©Copyright 2020
Jonas Pfab

# Deciphering Protein Complex Structures from Cryo-electron Microscopy Maps using Deep Learning

Jonas Pfab

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Computer Science and Software Engineering

University of Washington

2020

Reading Committee:

Dong Si, Chair

Michael Stiber

Clark Olson

Wooyoung Kim

Program Authorized to Offer Degree:
Computer Science and Software Engineering

University of Washington

**Abstract**

Deciphering Protein Complex Structures from Cryo-electron Microscopy Maps using Deep Learning

Jonas Pfab

Chair of the Supervisory Committee:
Assistant Professor Dong Si
Computing and Software Systems

Information about macromolecular structure of protein complexes such as SARS-CoV-2, and related cellular and molecular mechanisms can assist the search for vaccines and drug development processes. To obtain such structural information, we present DeepTracer, a fully automatic deep learning-based method for fast de novo multi-chain protein complex structure determination from high-resolution cryo-electron microscopy (cryo-EM) density maps. We applied DeepTracer on a previously published set of 476 raw experimental density maps and compared the results with a current state of the art method. The residue coverage increased by over 30% using DeepTracer and the RMSD value improved from 1.29Å to 1.18Å. Additionally, we applied DeepTracer on a set of 62 coronavirus-related density maps, among them 10 with no deposited structure available in EMDataResource. We observed an average residue match of 84% with the deposited structures and an average RMSD of 0.93Å. Additional tests with related methods further exemplify DeepTracer's competitive accuracy and efficiency of structure modeling. DeepTracer allows for exceptionally fast computations, making it possible to trace around 60,000 residues in 350 chains within only two hours. The web service is globally accessible at `https://deeptracer.uw.edu`.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# Chapter 1
# INTRODUCTION

## *1.1 Motivation*

Proteins constitute essential building blocks for all life on earth. Within every organism they perform a wide variety of functions from molecule transportation to the provision of cell structure. The determining factor of a protein's functionality is hereby its structure which is given by a unique sequence of amino acids that make up the protein, as well as their three-dimensional arrangement [2]. As a consequence, researchers can draw conclusions about the behavior of a protein solely based on information about its atomic structure. These kind of conclusions can be useful e.g. in the development of new vaccines. For many viruses, viral fusion proteins play a central role in how the virus invades cells. They fuse with the cell and hijack cellular processes such that they will produce virally encoded proteins that replicate the virus' genetic material [3]. In order to prevent such infections, researchers attempt to develop vaccines which target these fusion proteins. This is the case e.g. with the vaccine against the influenza virus. Here, structural information about the fusion proteins can be crucial for researchers to predict their behavior and ultimately find the right vaccine [4].

In order to determine the atomic structure of a protein, this thesis builds upon cryogenic electron microscopy (cryo-EM) data. Cryo-EM allows researchers to capture three-dimensional images of proteins, which describe the density of electrons at a near-atomic resolution. An example of such an image can be seen in Figure 1.1. The technology has gained wide popularity in recent years as an alternative to established structure determination methods, such as X-ray crystallography, due to its improved quality as well as its cheaper and faster process [5]. To derive the atomic structure of a protein based on its 3D

cryo-EM electron density image, researchers currently either have to manually fit the atoms or resort to existing prediction methods. Manually fitting the atomic structure represents an enormous effort as proteins usually consist of several thousand atoms and becomes virtually impossible for larger structures. Therefore, there is a great demand for a prediction method that automatically fits the atomic structure to a cryo-EM density map. However, existing prediction tools, which are discussed in more detail in Section 2.3, often lack in terms of accuracy and performance, predict only certain parts of the atomic structure, or require extensive manual processing steps. Particularly, due to the ability of cryo-EM to capture multiple large proteins in the course of a single study [6, 7], a fully automated, efficient tool to predict atomic structures would be crucial to increase the throughput of the technology.

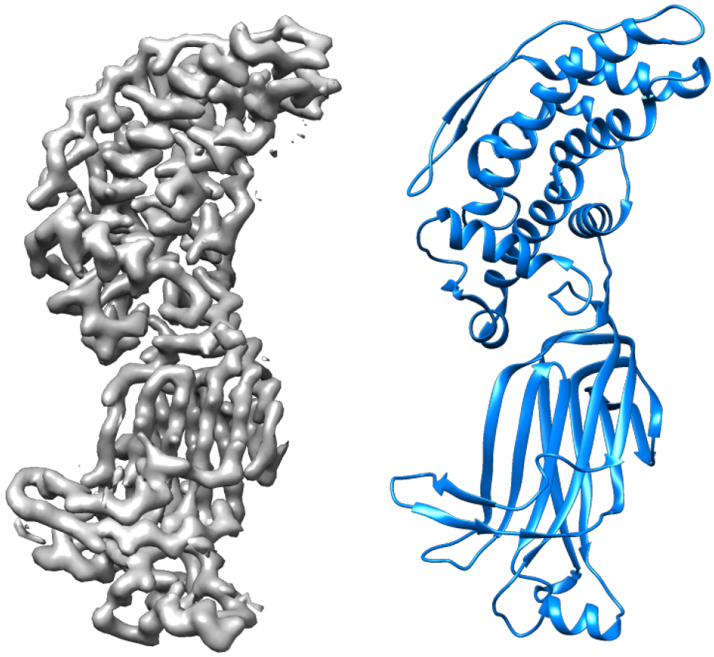

Figure 1.1: Cryo-EM density map (EMD-6272) of rotavirus protein VP6 on the left next to its fitted atomic structure (PDB-3j9s) on the right.

### *1.2 Problem Statement*

The goal of this thesis is to develop the DeepTracer, a software tool which predicts the complete atomic structure of a protein fully automated solely based on its cryo-EM density map as well as its amino acid sequence. No manual processing of the density map in any way should be necessary and the tool should require no further parameters to run predictions. The core of the prediction method will be a U-Net deep learning model which in combination with complex pre- and post-processing steps should achieve results similar or better than existing prediction methods with a significantly improved runtime.

### *1.3 Structure of Work*

In order to introduce important concepts as well as to review existing prediction methods, Chapter 2 elaborates on the background of the thesis work. Next, Chapter 3 presents all prediction steps of the DeepTracer as well as the implementation of its website. The performance of the DeepTracer is evaluated in Chapter 4 by comparing its prediction accuracy and runtime against existing methods using various metrics. Finally, in Chapter 5 we discuss the implications of our work, list possible future work, and draw a final conclusion.

# Chapter 2
# BACKGROUND

In order to gain a better understanding about the thesis' research area, this chapter covers some of its fundamental concepts. This includes the prevailing data types that the DeepTracer works with, which are cryo-EM density maps (Section 2.1) and atomic protein structures (Section 2.2). Additionally, it presents existing protein structure prediction methods, which we later use as a point of comparison to evaluate the DeepTracer's performance, in Section 2.3.

## *2.1 Cryo-EM Density Maps*

The central data type of this research is cryo-EM density maps. They are used as a basis for the protein structure prediction and understanding their structure as well as the meaning behind their values is crucial. Therefore, this section takes a closer look at what exactly density maps are, how they are stored, and how their content can be interpreted.

Cryo-EM density maps are captured by cooling a sample to cryogenic temperatures and then projecting beams of electrons from varying angles onto it. Based on the diffraction of the electrons, a three-dimensional image, the electron density map, can be created [8]. This density map can then be stored on a disk using the MRC file format [9]. It tracks the volume data of the density map in a 3D array, also referred to as a grid. Each value or voxel (pixel in 3D) on the grid represents the electron density, meaning the probability that an electron is present, for its location. Besides the volume data itself, an MRC file provides additional meta data about the map. This includes information necessary to transform the indices of a voxel to xyz coordinates that can be used to define the location of atoms. In particular, the voxel size determining the length of each voxel for all directions as well as the origin

which specifies the offset of every voxel for each axis. Both of those values are measured in Angstrom. For the voxel indices $i$, $j$, and $k$, voxel size $s$, and origin $o$, we can calculate the xyz coordinates as shown in (2.1). Similarly, Equation (2.2) shows how we can find the ijk indices of the voxel that lays at an xyz coordinate. The result is only an approximation as coordinates can express positions more accurately using floating point values compared to the integer indices of density maps.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} (k \times s_x) - o_x \\ (j \times s_y) - o_y \\ (i \times s_z) - o_z \end{pmatrix} \tag{2.1}$$

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} \lfloor (z - o_z) \div s_z \rceil \\ \lfloor (y - o_y) \div s_y \rceil \\ \lfloor (x - o_x) \div s_x \rceil \end{pmatrix} \tag{2.2}$$

When we visualize an MRC file using UCSF Chimera we have to specify a density threshold level. Only voxels with a density value above the specified threshold are visible. Figure 2.1 shows the EMD-6272 density map visualized at four decreasing threshold levels. Note that Chimera automatically smoothens the density map which is why it does not appear pixelated. The black rectangle around the map indicates the size of the grid, hence the dimensions of the 3D array containing the volume data. Below each map is the histogram of all density values with a black line to the left indicating the density value of zero and a gray line to the right specifying the threshold level used for the visualization. We can note that more voxels are shown as the threshold level decreases. This is only logical, as the number of voxels above a threshold can only increase for a lower threshold level. When the threshold level is set to zero all voxels are shown as every voxel has at least a density value of zero. This explains why the map is simply a rectangle of the size of the volume data grid itself in the rightmost visualization.

The central deposit of cryo-EM density maps is the EMDataResource website [10] which

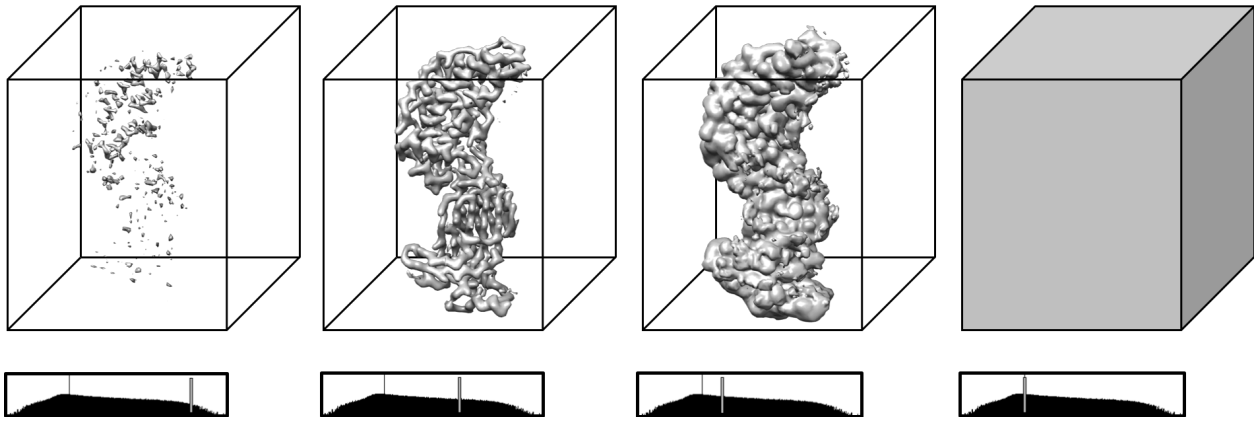

Figure 2.1: EMD-6272 density map visualized in UCSF Chimera. Maps are displayed with a decreasing density threshold from left to right. The enclosing rectangle defines the size of the grid. The histogram below indicates the number of voxels at a certain density value. The black line on the left in the histogram shows where the threshold level of zero is and the right line shows which threshold level is used for the visualization.

serves as a unified data resource for 3D electron microscopy data. Its database contains several thousand high-resolution density maps which are freely available to download.

### *2.2 Protein Structure*

The final output of the DeepTracer is a molecular structure prediction for the protein captured in the input density map. Therefore, this section covers the fundamentals of what the molecular structure of a protein consists of to better understand what this prediction comprises.

