AlphaFold

AlphaFold: A Beginner's Guide And In-depth Exploration Of The Revolutionary AI Tool And Its Inner Workings.

Shravan Saranyan
Branham High School, 1570 Branham Ln, San Jose, CA, 95118, United States

Abstract
AlphaFold is a machine learning program that predicts a 3D model of the protein's structure from a protein's sequence. This paper aims to analyze the purpose, methodology, and structure of AlphaFold's neural networks and code, and additionally covers current and potential innovations to the tool. AlphaFold predicts protein structures with high accuracy by analyzing the evolutionary history of a protein, similarities to known proteins, and pairwise evolutionary correlations. It creates a refined pair representation using the proprietary Evoformer neural network, which is converted into a 3D structure by an additional structure stage. Additionally, this paper shows specific customizable features which can change the type of model, MSA, and pair mode of the prediction by analyzing the code from both AlphaFold (found on GitHub) and ColabFold. These features change depending on whether the protein sequence is a monomer or multimer, or by user preference to optimize for speed, accuracy, etc. They demonstrate the methodology of AlphaFold by visualizing the structure of the code. While the tool struggles to predict novel, variable, and non-static proteins, it is still a greatly significant tool and greatly useful in most cases. Newer versions of AlphaFold (AlphaFoldv3) continue to iterate and expand the functionality of the tool by expanding predictions to ligands, free nucleic acids, ions, and complexes.

Keywords
AlphaFold; ColabFold; Protein Folding; Evolutionary History of Proteins; Multiple Sequence Alignments; Bioinformatics

Introduction
AlphaFold is an AI (artificial intelligence) program powered by a neural network developed by Google's DeepMind to predict protein structures [1, 5]. It is entirely open source and is used to predict the 3D structure of proteins from an amino acid chain; a quick summary of this process is shown in Figure 1.

While DNA acts as the blueprint of genetic information for our cells, proteins are the core molecules executing the processes and functions dictated by our DNA. The journey from DNA to proteins involves two key steps: transcription (DNA to mRNA) and translation (mRNA to protein). During translation, our cells read the mRNA and segment it into groups of three letters called codons. Each codon is then translated into one of twenty different amino acids, which are linked together in a chain. Following the conversion of the RNA sequence into a chain of amino acids, the chain detaches and folds into a protein in a highly complex process, resulting in a 3D tertiary protein structure (see Fig. 5).

The behavior of proteins is often guided by their 3D structure and shape, which dictates how they engage and coordinate with other proteins in the body [2]. Understanding a protein's structure is critical for discerning its function and potential relevance in research and medicine. For example, in drug discovery and development, knowing the shape of a viral protein can enable researchers to design drugs to neutralize its function more effectively [3, 13]. Accurate structural predictions can also help scientists see

how proteins malfunction or misfold to cause several proteinaceous diseases such as Parkinson's and Alzheimer's [4, 13]. Thus, a novel, cost-effective means to accomplish this became all the more valuable, which is what AlphaFold aims to achieve.

Computational methods for predicting accurate 3D protein structures have usually focused on physical interactions or evolutionary history. Using the physical interactions approach is generally impractical because it involves a lot of molecular physics and thermodynamics. Using the evolutionary history of protein sequences is more practical and efficient. To determine the protein structure, this method analyzes the evolutionary history of the protein, similarities to previously solved proteins, and the relationships between similar pairs of amino acids (residues) that have evolved across different species (known as pairwise evolutionary correlations). The sequences and pairs of residues that are similar to the ones of the original protein are then organized into multiple sequence alignments (MSAs) and pairwise features respectively [5, 13]. This method greatly benefits from experimental protein structures deposited in the Protein Data Bank (PDB) [30], the genome sequencing of more species to derive more protein sequences, and better deep learning neural network interpretations to interpret these protein structures more efficiently [6].

The evolutionary history method proved viable with the creation of AlphaFold, the neural network behind the first computational approach to predict protein structures with high accuracy. AlphaFold structures have a median backbone (core of the protein) accuracy of within 0.96 angstroms (an angstrom is 0.1 nm, used to measure atomic structures); the next best model has an accuracy of within 2.8 angstroms, and it also models highly accurate side chains. The overall accuracy of AlphaFold is within 1.5 angstroms compared to 3.5 angstroms of the nearest competitor; it is also highly scalable to very long proteins [5] (Fig. 2).

A lot of the processes that occur below in AlphaFold are powered by neural networks, which are a kind of machine learning model that mimics how the human brain works (see Fig. 3). The fundamental unit of neural networks are neurons, which receive an input, process it, and give an output. Several of these neurons are grouped into columns, known as layers. The neural network contains an input layer, an output layer, and hidden layers which perform computations. Each neuron in a layer will feed its output as an input to the neurons in the next layer. Still, the importance of each input the neurons in the next layer receive is determined by the weight or strength of the connection the neurons have; the stronger the weight, the more influential the input is. There are also additional biases in the network, which are independent parameters added to the inputs of neurons to adjust the output, along with the weights from the previous neurons. The neural network is then asked to provide a desired output, and it tweaks itself with all of its variables until its output is constantly close to the desired output [7, 13]. By doing this training to maximize the confidence and accuracy of its protein model, AlphaFold can consistently pump out accurate protein structures confidently given the input of a protein sequence.

The remarkable thing about neural networks is that their inner workings are incomprehensible, as the network creates its own architecture. Model trainers provide desired outputs given example inputs, and the model rewires itself throughout its training to provide the desired output given an input [7]. This is why neural networks are often extremely complicated and convoluted. There are many different variations of neural networks, including networks that monitor their progress layer by layer, cycle back to previous

layers, and use multiple dimensions to map spatial features [9, 13]. AlphaFold includes all of these different kinds of neural networks at some stage or another, and they are the reason AlphaFold can do what it does.

The two main stages of the AlphaFoldv2 neural networks are the Evoformer and Structural Stages, which are explained in detail in *Structure of Neural Networks within AlphaFold*.

AlphaFold predicts two kinds of protein structures, **monomers**, and **multimers**. Monomers are proteins folded from a single protein chain, whereas multimers are composed of several protein chains that fold together into a larger structure. Overall, AlphaFold inputs the primary protein structure, which is the amino acid chain, and during the structural stage, the model predicts the tertiary protein structure, which is either given as the output (if monomer) or combined in an additional stage with the other tertiary protein structures for each protein (amino acid) chain to form a quaternary protein structure (if multimer) [13] (see Fig. 5).

AlphaFold's code can be found on GitHub and run locally or on Google Colab. Google Colab is a tool that Google recently released that runs code on Google's servers. It's free to use, beginner-friendly, and generally more efficient than running code locally on a computer [11].

Here is the Colab file for AlphaFold (version 2.3.2) [15]:
https://colab.research.google.com/github/sokrypton/ColabFold/blob/main/AlphaFold2.ipynb

And here is the Github repo that contains the code [16]:
https://github.com/sokrypton/ColabFold

There are two significant parts of the code for AlphaFoldv2 that provide insight into how it runs and is applied; the Colab version of the code (front-end), and the run() function (back-end).

There are several stages of the code in the Colab version; they can be divided as such:

- Block 1 - Input Protein Sequences
- Block 2 - MSA options
- Block 3 - Model and other Advanced settings
- Block 4 - Run Prediction
- Block 5 - Display 3D structure

A detailed description and explanation of each block and how they connect to the system as a whole is showcased in *Customizations and Understanding of AlphaFold's Code*.

Despite all of the great advances pioneered by AlphaFold in the fields of molecular biology and artificial intelligence, the tool is still very new and has plenty of limitations, so some of these limitations as well as areas of improvement that could make AlphaFold more accessible and a better tool are included in *Limitations and Areas of Improvement of AlphaFoldv2*.

This paper mainly covers AlphaFoldv2 (AF2), which is the commonly available version of AlphaFold as of November 2024. However, *Improvements and Advancements of AlphaFoldv3* acknowledges the newer editions of AlphaFold (namely AlphaFoldv3 (AF3), which was released in May 2024 and released to the public in November 2024), and shows how these new versions of AlphaFold remedy many of the limitations that AlphaFoldv2 had.

Structure of Neural Networks within AlphaFold

This section investigates the structure of neural networks within the AlphaFoldv2 (AF2) network that allow it to function.

The AlphaFold network comprises of two main stages:

Evoformer Stage
In this main stage, the MSA (Multiple Sequence Alignments), and pairwise features of the input protein and its relatives are processed through repeated layers of Evoformer, a novel neural network block [5] (Fig. 4a).

1. MSA—Multiple Sequence Alignments are protein sequences similar to the desired protein sequences that are all aligned together. They provide key information about evolutionary history and conservation.
2. Pairwise Features - The relationships between any pair of two residues (what's left of amino acids when they join into a chain) within the sequence contained within a Pair representation.

In every single block (layer), the MSA representation (I) is first refined using the Pair representation containing the Pairwise Features (II), and then it updates the Pair representation. The MSA representation refines itself by using the Pairwise Features (III) to find evolutionary information and context and then transitioning itself to absorb this information (IV). Then, it provides this information back to the Pair representation (V), which updates its Pairwise Features by graphing them as triangles and adjusting each edge and node to minimize interference or error (VI, see Fig. 4b-c). This process is repeated iteratively 48 times, each time gaining more evolutionary context and becoming more accurate. The end product is the original protein sequence (amino acid chain) and the Pair Representation, which now contains all the evolutionary context and information about the protein that can be taken advantage of in the next stage in the form of relations between residues (VII) [5, 13].

Structural Stage
The structural stage takes in the refined pair representation of every possible pair of residues from the previous stage and the original protein sequence, then determines the 3D structure of the protein and generates it (Fig. 4d). The main body of the protein, known as the basic backbone structure, is represented as a series of independent rotations and translations to a frame (known as a residue gas representation, shown in Fig. 4e) for each residue in the sequence. The rotations and translations that iteratively update the frame are determined by the IPA module (which takes in the protein sequence and pair representation, shown in d) and then are enacted by an update operation. Initially, when modeling, the rotations and translations follow certain prioritizations, but the complex rules behind the geometry of peptide chains

(protein chains) are ignored to allow for the specific refinement of each part of the chain and then are factored at the end using a violation loss term. After every block has finished, the stage uses the protein sequence and fully updated backbone frames to predict a 3D molecular model of the folded protein and provides the accuracy of the predicted structure compared to the true structure if known (Fig. 4c) [5, 13].

