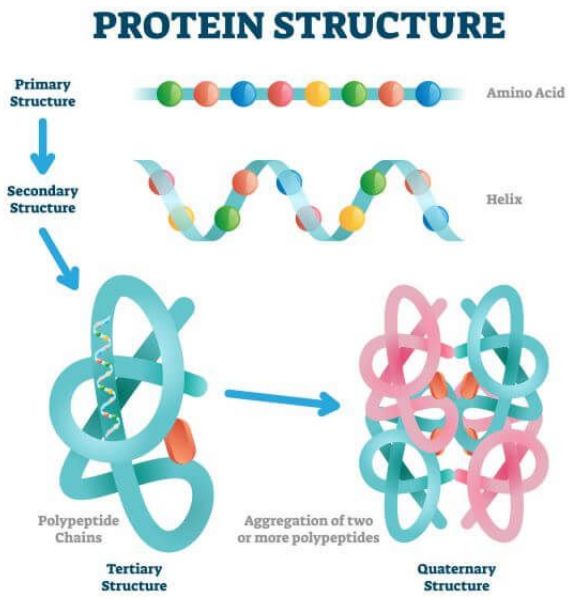


Figure 1.3: Protein folding. Figure taken from [5].

1.1.2 Protein misfolding

The proper folding of proteins into their native state is essential for their normal function. However, various factors can disrupt this process, leading to protein misfolding. Mutations in the genetic code, as well as environmental factors such as temperature or the presence of certain chemicals, can interfere with the folding process. When a protein misfolds, it means that it adopts an incorrect three-dimensional structure compared to its native state. This misfolded protein may become unstable, lose its functional properties and be unable to carry out its intended task effectively. The misfolding can disrupt the protein's ability to interact with other molecules, impair its enzymatic activity, or interfere with its structural role. Since proteins play crucial roles in numerous biological processes, the consequences of protein misfolding can be severe. Alzheimer's and Parkinson's diseases are two well-known examples of diseases associated with protein misfolding.

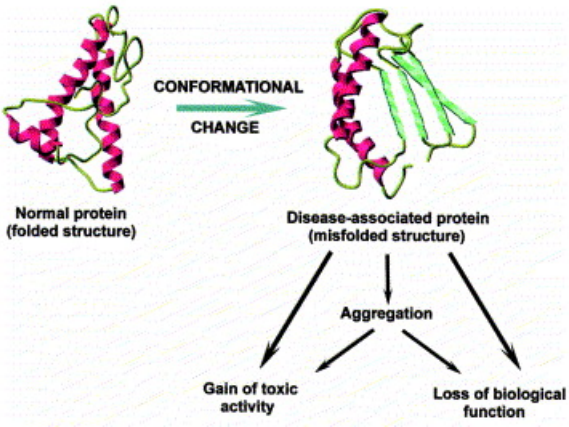


Figure 1.4: Protein misfolding. Figure taken from [6].

1.2 The problem

Knowing a protein's structure is very important for medical and biological research. The structure of nearly 100,000 proteins (compared to the billions of known sequences) has been found experimentally [7]. But determining a protein structure experimentally is not an easy task: it is expensive and time consuming, taking months to years to complete [7]. The scientists' goal is to be able to determine the structure of a protein in the three-dimensional space, given its amino acid sequence. If that is achieved, then they will be able to determine the structure of every protein. This is the so-called protein folding problem.

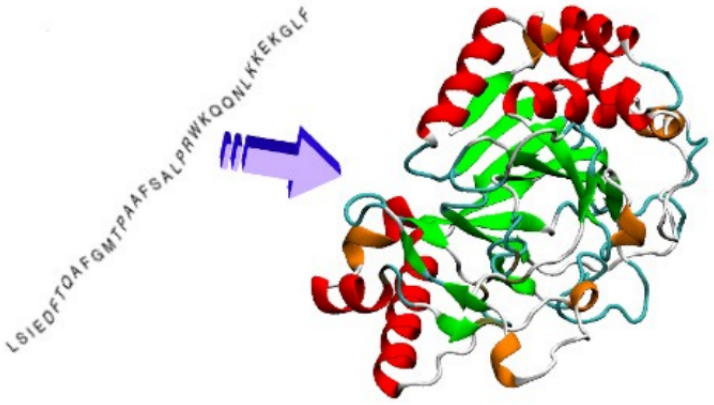


Figure 1.5: Protein folding problem. Figure taken from [8].

1.3 Computational solution

We know that a protein will fold so that its structure is most thermodynamically stable in the energy landscape of the intracellular environment [9]. So we could think of solving the problem by searching all the possible structures, to find the one that is most thermodynamically stable in the energy landscape. But this landscape contains an extremely big number of possible conformations for the protein (around 10^{300} for one protein), as estimated by Levinthal [10], making this problem impossible to solve by testing all structures possible, at least in less than time equal to the age of the known universe [11]. Still, proteins in nature fold spontaneously, in a matter of milliseconds. This is referred to as the Levinthal's paradox. So taking the brute force out of the picture as an impossible way of solving the problem, another computational method that can be used: supervised machine learning.

1.4 Neural networks

Neural networks are a famous subset of machine learning. They are widely being used in the last decade, forming a machine learning revolution, as they demonstrate an extremely good performance on certain tasks that could not be performed by computer programs before. Often referred to as artificial neural network, a neural network is inspired by the network of a biological brain. Its goal is to find underlying patterns and relationships in a set of data. It consists of artificial neurons, which are nodes that highly process information

and pass it to each other. Neurons are often organized into layers and a neural network with many layers is called a deep network.

A neural network is used after it has gone through a process called training. Training a neural network consists of showing it examples of the task it has to learn. For example, if we want to train a network to classify a text into a category between sports, fashion or politics, the input to the program will be examples of texts, which the neural network will use to build a statistical model that classifies a text in a category. Then, after training, the network is ready to classify a new text as referring to politics, fashion or sports, using that statistical model. The opposite of the training stage is the **inference** stage, where we put the trained model into operation. This means that the model simply outputs its prediction for the given input, without being trained. An additional procedure in neural networks is called fine-tuning. In fine-tuning of a neural network, the network is trained. Then, it goes through additional training on a specific dataset we want it to focus on.

There are various types of neural networks. The simplest type is called feed-forward neural network, where the output of a layer is the input to the next one. There are also other types including recurrent neural networks, convolutional neural networks, and **transformers**. Transformers, recently introduced by Google Brain in their famous paper Attention Is All You Need [12] led the way to language models such as ChatGPT [13]. Transformers are the type of neural networks used in AlphaFold 2, and are the key to the algorithm's success.

Transformers use a mechanism called attention. Attention is the technique of making the network focus only on some parts of the input that seem more useful than other input parts in predicting the output. That way, the network extracts more meaningful information from the input and outputs a highly accurate prediction. This technique is profoundly good in the field of Natural Language Processing (NLP). This figure demonstrates how attention works in translating a sentence: squares' brightness are proportional to the attention a word receives for translating each word of the sentence.

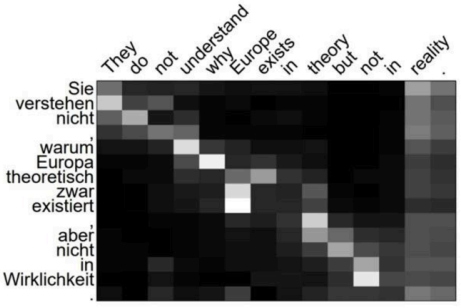


Figure 1.6: Attention mechanism. Figure taken from [14].

1.5 CASP competition

AlphaFold 2's success was demonstrated in the CASP14 competition. The CASP competition [15] is a biennial competition that calls scientific teams and communities to present their method on protein structure prediction. DeepMind took part in CASP14 (2020), where they presented AlphaFold 2. AlphaFold 2 is named after its predecessor, AlphaFold which took part in CASP13. Although a predecessor, AlphaFold was a different algorithm that did not perform so good as its second version. Still, AlphaFold was the best-performing method in CASP13. To understand AlphaFold 2's huge success in CASP14 competition,

we shall use the metric score $RMSD$ (root mean-square deviation). $RMSD$ is a metric showing how similar two protein structures are, based on their equivalent atoms distances. $RMSD$ is calculated by:

$$RMSD = \sqrt{\frac{1}{n}\sum_{i=1}^{n} d_i^2}$$

where n is the number of pairs of equivalent atoms and d_i is the distance between the i-th pair [16]. As we can see, a bigger $RMSD$-score implies greater distance between equivalent atoms in two structures, or bigger differences in the structure. In protein structure prediction, the ideal $RMSD$-score between the real structure and the predicted one would be zero (identical structures). Shown below is the median $RMSD$ for the top 15 entries in CASP14 (2020), measured in Å (1Å = 0.1 nm = 10^{-10} m). As we can see, AlphaFold 2 has an $RMSD_{95}$ score of 0.96 Å whereas the next best method has a 2.8 Å $RMSD_{95}$. The $RMSD$ was computed only for the C_α atoms of the structures (each amino acid has one C_α atom). AlphaFold 2's 95% confidence interval is 0.85 – 1.16 Å, meaning that 95% of the prediction scores will fall into that interval. The figure shows statistics for median data. *Median data* are samples of data which separate the first half data from the second half data, in a sorted dataset. This means that these are the middle scores among all scores, sorted. "GXXX" denotes the code of a group. AlphaFold 2's code is G427.