As mentioned in the introduction, all proteins consist of one or more sequences of the same 20 types of amino acids that are connected through peptide bonds. The linear sequence of amino acids specifies the primary structure of the protein. The secondary protein structure is defined as the three-dimensional form of local segments of this sequence. Here, we differentiate between α-helices where the amino acids form a helical shape, β-sheets which usually form a pleated sheet, and loops which follow no distinguishable pattern. When we visualize structures using UCSF Chimera, secondary structures can be made visible in the

ribbon view. In Figure 2.2, we can see examples for all three secondary structure elements in ribbon as well as atom view.

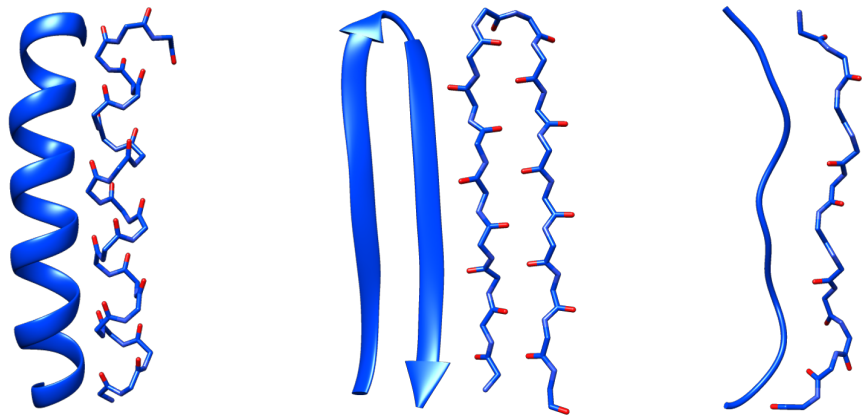

Figure 2.2: Visualization of an α-helix (left), β-sheet (center), and a loop (right) in ribbon view on the left side and atom view on the right side.

The tertiary structure of the protein defines the three-dimensional arrangement of the amino acids, meaning the positioning of their atoms. The position of each atom is determined through xyz coordinates. Each amino acid consists of the same backbone atoms, namely a nitrogen (N), carbon-alpha (Cα), carbon (C), and oxygen (O) atom. Additionally, every amino acid has a side-chain which is attached to its Cα atom. The atoms that make up the side-chain vary based on the type of the amino acid. However, for the same type of amino acid the side-chain atoms are always identical. In Figure 2.3, we can see the atoms of an example tyrosine amino acid.

Molecular protein structures are stored using the PDB file format where the structure's information is written to a human readable text file. The atoms are simply listed with their xyz coordinates as well as additional information. As a data resources for PDB files containing fitted molecular protein structures we utilize the RSCB protein data bank [11]. The fitted structures are used to train our deep learning model as well as to compare the accuracy of predictions made by the DeepTracer.

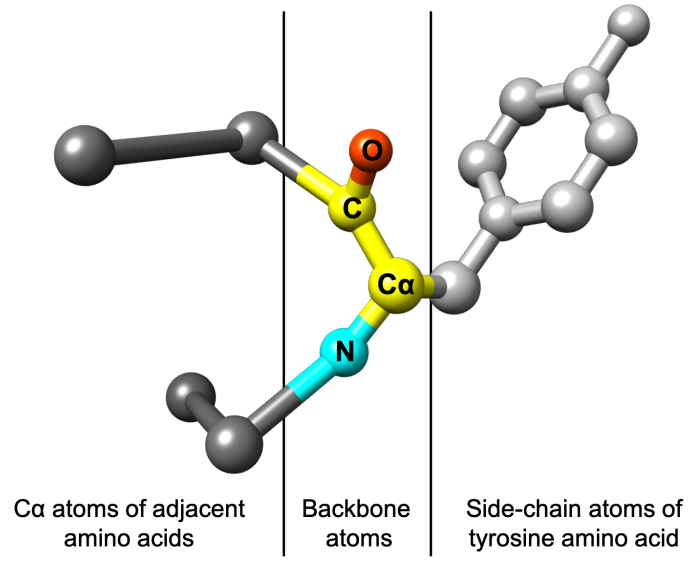


Figure 2.3: Tyrosine amino acid and Cα atoms of adjacent amino acids visualized in UCSF Chimera.

### *2.3 Related Work*

Due to the potential benefits of an efficient tool to accurately determine protein structures, researchers have invested significant effort in the development of diverse methods. Generally, these methods follow one of two major directions: Template-based and de novo. Template-based modelling refers to methods which require the solved structure of a homologue protein to perform the structure determination. While this makes predictions computationally less expensive and generally more accurate, it has the obvious disadvantage that it only works if the structure of a homologue protein is known. Therefore, this thesis focuses on the more challenging de novo prediction.

The majority of de novo prediction methods attempt to model the three dimensional structure based on the amino acid sequence of the protein [12, 13, 14]. The benefit of this approach is the vast availability of sequences, as their determination is relatively easy and has been possible for decades [15]. However, inferring the three dimensional structure

solely based on sequences is computationally very expensive, making it impossible to predict larger protein complexes [16]. Additionally, due to the limited information encoded in the amino acid sequence, such prediction methods generally lack in accuracy. As a consequence, researches have explored structure predictions from cryo-EM data in order to address these limitations.

Existing de novo cryo-EM prediction methods form the fundamental point of comparison for the performance of the DeepTracer. Particularly, the map to model function of the Phenix software suite constitutes our main focus as it also performs predictions fully automatic. Phenix is a widely popular molecular prediction suite that provides tools for a variety of problems related to molecular structure determination [17]. A paper published in 2018 presents its map to model function which we simply refer to as the Phenix method [18]. It uses a combination of automatic density map processing including sharpening and segmenting algorithms, and several map-interpretation methods which produce a collection of possible fits of the amino acid sequence in the density map segments. The final predicted structure is then derived from the best possible fit. As input data, Phenix expects the amino acid sequence and density map alongside its resolution. Note that this method does not use any machine learning techniques in its implementation. The authors of the paper presenting the map to model function provide a set of 476 density maps in combination with their predictions which we utilize as a test set for the DeepTracer.

RosettaES is another de novo prediction tool building upon cryo-EM data. Developed at the University of Washington, this method applies a sampling strategy to determine the molecular structure of a protein. Through fragment-based sampling, RosettaES finds a pool of possible structures which adhere to biological geometric rules while providing a good fit to the cryo-EM density map. The final solution is then determined through the use of an energy function.

# Chapter 3

# METHODS

To predict the molecular structure of a protein solely based on its cryo-EM density map and amino acid sequence we have to perform a variety of tasks. We need to pre-process each density map for the deep learning model, apply the model on the density map, and then transform its output into a protein structure. An overview of all prediction steps involved in this process is given in Figure 3.1. In this chapter we discuss each of these steps in detail, starting with the pre-processing steps in Section 3.2 and continuing with the deep learning model and post-processing steps in Sections 3.3 and 3.4. But, before we begin to look into each prediction step individually, Section 3.1 examines how the DeepTracer coordinates and manages the execution of the prediction pipeline.

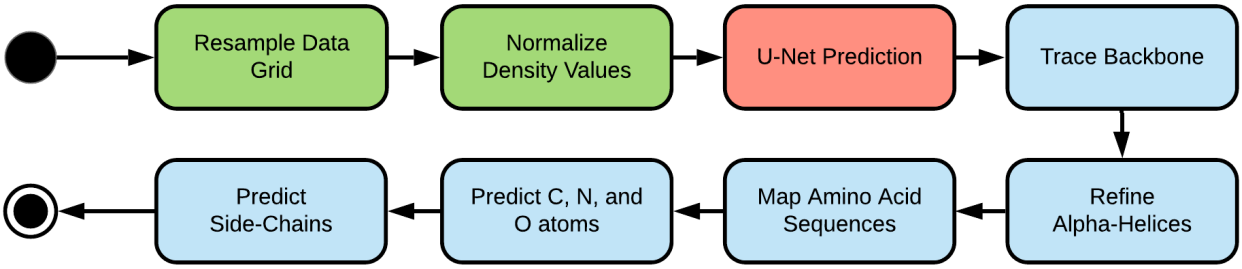


Figure 3.1: Pipeline containing all prediction steps of the DeepTracer. Pre-processing steps are depicted in green while the machine learning part is marked in red and all post-processing steps are shown in blue.

## *3.1 Prediction Pipeline Processing*

Coordinating the execution of each step in the prediction pipeline does not appear to pose a challenging software problem. After all, we can simply call a list of methods and provide the

output of each function as the input for the next one. However, we can quickly notice that there is a range of issues that such an implementation does not handle well. What happens if the last step requires the output of the first one? What if one step in the pipeline fails? How can we create checkpoints? What if we want to add/remove a prediction step? In order to see how the DeepTracer handles these issues this section takes a closer look at how the prediction pipeline is processed.

The central goal in mind, when deciding about the design of the DeepTracer, is an easy modification of the prediction pipeline. As other researchers work on this project changes to the pipeline are frequent which makes it important that their implementation is as easy as possible. In order to realize this we have to ensure that each prediction step is as modular as possible, meaning that it has few dependencies with other steps. We achieve this by introducing two entities. The `DeepTracer`, responsible for fetching data, managing executions, and updating results, as well as the `TracingContext`, which tracks the entire context of a prediction including all inputs, intermediate, and final results. A more detailed depiction of how these entities interact is given in the class diagram shown in Figure 3.2.

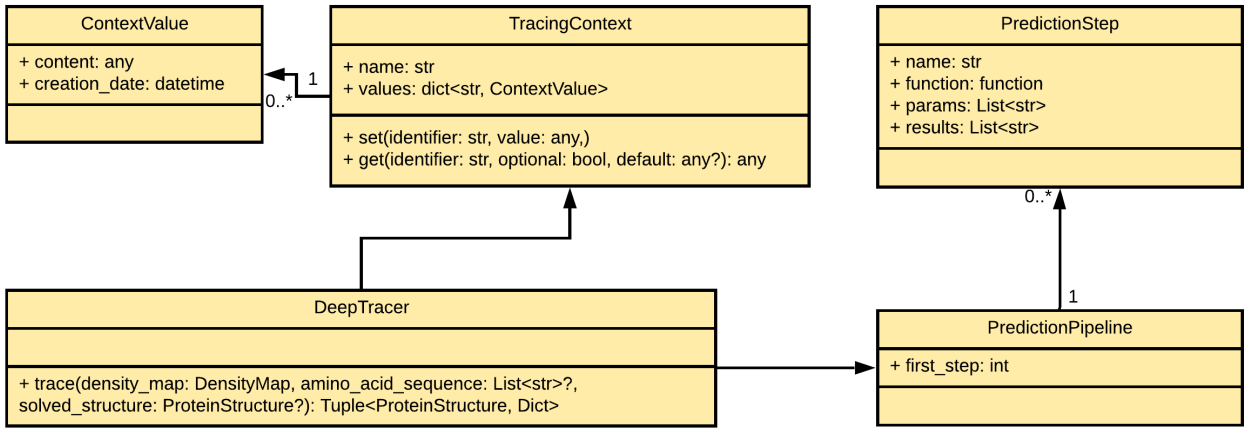


Figure 3.2: UML class diagram of the classes that are relevant for processing the prediction pipeline.

We can see that the `TracingContext` class has two properties: A name which corresponds

to the name of the density map and a dictionary containing the context's values. All values from a context are stored within the `ContextValue` class which additionally tracks the creation date of the value. The `TracingContext` class is referenced by the `DeepTracer` which uses it to track inputs and results throughout the prediction. The `DeepTracer` class has one method which can be invoked in order to trace density maps. It expects a density map, amino acid sequences used to refine predictions, as well as an optional solved protein structures used to calculate the prediction accuracy. The result of the `trace` function is a tuple containing the predicted structure as well as an evaluation stored in a dictionary. The `DeepTracer` references the `PredictionPipeline` class which tracks a list of `PredictionStep` objects. It also has a first step property which determines at which step the prediction should start. This is useful when a prediction is continued from a checkpoint. A detailed explanation on storing/resuming from checkpoints is given below. Each `PredictionStep` object has several properties related to its execution. First up is the name of the step. Next, is the `function` which points to the actual function that performs the prediction step. The `params`, and `results` properties both store a list of string identifiers used by the `DeepTracer` to fetch required arguments and store results of the step using the `TracingContext`. It is important that the function expects the arguments in exactly the same order as stored in the `params` property and returns results in the same order as specified in the `results` property.