Customizations and Understanding of AlphaFold's Code

This section looks at the Colab version of AlphaFoldv2 block by block in order to understand the available customizations and the process by which the code runs.

Block 1 - Input Protein Sequences

In this block, the desired protein sequence is inputted, and a test run is started (Fig. 6a).

The **query_sequence** is where the desired protein is inputted. Each letter stands for one amino acid residue, here is a table [18]. The sequence is formatted to capitalize and remove spaces. A colon should be used to separate each protein chain within a complex (multimer) [15].
jobname is used to create a name for the test run, a hash is generated using the inputted query_sequence and is tackled onto this string to provide a unique identifier for the test run; in this example, "test_a5e17"
num_relax determines how many "top ranked structures" go through an additional refinement process called **AMBER**, which uses molecular dynamics to predict a more realistic and accurate model [12, 13].
template_mode is used to find templates if asked. Templates are similar protein structures that have already been determined and can be used to help predict the structure [15, 19].
The output of this block is the identifier of the test run (jobname), the formatted sequence, and the length of the sequence in residues [13, 15].

Block 2 - MSA Options

In this block, the options for the Multiple Sequence Alignments (MSA) mode and pairing mode are configured (Fig. 6b).

The **msa_mode** determines what kind of MSA is used, the three main modes are:
mmseqs2_uniref(_env): These modes use a computer-generated MSA derived from the mmseqs2 server [11, 14], and are the default option.
custom: This mode uses a custom user-inputted MSA, and is useful as a more advanced option for testing certain MSAs or aiming for more customized results [11].
single_sequence: This mode only uses the original sequence and no others, and is usually used for benchmarking MSAs [13, 15].

The **pair_mode** determines if the model should use an unpaired MSA, paired MSA, or both [15]:
unpaired: Uses a regular, unpaired MSA, which identifies similar sequences for only one chain, and is used to find intra-chain coevolutionary information; is used by default for monomers [22, 23].

paired: Uses a paired MSA, which identifies similar sequences for multiple chains, and is used to find inter-chain coevolutionary information; is used by default for homo-multimers (complexes of identical chains) [22, 23].
unpaired_paired: Uses both an unpaired and paired MSA to gain coevolutionary information for both each individual chain and the entire complex as a whole; is used by default for hetero-multimers (complexes of differing chains) [22, 23].
Note: When using unpaired_paired, it will default the pair_mode to whatever is dictated by the model_type, which itself is dependent on whether the sequence is a monomer, homo-monomer, or hetero-monomer [23].

Block 3 - Advanced Settings

These settings for the model are advanced, optional, and mostly self-explanatory. The most important and interesting ones are covered (Fig. 6c).

The **model_type** is very important, and determines which version of AlphaFold to use, the pros and cons of each as well as an example are outlined in the supplemental research [11, 15, 21].
num_recycles determines how many times the program is recycled, i.e.**,** when the output of AlphaFold is fed back in for refinement, by default it happens three times [5, 11, 13, 15].
relax_max_iterations limits the amount of iterations AMBER can run for if AMBER is activated, 0 means no limit is imposed and it can take very long to run [15].
The **pairing_strategy** of the MSA can be either greedy or complete, complete means that every single pair of residues is checked, while greedy means that only selective, informative pairings are chosen. Greedy is the default and is more efficient, but complete is more accurate [13, 15].
max_msa gives the option to limit the size of the MSA for the sake of efficiency while trading off accuracy [13, 15].
use_dropout gives the option to use Dropout, a technique where neurons are randomly removed during the neural network stages to prevent overfitting; when the model becomes too comfortable in its current state and refuses to change and evolve. The randomness of this and other parts of the model are determined by random seeds (**num_seeds**); more seeds prevent variance from the random initialization of each seed [13, 15, 17].

Block 4 - Run Prediction

This block is where AlphaFold network is executed, so this section shows the most relevant processes that occur within this block and how the previous customizations tie in. The code relevant to this section includes both the code from the ColabFold block (Fig. 6d-e) and the run() function from GitHub (Fig. 7) [15, 16].

The portion shown in Figure 6d imports several modules from the colabfold libraries, the key module that runs AlphaFold is colabfold.batch.run [20], which calls the run() function from the file batch.py within the colabfold folder. Note: Several other libraries (such as get_queries) are run separately and inputted into the run() function as parameters rather than natively [16, 20].

The run() function takes in a lot of parameters (Fig. 6e), including the query sequence (Block 1) and all the MSA and advanced settings (Blocks 2-3). The run function within AlphaFold then outputs five 2D predicted structures and ranks them from most to least accurate [15]. The function behaves like a main function within Colabfold (although there is another main() function within the GitHub version), and as such, offloads most of its processes to other functions that it calls within itself or takes in as inputs [16, 20]. The most important functions that directly relate to the prediction itself are shown below in order of where they are defined in the code [20]:

predict_structure() (Lines 324-542) - This function contains the heart of AlphaFold; it takes the sequence and features from the MSA as inputs and outputs the predicted structure. It generates a random seed (more if requested), loads in the AlphaFold models, and the parameters that control how they behave, and preprocesses the sequence, MSA, etc. After this, it takes in all of these and predicts a structure, rates the model confidence of each part of the structure, and outputs it as a protein object. Finally it reranks the models based on the predicted confidence [13, 20] (See Fig. 7).

pair_sequences(): Pairs the query sequence(s) and a paired MSA together into a larger MSA, only used for multimers [13, 20].

pad_sequences(): Aligns the sequences within an unpaired MSA and formats them to make sure the sequences all have the same length by adding spaces, which are only used for an unpaired MSA (usually monomers) [13, 20].

get_msa_and_templates(): Finds and obtains the unpaired MSA, paired MSA, and templates (if chosen) that match the query_sequence(s) [13, 20].

build_monomer_feature(), build_multimer_feature(): These functions parse through the unpaired MSA and paired MSA respectively, and generate features for each depending on the MSA pair type. Features include key information derived from the MSA, the sequences within, and the MSA itself [13, 20].

process_multimer_features(): Additional processing for the multimer features, as they are more complex [13, 20].

pair_msa(): Takes in several unpaired MSA for each chain in a multimer and combines them into a paired MSA that contains additional features that describe how the chains link together and form a system [13, 20].

generate_input_features(): Generates sufficient paired and unpaired features using some of the functions above to input into the structure prediction [13, 20].

unserialize_msa(): Loads in an MSA stored in a FASTA format (text-based way to represent protein sequence data in an efficient and human-interpretable way) and converts it back to a machine-usable format for analysis and processing [13, 20, 24].

msa_to_str(): Adds an unpaired or paired MSA to a FASTA string (i.e. serializes the MSA) [13, 20].

Figure 8 shows a visual graph of these functions ordered by when they are run in the program [20].

To summarize what the run() function does in order, it gets the desired MSA(s) and templates (optional), converts the MSA(s) into a machine-readable format (unserialized them), then processes them (depending on monomer vs multimer it pairs or pads them) and serializes it back into text. Then, the input features are generated, the MSA(s) are processed again using the same method as before, and specific features are built and processed to input specifically into AlphaFold (again, depending on monomer vs multimer). Finally, the protein structure is predicted using the features derived from the MSA(s) [15, 20].

Block 5 - Display 3D Structure and Plots

The final portion of the Colab version of AlphaFold involves displaying the predicted 3D structure of the protein (Fig. 9-10c) [15].

During the structure prediction from the previous block, the model generates five predicted structures (by default) and ranks them 1 to 5, most to least accurate. **rank_num** simply states which one to use; by default, it uses "1" (see Fig. 9).

The **color** paints the protein structure based on the confidence of the model (lDDT), the different chains present (chain), or aesthetically (rainbow). The most practical of these options is lDDT, which is the default and utilized in the example model (Fig. 10a)

The model shown in Figure 10a is very confident, and as such is mostly blue, except for the ends, which are more orange, as the model is not very confident in that section. This 3D structure can be endlessly rotated in all axes to see the full structure, as well as zoomed in.

show_sidechains and **show_mainchains** are options that display the additional molecular chains that come along with the backbone structure predicted by AlphaFold; see Fig. 10b.

The model shown in Figure 10b is more accurate in terms of how the protein would appear, but it does look very messy and makes it harder to see the more relevant structure, so these modes are usually turned off.

The Colab version of AlphaFold additionally creates plots for bioinformatics purposes (Fig. 10c).

Limitations and Areas of Improvement of AlphaFoldv2

These sections showcase the limitations of AlphaFoldv2 and the small, medium, and large scale improvements that can be made.

Limitations of AlphaFold

AlphaFold sometimes struggles with predicting new or unusual proteins, known as “orphans,” because they have lesser close relatives, which is how it derives evolutionary information to predict protein structures. These protein predictions result in low-accuracy predictions with low confidence scores (red/orange in lDDT), and they can be made far worse if the “orphan” proteins have little to no related structures in the PDB (Protein Data Bank). This is why it’s essential to update the PDB with as many structures as possible, as AlphaFold excels at predicting new proteins if there are enough related sequences in the PDB to produce a quality MSA for prediction [25, 30].

Additionally, it struggles with high variation within proteins, whether that be regular point mutations within DNA that are then translated to altered residues within the protein or proteins with inherently high variation within sequences, such as antibodies. This is because of both a lack of data on the effect of variations within proteins and AlphaFold’s focus on long-term evolutionary changes rather than physical forces that cause short-term changes such as radiation, mutation, etc [25].

While performing their functions, proteins often change their structure to adapt their function (a great example is the complement system [26]), but only a few structures have their potential conformations in the PDB (which AlphaFold uses for reference. By default, AlphaFold only predicts static structures, but researchers have found ways to trick AlphaFold to predict different versions of the proteins) [25].

Figure 11 shows an example of how a protein can conform to a completely different structure when bonding with a different molecule. Here, when the protein Hexokinase bonds with a sugar molecule, it conforms to a sugar-bound structure, and when AlphaFold predicts both, it has a lot more trouble predicting the sugar-bound structure (within 3.02 angstroms to within 0.67 angstroms, the lower, the more accurate) [25].