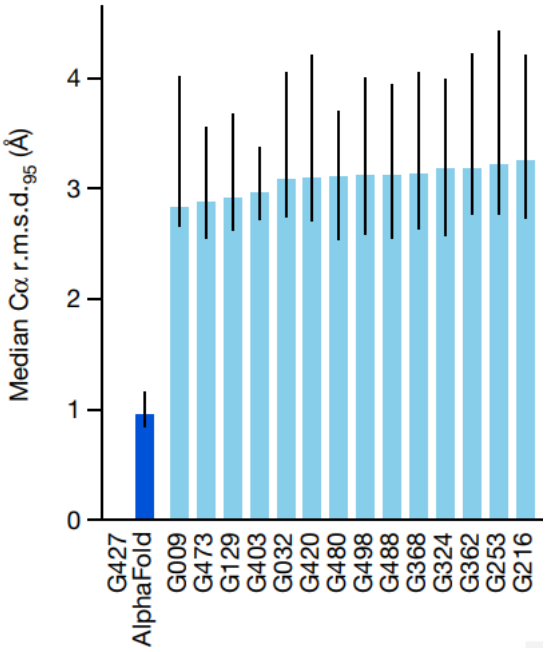


Figure 1.7: All-atom RMSD in CASP14 competition (top 15 entries out of 146). Data are median and the 95% confidence interval of the median, estimated from 10,000 bootstrap samples. Figure taken from [7].

2. THE ALPHAFOLD 2 ALGORITHM

2.1 An overview

AlphaFold 2 takes a protein amino acid sequence as input and outputs the most probable structure of that sequence. As an example, in Figure 1.6, the sequence would be the input to the program and the structure would be the output. The program's output is a structure for the protein in question and confidence scores (ranging from 0 to 100%) for each amino acid (note: amino acid is also referred to as **residue**). The program is split into two big parts: the **feature embedding and Evoformer** (first part) and the **Structure module** (second part). In the first part, information from other proteins is highly processed and used to build some structures. These structures will then be the input to the second part, which will use them to end up in a protein structure. The network uses recycling: the outputs of an iteration are used to update the input to the next one. The number of iterations is defined in the program.

2.2 Data

AlphaFold 2 widely uses neural networks, as discussed before. To sufficiently train those networks, AlphaFold 2 relies on a large amount of protein data. These data can be found by searching protein databases. The two main databases searched are the Protein Data Bank(PDB) [17] and the UniRef90 [18] database. The proteins searched are evolutionarily related to the protein in question, meaning they also share common amino acids. PDB provides structural data, while UniRef90 provides sequences belonging to the same protein family as the one in question. That means that PDB proteins have known structures, while the ones taken from UniRef90 not necessarily. Using information from proteins with known structures, the model can see how evolutionarily related proteins fold and that way extract information about the structure in question. Using information from proteins with unknown structures, it extracts information by aligning their sequences and finding patterns which reveal the proximity of two amino acids in the final protein shape.

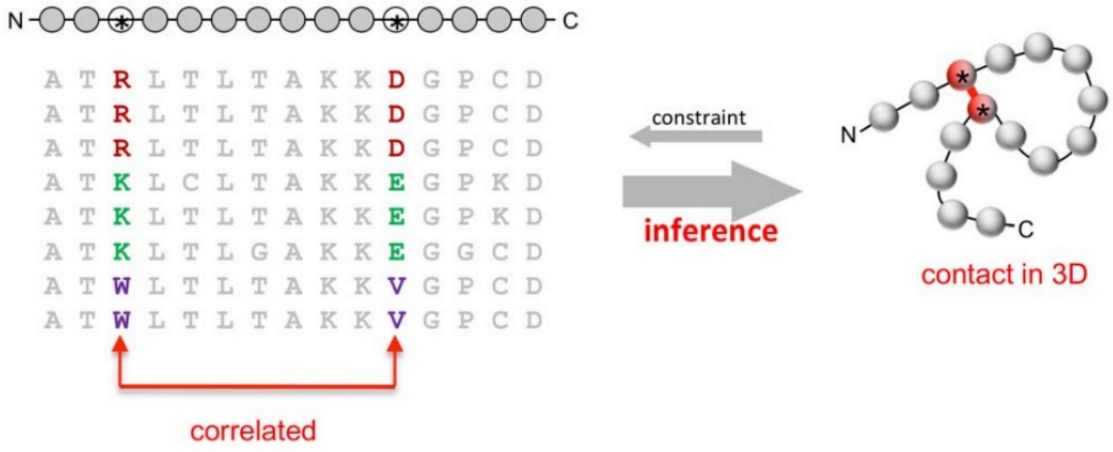


Figure 2.1: How sequence aligning gives us information about the structure. Figure taken from [19].

As shown in Figure 2.1, when mutations in two amino acids are correlated (a change in an

amino acid leads to the change of the other), we can conclude that those two amino acids are possibly close to each other in the 3-D structure. *Why?* Suppose a protein structure has two amino acids close to each other: A, negatively charged and B, positively charged. If the negatively charged amino acid A is substituted by a positively charged one as a result of a mutation, then the positively charged B should also mutate, otherwise the structure may be forced to change. But the structures tend to maintain their shape in order to preserve their functionality, so eventually amino acid B will mutate to become a negatively charged amino acid[20]. That way, when we observe such a correlation between two mutations, the corresponding amino acids *could* be close to each other in the structure (attention: they may also not be, since other factors may affect both of them).

In the first stages of the AlphaFold 2 algorithm, an MSA (Multiple Sequence Alignment) table of such proteins is made. This is an alignment of all amino acid sequences, from which we can extract knowledge for the protein structure in question. When aligning a group of proteins, we can see which amino acids are common and which of them have correlated mutations. An MSA table looks like this:



Figure 2.2: Multiple Sequence Alignment. Figure taken from [21].

2.3 Network inference

During inference, AlphaFold 2 receives input features from the input protein sequence, the MSA table and templates, and outputs atom coordinates and confidence score per amino acid. *Templates* are proteins with similar structures to the one we want to predict. After searching protein databases, the data are organized into an MSA table and a set of templates.

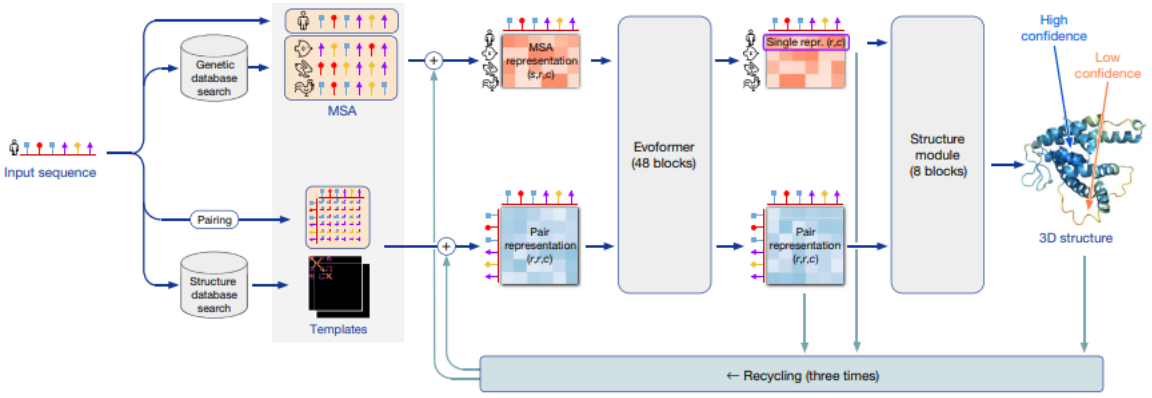


Figure 2.3: Overview of AlphaFold 2 inference. Figure taken from [7].

2.3.1 First part of network

Starting, we have an MSA table, which contains sequences of proteins evolutionarily related to the input protein. All these sequences are informative, but the memory cost required to use all of them is too high. To overcome this issue, some representatives are chosen among them. The non-representative sequences left are clustered: clustering is the process of distributing the protein sequences in groups. In each of those groups, or clusters, the central protein (referred to as a *centroid*) is a representative one, while all the others are non-representatives and similar to the central one. The information from non-representative sequences will not be thrown away: those will also be used at some point. Initially, a first version of the MSA and pair representation is created from an MSA sample. An MSA representation arithmetically encodes information of the MSA protein sequences. It is constantly changed and updated. A pair representation arithmetically represents the relationships between all pairs of amino acids of the protein in question: an $N \times N$ table, where N is the length of the input protein sequence (the number of its amino acids) and each cell ij contains information on the relationship between amino acids i and j in the final structure (e.g. their proximity).

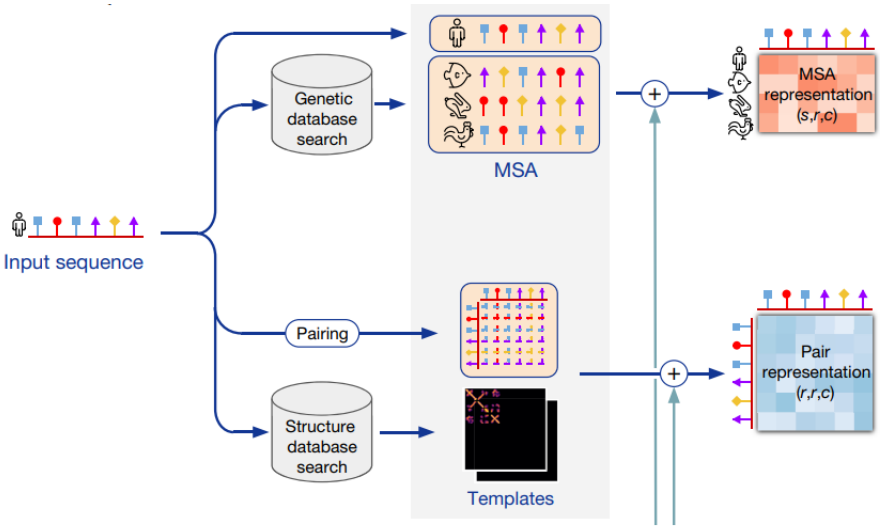


Figure 2.4: MSA and pair representation. Detail of figure taken from [7].