In the activity diagram shown in Figure 3.3 we can see how the DeepTracer processes the prediction pipeline. It begins by loading all input density maps that should be traced from the disk. Next, it checks whether all prediction steps have been executed. If that is not the case then it fetches the next prediction step, fetches the required arguments from the tracing context as specified by the step, and calls the step's setup function. Then, it checks whether the step can be executed in parallel. If that is the case the step's execution function is invoked in parallel for every input density map. Otherwise, it is called sequentially for every map. After the prediction step completed, its tear down function is invoked and the results of the step are written to the tracing context. Next, the DeepTracer checks again if there are any prediction steps left. Once all steps are executed all final results are fetched

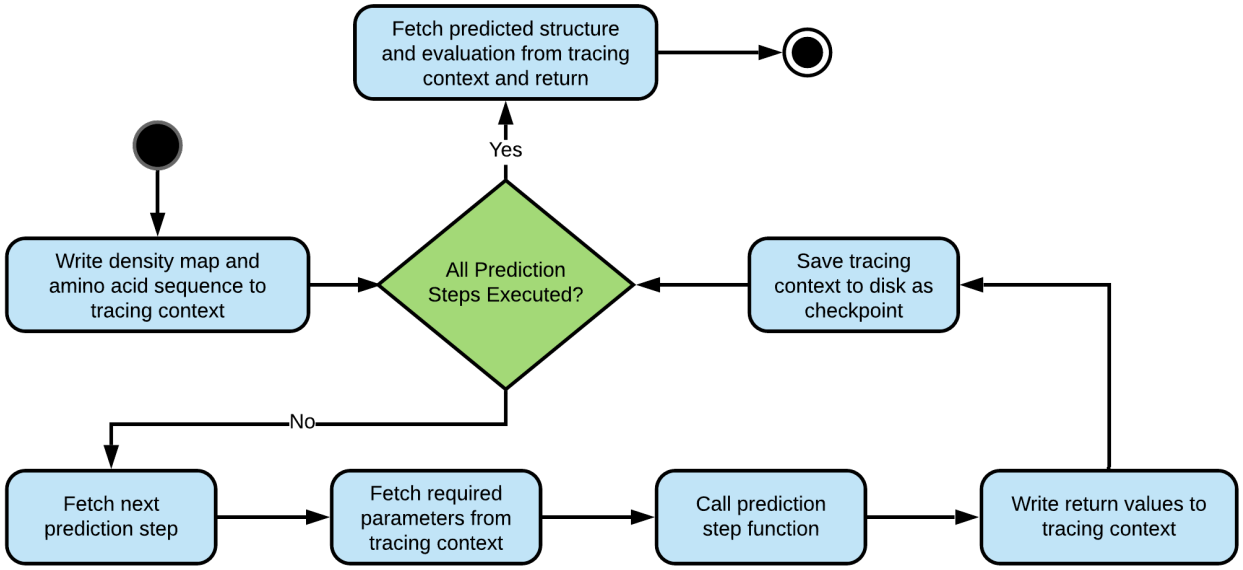


Figure 3.3: UML activity diagram depicting how the DeepTracer processes its prediction pipeline.

from the tracing context and returned as the results.

The presented design allows the DeepTracer to centrally manage the execution of all steps. This means that e.g. error handling and memory optimizations can be implemented in the DeepTracer without the need for any modification of the prediction steps. Additionally, it allows for a straightforward implementation of a checkpointing system. Through the use of the pickle library [19], we can store the `TracingContext` object to the disk as a checkpoint after each prediction step. In order to resume from a checkpoint we can then just load the context from the disk and continue the execution. However, the most important benefit of the chosen design is the simple modification of the prediction pipeline. In order to add a new prediction step no modifications are necessary to any existing parts of the project. The code snippet shown in Figure 3.4 contains all of the code that is necessary to append a new post-processing step to the prediction pipeline.

```python
from deep_tracer.postprocessing import pipeline

@pipeline.step(
    name='Trace Backbone',
    params=['density_map', 'ca_unet_prediction'],
    results=['predicted_structure']
)
def trace_backbone(density_map, confidence_maps):
    # Implementation of step
    return predicted_structure
```

Figure 3.4: Python code snippet showing how a new step can be added to the prediction pipeline.

## *3.2 Pre-Processing*

The goal of the pre-processing steps is to prepare the cryo-EM density maps for the neural network. As we use experimental maps for the training as well as prediction these steps are crucial as maps can differ significantly in terms of shape, quality, resolution, and more. Therefore, we need to process the maps and convert them into a consistent format such that the neural network can understand connections across density maps. The pre-processing steps that achieve this are examined in Sections 3.2.1, 3.2.2, and 3.2.3.

### *3.2.1 Data Grid Resampling*

The first pre-processing step is concerned with standardizing the voxel size of all density maps. As mentioned in Section 2.1, the grid storing the volume data of the density map has an associated voxel size which determines the size of a single grid element or voxel in Angstrom. Without standardizing this voxel size to a fixed value the neural network could make no conclusions about how far two voxels are apart from each other making it difficult to predict the location of any amino acids. Therefore, this step ensures that each density maps has a voxel size of exactly 0.5Å. The value 0.5 was chosen as a trade-off between prediction precision and memory usage of the resulting grids based on several rounds of testing.

In order to set the voxel size of a density map to 0.5Å we cannot simply change the meta data of the map. We have to resample the volume data onto a new grid in which each voxel represents 0.5Å. An example of a resampling process from an origin grid to a grid with half the voxel size is shown in Figure 3.5. As we can see, the shape of the volume remains the same, however, we require eight times the number of voxels to represent it. To realize the resampling step, the DeepTracer utilizes UCSF Chimera [20]. First, it creates a new grid of the same size in Angstrom as the original density map, but with a voxel size of 0.5Å. Then, it uses Chimera's resampling command [21] to resample the original density map onto the newly created grid.

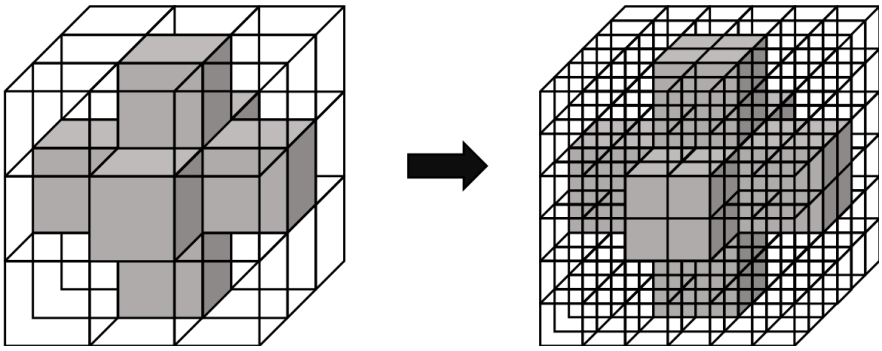

Figure 3.5: Visualization of the resampling process from an origin grid onto a grid with half the voxel size.

### *3.2.2 Density Value Normalization*

The absolute value of a voxel in itself contains little information. Only relative to the density values of other voxels can we make conclusions about the protein structure. Consequently, we can normalize the density values without the risk of loosing any information, which constitutes the second and final pre-processing step of the DeepTracer. The normalization process makes sure that the range of density values is identical for all density maps. In the case of experimental maps this range can initially differ substantially with some maps containing values from -0.1 to 0.1 and other maps ranging from -10 to 20.

In order to normalize values we can usually divide each value by the overall highest value. However, this is problematic for some density maps as there are outlier density values which have values that are much higher than all other values. If we were to divide all other values using such an outlier, all other density values would end up being close to zero. Therefore, we use the 95th percentile of the density values to divide all other values with. Afterwards, we simply set the few values that are greater than 1 to 1. Additionally, we also set all values below 0 to 0 as they contain no valuable information for our use-case. This leaves us with a range from 0 to 1 which contains all density values. An example of the density value histograms before and after the normalization step can be seen in Figure 3.6.

Figure 3.6: Histograms of the EMD-6272 density map depicting the relative frequency of density values before (top) and after (bottom) normalization. The vertical black line in the upper histogram indicates the value 0 and is not visible in the lower one as there are no density values below 0 anymore.

### *3.2.3 Grid Division*

In order for the deep learning model to make any predictions we have to feed the density map to the input layer of the model. Therefore, we have to make sure that the dimensions of the volume data grid of the density map are identical to those of the input layer. However, the dimensions of the grid vary from map to map which means that we have to modify the grid in order to match its dimensions to the input layer. Unfortunately, we cannot simply scale the density map such that it fits the input layer as this would change the size each voxel

represents in Angstrom, which has to remain one as mentioned in Section 3.2.1. Therefore, we divide the grid into multiple sub grids each the size of the input layer of the deep learning model.

We divide the volume data grid into sub grids of size $64^3$. The number 64 was chosen as it is creates a relatively small input layer that is still broad enough for the deep learning model to detect larger patterns, such as secondary structure elements. Dividing the grid, however, can aggravate predictions in areas close to the border of sub grids as relevant information from neighboring voxels might be cut off. Therefore, we introduce a core grid of size $50^3$ in the center of each sub grid. Although each sub grid has a size of $64^3$ we only use the predictions from the inner core grid. Consequently, when dividing the grid we have to overlap the $64^3$ sub grids such that core grids of all sub grids cover the entire original grid without overlap. An example of such a division for a two-dimensional grid can be seen in Figure 3.7.

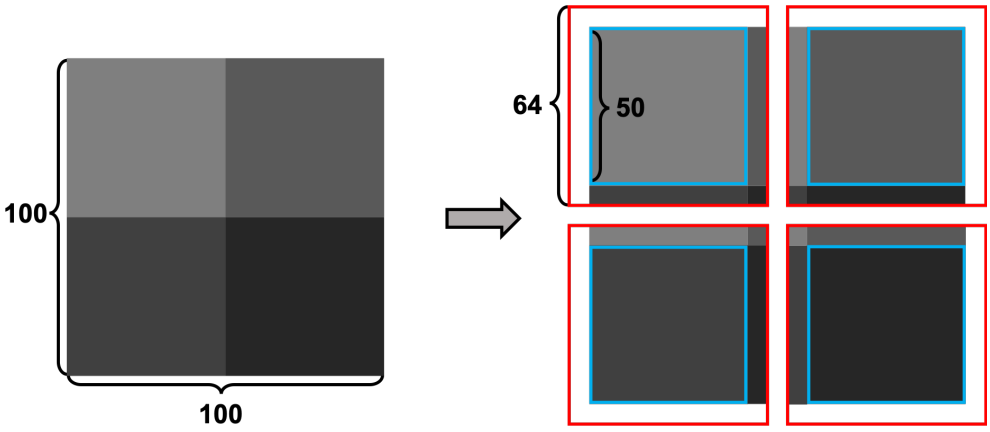


Figure 3.7: Visualization of the division of a $100^2$ grid into four grids each with a dimension of $64^2$ and a core size of $50^2$. The red lines indicate the complete sub grids while the blue lines show their cores.

### *3.3 U-Net Deep Learning Model*

The deep learning model represents the central entity in the structure prediction. Its job is to predict not only the locations of amino acids but also secondary structure positions

and amino acid types. To explain its realization, we start by looking at its architecture in Section 3.3.1. Then, we move on to the data collection and finally describe the model's training process in Section 3.3.2 and 3.3.3.

### *3.3.1 Model Architecture*

In this subsection we will take a closer look at the architecture of the deep learning model as used by the DeepTracer. To do so, we start by looking at the high-level design of the model and then examine the details of the U-Net architecture.