There are also several functions within the process of protein folding and interactions that AlphaFold wasn't designed to do [25]. These include but are not limited to:

- Other molecules that interact (but don’t bond) with proteins, such as ions, nucleic acids (DNA, RNA), and other non-proteins that work closely with proteins and may provide key context on the protein’s function.
- Any modifications that happen to the protein after translation.
- Free nucleic acids (unfolded DNA or RNA that didn’t translate into proteins).
- Any ions or other molecules (ligands) that bond post-translation (AlphaFold usually predicts them when absent, a false positive).
- Anything to do with the Membrane Plane (a fluid membrane made up of fat lipids, complex molecules, and proteins within cells), including a protein’s relative position to it (if the protein spans through it).

Potential Advancements for AlphaFold

Below, potential advancements are shown that can be made to AlphaFold, ranging from small (portion of AlphaFold), to medium (AlphaFold as a whole), to large (the field as a whole).

Small advancements:

When trying to input a protein into AlphaFold, scientists often go onto the NIH library [27], and search for proteins, looking for the protein sequence listed at the bottom. The problem is the sequence usually appears in a grid-like formation (Fig. 12a).

This formation (Fig. 12a) when copy-pasted isn't compatible with the input the way it is formatted. This problem can easily be fixed by adding a snippet of code (Fig. 12b) which removes the spaces (already done in the current version [15]) and then numbers and capitalizes the sequence.

When added, this can save scientists a lot of time and effort from removing the digits and capitalizing manually, and more seamlessly integrates the library of protein sequences and the program.

Medium advancements:

There are several areas where AlphaFold could be advanced, from the code to the interface to the content of the program. Here are some potential improvements below:
(Note: These are all just ideas, which may vary in actual feasibility)

- The ColabFold program takes a long time to run, especially for multimers and complex proteins. There are modes within the advanced settings of ColabFold that reduce the runtime [15] (such as reducing the size of the MSA or reducing the number of ranks generated), but all of those modes have to be inputted manually, and for a novel protein, it can be hard to tell the correct tradeoff between accuracy and efficiency. So potentially, a time-saving mode could be created that would automatically tweak specific modes to reduce the runtime if it deems the tradeoff of accuracy to be worth it; this automatic mode could be powered by some neural network to run the ColabFold program with more efficiency and manage the tradeoff of accuracy.

- Instead of manually inputting a query sequence, the ColabFold program could have a search function that saves the user from having to search and find a protein sequence; they would only have to type the name of a protein, and it would predict the most common version (For searching for particular proteins the specific tag used in the NIH library [27] would have to be used)
  Note: This has been incorporated into AlphaFoldv3.

- Because Google Colab uses a different computer when a runtime is restarted, any one-time-only downloads that AlphaFold requires must be repeated, which can take a lot of time (if using both amber and templates, 2 minutes are used just to download their libraries, and another 90 seconds are used to download the specific AlphaFold model (ptm) [21]). This problem could be fixed by potentially working with Google and downloading all of these files onto some of their machines and assigning those specific machines to ColabFold users to save time on downloads.

- Additionally, restructuring the order in which ColabFold is run by placing any user-inputted parameters (model_type, pair_mode, template_mode, etc.) nearer to the end would make the turnaround for quick runs of the model with slight changes to certain parameters (for purposes of

experimentation) far more efficient. Anything unchanged within ColabFold (such as downloading libraries, defining functions, etc.) could be run only once, so subsequent runs of ColabFold could be processed far quicker.

- Looking at the model_type options [21], some models had better overall accuracy (**ptm**), and some models had better point accuracy (**multimer_v2**, **deepfold_v1**). By using two algorithms, one to determine where each model does best and focus that model on what section it does best, and another one to stitch each piece of the protein model together, a "Frankenstein" model could consist of different predictions stitched together with the best attributes from every available model. This model would be extremely time-intensive (measures like the ones above could be used to save time) but could potentially be more confident and accurate than its parts.

- There are three general types of tertiary structures (what AlphaFold outputs for monomers): **globular**, **fibrous**, and **membrane** [29]. While it's usually pretty easy to tell which is which based on their shape, sometimes the protein may be more ambiguous. AlphaFold could help out by labeling every 3D structure it predicts as one of these protein types (or, in the case of multimers, what type each strand within is), which would make the model easier to interpret.

- AlphaFold provides plenty of plots and graphs with its output for bioinformatics purposes, but some of these graphs could do with adding in a key and labels for each axis, as well as a quick explanation on how to interpret the plots and graphs for beginners to more easily use the data, and to make the data more accessible.

Large advancements:

Large advancements are the ideas and concepts that will not just improve AlphaFold but elevate the entire field. There are three of these ideas below.

Improved accessibility to new proteins from the Protein Data Bank using web parsers
There are thousands of new proteins discovered every year, but not all of the proteins discovered end up on the PDB. These novel proteins are usually from unconventional hosts (any host besides humans/mice) and are usually still present in research documents across the internet. A web parser (or scraper) could theoretically comb through all of these overlooked proteins and automatically run them through AlphaFold (if enough similar proteins are found) and add them and their structure to PDB. This would significantly improve cooperation between researchers by making these overlooked proteins more accessible and shed more light on the fields they represent [30].

Adding features to AlphaFold to overcome some of its limitations
Now that the concept of using machine learning to predict molecular structures has been proven, AlphaFold can move beyond some of its limitations by adding features, such as training the model to predict several variations of the protein, molecules binding to the protein, free DNA and RNA strands, and protein-nucleic acid complexes. This would dramatically improve the scope of AlphaFold's reach and would be an important stepping stone for future endeavors.
Note: These features are now present in AlphaFoldv3, which is shown more in detail in Theme 4.

Predicting Protein Mimicry

Protein Mimicry is a phenomenon where bacterial (or other foreign organism) proteins mold themselves into a shape similar to that of the host's own proteins, usually to avoid detection. When AlphaFold predicts a protein, it could also show bacterial proteins that have been known to mimic that specific protein. This would give important insights and information to researchers studying protein mimicry and expose more people to the field [31].

Improvements and Advancements of AlphaFoldv3

AlphaFoldv3 (AF3) was released in May 2024 to researchers and in November 2024 to the general public. The new model allows for the accurate structural prediction of many other biomolecular interactions in addition to protein structures. These interactions mainly consist of complexes of proteins, nucleic acids, small molecules, ions, and/or modified residues. AF3 was created to tackle the problem of predicting these interactions in the same vein as previous versions of AlphaFold did for protein structures; it can predict nearly all molecular types in the PDB.

New Methods

The structure of AF3's neural networks is based off of AlphaFoldv2 (AF2) but contains some key differences, the Evoformer stage has been replaced by a simpler Pairformer module, which decreases the number of MSA computations required. And instead of updating a residue gas model to obtain the final structure, AF3 predicts the actual atom coordinates for the entire model using a diffusion module; this gives the model more flexibility from having to be reliant on amino-acid (residue) relationships.

The main difference between AF3's Pairformer and AF2's Evoformer is the de-emphasis of MSAs within the neural networks. MSAs are used beforehand to provide information to the Pair representation in a conditioning network, and Pairformer itself doesn't rely on MSAs at all [33]. It simply uses the pair and updated single representations to obtain a plethora of information about the relationships between any and all pairs of residues; this reduces the burden of MSA processing.

The resulting pair representation and single representation are then thrown into the brand-new diffusion module, which replaces the structural module of AF2. Unlike the structural module, which incrementally updates a frame via torsions along different axes, the diffusion module predicts the actual coordinates of every atom within the predicted structure.

It had been observed that the backbone frame and torsions of AF2 added a lot of complexity, and that removing most of AF2's complexity had a lesser impact on accuracy. The AF2 structure module also required very sensitive structural violation penalties to ensure chemical plausibility. The AF3 diffusion model uses a different approach; it receives approximate atom coordinates and pinpoints their exact location. Even if the model is uncertain, it will produce an exact atomic model of the structure [32].

However, the model sometimes "hallucinates", which is when it predicts structures even where there aren't any. This effect can be remedied by using predicted structures from the multimer version of AF2 in

conjunction with the diffusion module when training. Confidence measurements like the ones present in AF2 are used, but since the diffusion module is only trained one step at a time, a "mini-rollout" is required, where a partial or full structure is predicted after each step to gauge accuracy.

The representation in Figure 13c shows that modelling smaller, single representations (known as "local structures"; includes ligands, proteins, DNA, RNA) is easier than modelling large complexes (known as "global constellations"). As training increases, some of these graphs peak in accuracy and decline (due to overfitting), and some of them take more time to reach 97% potential accuracy, so it customizes the training weights to find the best checkpoint for the highest accuracy [32].

Overall, the overhaul of the neural networks within AF3 simplify many of the existing complexities within the networks, such as the long MSA processing runtimes. They also add new complexities and details such as the addition of a diffusion module to pinpoint the exact atom coordinates of the entire structure. These bring forth new solutions and new challenges, but they do vastly improve and polish the model, and enable it to perform far more utility than its predecessor.

As of the end of 2024, AF3's code has yet to be fully published and made open-source by the AlphaFold team like AF2 was, but it is available for non-commercial use on a server [34, 35].

Conclusion

The purpose of AlphaFold is to provide an accurate prediction of the structure of a protein (and other organic structures in newer versions). It achieves this purpose by using a neural network to determine the evolutionary history of a protein; the MSAs and/or pairwise features garnered as a result can be used to predict how the protein can fold. In AF2, the Evoformer stage is the main innovation that allowed this tool to function; within each of its iterations, the MSA and Pair representation update each other to share and collectively express the evolutionary history of the sequence and each residue pair. The structural stage then uses the context derived from the Evoformer stage to rotate and translate a frame into the finished prediction using strict regulations to ensure accuracy. The AlphaFold model can be customized to the user's preference within the open-source, user-friendly version of AF2 on Google Colab (known as ColabFold). Within ColabFold the user can input their sequence, determine how much additional refinement is needed, use existing templates, and most importantly they can determine how to pair MSAs and which specific model within AF2 to use. In the supplemental research, great variation was found between the finished predictions of these models (ptm behaved best for monomers, multimer_v3 behaved best for multimers, and deepfold_v1 occasionally behaved better). The visualization of the run() function (which was the main function in the code running everything) shows how every customization is applied throughout the prediction and modelling process. The final prediction is colored based on model confidence (lDDT); five separate predictions are made, ranked, and shown in terms of accuracy per position, and the MSA for all the protein sequence(s) is shown as well. AF2 has plenty of limitations, chief of which is the inability to predict other sequences that interact with proteins. AF3 improves upon AF2 because it can predict these additional sequences, and unlike AF2 it can predict the precise atomic coordinates of the entire model using a diffusion module. When predicting, AF3 has the highest accuracy with ligands and proteins, and the lowest accuracy with complexes. AF3 has more features and utility than AF2, but is yet to have its code published. In addition to these advancements, AlphaFold could

continue to improve with ideas such as combining several models to increase point accuracy, more accessible bioinformatic data, categorization of protein tertiary structures, prediction of protein mimicry, and the implementation of techniques such as web parsers to add non-model protein samples to the PDB.