2.3.1.1 Input embeddings

By using information from the input sequence and the MSA centroids, we build the MSA representation and the pair representation. The input sequence and the MSA centroids are arithmetically represented. To build the MSA and pair representations, the input sequence representation (*target_feat* below) and the MSA centroid representations (*msa_feat* below) go through networks of linear layers (first use of neural networks in AlphaFold 2). By linear layers, we mean that linear transformations are applied to the input. The network is shown below, in the figure from the Supplementary paper of [7]. We can see that for the pair representation, we only need the input sequence, which is expected since the pair representation encodes information on the pairwise relationship between amino acids of the input protein. For the MSA representation, we need the MSA sequences (only the representative ones) as well as the input sequence. After that, the first versions of the MSA and pair representations are ready.

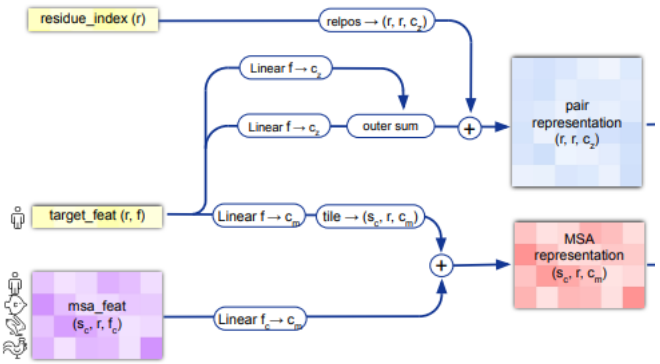


Figure 2.5: First MSA and pair representations. Detail of figure taken from [7].

We mentioned that the algorithm is executed multiple times (recycling). So in this stage, we want to inject information from the previous iteration: the final MSA and pair representations generated by the previous iteration, as well as the 3-D protein structure predicted by it. So, after having generated the first MSA and pair representations in the current iteration, we use the data generated by the previous one in order to update our representations. That way, we now have a pair and MSA representation which also contain encoded information from the previous iteration.

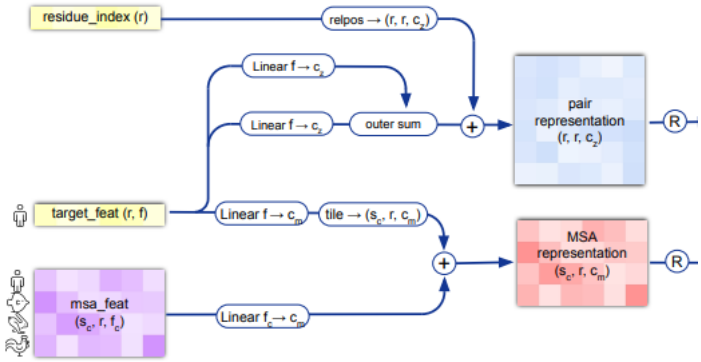


Figure 2.6: Injection of previous iterations information. R corresponds to recycling. Detail of figure taken from [7].

As mentioned, templates are known structures of proteins related to the input one. After

having searched and retrieved a set of templates for our input protein, we now have to make use of it. The set of templates is encoded in two structures: one encodes their angles (let us call it A) and the other their residues and their pairwise relationships (let us call it B). After passing structure A through a neural network, we concatenate it to the MSA representation. Now the MSA representation is updated and contains information from the templates. As for structure B, it will be used to update the pair representation. Knowledge on pair-wise relationships between residues in similar protein structures seems useful for determining the pairwise relationships between residues of the input protein. On figure below, A is the *template_angle_feat* while B is the *template_pair_feat*. This procedure embodies useful information from the templates in the MSA and pair representation in general.

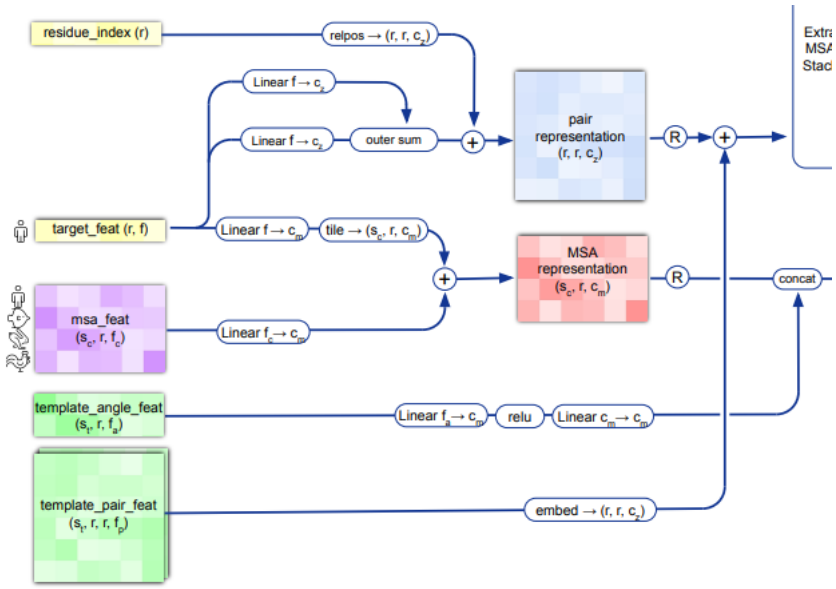


Figure 2.7: Injection of templates information. Detail of figure taken from [7].

At this stage, we need to extract information from the sequences we did not select as representatives and subsequently clustered. These sequences are represented by the structure called *extra MSA representation*. To use this information, we pass the extra MSA representation through an algorithm called the extra MSA stack. This algorithm has two inputs: the pair representation and the extra MSA representation. It outputs only an updated version of the pair representation. The pair representation is now highly processed using information from the extra MSA representation. This is the only time the extra sequences will be used. After that step, we now have the final MSA and pair representations which are ready to be given to the algorithm's main trunk: the Evoformer. This trunk will let information flow between MSA and pair representations, and output their final version. Then, from those representations, the algorithm will be ready to determine the protein's structure.

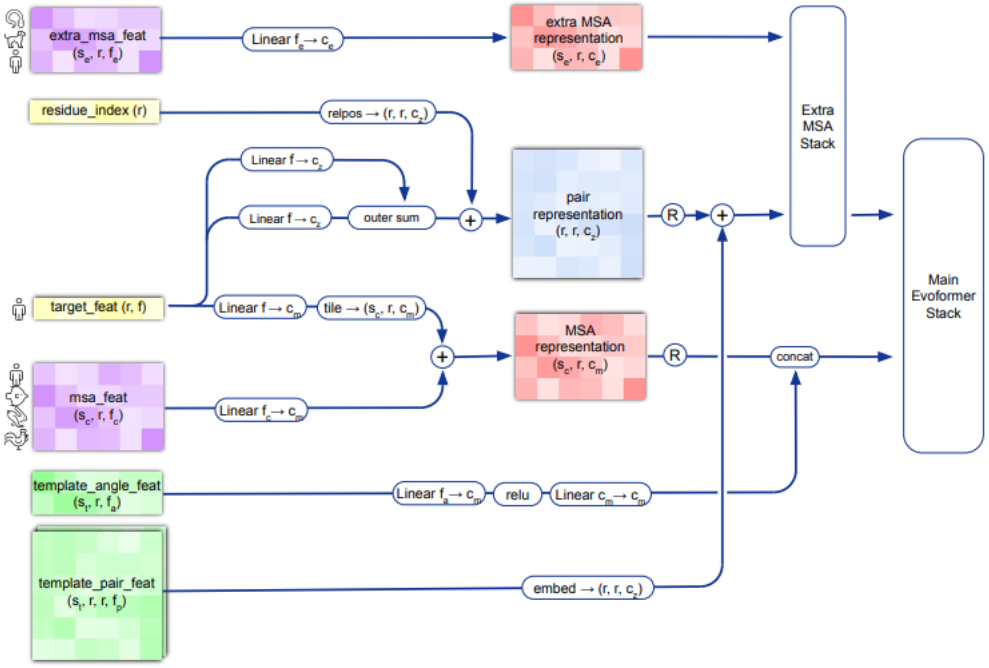


Figure 2.8: Full first part of model. Image taken from [7]

2.3.1.2 Evoformer

The Evoformer module focuses solely on MSA and pair representation and has a ”two-tower” architecture: one for each representation. What it does is to perform a bidirectional information transfer between the MSA and pair representations, in order to update both of them. This module uses attention mechanism (discussed in Chapter 1.4) in both MSA and pair representation update, and lets the two representations send information to each other, so that both can be updated using that information. The output consists of the processed and final MSA representation, final pair representation, and final single sequence representation (taken from MSA’s first row). This single representation is the amino acid sequence representation of the input protein. What happens in every iteration of the Evoformer is the following: first, we have row-wise self-attention of the MSA representation. That way, the network learns which residues are important for the structure determination (since it operates row-wise). This comes with a bias from the pair representation, allowing the pair representation to transfer its knowledge to the MSA representation. The result is an updated MSA representation.