As mentioned in the beginning of this section, the deep learning model should predict multiple things, including the atom positions, secondary structure elements, as well as the amino acid types. Additionally, for the post-processing step which connects the predicted atoms described in Section 3.4.2, the model needs to predict the backbone location of the protein structure. For each of those predictions we use separate U-Nets which are all combined to a single model as shown in Figure 3.8. The input of the model is the $64^3$ volume data grid from the pre-processed density map. The atoms U-Net is responsible for predicting whether each voxel contains either a carbon alpha atom, a nitrogen atom, a carbon atom, or no atom at all. Therefore, the output of this U-Net has four channels, one for each predicted class. The backbone U-Net predicts for each voxel if it is part of the backbone, meaning either carbon alpha, carbon, or nitrogen atom, part of a side chain, or not part of the protein at all, which leads to three different output channels. The secondary structure U-Net is responsible for predicting the secondary structure of each voxel. Therefore, we have a four channel output for loops, sheets, helices, and no structure. Finally, the amino acid type U-Net predicts the amino acid type for every voxel. As there are 20 different types of amino acids found in nature we have 21 output channels one for each type and one in case the voxel is not part of the protein.

Now, that we have seen how the different U-Nets are connected to form the DeepTracer's deep learning model, we can take a closer look the U-Net itself. The U-Net is a convolutional network architecture developed by researchers at the University of Freiburg. Its name derives

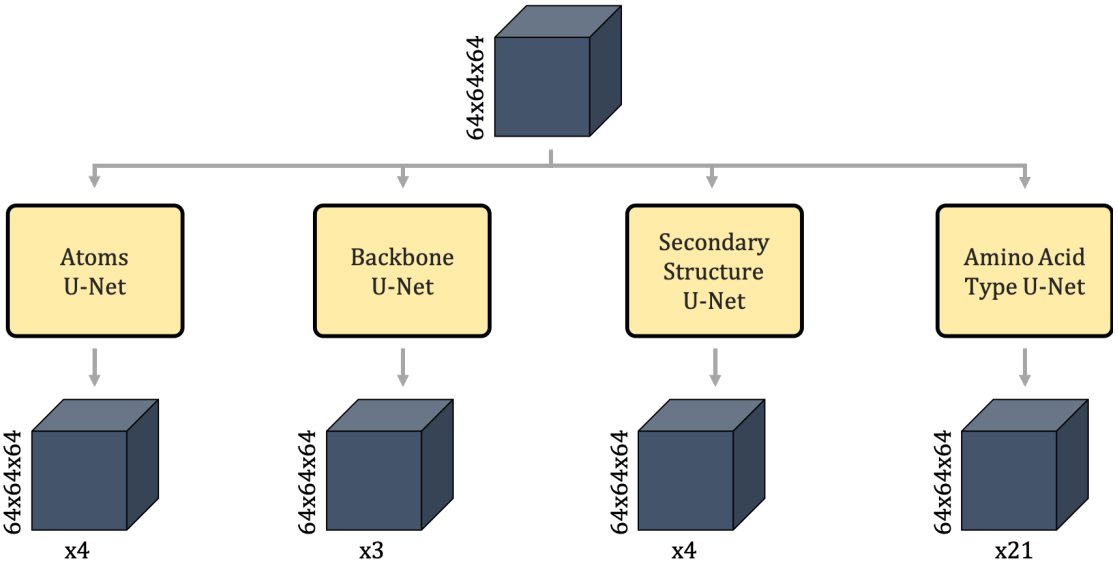


Figure 3.8: Architecture overview of the DeepTracer's deep learning model consisting of four parallel U-Nets. The gray boxes show the input and output maps of the model, with their dimensions noted to the left and the number of channels marked below.

from the U-shape of its architecture. The U-Net excels in fast and precise image segmentation tasks particularly for biomedical applications [1]. For the DeepTracer we modified its original 2D architecture for 3D images. The detailed architecture of the model used by the DeepTracer can be seen in Figure 3.9. The pre-processed cryo-EM density maps are fed to the $64^3$ input layer. The output layer has the same $64^3$ shape with `N` different channels. The number of channels varies between each parallel U-Net (see Figure 3.8).

### *3.3.2 Data Collection*

Before we can begin training the U-Net model, we first have to collect a training dataset. Previous projects, such as [22] use simulated density maps to train their neural networks. However, in order for the network to learn common noise patterns in cryo-EM density maps we decided to use experimental maps. The maps were downloaded from the EMDataResource website [10] in combination with their fitted protein structures that serve as the ground truth

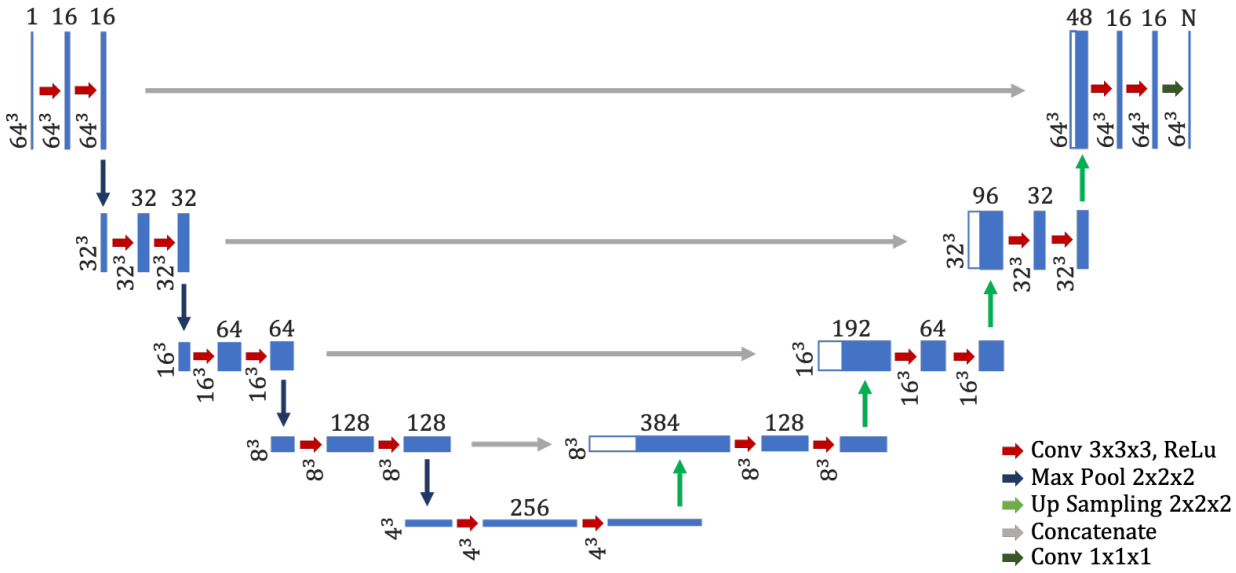


Figure 3.9: Architecture of the U-Net as used by the DeepTracer. The blue boxes show the output maps of the different layers where the dimensions of the maps are depicted on the left and the number of channels is depicted on top. A more detailed description can be found in [1].

in the training process and were fetched from RCSB Protein Data Bank [11]. As this work focuses on high resolution maps we only used density maps with a resolution of 4Å or better. In total, we downloaded 1,800 experimental density maps and their corresponding solved PDB structures.

In addition to pre-processing the density map as described in Section 3.2, we also have to label the maps before we can start training the model. In order to do so, we create masks with the same dimensions as the grid of the density map, which provide a label for each voxel. The labels of the masks are hereby created based on the ground truth structures of each density map. As shown in Figure 3.8, the model has four different outputs, for each of which we have to create separate masks. The atoms mask should provide a label for each voxel whether or not it contains a Cα, C, or N atom. Therefore, we filter out these atoms from the protein structure, calculate the corresponding grid indices for their location

using Equation (2.2), and set that voxel and all directly neighboring voxels to the value representing the atom (1 for Cα, 2 for C and 3 for N atoms). In Figure 3.10 we can see a portion of the atom mask next to the corresponding ground truth structure.

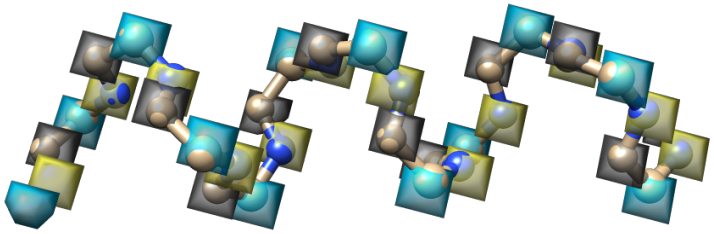

Figure 3.10: Portion of the atom mask containing backbone atoms for part of a helix from the PDB-6NQ1 structure. The gray labels indicate carbon alpha atoms, the blue labels carbon atoms, and the yellow labels nitrogen atoms.

The masks for the backbone, secondary structure, and amino acid type U-Net, are created in a similar fashion. The backbone mask filters all backbone atoms and side chain atoms and sets the respective voxels and all surrounding voxels with a distance of 2 to 1 for backbone and 2 for side chain. To create the secondary structure mask we filter all atoms for helices, sheets, and loops and then set all voxels with a distance of 4 surrounding the atoms to 1 for loop, 2 for helix, and 3 for sheet. Finally, for the amino acid type mask, all Cα atoms for each of the 20 amino acid types are filtered out and all surrounding voxels within a distance of 3 are set to a value between 1 and 20, where each value corresponds to a specific amino acid type. An example of all masks can be seen in Figure 3.11.

The density maps that are added to the dataset first undergo the pre-processing steps described in Section 3.2, which include splitting each density map into sub-maps of size $64^3$. However, this splitting process can become problematic when the grid size of different density maps varies significantly, as larger maps end up producing many more data entries than small ones leading to an over-representation in the final dataset. To tackle this issue we set a fixed number of sub-maps (100,000) that we want to add to the dataset and sample the same number of sub-maps from each density map.

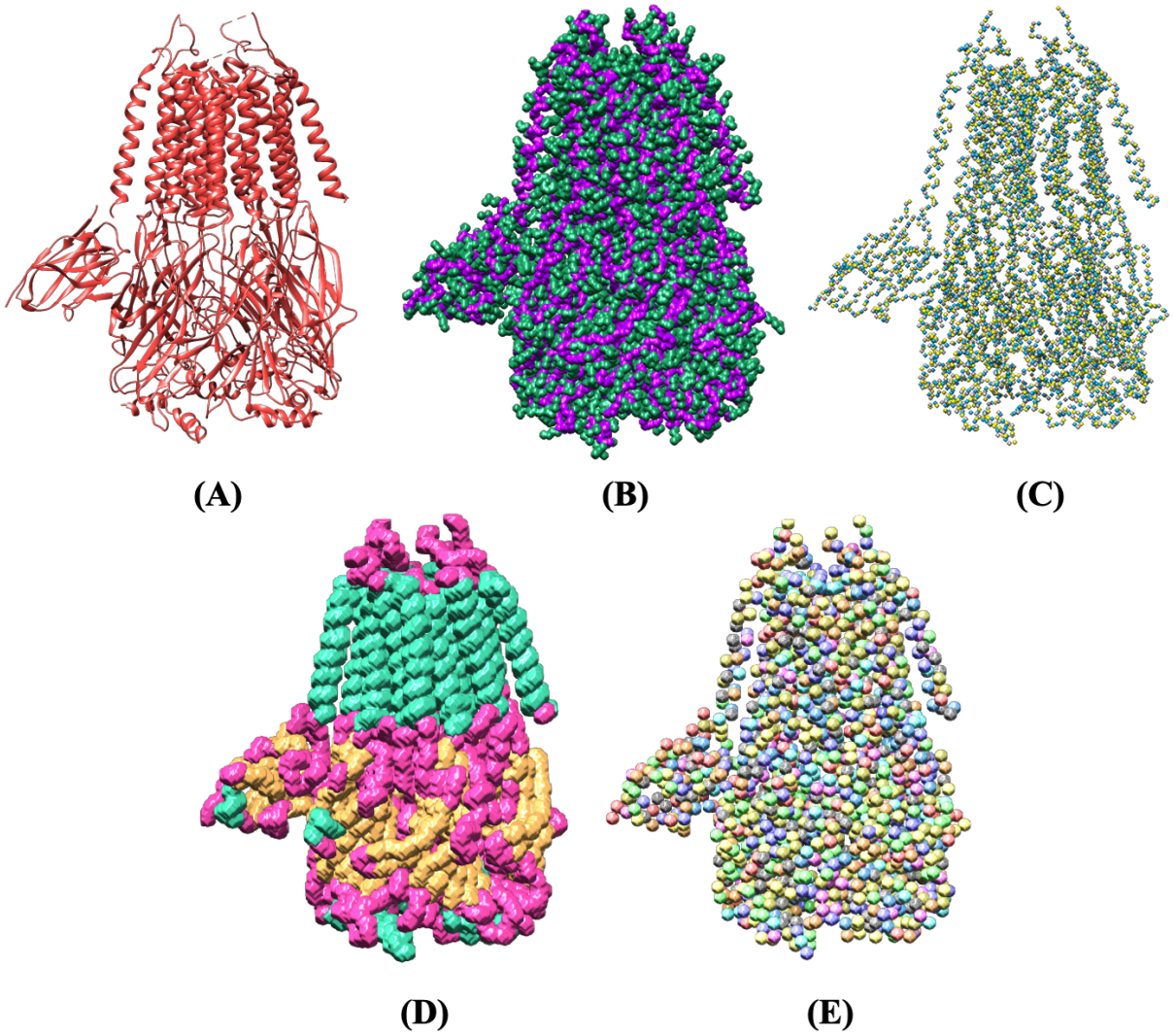


Figure 3.11: Example masks from the training dataset based on the PDB-6NQ1 ground truth structure. (A) Ground truth structure. (B) Backbone (Cα, C, and N atoms) in purple and side chains in green. (C) Atoms mask with labels for Cα, C, and N atoms. (D) Secondary structure mask with helices in turquoise, loops in pink and sheets in orange. (E) Amino acid type mask with different colors for each amino acid type.