Acknowledgements

I would like to thank Vera Bellinson (vbellins@caltech.edu) for mentoring me throughout this project and the Polygence Core Program for bringing us together and helping with the management of this project.

The author(s) declare that there are no conflicts of interest regarding the publication of this article.

References

1. AlphaFold Protein Structure Database. Available from: https://alphafold.ebi.ac.uk/ (accessed on 2024-08-31).

2. Alberts, B., Johnson, A., Lewis, J., et al., Molecular Biology of the Cell, 4th ed., Garland Science, 2002. The Shape and Structure of Proteins. Available from: https://www.ncbi.nlm.nih.gov/books/NBK26830/.

3. Hughes, J. P., Rees, S., Kalindjian, S. B., and Philpott, K. L., "Principles of early drug discovery," Br. J. Pharmacol., vol. 162, no. 6, pp. 1239–1249, 2011. doi: 10.1111/j.1476-5381.2010.01127.x.

4. Ashraf, G. M., et al., "Protein misfolding and aggregation in Alzheimer's disease and type 2 diabetes mellitus," CNS Neurol. Disord. Drug Targets, vol. 13, no. 7, pp. 1280–1293, 2014. doi: 10.2174/18715273136661409170955 14.

5. Jumper, J., Evans, R., Pritzel, A., et al., "Highly accurate protein structure prediction with AlphaFold," Nature, vol. 596, pp. 583–589, 2021. doi: 10.1038/s41586-021-03819-2.

6. Pakhrin, S. C., et al., "Deep learning-based advances in protein structure prediction," Int. J. Mol. Sci., vol. 22, no. 11, p. 5553, 2021. doi: 10.3390/ijms22115553.

7. Haggerty, M. A., "Explained: Neural networks and deep learning," MIT News. Available from: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414 (accessed on 2024-08-31).

8. Williams, J. H., "Language modeling from scratch: Part 2," Towards AI. Available from: https://towardsai.net/p/data-science/language-modeling-from-scratch-part-2 (accessed on 2024-08-31).

9. Sarker, I. H., "Deep learning: A comprehensive overview on techniques, taxonomy, applications, and research directions," SN Comput. Sci., vol. 2, no. 6, p. 420, 2021. doi: 10.1007/s42979-021-00815-1.

10. Reading Protein Structure. Nursing Hero. Available from: https://www.nursinghero.com/study-guides/bio1/reading-protein-structure (accessed on 2024-09-01).

11. Mirdita, M., Schütze, K., Moriwaki, Y., et al., “ColabFold: making protein folding accessible to all,” Nat. Methods, vol. 19, pp. 679–682, 2022. doi: 10.1038/s41592-022-01488-1.

12. Hornak, V., Abel, R., Okur, A., et al., “Comparison of multiple Amber force fields and development of improved protein backbone parameters,” Proteins, vol. 65, no. 3, pp. 712–725, 2006. doi: 10.1002/prot.21123.

13. ChatGPT (GPT-4o). OpenAI. Available from: https://chat.openai.com/chat (accessed on 2024-09-01).

14. ColabFold Documentation. UMass Unity. Available from: https://docs.unity.rc.umass.edu/documentation/tools/colabfold/ (accessed on 2024-09-01).

15. ColabFold. AlphaFold2.ipynb, 2021. Available from: https://colab.research.google.com/github/sokrypton/ColabFold/blob/main/AlphaFold2.ipynb (accessed on 2024-09-01).

16. ColabFold. GitHub repository, 2021. Available from: https://github.com/sokrypton/ColabFold (accessed on 2024-09-01).

17. torch.nn.Dropout. PyTorch Documentation. Available from: https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html (accessed on 2024-09-01).

18. Table 2.1. Western Oregon University, Apr. 2020. Available from: https://wou.edu/chemistry/files/2020/04/Table-2.1.jpg (accessed on 2024-09-01).

19. Soeding Lab. hhdatabase_cif70, GitHub repository, 2021. Available from: https://github.com/soedinglab/hhdatabase_cif70 (accessed on 2024-09-01).

20. ColabFold. batch.py, GitHub repository, 2021. Available from: https://github.com/sokrypton/ColabFold/blob/main/colabfold/batch.py (accessed on 2024-09-01).

21. Saranyan, S., “Investigations into AlphaFold Modelling Types,” Google Docs. Available from: https://docs.google.com/document/d/1qSxyBQB7Vwt77xJsdGWz8se-t6XDk7AVuZNLEb4vKSs/edit (accessed on 2024-09-01).

22. Bryant, P., Pozzati, G., and Elofsson, A. "Improved prediction of protein-protein interactions using AlphaFold2," Nat. Commun., vol. 13, p. 1265, 2022. doi: 10.1038/s41467-022-28865-w.

23. Liu, J., et al., "Enhancing AlphaFold-Multimer-based protein complex structure prediction with MULTICOM in CASP15," Commun. Biol., vol. 6, no. 1, p. 1140, Nov. 2023. doi: 10.1038/s42003-023-05525-3.

24. "FASTA Format Overview," Zhang Lab. Available from: https://zhanggroup.org/FASTA/ (accessed on 2024-09-01).

25. "Strengths and Limitations of AlphaFold," EMBL-EBI. Available from: https://www.ebi.ac.uk/training/online/courses/alphafold/an-introductory-guide-to-its-strengths-and-limitations/strengths-and-limitations-of-alphafold/ (accessed on 2024-09-01).

26. "The Complement System," NCBI Bookshelf. Available from: https://www.ncbi.nlm.nih.gov/books/NBK27100/ (accessed on 2024-09-01).

27. "National Center for Biotechnology Information (NCBI)," NCBI. Available from: https://www.ncbi.nlm.nih.gov/ (accessed on 2024-09-01).

28. "Modified Copy of AlphaFold2.ipynb," Google Colab. Available from: https://colab.research.google.com/drive/1aZmUSm1k3XBO6W18gfu_lnFhUJWPx4H- (accessed on 2024-09-01).

29. "Protein Classes," Rose-Hulman Institute of Technology. Available from: https://www.rose-hulman.edu/~brandt/Chem330/Protein_classes.pdf (accessed on 2024-09-01).

30. "Research Collaboratory for Structural Bioinformatics (RCSB) Protein Data Bank," RCSB. Available from: https://www.rcsb.org/ (accessed on 2024-09-01).

31. Maoz-Segal, R., and Andrade, P., "Molecular mimicry and autoimmunity," in Infection and Autoimmunity, pp. 27–44, 2015. doi: 10.1016/B978-0-444-63269-2.00054-4.

32. Abramson, J., Adler, J., Dunger, J., et al., "Accurate structure prediction of biomolecular interactions with AlphaFold 3," Nature, vol. 630, pp. 493–500, 2024. doi: 10.1038/s41586-024-07487-w.

33. Springer Nature, "Supplementary Information: Accurate structure prediction of biomolecular interactions with AlphaFold 3," Nature, vol. 630, pp. 493–500, 2024. Available from: https://static-content.springer.com/esm/art%3A10.1038%2Fs41586-024-07487-w/MediaObjects/41586_2024_7487_MOESM1_ESM.pdf (accessed on 2024-12-11).

34. "AlphaFold3 — why did Nature publish it without its code?," Nature, Jun. 2024. Available from: https://www.nature.com/articles/d41586-024-01463-0 (accessed on 2024-12-31).

35. "AlphaFold Server," AlphaFoldServer.com. Available from: https://alphafoldserver.com/ (accessed on 2025-01-02).

Tables and Figures

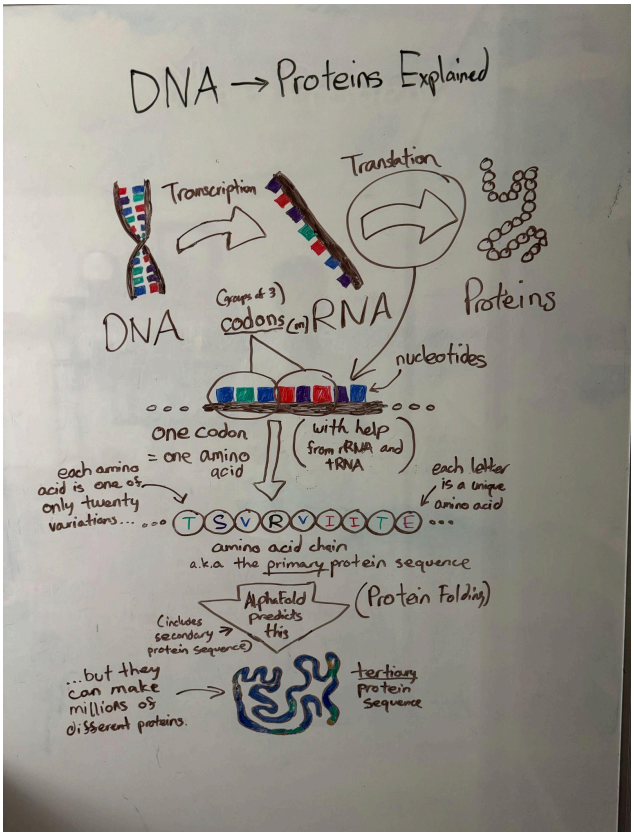


Figure 1. **DNA to Proteins Explained**. A refresher on how cells transcript and translate DNA into folded 3D proteins.