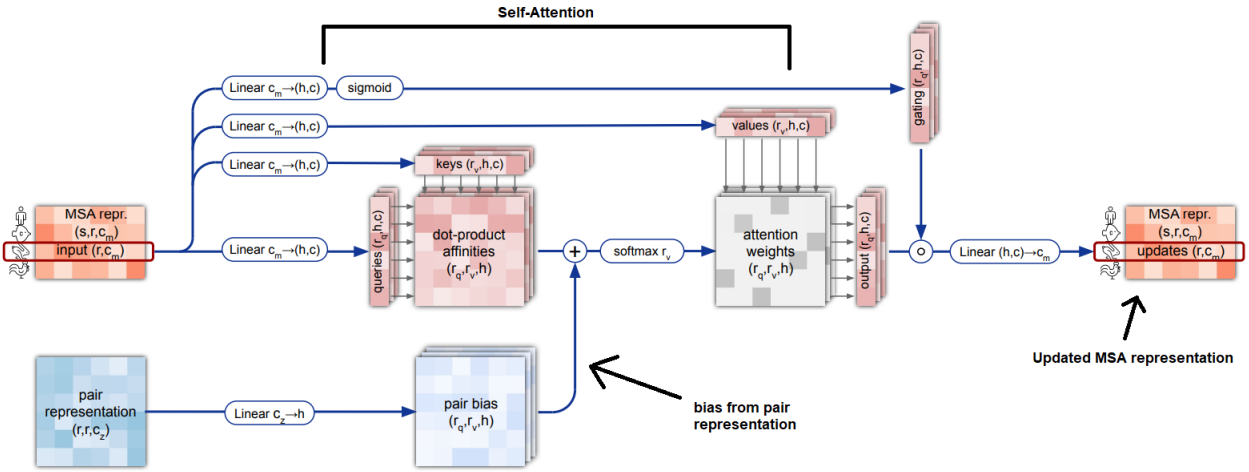


Figure 2.9: Row-wise self-attention with bias from pair representation. Supplementary Figure 2, [7] with some sketches on it.

After the MSA row-wise attention, we also need column-wise attention for the MSA representation, in order to establish which *sequences* could be of some usage to our structure determination, by giving useful information on possible mutations that have taken place. This results in an updated MSA representation. At this point, we did not include information from the pair representation, since as said, we are interested in which sequences are informative and not on the relationship between residues.

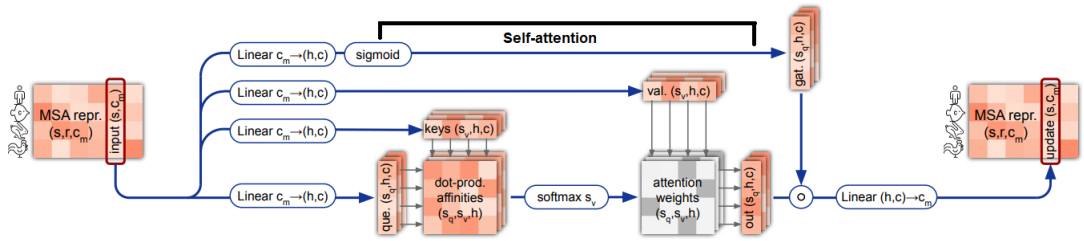


Figure 2.10: Column-wise self-attention. Supplementary Figure 3, [7] with some sketches on it.

So far, we went through attention operations in the MSA representation and let information flow from the pair representation to the MSA representation. What remains to be done is for the information to flow the other way around. So the pair representation is updated using MSA representation bias:

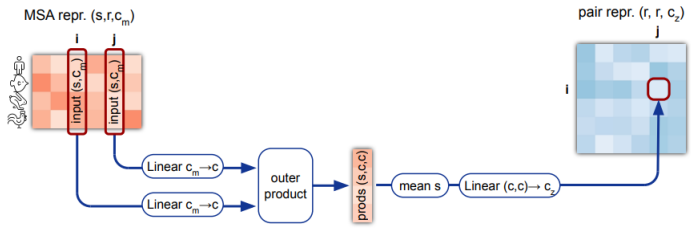


Figure 2.11: Pair representation update using MSA representation (Figure taken from [7]).

As we can see, in order to update the ij cell of the pair representation (which encodes the relationship between residues i,j in the target protein structure), we use information from the i-th and j-th columns of the MSA representation. And that way, we achieved

communication between both representations. Now the pair representation needs to go through some more extensive processing to make sure it is correct in terms of triangular inequality. Pair representation will also receive attention operations.

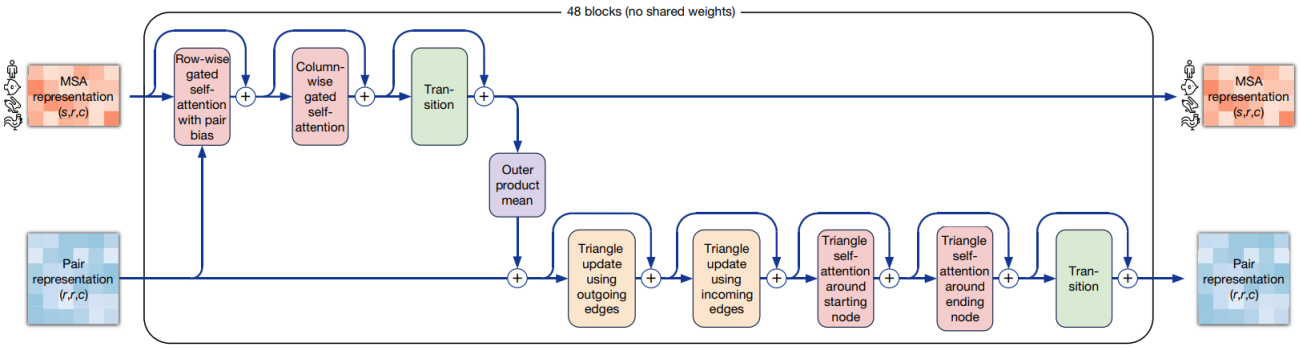


Figure 2.12: Overview of Evoformer module. Figure taken from [7].

The first stage (input embedding and Evoformer) is executed $N_ensemble$ times, each time the input is different samples from the MSA, and the stage's outputs (these are the Evoformer outputs) are averaged. This averaged output is then given as input to the second part of the network, the structure module. Ensembling (multiple executions of Evoformer) takes place only in inference and is optional, since it does not dramatically improve prediction scores.

2.3.2 Second part of the network

We have the final form of the MSA and pair representations. These are the tools that will be used to build a 3-D protein structure out of them. This is a job for the second part of the network, or the *Structure module*.

2.3.2.1 Structure module

The structure module takes two of the Evoformer's outputs as input: the pair representation and the single representation. As mentioned before, the single representation is taken from the MSA representation. The module consists of 8 layers. In each layer the single representation (given as input) and the 3D representation are updated. The 3D representation is initialized in the algorithm and is a representation of the protein 3-D structure. Its processed version will be the output of the entire algorithm. The 3-D representation is a set of frames (one frame per amino acid), each consisting of a rotation matrix and a translation vector. A frame is essentially a Euclidean transform $T_i = (R_i, \vec{t_i})$, which transfers a residue from a local to a global frame, meaning that it groups all amino acids together in a single structure. Updating the single representation is needed because this representation will give us information on the angles of the 3-D structure.

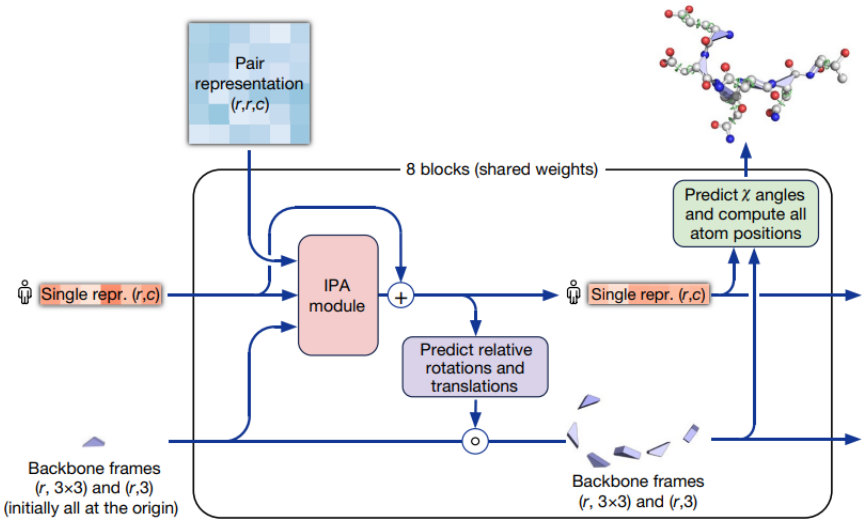


Figure 2.13: Structure module. Figure taken from [7].

The structure module algorithm is the following:

Algorithm 1 Structure module: inference mode

Input: $\{\mathbf{s}_i^{initial}\}, \{\mathbf{z}_{ij}\}, N_{layer} = 8, c = 128$ ▷ $\mathbf{s}_i^{initial} \in \mathbb{R}^{c_s}$

$\mathbf{s}_i^{initial} \leftarrow \text{LayerNorm}(\mathbf{s}_i^{initial})$

$\mathbf{z}_{ij} \leftarrow \text{LayerNorm}(\mathbf{z}_{ij})$

$\mathbf{s}_i = \text{Linear}(\mathbf{s}_i^{initial})$ ▷ $\mathbf{s}_i \in \mathbb{R}^{c_s}$

for all $l \in [1, ..., N_{layer}]$ **do**

  $T_i = (\mathbf{l}, \vec{\mathbf{0}})$ ▷ $\mathbf{l} \in \mathbb{R}^{3x3}$

  $\{\mathbf{s}_i\} += \text{InvariantPointAttention}(\{\mathbf{s}_i\}, \{\mathbf{z}_{ij}\}, \{T_i\})$

  $\mathbf{s}_i \leftarrow \text{LayerNorm}(\text{Dropout}_{0.1}(\mathbf{s}_i))$

  $\mathbf{s}_i \leftarrow \mathbf{s}_i + \text{Linear}(\text{relu}(\text{Linear}(\text{relu}(\text{Linear}(\mathbf{s}_i)))))$ ▷ all intermediate activations $\in \mathbb{R}^{c_s}$

  $\mathbf{s}_i \leftarrow \text{LayerNorm}(\text{Dropout}_{0.1}(\mathbf{s}_i))$

  $T_i \leftarrow T_i \circ \text{BackboneUpdate}(\mathbf{s}_i)$

  $\mathbf{a}_i = \text{Linear}(\mathbf{s}_i) + \text{Linear}(\mathbf{s}_i^{initial})$ ▷ $\mathbf{a}_i \in \mathbb{R}^c$