The last step before we can start training the model is to split the density maps into a training and validation set and store them as an HDF5 file [23]. We use an 80/20 split for the training and validation dataset. Both sets are written into the same HDF5 file which allows for faster reading and writing of the data to improve the training performance. The HDF5 file represents the final dataset that is used to train the neural network.

### *3.3.3 Training Process*

In this section, we will take a look at the training process of the deep learning model presented in Section 3.9. Particularly, we will explain how we address the class imbalance problem of the training data, which metrics we use for training, and the training progress itself. Finally, we will look at a raw example output of the trained model.

A central challenge in training the U-Net model is the class imbalance between background and non-background voxels in the training data. In order to minimize the loss in the training process with such a class imbalance the model eventually ends up predicting every single voxel as background since this is the correct choice in the vast majority of cases. Usually, class imbalance problems are addressed either by under-sampling the majority class or over-sampling the minority class [24, 25]. However, in this particular problem neither method is applicable as the class imbalance exists for the voxels within each data entry meaning that any over- or under-sampling of entries themselves would not have any effect on the class imbalance of the dataset. Therefore, we decided to use a weighted loss function, particularly a weighted cross-entropy, to train the deep learning model [26]. In order to find the appropriate weights for each class we first analyzed the training data and calculated the ratio of non-background to background voxels. Then, we slightly adjusted those ratios based on training results. In Table 3.1 we can see the weight for each class that was used in the final training process. It is interesting to note the weight differences between different non-background classes, e.g. in the secondary structure elements as well as the amino acid types. They can be seen as an indicator for the relative frequency with which they occur in the protein structures.

Table 3.1: Class weights used for the training of the deep learning model. SSE stands for secondary structure elements and BG for background. Amino acid classes use three character abbreviation of each amino acid type.

| **Atoms** | | **Backbone** | | **SSE** | | **Amino Acids** | |
|---|---|---|---|---|---|---|---|
| Class | Weight | Class | Weight | Class | Weight | Class | Weight |
| BG | 1 | BG | 1 | BG | 1 | BG | 1 |
| Cα | 300 | Backbone | 28 | Loop | 12 | ALA | 277 |
| C | 300 | Side Chain | 21 | Helix | 18 | ARG | 419 |
| N | 300 | | | Sheet | 35 | ASN | 419 |
| | | | | | | ASP | 403 |
| | | | | | | CYS | 1398 |
| | | | | | | GLN | 487 |
| | | | | | | GLU | 389 |
| | | | | | | GLY | 324 |
| | | | | | | HIS | 1048 |
| | | | | | | ILE | 324 |
| | | | | | | LEU | 219 |
| | | | | | | LYS | 404 |
| | | | | | | MET | 847 |
| | | | | | | PHE | 469 |
| | | | | | | PRO | 412 |
| | | | | | | SER | 307 |
| | | | | | | THR | 296 |
| | | | | | | TRP | 1438 |
| | | | | | | TYR | 562 |
| | | | | | | VAL | 275 |

In order to evaluate the performance of the deep learning model it is crucial to choose appropriate metric functions. As a consequence of the class imbalance, computing the voxel-vise accuracy of predictions provides little useful information. Therefore, we chose to compute a confusion matrix based on which we can calculate the recall and precision of the model for each individual class [27]. The recall tells which proportion of voxels that belong to a certain class were actually predicted as such. The precision gives us insights on the share of voxels that were predicted as a certain class actually belong to that class.

The training of the U-Net model was performed on an Nvidia Titan RTX GPU. It continued until the validation loss stopped improving which was the case after 14 epochs or around 5 days of training. In Figure 3.12, we can see an example of what the prediction of the trained U-Net looks like.

## *3.4 Post-Processing*

The post-processing steps of the DeepTracer are responsible for transforming the U-Net predictions into a protein structure. In contrast to many other machine learning applications, the post-processing steps of the DeepTracer perform complex functionalities and their performance is crucial for accurate predictions. As we can see from Figure 3.1, there are six separate post-processing steps, which are described in Sections 3.4.2 through 3.4.6. However, in order to get a better understanding how the DeepTracer handles protein structures, we first take a closer look at their implementation in Section 3.4.1.

### *3.4.1 Protein Structure Implementation*

The final output of the DeepTracer is a predicted protein structure which can be saved as a PDB file. The implementation of a protein structure in the DeepTracer closely follows the specifications given by the PDB file format [28]. The class diagram of this implementation can be seen in Figure 3.13.

The `ProteinStructure` class represents a protein structure and provides a static method to open PDB files and a method to save the protein structure to a PDB file. Its only property

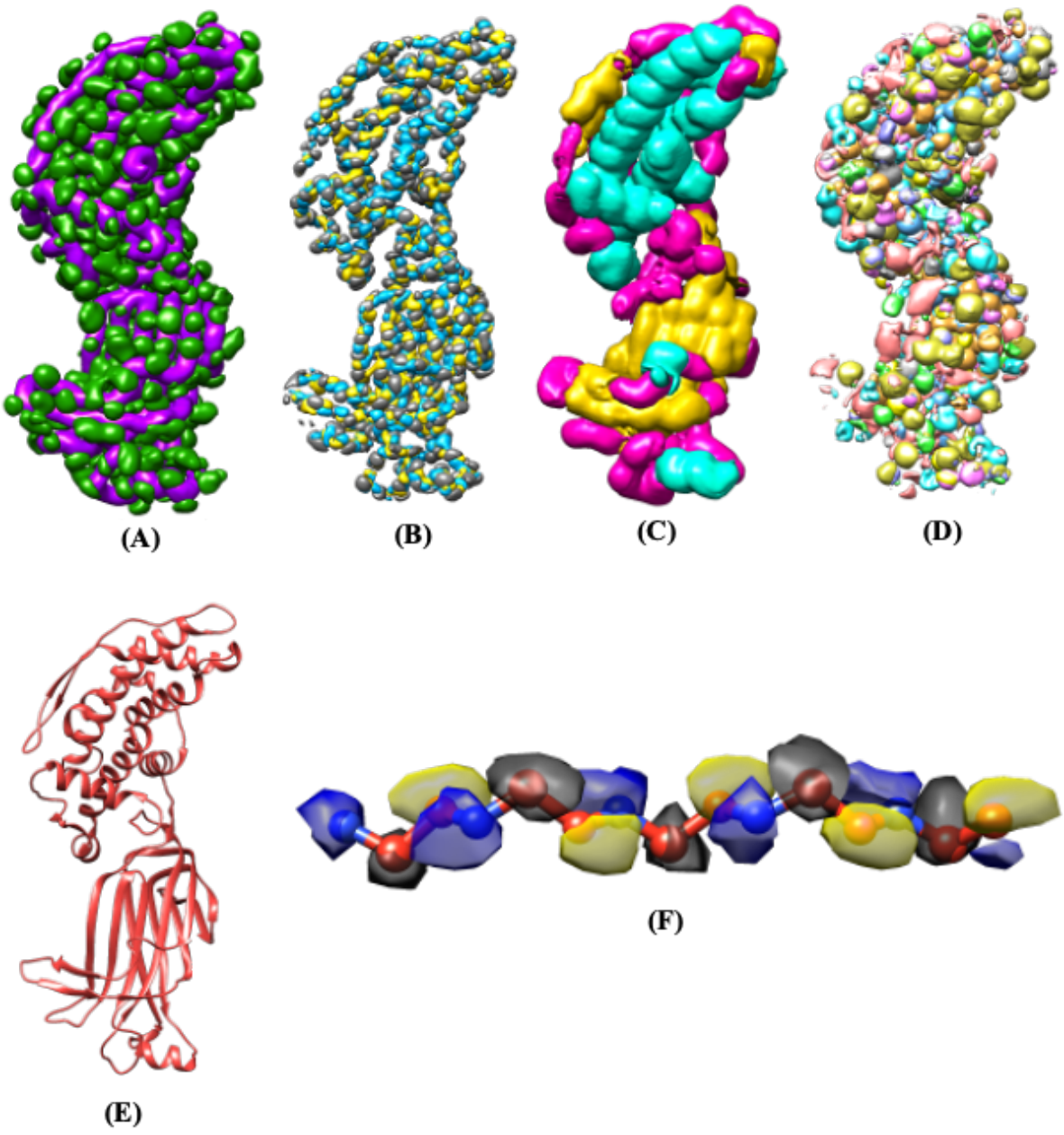


Figure 3.12: Raw prediction of the U-Net deep learning model for the EMD-6272 density map. (A) Backbone prediction with backbone in purple and side-chains in green. (B) Atoms prediction with Cα atoms in gray, C atoms in yellow, and N atoms in blue. (C) Secondary structure prediction with α-helices in turquoise, β-sheets in yellow and loops in pink. (D) Amino acid type prediction with a different color for each type. (E) Solved structure (PDB-3j9s) of density map. (F) Atoms prediction segment next to solved structure.

references a list of `Model` objects that make up the structure. Most structures, particularly all structures predicted by the DeepTracer, consist only of a single model. However, some solved protein structures are split up into several models. Each `Model` consists of a list of `Chain` objects which represent the structure's amino acid chains. Each chain tracks a list of `AminoAcid` instances that make up the chain. Amino acids that are adjacent in this list are considered connected. The `AminoAcid` class provides several defining properties. First, the `name` of the amino acid, which tracks the three letter identifier of the amino acid type and the `secondary_structure` property determining the secondary structure of the amino acid. Next, we have the properties that track the atoms which make up the amino acid and include a `carbon_alpha` atom, a `carbon` atom, a `nitrogen` atom, an `oxygen` atom, and finally a list of `side_chain` atoms. Each `Atom` object has a name (e.g. Ca or N) as well as x,y, and z coordinates that determine the atom's location in 3D space. In addition to the properties, each class except for the `ProteinStructure` itself provides a static `from_pdb` method that creates an instance of that class based on lines of a PDB file as well as a `to_pdb` method which returns the object as a string in PDB file format. These methods are invoked by the `ProteinStructure` class in order to open and save PDB files.

### *3.4.2 Tracing Backbone*

The first post-processing step uses the the output of the U-Net to create an initial protein structure prediction which only contains Cα atoms connected into chains. This is a central post-processing step and its accuracy determines to a great extent how well the remaining post-processing steps will perform. The step can be split into three different parts. First, we identify disconnected chains which can be processed independently. Next, we calculate the x,y, and z coordinates of the Cα atoms and, finally, connect them into chains by applying a modified travelling salesman algorithm.