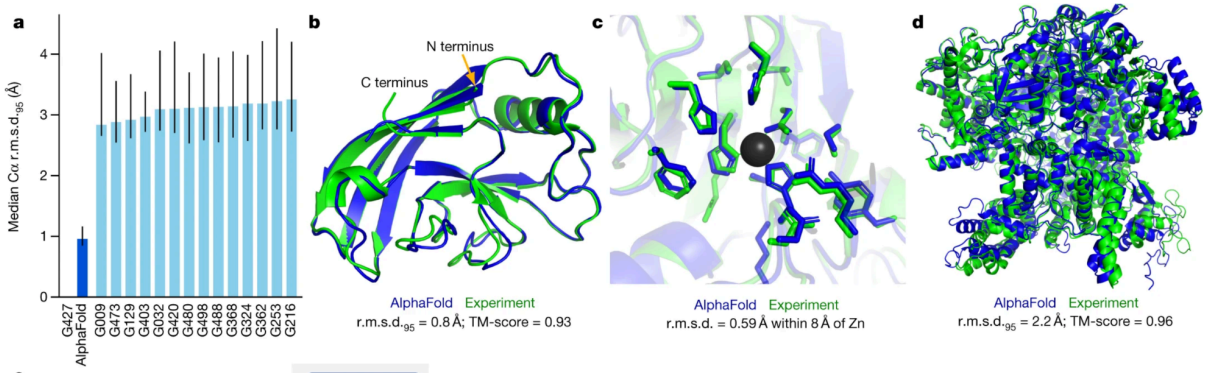


Figure 2. **Examples of AlphaFold Accuracy**. Above is the median backbone accuracy of AlphaFold (G427) in angstroms compared to other models (G009-G216) and some example outputs (a, b, c, d). [5].

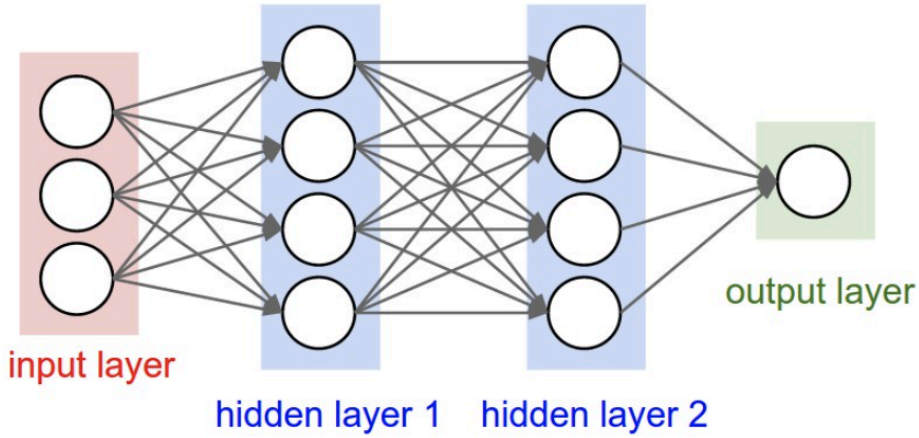


Figure 3. **Neural Network Diagram**. In the example above, the layers are labeled, the dots are the neurons, and the arrows are the weights from the previous layers. [8].

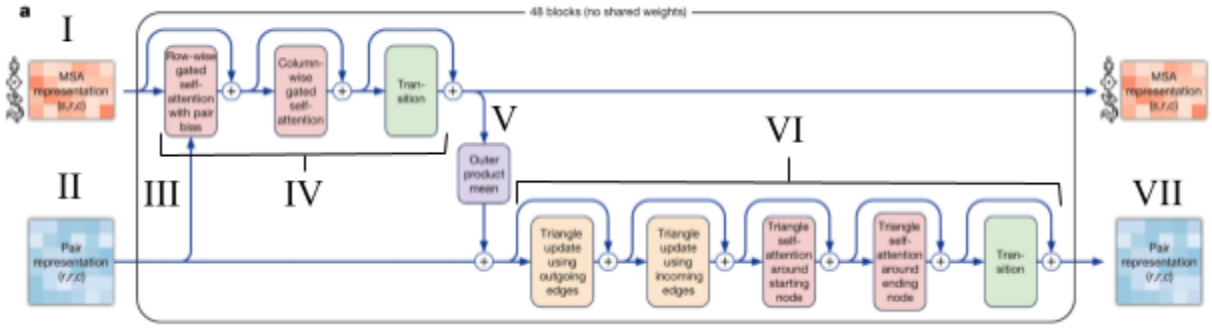
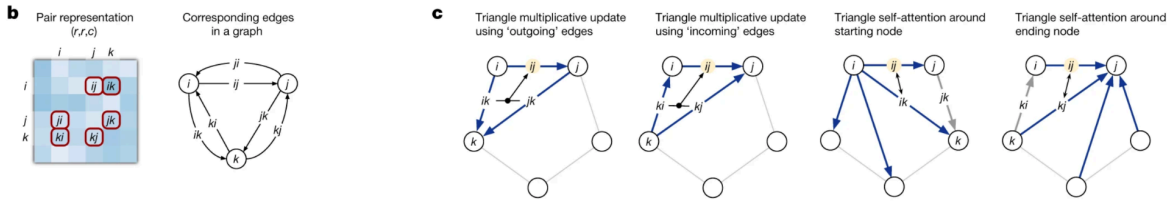
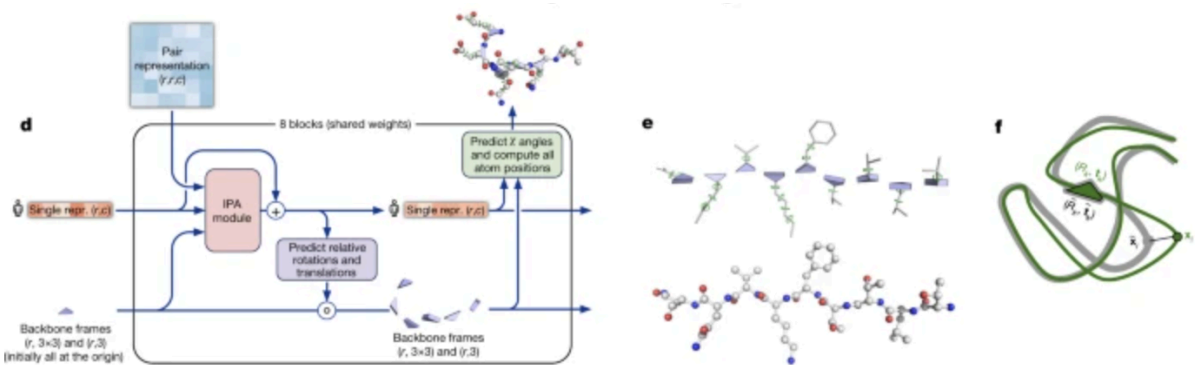


Figure 4a. **Architectural Details of Evoformer Stage**. A visual representation of the Evoformer Stage of AlphaFold, contains seven major steps (I-VII). [5].

Legend for Fig 4a:
I - MSA Representation
II - Pairwise Features
III - MSA Representation uses Pairwise Features to refine itself
IV - Process of MSA Representation refinement
V - Provides information back to Pair representation
VI - Pairwise Features updating using the graph interference problem
VII - The finished Pair Representation and MSA Representation containing evolutionary context



Figures 4b-c. **Pair Representation Details**. Above is a visual representation of how select pairs of residues are weighted in a pair representation and shown as a graph representation (b), as well as how it self-updates the graph to minimize interference (c). [5].



Figures 4d-f. **Architectural Details of Structural Stage**. Above is the Architectural Details of the Structural Stage of AlphaFold (d), example of the backbone frames (e), and the algorithm to check structural accuracy (f). [5].

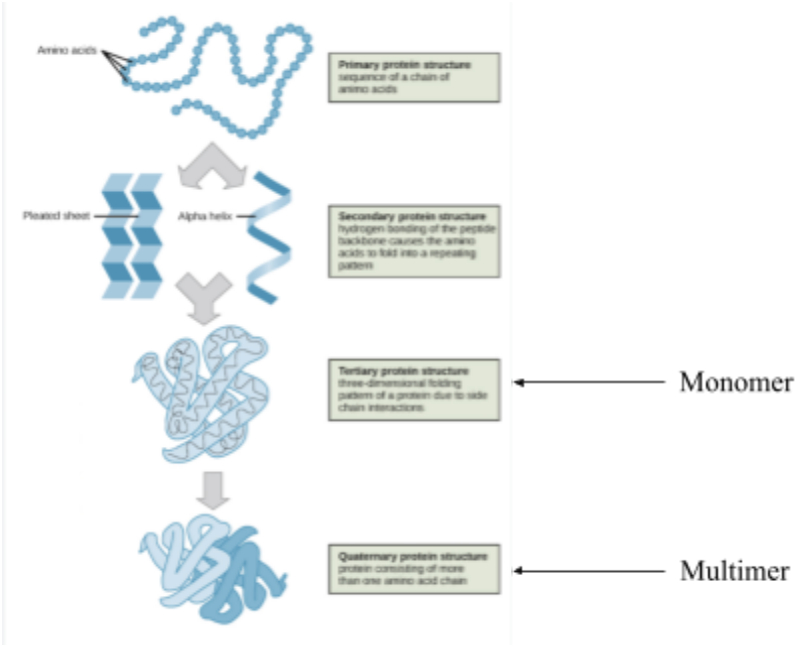


Figure 5. **Four Levels of Protein Structure**. Above are the four stages of protein structure as the protein folds from a one-dimensional chain into a three-dimensional structure, and which stage AlphaFold predicts as its output depending on the protein type (monomer/multimer). [10].