  $\mathbf{a}_i \leftarrow \mathbf{a}_i + \text{Linear}(\text{relu}(\text{Linear}(\text{relu}(\mathbf{a}_i))))$ ▷ all intermediate activations $\in \mathbb{R}^c$

  $\mathbf{a}_i \leftarrow \mathbf{a}_i + \text{Linear}(\text{relu}(\text{Linear}(\text{relu}(\mathbf{a}_i))))$ ▷ all intermediate activations $\in \mathbb{R}^c$

  $\vec{\alpha}_i^f = \text{Linear}(\text{relu}(\mathbf{a}_i))$ ▷ $\vec{\alpha}_i^f \in \mathbb{R}^2, f \in S_{torsion\,names}$

  $(R_i, \vec{\mathbf{t}}_i) = T_i$

  $\vec{\mathbf{x}}_i^{C\alpha} = \vec{\mathbf{t}}_i$

end for

$T_i^f, \vec{\mathbf{x}}_i^{\alpha} = \text{computeAllAtomCoordinates}(T_i, \vec{\alpha}_i^f)$ ▷ $\alpha \in S_{atom\,names}$

$T_i^f \leftarrow \text{concat}(T_i, T_i^f)$

$\{r_i^{pLDDT}\} = \text{predictPerResidueLDDT_Ca}(\{\mathbf{s}_i\})$

return $\{\vec{\mathbf{x}}_i^{\alpha}\}, \{r_i^{pLDDT}\}$

When starting this algorithm, every amino acid's frame is initialized to identity transformation i.e. $T_i = (I, \vec{0})$. We start by updating the so-far single representation (line 6). This is done by using the current pair representation and frames. The single representation is updated by an algorithm called Invariant Point Attention (IPA). This algorithm performs self-attention on the single representation, using also bias from the pair representation. The result is an update of the single representation.

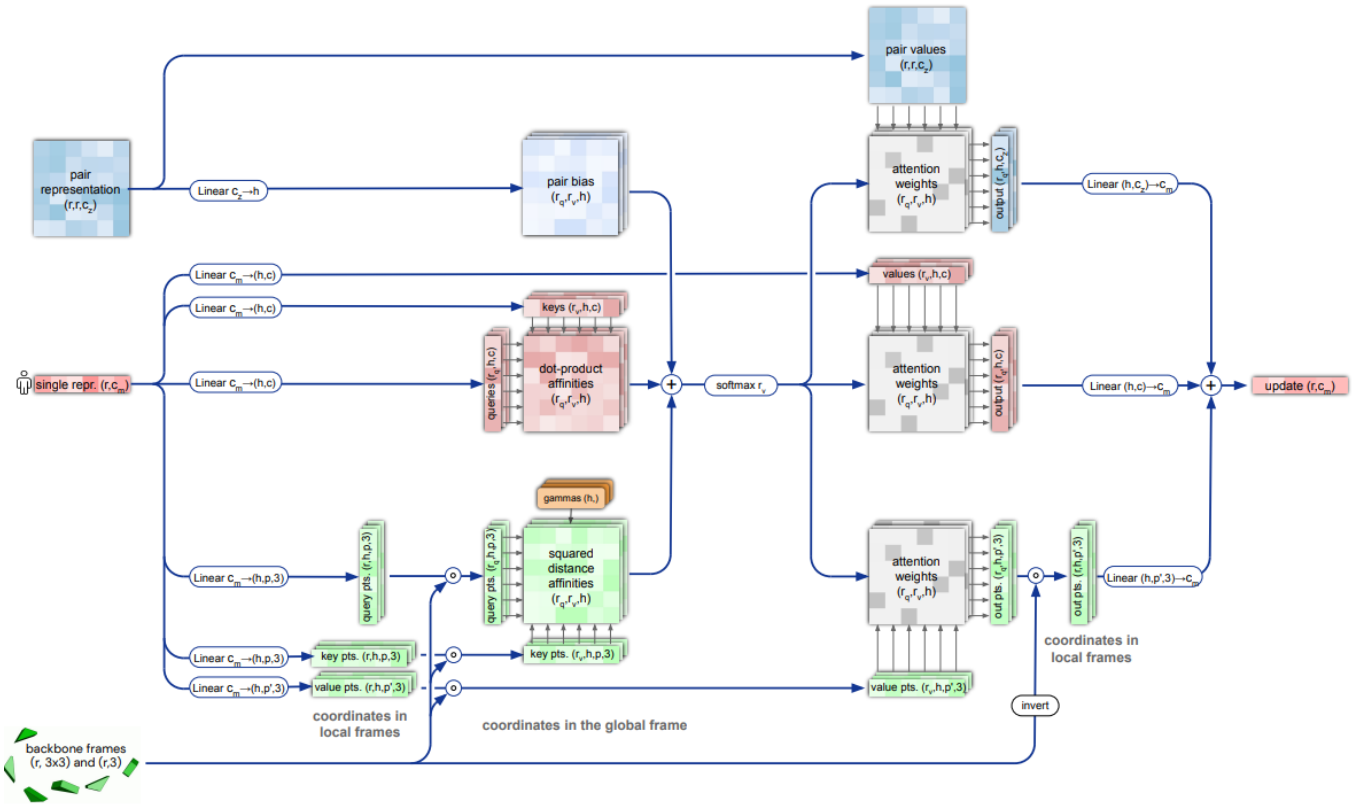


Figure 2.14: Invariant Point Attention algorithm. Figure taken from [7].

After running the IPA algorithm which has resulted in an updated single representation, we enter this single representation into a deep network (lines 7-9) and then use the output of the network (a reprocessed single representation) to update the frames with an algorithm called *BackboneUpdate*(line 10):

Algorithm 2 BackboneUpdate

Input: $\mathbf{s}_i$

1: $(b_i, c_i, d_i, \vec{\mathbf{t}}_i) = \text{Linear}(s_i)$ ▷ $b_i, c_i, d_i \in \mathbb{R}, \vec{\mathbf{t}}_i \in \mathbb{R}^3$

2: $(a_i, b_i, c_i, d_i) \leftarrow (1, b_i, c_i, d_i)/\sqrt{1 + b_i^2 + c_i^2 + d_i^2}$

3: $R_i = \begin{pmatrix} a_i^2 + b_i^2 - c_i^2 - d_i^2 & 2b_ic_i - 2a_id_i & 2b_id_i + 2a_ic_i \\ 2b_ic_i + 2a_id_i & a_i^2 - b_i^2 + c_i^2 - d_i^2 & 2c_id_i - 2a_ib_i \\ 2b_id_i - 2a_ic_i & 2c_id_i + 2a_ib_i & a_i^2 - b_i^2 - c_i^2 + d_i^2 \end{pmatrix}$

4: $T_i = (R_i, \vec{\mathbf{t}}_i)$

5: **return** T_i

The *BackboneUpdate* algorithm uses a neural network that outputs values that will update the frames. Also, we will use the updated single representation along with its non-updated version (the single representation before going through the IPA algorithm), in order to predict the torsion angles of the 3-D protein structure. Torsion angles in a protein structure are angles between planes formed by atoms of one or two amino acids.

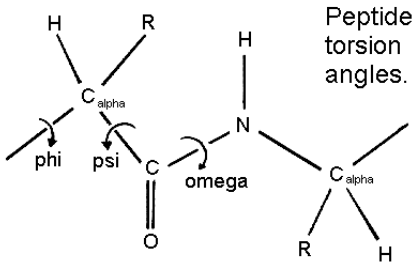


Figure 2.15: The three main torsion angles of an amino acid: phi, psi and $omega$. Figure taken from [22].

They will be later used to predict the exact atoms' coordinates for all amino acids of the protein. Both single representations are used as input to a deep network, which will output all torsion angles for each residue (lines 11-14). Then, using the predicted frames and torsion angles, we run the algorithm that will give us the final frames and atom coordinates (line 18). The final 3-D structure has been determined, but the structure confidence remains to be computed. As mentioned, the model is trained not only to predict a protein structure, but also on predicting its confidence. In inference mode, the confidence is predicted by simply inputting the single representation into a deep network, that will output the per-residue confidence. This is done in algorithm *predictPerResidueLDDT_Ca*, line 20:

Algorithm 3 predictPerResidueLDDT_Ca: inference mode

Input: $\{\mathbf{s}_i\}, \mathbf{v}_{bins} = [1, 3, 5, ..., 99]^\top, c = 128$

1: $\mathbf{a}_i = \text{relu}(\text{Linear}(\text{relu}(\text{Linear}(\text{LayerNorm}(\mathbf{s}_i)))))$ ▷ $\mathbf{a}_i$, intermediate activations $\in \mathbb{R}^c$

2: $\mathbf{p}_i^{pLDDT} = \text{softmax}(\text{Linear}(\mathbf{a}_i))$ ▷ $\mathbf{p}_i^{pLDDT} \in \mathbb{R}^{|\mathbf{v}_{bins}|}$

3: $\mathbf{r}_i^{pLDDT} = \mathbf{p}_i^{pLDDT^\top} \mathbf{v}_{bins}$ ▷ $\mathbf{r}_i^{pLDDT} \in \mathbb{R}$

4: **return** $\{\mathbf{r}_i^{pLDDT}\}$

Returned by the structure module are the atom coordinates and the confidence measure. Lastly, relaxation is performed on the predicted structure. When a molecular structure is predicted by a computer program, it may contain structural violations, meaning violations that do not appear in natural structures. So the structure undergoes a procedure called relaxation in order to resolve the most violations possible. Both relaxed and unrelaxed versions of the protein structure are returned by AlphaFold 2. Structure module's outputs (coordinates and confidence measure) will be organized in a PDB file.