Identifying chains prior to any atom prediction has two advantages. First, it improves the performance of the step as each chain will contain a lower number of atoms that have to be connected with the travelling salesman algorithm. Second, it decreases the number of

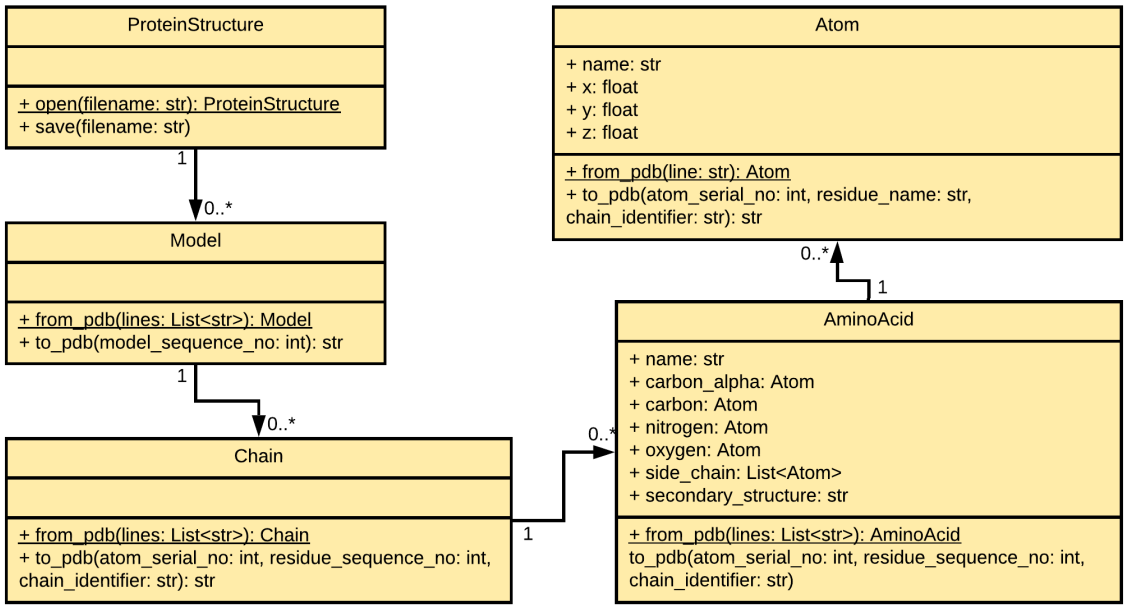


Figure 3.13: UML class diagram depicting the topology of a protein structure as implemented in the DeepTracer. The topology closely follows the specifications given by the PDB file format. Underlined function names indicate that they are static.

incorrect connections between atoms of separate chains as they are processed independently. In order to identify chains we use the output of the backbone U-Net (see 3.12). We round each voxel of the confidence map to either zero or one and then find connected areas of voxels with a value of one. Disconnected areas are then identified as separate chains. An example of the chain identification process visualized for the EMD-0478 density map can be seen in Figure 3.14.

To find the x, y, and z coorindates of the Cα atoms we utilize the Cα channel from the output of the atoms U-Net. Every voxel value in this map describes the confidence of the deep learning model that this voxel contains a Cα atom. The coordinates are now calculated in two steps. First, we find the indices of all local maximums in the confidence map within a distance of 4 voxels with that have a minimum value of 0.5. Next, we can refine the indices by calculating the center of mass of all voxels within a distance of 4 surrounding the local maximums. This is possible as we now move away from integer indices towards

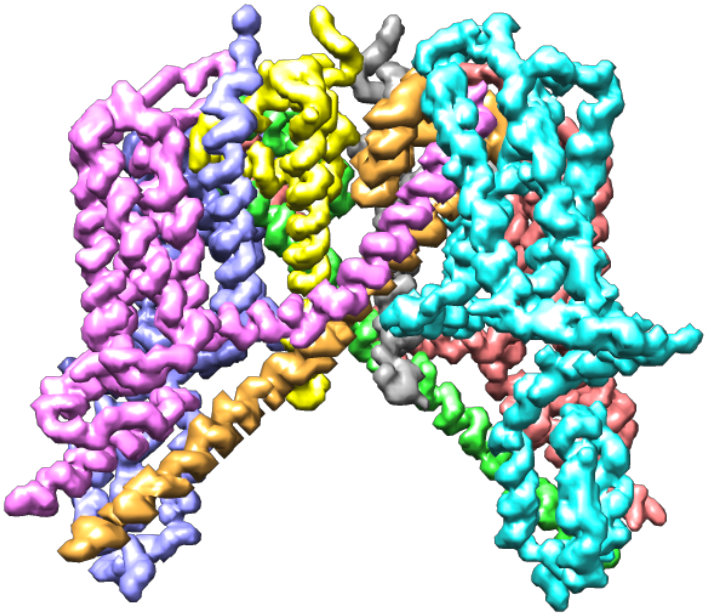

Figure 3.14: Backbone confidence map of the EMD-0478 density map with identified chains annotated in different colors.

floating point coordinates giving us the opportunity to express locations more precisely. The resulting values of the refinement can now be used to calculate x, y, and z coordinates through Equation (2.1).

The most challenging part of this prediction step is to correctly connect the predicted Cα atoms into chains. The factorial growth of the number of ways in which the atoms can be connected makes it infeasible to test all possible solutions even for a low number of atoms. Therefore, we decided to solve the problem using an optimization algorithm, particularly, one for the travelling salesman problem (TSP). However, our problem does not match every criteria of the traveling salesman problem. The shortest possible path is not necessarily the correct one as the ideal distance between Cα atoms is 3.8A [29]. Deviations from this value are however possible due to prediction inaccuracies. Additionally, it is often difficult to decide only based on distance which atoms to connect if there are multiple possibilities with a similar distance. In order to address these issues, we developed a custom confidence

function instead of solely relying on the Euclidean distance between atoms. The idea of the confidence function is to return a score between 0 and 1 which expresses how confident we are that these two atoms are connected. The goal of the TSP algorithm is then to connect the atoms such that the sum of all confidence scores between connected atoms is maximized.

The calculation of the confidence score between Cα atoms takes two factors into consideration. The euclidean distance between the atoms as well as the average values of voxels, that lay in between the atoms, in the backbone confidence map predicted by the backbone U-Net. The latter factor is meant to ensure that connections are made along the backbone of the structure. The voxels that lay between the atoms are found using Bresenham's algorithm [30]. To transform these metric values to a confidence score we use a probability density function $p(x, \mu, \sigma)$ with a mean $\mu$, which represents the ideal metric value, and a standard deviation $\sigma$. In order to make sure that the function returns exactly 1 at the mean, we normalize it by dividing it through the probability density value at the mean. For the euclidean distance we use a mean of 3.8 and a standard deviation of 1. The average backbone confidence has a mean of 1 and a standard deviation of 0.3. The standard deviations were determined based on several rounds of testing. In Figure 3.15 we can see both probability density functions plotted for a range of metric values. In order to combine both results into a single confidence score we simply multiply both values. As the TSP algorithm is designed to minimize distances between paths we now just have to subtract the confidence score from 1 and then provide it to the algorithm.

In order to apply the TSP algorithm, we also need to specify a start/end point. However, we do not know yet at which atom the chain will start and end. Therefore, we add a new atom that is connected to every other atom with a confidence of 1. This atom is then specified as the start/end and later on removed from the actual chain. An example of the application of the TSP on a list of Cα atoms can be seen in Figure 3.16.

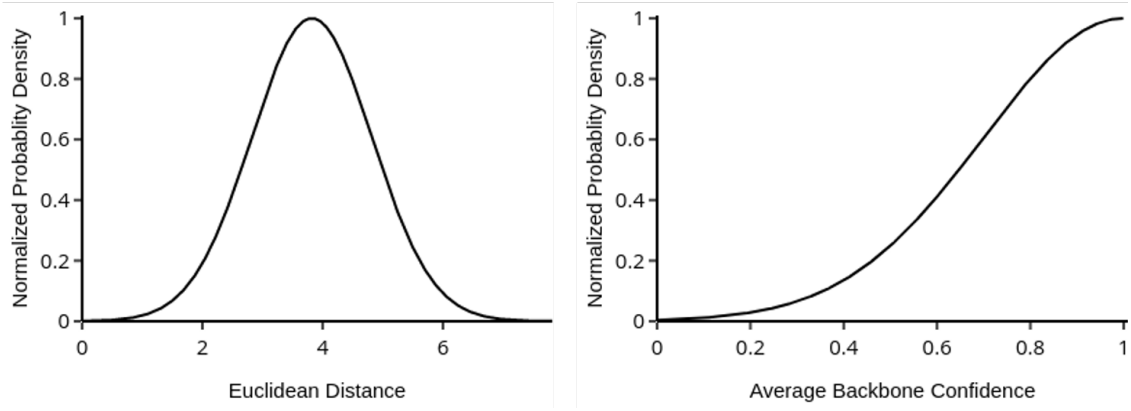


Figure 3.15: Normalized probability density function used to calculate confidence score for the euclidean distance and average backbone confidence between two Cα atoms.

### *3.4.3 Refining Alpha Helices*

This prediction step is responsible for refining modelled α-helices. Here for, we can exploit the fact that the shape of an α-helix has a general definition which is valid across proteins [23]. Since the U-Net predicts the confidence of secondary structure elements, as shown in Figure 3.12, we know which amino acids belong to an α-helix based on the confidence of their region in space. We combine this knowledge of α-helix locations and their shape attributes in order to adjust the appropriate Cα atoms to better fit the shape of a natural α-helix structure.

For an α-helix which centers around the z-axis, we can use Equation (3.1) to model its shape where the variables s and r represent the initial shift and rotation of the helix. The values 2.11 and 1.149 are constants that define the radius and pitch of the helix to best match those of an α-helix.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 2.11 \times \sin(1.49 \times t) - s + r \\ 2.11 \times \cos(1.49 \times t) - s + r \\ t \end{pmatrix} \tag{3.1}$$

Equation (3.1) however, cannot be used to describe an α-helix which does not center

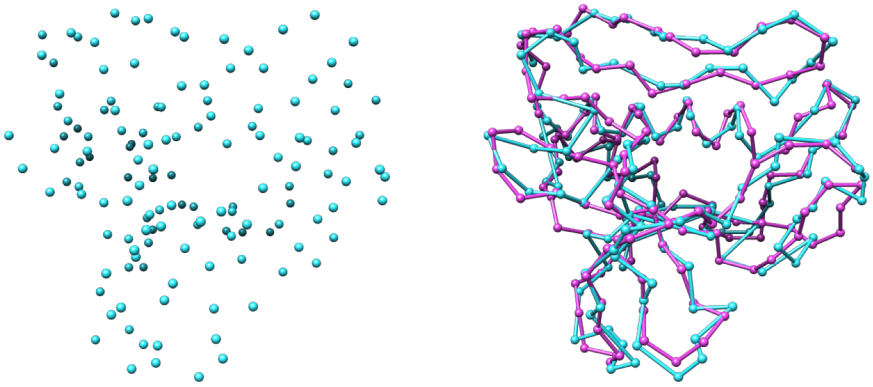

Figure 3.16: Predicted Cα atoms for the EMD-4054 density map in blue before (left) and after (right) the backbone tracing step compared to the true structure in pink.

around the z-axis or whose shape is not a straight cylinder. Since this is the case for most α-helices, it is necessary to adjust the equation in such way that it will address these issues. With the aim of doing so, we first locate the screw axis, the center line around which the helix winds itself, for each α-helix. This is achieved by calculating the centroid of consecutive intervals of the α-helix and then connecting them to approximate the true curve. An example of an α-helix and its calculated screw axis can be seen in Figure 3.17.

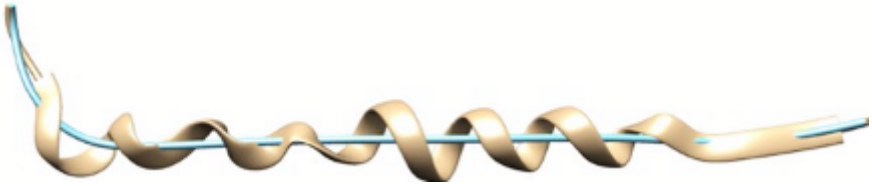

Figure 3.17: α-Helix extracted from the backbone prediction of the EMD-8515 density map in tan color and its screw axis in teal color.