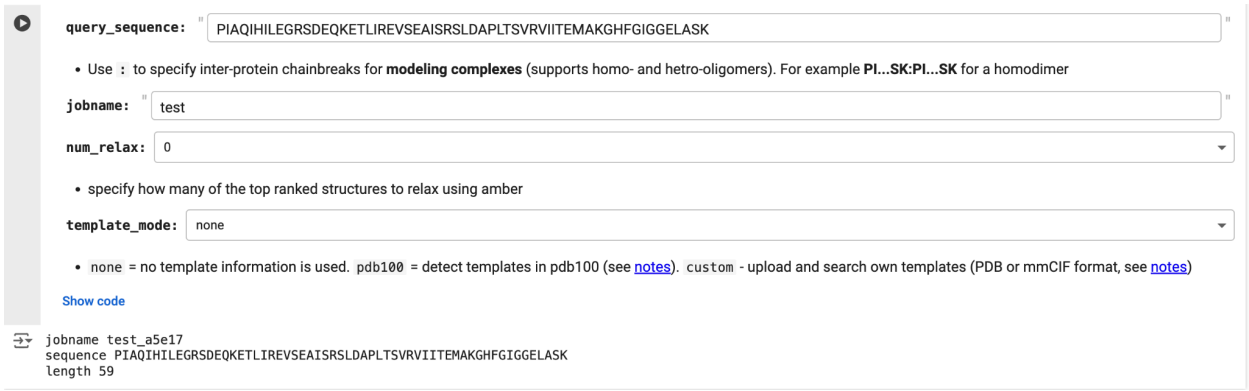


Figure 6a. **Block 1**. Screenshot of Block 1 in AlphaFold2.ipynb (Colabfold). [15].

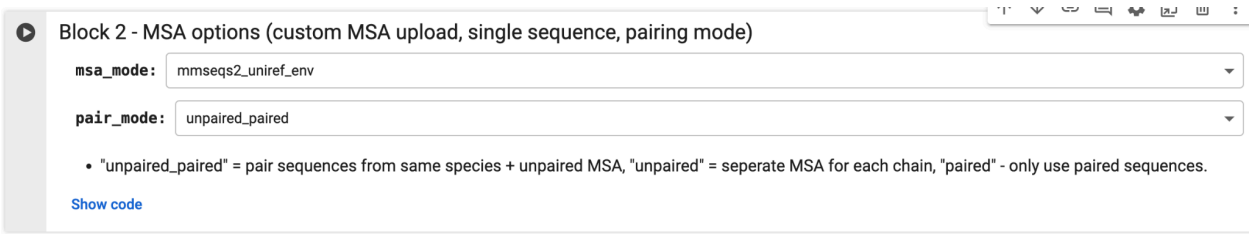


Figure 6b. **Block 2**. Screenshot of Block 2 in AlphaFold2.ipynb (Colabfold). [15].

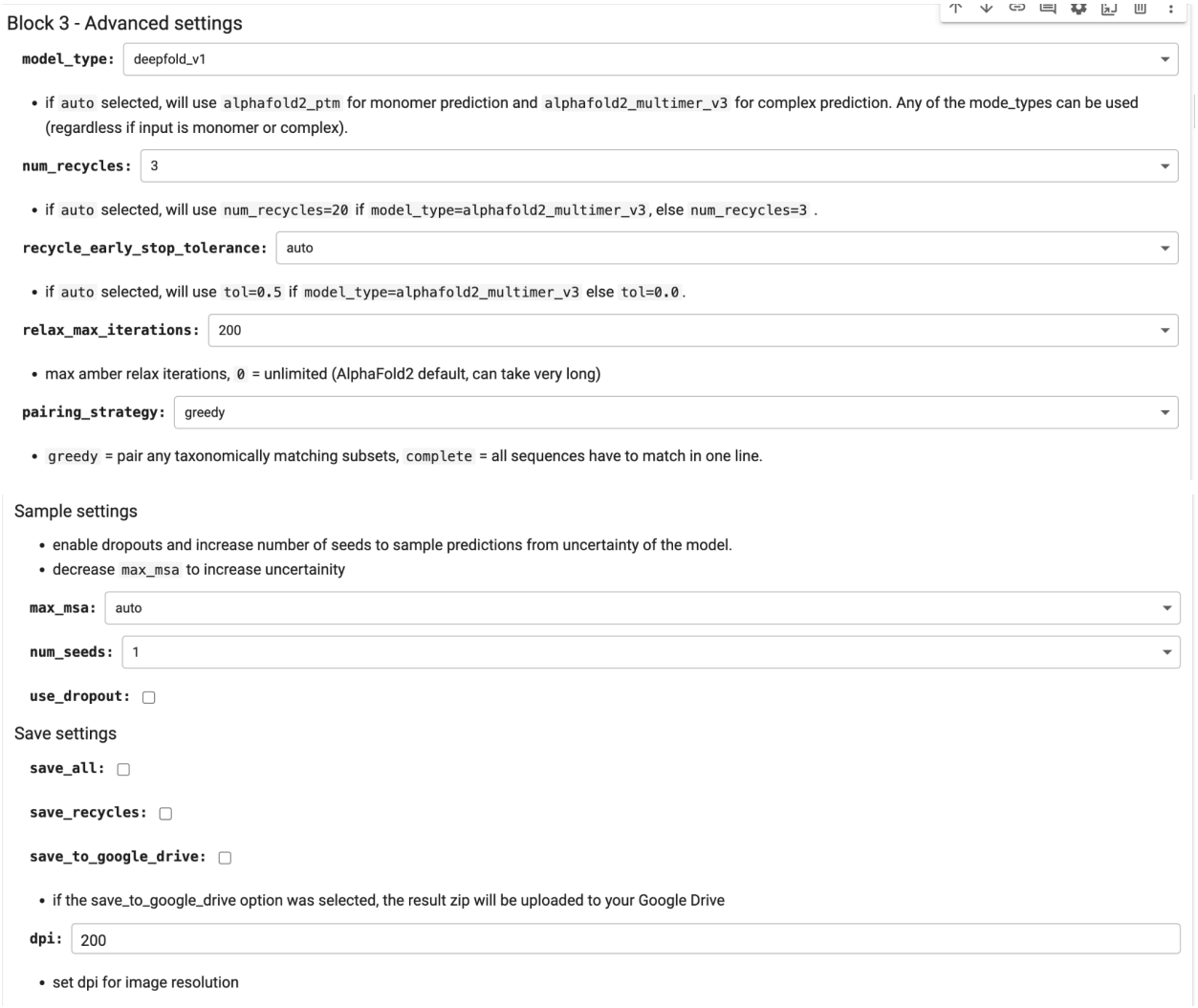


Figure 6c. **Block 3**. Screenshot of Block 3 in AlphaFold2.ipynb (Colabfold). [15].

```
# Imports required libraries and functions for ColabFold
import sys
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
from Bio import BiopythonDeprecationWarning
warnings.simplefilter(action='ignore', category=BiopythonDeprecationWarnin
from pathlib import Path
from colabfold.download import download_alphafold_params, default_data_dir
from colabfold.utils import setup_logging
from colabfold.batch import get_queries, run, set_model_type
from colabfold.plot import plot_msa_v2

import os
import numpy as np
```

Figure 6d. **Imports for Block 4**. Screenshot of required libraries and functions for ColabFold being imported in AlphaFold2.ipynb (Colabfold). [15, 16].

```
results = run(
    queries=queries,
    result_dir=result_dir,
    use_templates=use_templates,
    custom_template_path=custom_template_path,
    num_relax=num_relax,
    msa_mode=msa_mode,
    model_type=model_type,
    num_models=5,
    num_recycles=num_recycles,
    relax_max_iterations=relax_max_iterations,
    recycle_early_stop_tolerance=recycle_early_stop_tolerance,
    num_seeds=num_seeds,
    use_dropout=use_dropout,
    model_order=[1,2,3,4,5],
    is_complex=is_complex,
    data_dir=Path("."),
    keep_existing_results=False,
    rank_by="auto",
    pair_mode=pair_mode,
    pairing_strategy=pairing_strategy,
    stop_at_score=float(100),
    prediction_callback=prediction_callback,
    dpi=dpi,
    zip_results=False,
    save_all=save_all,
    max_msa=max_msa,
    use_cluster_profile=use_cluster_profile,
    input_features_callback=input_features_callback,
    save_recycles=save_recycles,
    user_agent="colabfold/google-colab-main",
)
```

Figure 6e. **Run function in Block 4**. Screenshot of the run function [20]; which is being called in AlphaFold2.ipynb (Colabfold). [15, 20].

```
422             # predict
423             result, recycles = \
424             model_runner.predict(input_features,
425                 random_seed=seed,
426                 return_representations=return_representations,
427                 callback=callback)
428
429             prediction_times.append(time.time() - start)
430
431             ########################
432             # parse results
433             ########################
434
435             # summary metrics
436             mean_scores.append(result["ranking_confidence"])
437             if recycles == 0: result.pop("tol",None)
438             if not is_complex: result.pop("iptm",None)
439             print_line = ""
440             conf.append({})
441             for x,y in [["mean_plddt","pLDDT"],["ptm","pTM"],["iptm","ipTM"]]:
442               if x in result:
443                 print_line += f" {y}={result[x]:.3g}"
444                 conf[-1][x] = float(result[x])
445             conf[-1]["print_line"] = print_line
446             logger.info(f"{tag} took {prediction_times[-1]:.1f}s ({recycles} recycles)")
447
448             # create protein object
449             final_atom_mask = result["structure_module"]["final_atom_mask"]
450             b_factors = result["plddt"][:, None] * final_atom_mask
451             unrelaxed_protein = protein.from_prediction(
452                 features=input_features,
453                 result=result,
454                 b_factors=b_factors,
455                 remove_leading_feature_dimension=("multimer" not in model_type))
```

Figure 7. **Portion of the predict_structure() function.** Screenshot of a relevant portion of the predict_structure() function within the run() function. [15, 20].

Figure 7 shows the prediction (422-429), model confidence (435-446), and creation of protein objects (449-455) [20].