2.3.3 PDB file as an output of AlphaFold 2

A PDB file [23] is a file that represents a molecule structure, giving coordinates for all its atoms. An instance of a PDB file is shown in Figure 2.16. In a PDB file returned by AlphaFold 2, each row represents an atom. The fourth column denotes the amino acid type that the atom belongs to (amino acid types are listed in table 1.1). Columns 7,8,9 are the atom's x,y and z coordinates respectively. The eleventh column is the confidence measure of the amino acid the atom belongs to (0-100%). Finally, the twelfth and last column denotes the atom type: carbon (C), hydrogen (H), oxygen (O), nitrogen (N) or sulfur (S). Those are the atom types that make up all 20 amino acids.

```
ATOM     79  H   THR A   6      -8.853  11.980  -0.900  1.00 62.67           H
ATOM     80  CA  THR A   6      -7.938  12.396   0.982  1.00 62.67           C
ATOM     81  HA  THR A   6      -6.920  12.777   1.058  1.00 62.67           H
ATOM     82  C   THR A   6      -8.163  11.388   2.116  1.00 62.67           C
ATOM     83  CB  THR A   6      -8.933  13.562   1.082  1.00 62.67           C
ATOM     84  HB  THR A   6      -8.804  14.054   2.046  1.00 62.67           H
ATOM     85  O   THR A   6      -8.848  10.375   1.935  1.00 62.67           O
ATOM     86  CG2 THR A   6      -8.735  14.593  -0.033  1.00 62.67           C
ATOM     87 HG21 THR A   6      -8.976  14.154  -1.001  1.00 62.67           H
ATOM     88 HG22 THR A   6      -7.698  14.927  -0.039  1.00 62.67           H
ATOM     89 HG23 THR A   6      -9.386  15.449   0.144  1.00 62.67           H
ATOM     90  OG1 THR A   6     -10.254  13.083   0.982  1.00 62.67           O
ATOM     91  HG1 THR A   6     -10.864  13.809   1.137  1.00 62.67           H
ATOM     92  N   GLU A   7      -7.629  11.649   3.310  1.00 63.67           N
ATOM     93  H   GLU A   7      -6.964  12.402   3.412  1.00 63.67           H
ATOM     94  CA  GLU A   7      -7.925  10.819   4.490  1.00 63.67           C
ATOM     95  HA  GLU A   7      -7.422   9.856   4.397  1.00 63.67           H
ATOM     96  C   GLU A   7      -9.434  10.556   4.646  1.00 63.67           C
ATOM     97  CB  GLU A   7      -7.440  11.525   5.759  1.00 63.67           C
ATOM     98  HB2 GLU A   7      -7.803  10.962   6.619  1.00 63.67           H
ATOM     99  HB3 GLU A   7      -7.881  12.521   5.789  1.00 63.67           H
ATOM    100  O   GLU A   7      -9.832   9.479   5.081  1.00 63.67           O
ATOM    101  CG  GLU A   7      -5.913  11.640   5.873  1.00 63.67           C
ATOM    102  HG2 GLU A   7      -5.486  10.637   5.848  1.00 63.67           H
ATOM    103  HG3 GLU A   7      -5.536  12.199   5.017  1.00 63.67           H
ATOM    104  CD  GLU A   7      -5.485  12.352   7.170  1.00 63.67           C
ATOM    105  OE1 GLU A   7      -4.263  12.420   7.421  1.00 63.67           O
ATOM    106  OE2 GLU A   7      -6.373  12.845   7.907  1.00 63.67           O
ATOM    107  N   GLU A   8     -10.270  11.508   4.216  1.00 74.00           N
```

Figure 2.16: PDB file

2.4 Additional operations in training mode

In training mode some additional operations take place, as expected. In training mode, data from known protein structures are used in order to train the neural networks of the algorithm. Since the training mode has the goal of training the neural networks present in the algorithm, some losses are computed, which will help refine those networks. Those losses take into consideration the true coordinates and frames of a structure and compare them to the predicted ones. The model confidence is computed using the distance between the real and predicted structure, rather than being predicted by a neural network as done in algorithm 3. The distance metric used to compute this distance is the $lDDT$ metric [24]. Below is the *predictPerResidueLDDT_Ca* algorithm used in training mode.

Algorithm 4 predictPerResidueLDDT_Ca: training mode

Input: $\{\mathbf{s}_i\}, \mathbf{v}_{bins} = [1, 3, 5, ..., 99]^\top, \{r_i^{trueLDDT}\}, c = 128$

1: $\mathbf{a}_i = \text{relu}(\text{Linear}(\text{relu}(\text{Linear}(\text{LayerNorm}(\mathbf{s}_i)))))$ ▷ $\mathbf{a}_i$, intermediate activations $\in \mathbb{R}^c$
2: $\mathbf{p}_i^{pLDDT} = \text{softmax}(\text{Linear}(\mathbf{a}_i))$
3: $\mathbf{p}_i^{trueLDDT} = \text{one_hot}(r_i^{trueLDDT}, \mathbf{v}_{bins})$
4: $L_{conf} = \text{mean}_i(\mathbf{p}_i^{trueLDDT^\top} \log \mathbf{p}_i^{pLDDT})$ ▷ $\mathbf{p}_i^{pLDDT} \in \mathbb{R}^{|\mathbf{v}_{bins}|}$
5: $\mathbf{r}_i^{pLDDT} = \mathbf{p}_i^{pLDDT^\top} \mathbf{v}_{bins}$ ▷ $\mathbf{r}_i^{pLDDT} \in \mathbb{R}$
6: **return** $\{\mathbf{r}_i^{pLDDT}\}, L_{conf}$

Lines 3 and 4 are additional and not used in inference mode. As we can see, in these lines the loss between the real and the predicted confidence is computed and returned by the algorithm. This loss helps refine the network that predicts the model confidence in inference mode.

During training, AlphaFold 2 outputs not only the structure and the confidence measure, but also the loss metrics.

3. RUNNING THE ALGORITHM

AlphaFold 2 is an open-source software available on GitHub [25], with detailed instructions on how to run it. AlphaFold 2 can only be run in inference mode and is not available for training or fine-tuning. Running it requires a Linux-running machine, around 3 TB space for the full download of protein databases and a powerful GPU. For systems with less memory, there is also the option of downloading a reduced version of databases (a database of 17GB will be downloaded instead of the full one of 1.8 TB, still leaving the size needs to around 1.2 TB). AlphaFold 2 needs a significant time to set up: many hours (even a day or more) are required to download the data needed for running the algorithm.

OpenFold [26] is a faithful reproduction of AlphaFold 2 implemented in PyTorch [27], whereas AlphaFold 2 was developed for JAX workflow [28]. OpenFold does give the option for training or fine-tuning and can be faster in inference mode [26]. Still, its setup requires the same resources mentioned for AlphaFold 2 (multiple hours of downloading databases as well as storage). In an attempt to run the AlphaFold 2 algorithm, we chose to run OpenFold, since it would allow us to also experiment with some fine-tuning.

3.1 Running locally

The following is a procedure of predicting a protein structure using OpenFold. The whole process is done in Linux environment, using the Command Line Interface (terminal).

3.1.1 Installing OpenFold and dependencies

The first step in installing OpenFold is to download the source code stored in the GitHub repository [26] by running

```
$ git clone https://github.com/aqlaboratory/openfold.git
```

After successfully running the command, a directory called ”openfold” will appear in the directory where the download command was run. All the following commands will have to be run in the ”openfold” directory.

The command that follows and downloads all dependencies is the following:

```
$ scripts/install_third_party_dependencies.sh
```

This command will need a few minutes to complete.

3.1.2 Downloading databases

The protein databases will need to be downloaded by running:

```
$ bash scripts/download_alphafold_dbs.sh data/
```

This command will take a long time to complete. In our case, it took nearly a day, but that depends on internet speed and other factors.

After the downloads are done, OpenFold is ready to be run.

3.1.3 Running a simple example

You will need to create a file with the extension *.fasta*. This is called a FASTA file and it includes an amino acid sequence and its name. Shown below is a FASTA file named *example.fasta*, which will include the amino acid sequence of the MCHU Calmodulin. MCHU Calmodulin is a protein that acts as an intermediary protein sensing calcium levels and relaying signals to various calcium-sensitive enzymes, ion channels and other proteins [29]. It is found in humans, rabbits, bovines, rats and chickens. The file will look like this:

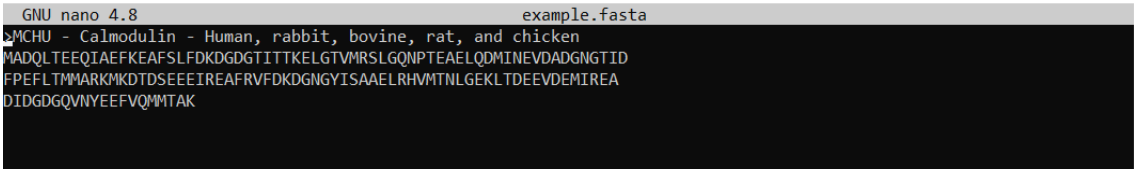


Figure 3.1: FASTA file of MCHU Calmodulin

This file will need to be stored in a directory named *fasta_dir*, which will be a subfolder of the *openfold* directory.

A PDB file that contains the 3-D structure of the MCHU Calmodulin protein is the expected output of the program.

The *example.fasta* file will be the input of the program. The command following is the command for the OpenFold inference using Docker. Docker [30] is a software that allows a user to perform any installations and runs in a secluded environment, called a container. That way, the program will not have to interfere with the whole system where the program will be run and allows independence, especially in systems that many people use. Docker is a simple way of running OpenFold.

In *openfold* folder, you will need to run:

```
$ docker build -t openfold .
```

in order to prepare the docker environment.