Now, that we know the location and shape of the screw axis for the α-helix, we need to incorporate this information into Equation (3.1). This is achieved by interpreting t as the distance that we travelled on the screw axis and use the unit direction vector of the screw axis at a certain point t as the new z axis. Next, we can find the new y-axis by calculating

the cross product of the x-axis and the new z-axis and then normalizing it. Finally, we can get the new x-axis by calculating the cross product of the new z and y-axis and normalizing it again. By concatenating the three new axes we can get a rotation matrix $R$ with which we can calculate the point of the α-helix for any value t as shown in Equation (3.2).

$$\alpha\text{-helix}(t) = \text{screw-axis}(t) + R \begin{pmatrix} 2.11 \times \sin(1.49 \times t) - s + r \\ 2.11 \times \cos(1.49 \times t) - s + r \\ 0 \end{pmatrix} \tag{3.2}$$

Now, we need to know the values t at which we have to insert Cα atoms. Since we know that an α-helix has a rise of 1.5Å per residue [23] we can increase t in steps of 1.5 and add a new Cα atom at α-helix(t).

In the final step we minimize the average distance from the Cα atoms of the refined α-helix to the Cα atoms of the original prediction. This is done by applying a minimization algorithm1 over the variables s and r to try different initial shifts and rotations. The final results of the α-helix refinement step are shown in Figure 3.18.

### *3.4.4 Amino Acid Sequence Mapping*

In order to realize the side-chain prediction for the protein structure, we first need to know the type of each amino acid. As discussed in Section 3.3, one output of the deep learning model is the amino acid type prediction. However, as there are twenty different possible types, some of which have a very similar appearance in electron density maps, this prediction is of limited accuracy with around 10% to 50% depending on the resolution of the density map. The goal of this prediction step is to improve the amino acid type accuracy by aligning intervals of the initially predicted sequence to the known true amino acid sequence (protein primary structure) and then updating the types of the predicted amino acids accordingly (see Figure 3.19).

Aligning amino acid sequences is a common problem in the field of bioinformatics and past research has lead to the development of multiple algorithms [31, 32, 33]. However,

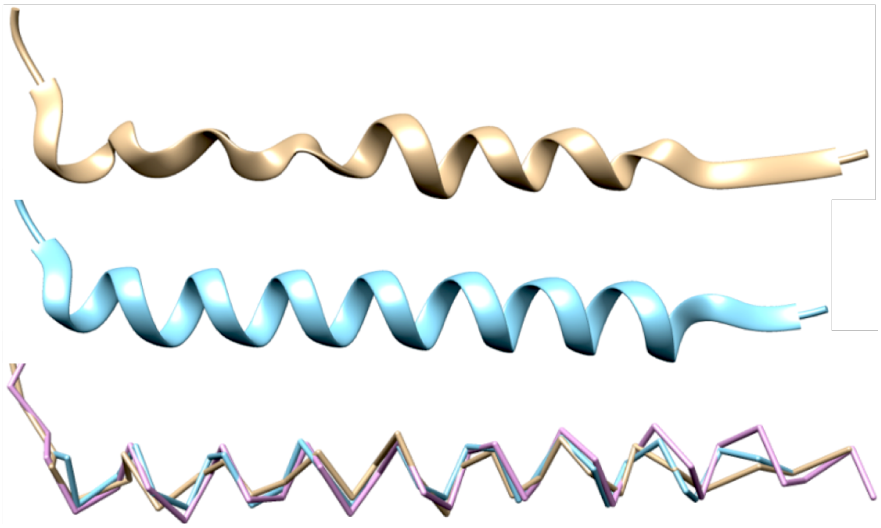

Figure 3.18: α-Helix extracted from the prediction of the EMD-8515 density map. (Top) Original prediction before the helix-refinement step. (Center) α-Helix after the refinement. (Bottom) Direct comparison of original prediction in tan color, refined prediction colored in teal, and the solved structure (PDB-5u70) in pink color.

usually these algorithms are applied between different proteins in order to measure their sequence similarities which does not quite fit our use-case. The main problem is hereby that we require an algorithm that does not treat all matches and mismatches the same. This stems from the fact that some amino acid types have a more similar appearance in density maps than others, which leads to some mismatches of the U-Net being more likely than others. To analyze the relative frequency of a certain match of predicted and true amino acid type, we applied the U-Net to 200 different density maps and compared the predicted amino acid types with the actual types from the solved PDB structures. The heatmap in Figure 3.20, shows the results of this analysis. As expected, the most frequent matches are those of the same predicted and true amino acid type. However, we can also see that the U-Net often mixes up some types (e.g. ALA and SER) and struggles more with other types (e.g. CYS).

In order to incorporate the U-Net prediction behavior described in the previous section into the alignment algorithm, we define a reward function $r$ which returns a score denoting

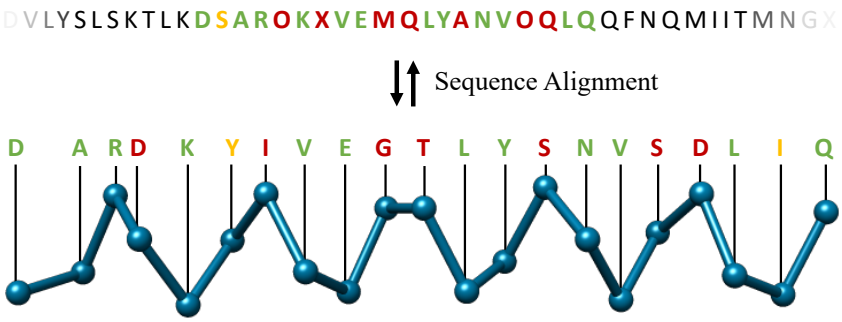


Figure 3.19: Sequence alignment between true (top) and predicted (bottom) amino acid sequence. Green color indicates a match, red color a mismatch, and yellow gaps. Letters represent one character identifiers of the amino acid type.

how valuable a certain match of predicted type $p$ and true type $t$ is. With $f(p,t)$ defined as the relative frequency of a match as depicted in Figure 3.20, we construct the reward function shown in Equation (3.3). The constant 100 as a multiplier is used to balance the match rewards with gap penalties described in the next section, and was chosen based on multiple rounds of testing. The 0.05 constant was chosen as this represents the likelihood of a correct match if we would chose the amino acid type randomly, since there are 20 different types of amino acids. This achieves that the score is zero if the relative frequency equals this random likelihood.

$$r(t_p, t_t) = 100 \times (f(p,t) - 0.05) \tag{3.3}$$

In addition to the match reward, our algorithm also requires a gap penalty. A gap represents a skipped amino acid in either the predicted or true sequence. This penalty, however, cannot simply be a static value as not all gaps are the same. For example gaps in the beginning of the sequence before any matches were made should not result in any penalties as we only match short intervals of the predicted sequence which means it is highly unlikely that they align at the first amino acid of the true sequence. Additionally, the number of consecutive gaps is important. Cases where the DeepTracer misses an amino acid

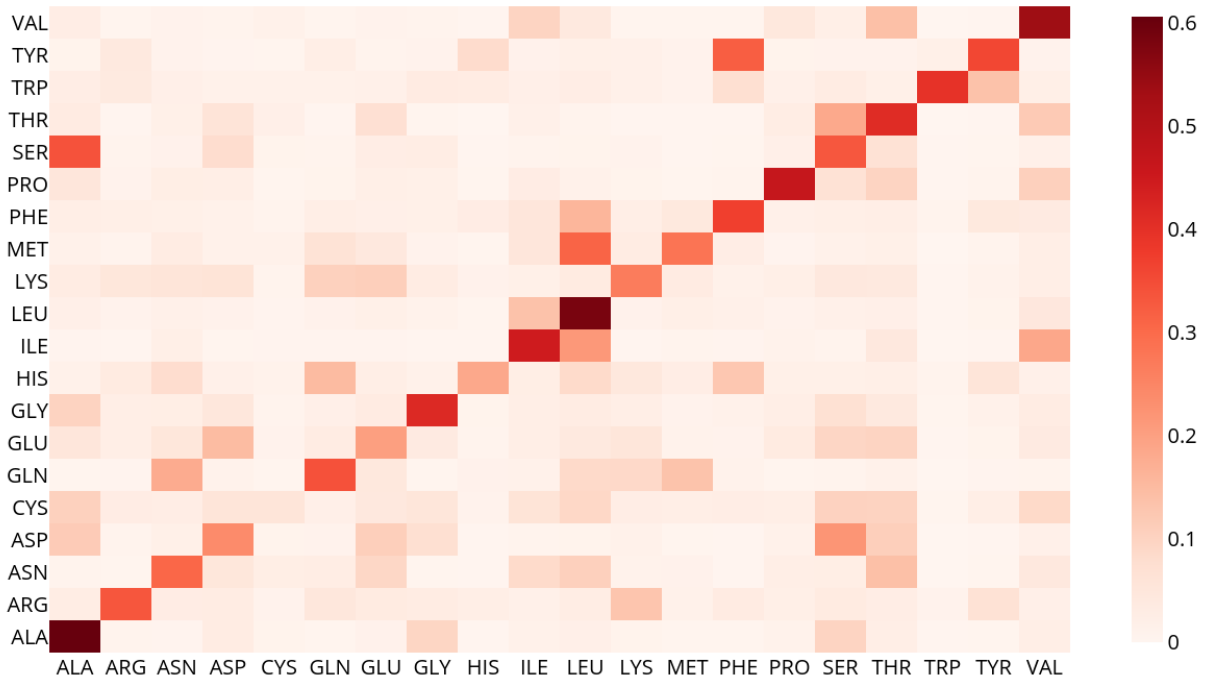


Figure 3.20: Heatmap depicting the relative frequencies of matches of predicted and true amino acid types. The x-axis shows the predicted amino acid type and the y-axis the true type.

or predicts an extra amino acid appear relatively frequent meaning that a single gap is not unlikely. However, two missed amino acids in a row is very uncommon and three gaps in a row virtually never happens. Therefore, we must define our penalty function $p$ such that it takes the number of consecutive gaps $g$ into account. Let $i$ be the index of the amino acid that is not skipped. Then we can define $p$ as shown in Equation (3.4). The constants 20 and 30 were chosen based on test runs to create a good balance with the rewards function.

$$p(g,i) = \begin{cases} 0, & \text{if } i = 0 \\ \infty, & \text{if } g \geq 3 \\ 20 + (g \times 30), & \text{otherwise} \end{cases} \tag{3.4}$$

Now, that we have defined a reward and penalty function we can find the ideal alignment by maximizing the sum of all rewards and penalties using a dynamic algorithm. To do so, we define a recursive equation which calculates the optimal solution based on an index $i$ which points to the current amino acid in the true sequence, an index $j$ which points to the current amino acid in the predicted sequence, as well as $g$ which counts the number of previous consecutive gaps. With $t$ and $p$ as the true and predicted sequence we can define this function as shown in Equation (3.5). To efficiently find the solution we can now simply apply the dynamic programming "bottom up" approach [34].

$$\mathrm{OPT}(i,j,g) = \begin{cases} 0, & \text{if } i = 0 \text{ or } j = 0 \text{ or } g \geq 3 \\ \max\{\mathrm{OPT}(i-1,j-1,0) + r(t_i,p_j), & \\ \qquad \mathrm{OPT}(i,j-1,g+1) + p(g,i), & \\ \qquad \mathrm{OPT}(i-1,j,g+1) + p(g,j)\}, & \text{otherwise} \end{cases} \tag{3.5}$$

### *3.4.5 Carbon and Nitrogen Prediction*

So far, the predicted amino acids consist solely of Cα atoms. To complete the backbone of the structure, this step adds carbon and nitrogen atoms to each amino acid. This is necessary in order to predict the side-chain atoms in the next prediction step.