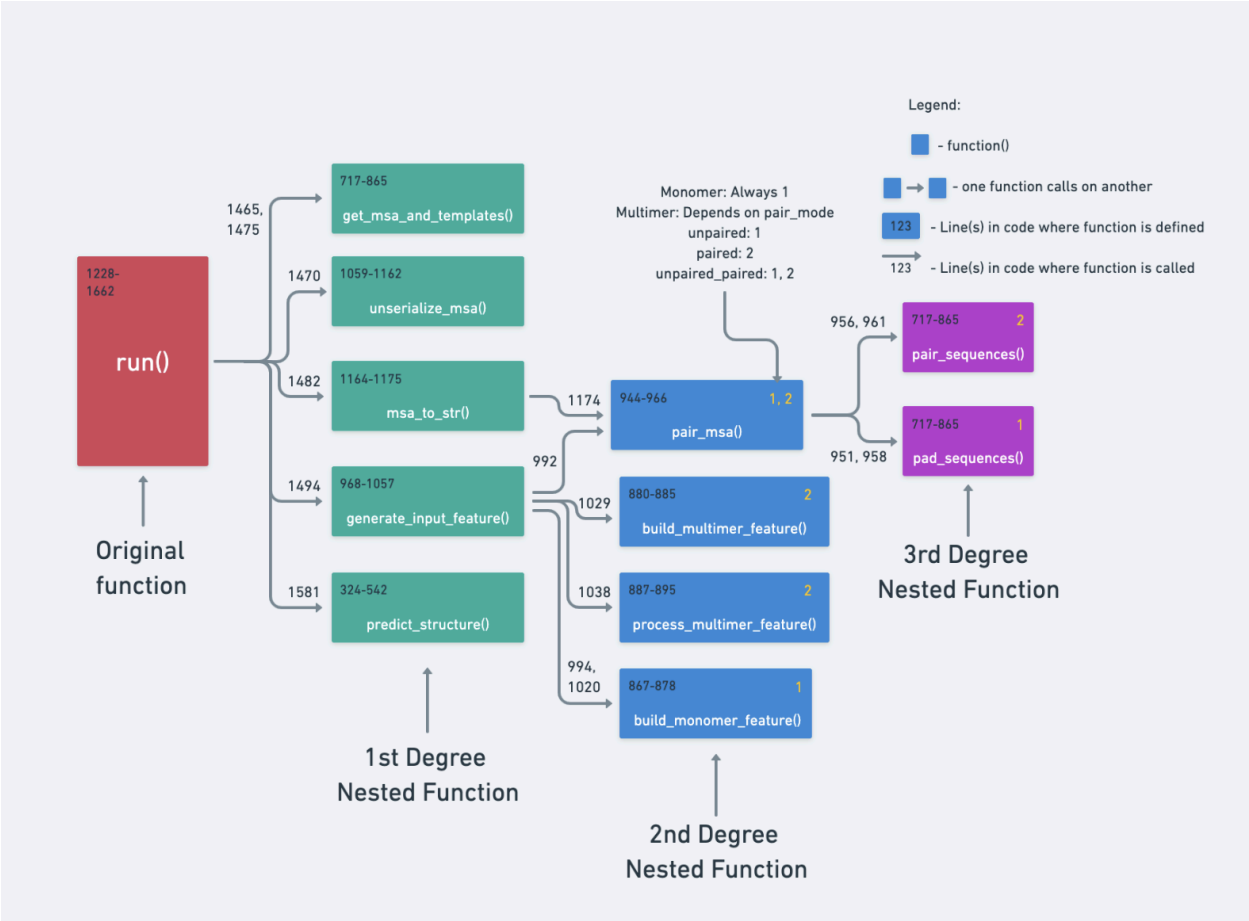


Figure 8. **Visualization of how the run() function is executed.** A visualization of how the run() function is executed, containing information about where each function is defined and called with line numbers and additional info on how the functions are structured when executed within the run function code on Github.

To read Figure 8, start from run() and follow the arrows from top to bottom, left to right, and from box to box (function to function). The program runs these functions in this order: each colored box represents how far each function is called respective to run() (1st, 2nd, 3rd), the numbers inside the boxes represent the lines of code where the function is defined, and the numbers adjacent to the arrows represent the line(s) of code where the function is called upon by the previous function. Additionally, the yellow numbers show which functions are run depending on the pair_mode (which in turn is dependent on monomer vs multimer) [22, 23].

Figure 9. **Block 5.** Screenshot of Block 5 in AlphaFold2.ipynb (Colabfold). [15].

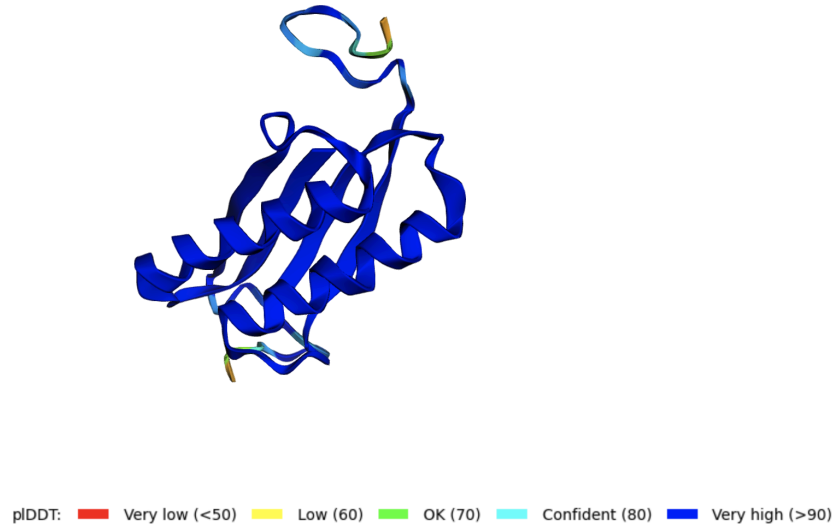


Figure 10a. **Default 3D Predicted Protein Structure.** Screenshot of the 3D protein structure predicted by AlphaFold2.ipynb (Colabfold). [15].

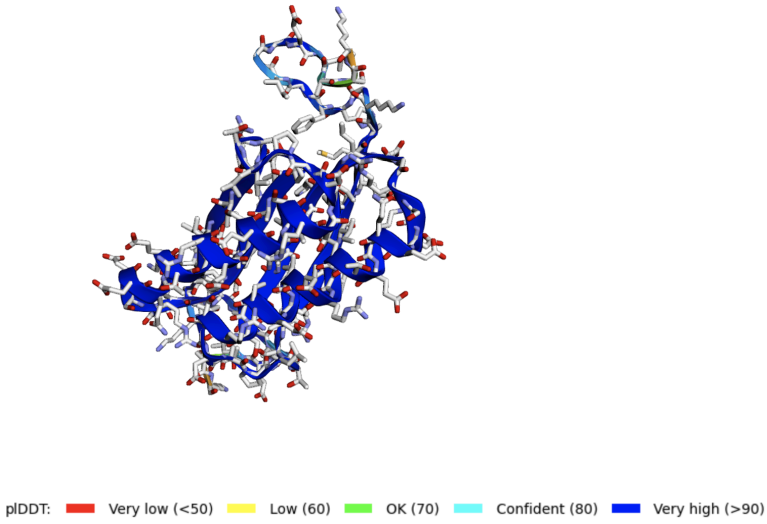


Figure 10b. **3D Predicted Protein Structure with sidechains and mainchains.** Screenshot of the 3D protein structure predicted by AlphaFold2.ipynb (Colabfold) with sidechains and mainchains displayed as well. [15].

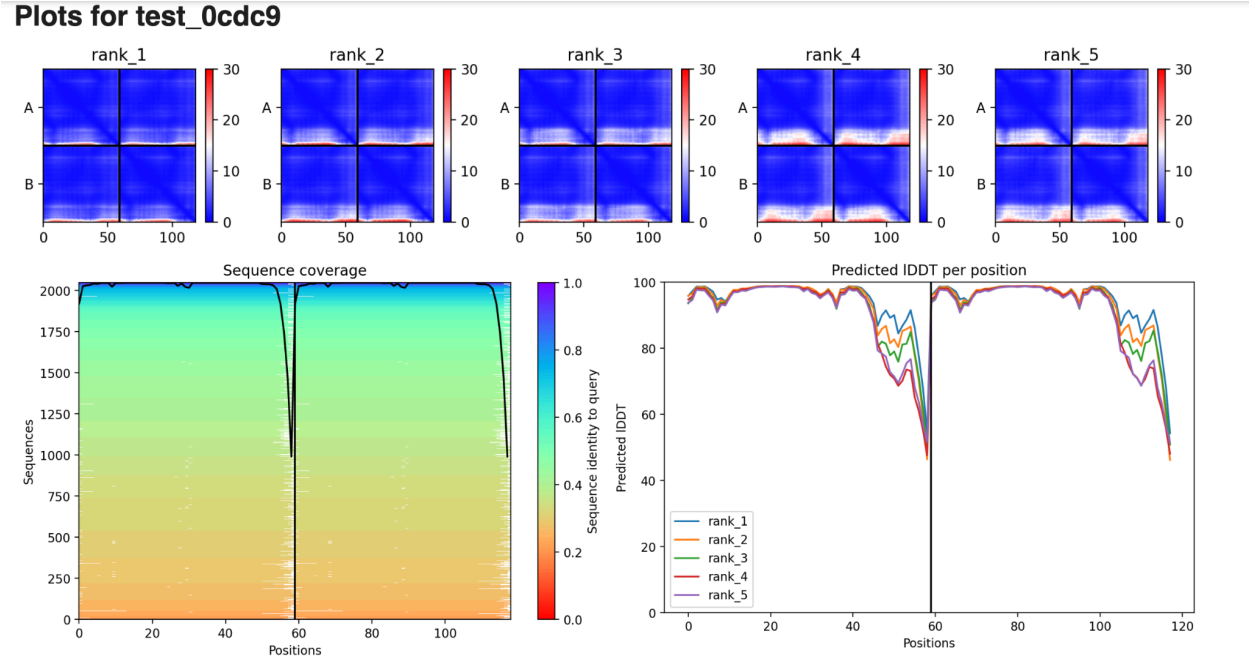


Figure 10c. **Assorted plots for the model**. Screenshot of the plots generated by AlphaFold2.ipynb (Colabfold) for the test model. [15].

The square graphs show the accuracy and correlation of the pairwise representations for every ranked model, the bottom two graphs show MSA coverage (the bluer the line the more related it is to the original) and the confidence level by position of the model for every ranked model [15].