Run also

```
$ docker run --rm --gpus all nvidia/cuda:11.0-base nvidia-smi
```

in order to establish communication between the Docker installation and the GPU.

After preparing the Docker environment, the inference command can be run:

```
$ docker run --gpus device=0
-v $PWD/:/data -v /mnt/hdd2/vpitsilou/openfold/data/:/database
-ti openfold:latest python3 /opt/openfold/run_pretrained_openfold.py
/data/fasta_dir
/database/pdb_mmcif/mmcif_files/
--uniref90_database_path /database/uniref90/uniref90.fasta
--mgnify_database_path /database/mgnify/mgy_clusters_2018_12.fa
--pdb70_database_path /database/pdb70/pdb70
--uniclust30_database_path
```

```
/database/uniclust30/uniclust30_2018_08/uniclust30_2018_08
--output_dir /data
--bfd_database_path
/database/bfd/bfd_metaclust_clu_complete_id30_c90_final_seq.sorted_opt
--model_device cuda:0
--jackhmmer_binary_path /opt/conda/bin/jackhmmer
--hhblits_binary_path /opt/conda/bin/hhblits
--hhsearch_binary_path /opt/conda/bin/hhsearch
--kalign_binary_path /opt/conda/bin/kalign
--config_preset "model_2_ptm"
--openfold_checkpoint_path /database/openfold_params/finetuning_ptm_2.pt
```

In *device=0* you have to enter the GPU that will be used. In the system with two GPUs, you can enter *device=0*, *device=1* or *all*. Using all GPUs available is preferable for faster inference. Next, Docker creates volumes: the current directory (PWD) will now be called *data* for the whole command, while the directory */mnt/hdd2/vpitsilou/openfold/data/* will be called *database*. The *database* file is the directory where all protein databases were downloaded. The *fasta_dir* directory must contain the *example.fasta* file. All other arguments refer to databases downloaded. One can choose between many models in *config_preset* and *openfold_checkpoint_path* argument. As you will see, in the directory *data/openfold_params*, there are many differently trained models to choose from.

For this particular protein, the running time will be some minutes, depending on the running environment. Running it on an 8GB NVIDIA GeForce RTX 3060 Ti GPU, it took nearly 15 minutes. When the command is completed, a directory called *predictions* will appear, containing pdb files: some named *relaxed* and some *unrelaxed*. Those two files correspond to the original structure returned as well as the one relaxed with AMBER relaxation [7].

A PDB file can then be used in a visualizing software like Chimera [31]. Such softwares visualize a structure as encoded in a PDB file. The PDB file returned by OpenFold for the MCHU Calmodulin protein can be visualized into the following 3-D structure, using Chimera 1.16:

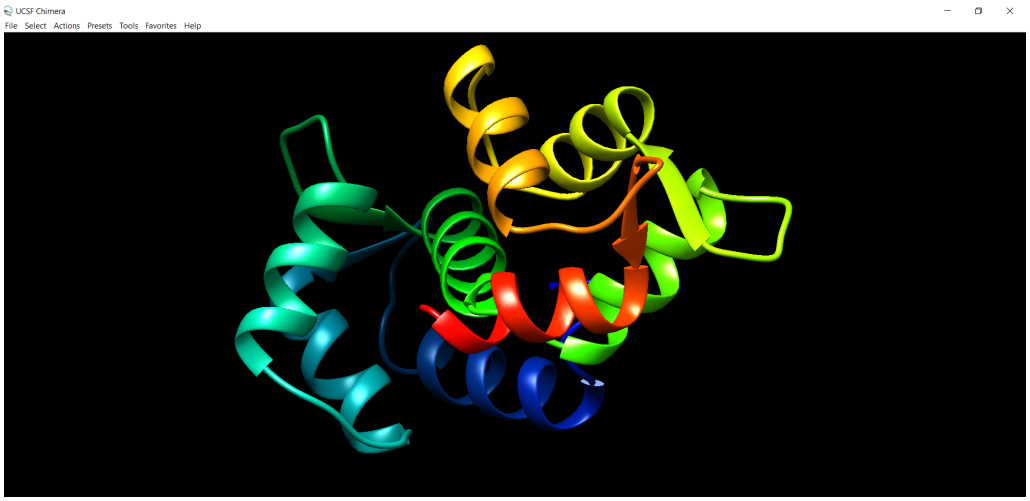


Figure 3.2: Protein structure of MCHU Calmodulin as returned by OpenFold

The program was run on Linux Ubuntu 20.04, using AMD Ryzen Threadripper 3960X 24-Core Processor and an 8 GB NVIDIA GeForce RTX 3060 Ti GPU.

3.2 Running OpenFold on Google Colab

OpenFold offers the possibility of running the program in Google Colab [32]. Google Colab is an environment where Jupyter notebooks can be run without the use of setting up GPU or CPU, since they are provided by Google. This is a pretty simple way of running OpenFold without having to download and run it manually. To run OpenFold on Google Colab, simply enter the amino acid sequence on the first cell, choose the model you want to use (OpenFold and AlphaFold 2 have slight differences) and then run all the cells. This program downloads all the databases on the Google cloud environment. The running time for an amino acid sequence of 70 amino acids was 40 minutes, including all databases downloads. After being completed, the program downloads a zip file containing the PDB file and the predicted error in a JSON file. It also visualizes the 3D structure along with its confidence prediction. In the 3D structure, the colouring denotes the confidence areas:

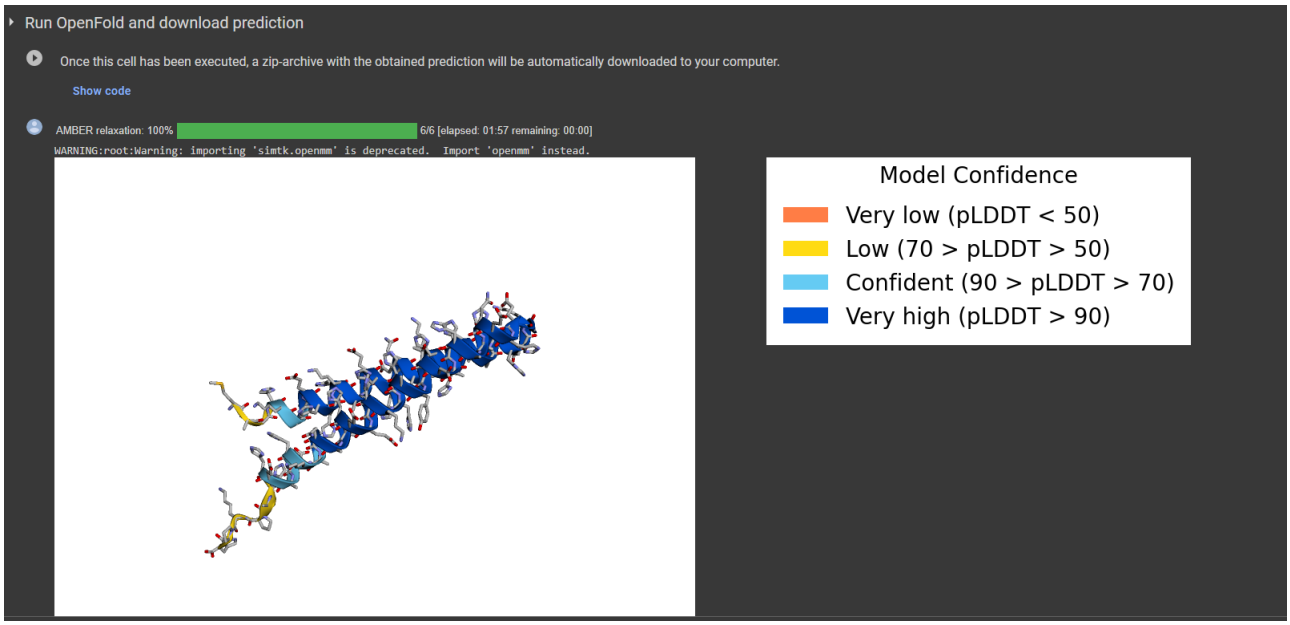


Figure 3.3: Output of running OpenFold on Google Colab

4. RESULTS

We saw how AlphaFold 2 can be run using OpenFold. All we used was a simple computer system running Linux and an 8 GB GPU. Even though we needed a large space of around 3 TB (which can also be reduced to 1.2 TB as mentioned in Chapter 3), it is still impressive how anyone can obtain a protein structure estimation, corresponding to an amino acid sequence of their choice within minutes, simply by running a program on their computer or on Google Colab. Until some years ago, a protein structure could only be obtained experimentally by scientists, after months of effort. This demonstrates the power of computer science, computer systems and machine learning.

In the previous chapter, predictions were obtained using trained models. But taking a step further, what would happen if someone wanted to fine-tune the model, or even train it from scratch, needing even higher resources? AlphaFold 2 does not offer the option of training or fine-tuning, probably because they require resources available only within scientific communities. But OpenFold remains an option for fine-tuning or training the model.

In order to fine-tune the model on a selected group of proteins, those proteins first need to be aligned. Alignment will be used for the construction of the MSA (Multiple Sequence Alignment) table discussed before. In an alignment procedure, all proteins are aligned based on a model protein, in order to find common amino acids with it. Since all proteins have to take the role of the model protein (all proteins have to be compared with all others), the alignment procedure leads to a high complexity. OpenFold offers two alignment scripts:

Training

To train the model, you will first need to precompute protein alignments.

You have two options. You can use the same procedure DeepMind used by running the following:

```
python3 scripts/precompute_alignments.py mmcif_dir/ alignment_dir/ \
    --uniref90_database_path data/uniref90/uniref90.fasta \
    --mgnify_database_path data/mgnify/mgy_clusters_2018_12.fa \
    --pdb70_database_path data/pdb70/pdb70 \
    --uniclust30_database_path data/uniclust30/uniclust30_2018_08/uniclust30_2018_08 \
    --bfd_database_path data/bfd/bfd_metaclust_clu_complete_id30_c90_final_seq.sorted_opt \
    --cpus_per_task 16 \
    --jackhmmer_binary_path lib/conda/envs/openfold_venv/bin/jackhmmer \
    --hhblits_binary_path lib/conda/envs/openfold_venv/bin/hhblits \
    --hhsearch_binary_path lib/conda/envs/openfold_venv/bin/hhsearch \
    --kalign_binary_path lib/conda/envs/openfold_venv/bin/kalign
```

As noted before, you can skip the `binary_path` arguments if these binaries are at `/usr/bin`. Expect this step to take a very long time, even for small numbers of proteins.

Alternatively, you can generate MSAs with the ColabFold pipeline (and templates with HHsearch) with:

```
python3 scripts/precompute_alignments_mmseqs.py input.fasta \
    data/mmseqs_dbs \
    uniref30_2103_db \
    alignment_dir \
    ~/MMseqs2/build/bin/mmseqs \
    /usr/bin/hhsearch \
    --env_db colabfold_envdb_202108_db
    --pdb70 data/pdb70/pdb70
```

Figure 4.1: Alignment procedure by OpenFold [26]

The problem is that this step is not that simple. After trying to align a set of 900 proteins using both alignment scripts, the result was the following:

- For the former, alignment of approximately one protein per day (which would result in 900 days of aligning).
- For the latter, usage of **90% of computer memory** on average. This amount of memory consumption prevents others from using the same computer machine. Alignment running for 3 days consecutively, was killed manually afterwards. This method is faster than the previous one.

These data show us that is infeasible to run the alignment script in a short amount of time, but one can overcome this obstacle if having precomputed alignments available. In that case, alignment of proteins is not needed. Besides alignment, we also expect fine-tuning to take a long time (at least a few days).

The amount of time needed for aligning prevented us from fine-tuning the model. Fine-tuning is a much harder task, although not infeasible, given that one can invest time and effort in it. But if alignment could become more efficient, many scientific teams or individuals would experiment with training or fine-tuning, leading to a better understanding of the algorithm and even improvement of it. At this point, we could not expect the same thing for training or fine-tuning, since solving such a problem using machine learning is by its nature resource-intensive and training this model using such a large amount of data could not be faster. AlphaFold 2, even with its weaknesses in terms of resources or time needed to train or fine-tune it, is still a very sophisticated algorithm, which using state-of-the-art techniques has managed to solve a very important biological problem, at least to a certain extent. But as science by its nature aims at solving problems (even after having just solved a very important one), scientists will try to improve the algorithm efficiency and more importantly, obtain even more accurate protein structures.

5. CONCLUSIONS AND FUTURE WORK

AlphaFold 2 has been able to predict protein structures given their amino acid sequence, with a high accuracy for the majority of proteins. The algorithm has been considered revolutionary and is indeed remarkable, since it has achieved significant advancements in predicting protein structures and thus managed to solve the protein folding problem to a large degree. That way, it expanded our knowledge on previously unknown protein structures and the molecular world and showed how machine learning algorithms can be reliable and powerful tools for solving biological problems. It has also given the scientific world new tools and ideas on how to tackle specific problems using machine learning techniques, opening up new possibilities for scientific research. However, we should keep in mind that machine learning algorithms make predictions based on data they have been trained on. So one should be careful when using a predicted protein structure and not consider it as ground truth, but also take into consideration the confidence scores provided with every structure. Those confidence scores indicate, as mentioned, the model's certainty for the prediction of each amino acid's position in the 3-D structure. The scientific community will strive for even bigger breakthroughs in the field of structural biology. AlphaFold 2 algorithm has been a precious tool in the hands of scientists, which will continue playing a vital role in advancing our knowledge of proteins and facilitating various research endeavors.

What we think of as a future project is testing the algorithm's generalization, using the class of membrane proteins. Membrane proteins [33] are proteins existing in or interacting with biological membranes. The generalization test can be done by fine-tuning OpenFold on the class of membrane proteins and then testing its performance on membrane protein structures. Then test performance of the non-tuned model on the same dataset. Finally, compare the performances of the two programs. If the fine-tuned version does not perform much better than the non-tuned one, we would conclude that AlphaFold 2 can perform well on predicting protein structures it has not been specifically fine-tuned on. Ideally, we would expect similar performances for both models, which would mean that AlphaFold 2 performs well for all protein classes, without the need for fine-tuning. For this project, we will again confront the alignment problem discussed in the previous chapter, but we will invest a significant amount of time both in protein alignment and fine-tuning of the model, hoping to overcome it.

ABBREVIATIONS - ACRONYMS

PDB	Protein Data Bank
MSA	Multiple Sequence Alignment
RMSD	Root Mean-Square Deviation

BIBLIOGRAPHY

[1] Alphafold. https://www.deepmind.com/research/highlighted-research/alphafold. Accessed: 2023-06-29.

[2] Google deepmind. https://www.deepmind.com/. Accessed: 2023-06-29.

[3] Adrian Linacre. Dna is often used in solving crimes. but how does dna profiling actually work? https://theconversation.com/dna-is-often-used-in-solving-crimes-but-how-does-dna-profiling-actually-work-191937/, 2022. Accessed: 2023-06-13.

[4] Madprime, cc0, via wikimedia commons. https://commons.wikimedia.org/wiki/File:Genetic_code.svg. Accessed: 2023-06-13.

[5] Kate Latham. Protein structure. https://biologydictionary.net/protein-structure/. Accessed: 2023-06-13.

[6] Claudio Soto. Protein misfolding and disease; protein refolding and therapy. *FEBS letters*, 498:204–207, 06 2001.

[7] John M. Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Zídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A A Kohl, Andy Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David A. Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583 – 589, 2021.

[8] Deepmind makes alphafold to solve a 50 year old biology problem. https://dev.to/amananandrai/deepmind-makes-alphafold-to-solve-a-50-year-old-biology-problem-2nik. Accessed: 2023-06-13.

[9] Enrique Reynaud. Protein misfolding and degenerative diseases. *Nat. Educ.*, 3, 01 2010.

[10] Levinthal's paradox. https://web.archive.org/web/20110523080407/http://www-miller.ch.cam.ac.uk/levinthal/levinthal.html. Accessed: 2023-06-24.

[11] Alphafold: a solution to a 50-year-old grand challenge in biology. https://www.deepmind.com/blog/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology. Accessed: 2023-06-24.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017.

[13] Chatgpt - wikipedia. https://en.wikipedia.org/wiki/ChatGPT. Accessed: 2023-06-24.

[14] Yin-Fu Huang and Yi-Hao Li. Sentiment translation model for expressing positive sentimental statements. 12 2019.

[15] Openfold.ipynb - colaboratory. https://colab.research.google.com/github/aqlaboratory/openfold/blob/main/notebooks/OpenFold.ipynb. Accessed: 2023-06-21.

[16] Abagyan R. Kufareva I. Methods of protein structure comparison. *Methods Mol Biol.*, 857:231–257, 2012.

[17] Rcsb protein data bank. https://www.rcsb.org. Accessed: 2023-06-16.

[18] Uniref. https://www.uniprot.org/help/uniref. Accessed: 2023-06-16.

[19] Debora S. Marks, Lucy J. Colwell, Robert Sheridan, Thomas A. Hopf, Andrea Pagnani, Riccardo Zecchina, and Chris Sander. Protein 3d structure computed from evolutionary sequence variation. *PLOS ONE*, 6:1–20, 12 2011.

[20] Carlos Outeiral Rubiera. Alphafold 2 is here: what's behind the structure prediction miracle. https://www.blopig.com/blog/2021/07/alphafold-2-is-here-whats-behind-the-structure-prediction-miracle/. Accessed: 2023-06-13.

[21] Wikid25, cc0, via wikimedia commons. https://commons.wikimedia.org/wiki/File:An_excerpt_of_a_multiple_sequence_alignment_of_TMEM66_proteins.png. Accessed: 2023-06-13.

[22] Peptide torsion angles and secondary structure. https://www.cryst.bbk.ac.uk/PPS95/course/9_quaternary/3_geometry/torsion.html. Accessed: 2023-06-13.

[23] Protein data bank (file format) - wikipedia. https://en.wikipedia.org/wiki/Protein_Data_Bank_(file_format). Accessed: 2023-06-27.

[24] Valerio Mariani, Marco Biasini, Alessandro Barbato, and Torsten Schwede. lddt: a local superposition-free score for comparing protein structures and models using distance difference tests. *Bioinformatics*, 29:2722–2728, 11 2013.

[25] Github - deepmind/alphafold: Open source code for alphafold. https://github.com/deepmind/alphafold. Accessed: 2023-06-20.

[26] Openfold: A faithful but trainable pytorch reproduction of deepmind's alphafold 2. https://github.com/aqlaboratory/openfold. Accessed: 2023-06-20.

[27] Pytorch homepage. https://pytorch.org/. Accessed: 2023-06-20.

[28] Github - google/jax: Composable transformations of python+numpy programs: differentiate, vectorize, jit to gpu/tpu, and more. https://github.com/google/jax. Accessed: 2023-06-20.

[29] Pdb-101: Molecule of the month: Calmodulin. https://pdb101.rcsb.org/motm/44. Accessed: 2023-06-20.

[30] Docker: Accelerated, containerized application development. docker.com. Accessed: 2023-06-20.

[31] Ucsf chimera home page. https://www.cgl.ucsf.edu/chimera/. Accessed: 2023-06-21.

[32] Openfold.ipynb - colaboratory. https://colab.research.google.com/github/aqlaboratory/openfold/blob/main/notebooks/OpenFold.ipynb. Accessed: 2023-06-21.

[33] Membrane protein. https://en.wikipedia.org/wiki/Membrane_protein. Accessed: 2023-06-17.