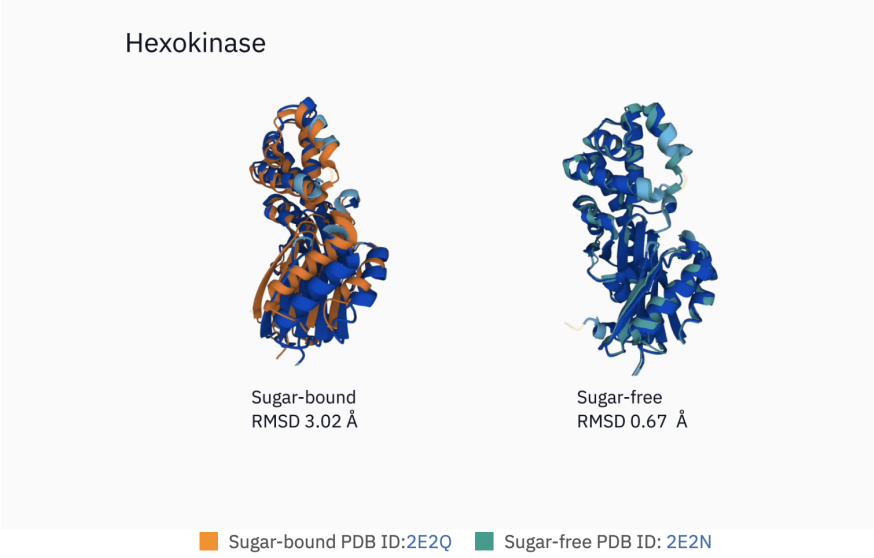


Figure 11. **Accuracy of Sugar-bound vs Sugar-free versions of Hexokinase**. Screenshot of the plots generated by AlphaFold2.ipynb (Colabfold) for the test model. [25].

```
ORIGIN
        1 mttcsrqfts sssmkgscgi gggigggssr issvlaggsc rapstygggl svssrfssgg
       61 acglgggygg gfsssssfgs gfgggygggl gagfggglga gfgggfaggd gllvgsekvt
      121 mqnlndrlas yldkvralee anadlevkir dwyqrqrpse ikdyspyfkt iedlrnkiia
      181 atienaqpil qidnarlaad dfrtkyehel alrqtveadv nglrrvldel tlartdlemq
      241 ieglkeelay lrknheeeml alrgqtggdv nvemdaapgv dlsrilnemr dqyeqmaekn
      301 rrdaetwfls kteelnkeva snselvqssr sevtelrrvl qgleielqsq lsmkaslens
      361 leetkgrycm qlsqiqglig sveeqlaqlr cemeqqsqey qilldvktrl eqeiatyrrl
      421 legedahlss qqasgqsyss revftssss ssrqtrpilk eqssssfsqg qss
```

Figure 12a. **Example of Protein sequence from NIH library**. Screenshot of how a protein sequence would appear in the NIH library. [27].

```
# remove whitespaces
query_sequence = "".join(query_sequence.split())
# Allows for spaces and numbers in query sentence by removing them
query_sequence = ''.join([i for i in query_sequence if not i.isdigit()])
query_sequence = query_sequence.upper()
```

Figure 12b. **Code snippet that fixes the formatting issue**. Screenshot of a code snippet that would fix the formatting issue with the NIH library (the two lines below the second comment). [27].

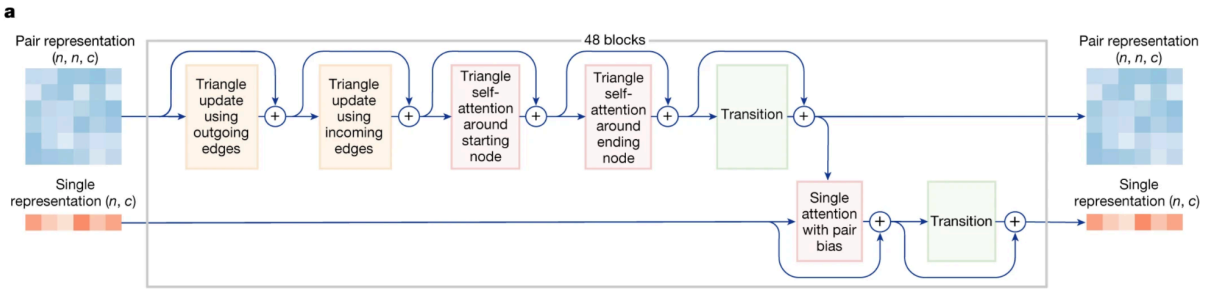


Figure 13a. **Architectural Details of Updated Pairformer Stage**. A visual representation of the Updated Pairformer Stage of AF3. [32].

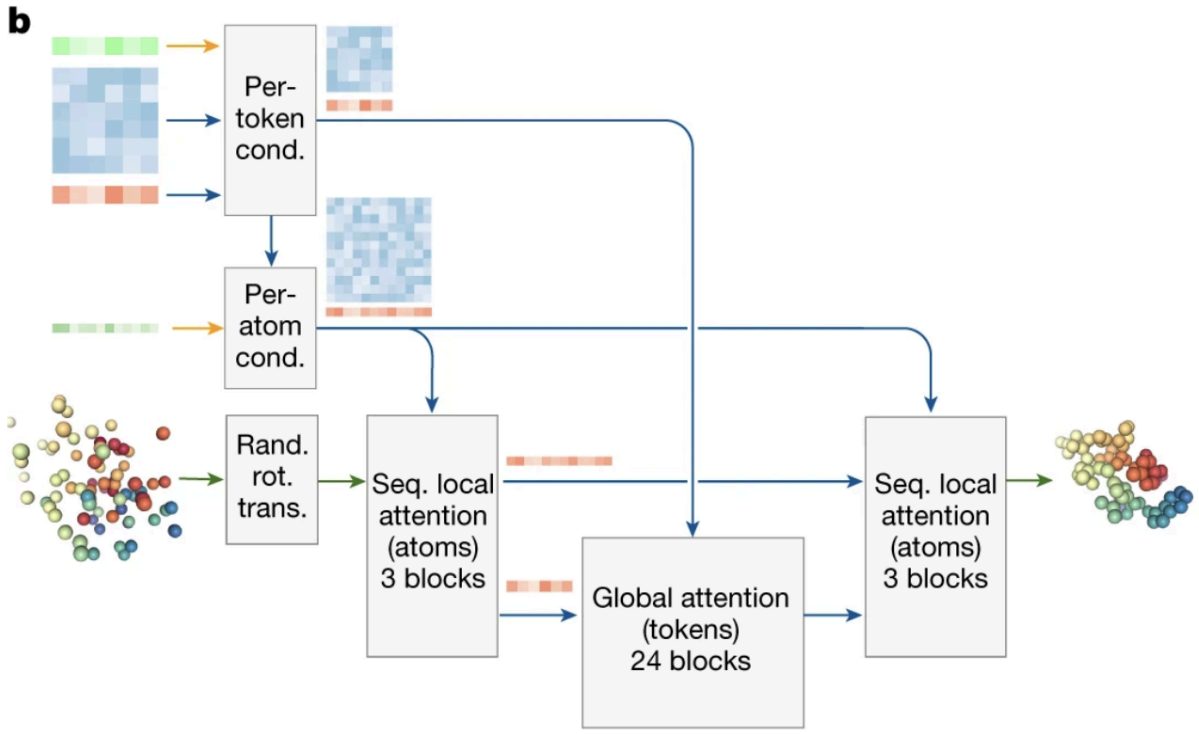


Figure 13b. **Architectural Details of the Diffusion Module**. A visual representation of the new diffusion module of AF3 which predicts atomic coordinates for the structure [32].

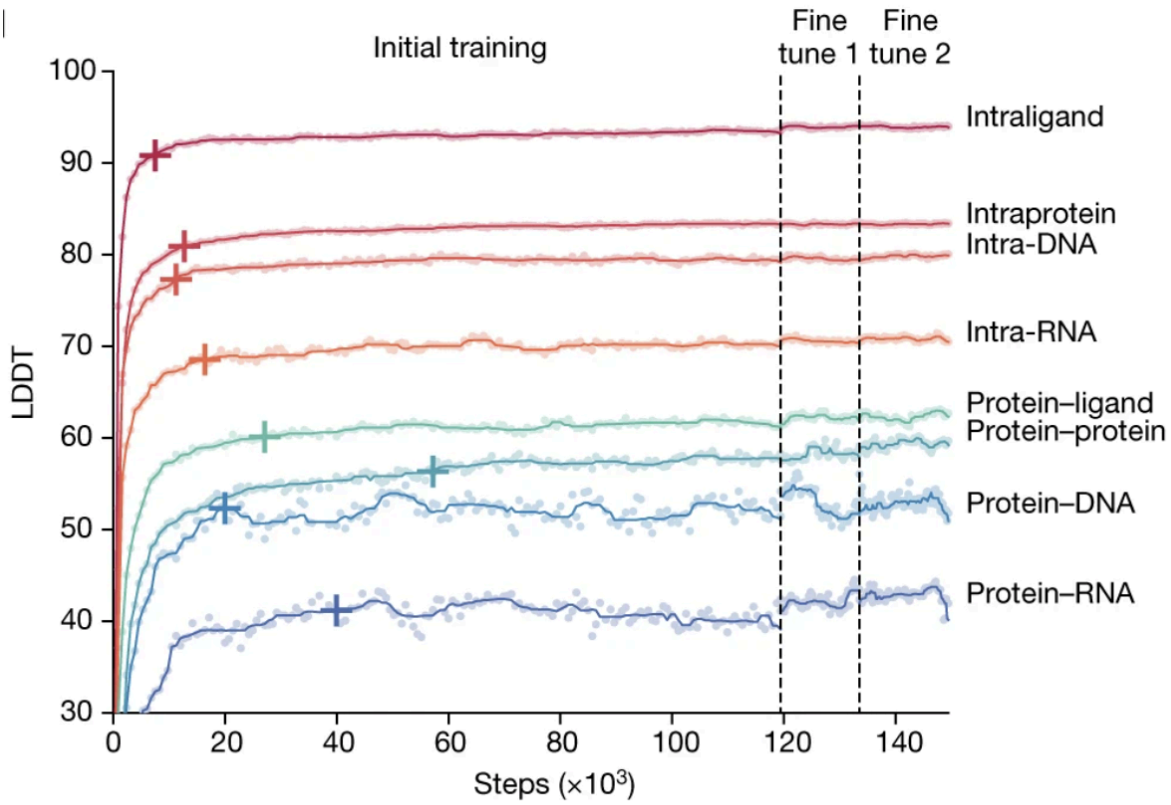


Figure 13c. **Accuracy while training to model different organic structures**. A visual representation of the lDDT accuracy vs training steps for different organic structures [32].