As shown in Figure 3.12, the U-Net predicts not only the Cα atoms of the structure but also the carbon and nitrogen atoms that form the peptide bonds connecting the amino acids. We can utilize this prediction in combination with the previously predicted Cα atom positions to place the carbon and nitrogen atoms. From Figure 2.3 we can see that in between the Cα

atoms of two connected amino acids, there is always a nitrogen and carbon atom. Therefore, we can guess the initial position of these atoms by calculating the vector from one Cα atom to the other and then placing the nitrogen and carbon atoms at $\frac{1}{3}$ and $\frac{2}{3}$ of the distance of this vector. To refine these initial positions we calculate the center of mass around them in the carbon and nitrogen confidence maps predicted by the U-Net. In Figure 3.21 we can see an example for the initial and refined prediction of the carbon and nitrogen atoms.

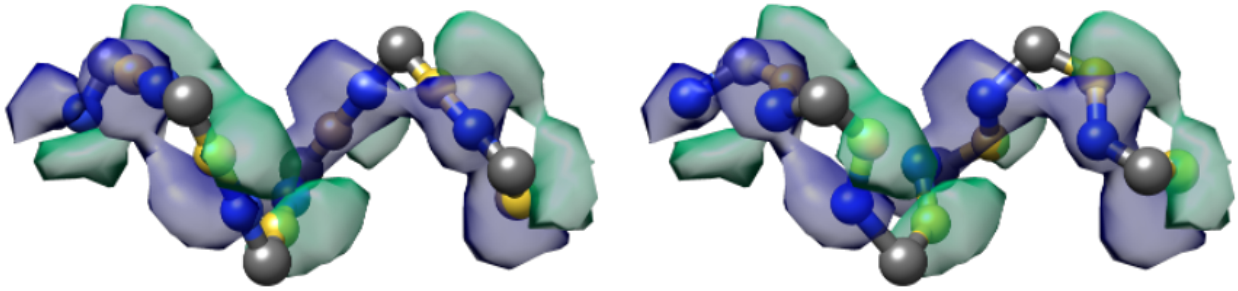

Figure 3.21: Initial positioning of carbon (yellow) and nitrogen (blue) atoms in between the Cα atoms (gray) on the left and their refined positioning, which fits the U-Net prediction of carbon atoms (green volume) and nitrogen atoms (blue volume), on the right.

### *3.4.6 Side Chain Prediction*

The final step of the DeepTracer is the side chain prediction. Its goal is to position the side chain atoms of each amino acid based on its type and backbone structure. This is done through the use of SCWRL4, a tool developed by the Dunbrack lab which predicts side chain atoms for structures that have a complete backbone and amino acid types set. The tool is integrated in the prediction pipeline of the DeepTracer and runs fully automatic as well. In Figure 3.22 we can see an example of an α-helix after the side chain prediction step.

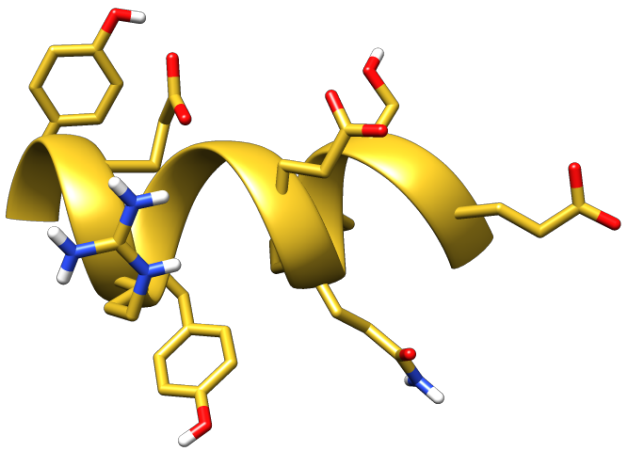

Figure 3.22: Predicted side chains for an extract of an α-helix. Backbone atoms are displayed in the ribbon view and side chain atoms in atom view.

# Chapter 4

# RESULTS

In this chapter we apply the DeepTracer method outlined in the previous chapter to experimental density maps in order to evaluate the quality of its predictions. Particularly, we want to compare its effectiveness with the existing Phenix map-to-model function. Section 4.1 describes the metrics we utilize to facilitate this comparision. Next, Section 4.2 and 4.3 present the prediction results of both methods for two separate test datasets. Finally, Section 4.5 conducts a short analysis of the DeepTracer's computation time.

## *4.1 Metrics*

In order to achieve a fair comparison with the existing Phenix method we utilize their `phenix.chain_comparison` tool, which is freely available as part of the Phenix software suite, to evaluate the accuracy of predicted structures. It compares predicted and native protein structures by finding a one-to-one matching between native and predicted residues based on Cα positions. In order for a native residue to be matched with a predicted residue they cannot be further apart from each other than 3A. Based on this matching several metrics determining the prediction accuracy are calcluated. First, the root-mean-square deviation (RMSD) which expresses the average distance between Cα atoms of matched native and predicted residues. Its calculation is shown in Equation (4.1) with $M$ as a set of tuples of matched native residues $n$ and predicted residues $p$. The coverage of the prediction is expressed using the matching percentage. This value represents the proportion of native residues which have a matching predicted residue and is calculated by dividing the number of matches by the total number of native residues. To evaluate how well the amino acid types were predicted the `chain_comparison` tool calculates the sequence matching percentage which denotes the

percentage of matched native and predicted residues that have the same amino acid type. Finally, to get a sense of how well the predicted residues are connected, the mean length of matched segments is calculated where consecutive matches are connected both in the native and predicted structure.

$$\text{RMSD} = \sqrt{\frac{1}{|M|} \sum_{(n,p)\in M} (n_x - p_x)^2 + (n_y - p_y)^2 + (n_z - p_z)^2} \tag{4.1}$$

## *4.2 Phenix Test Dataset*

The first test dataset on which we apply the DeepTracer was put together by the authors of Phenix's map-to-model method and consists of a total of 476 density maps [18]. As part of the supplementary material the authors provided Phenix's prediction results for these density maps which we use to compare the DeepTracer's predictions. Utilizing the results provided by the authors themselves ensures that the Phenix method was applied correctly allowing for a fair comparison.

The results of the predictions can be seen in Figure 4.1. We can see that the DeepTracer achieves better results than the Phenix method for every metric calculated by the `phenix.chain_comparison` tool. The matching percentage of native residues is on average 76.93% compared to 45.65% with Phenix, which represents an improvement of over 30%. The DeepTracer achieved a matching percentage above 70% for almost all density maps excluding a few outliers. The average RMSD value of the DeepTracer has improved by 0.11 compared to the Phenix method from 1.29 to 1.18. We can see that the distribution of the RMSD values of the DeepTracer follows a similar pattern as Phenix, with a strong correlation between RMSD and the resolution of the density map. The most significant improvements of the DeepTracer were measured for the sequence matching, so the percentage of matched native and predicted residue that have the same amino acid type. For this metric the DeepTracer achieved 49.83%, which is more than four times as much as the 12.29% of the Phenix method. Although 49.83% still sounds fairly low, looking at the distribution of

the values we can note that there is a steep improvement of the sequence matching with more accurate maps. There are two factors contributing to this trend. First, side chain atoms, which determine the amino acid type, are only visible in very high resolution maps making it almost impossible to accurately predict the amino acid type for lower resolutions. Second, the amino acid type mapping of every segment can either be correct or incorrect. This means that either all amino acid types will be correct for this segment or in case of an incorrect mapping the amino acid types are entirely random. This amplifies the steep incline in accuracy for higher resolution maps. Finally, for the last evaluated metric, the mean length of matched segments, improved from 8.16 with Phenix to 14.05 with the DeepTracer. While this number is influenced by several factors including the average length of connected segments in the native structure, this is an indicator that the DeepTracer connects residues better than the Phenix method.

Figures 4.2 and 4.3 show two example predictions of the DeepTracer and Phenix compared to the native structure. In both figures we can note that the DeepTracer's prediction is more complete. In Figure 4.2, we can particularly see the greater coverage and more precise prediction of the residues in direct comparison with the ground truth. Figure 4.3 shows well that, even though Phenix predicted most of the residues correctly, the DeepTracer connected the residues better creating a less fragmented prediction.

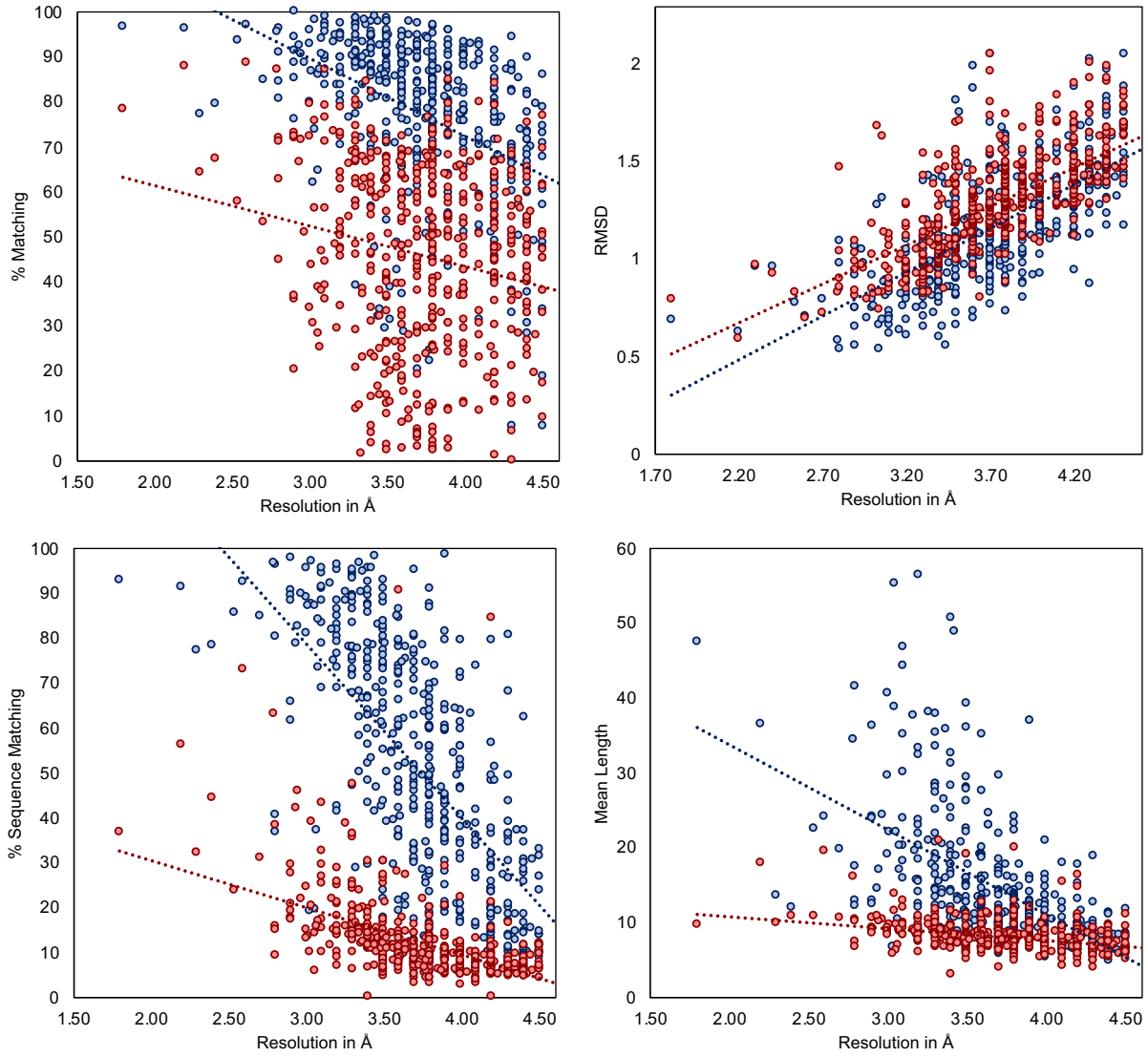


Figure 4.1: Evaluation of prediction results from the DeepTracer (blue) and Phenix (red) for 476 density maps. The dotted lines represent the trend for each prediction method.

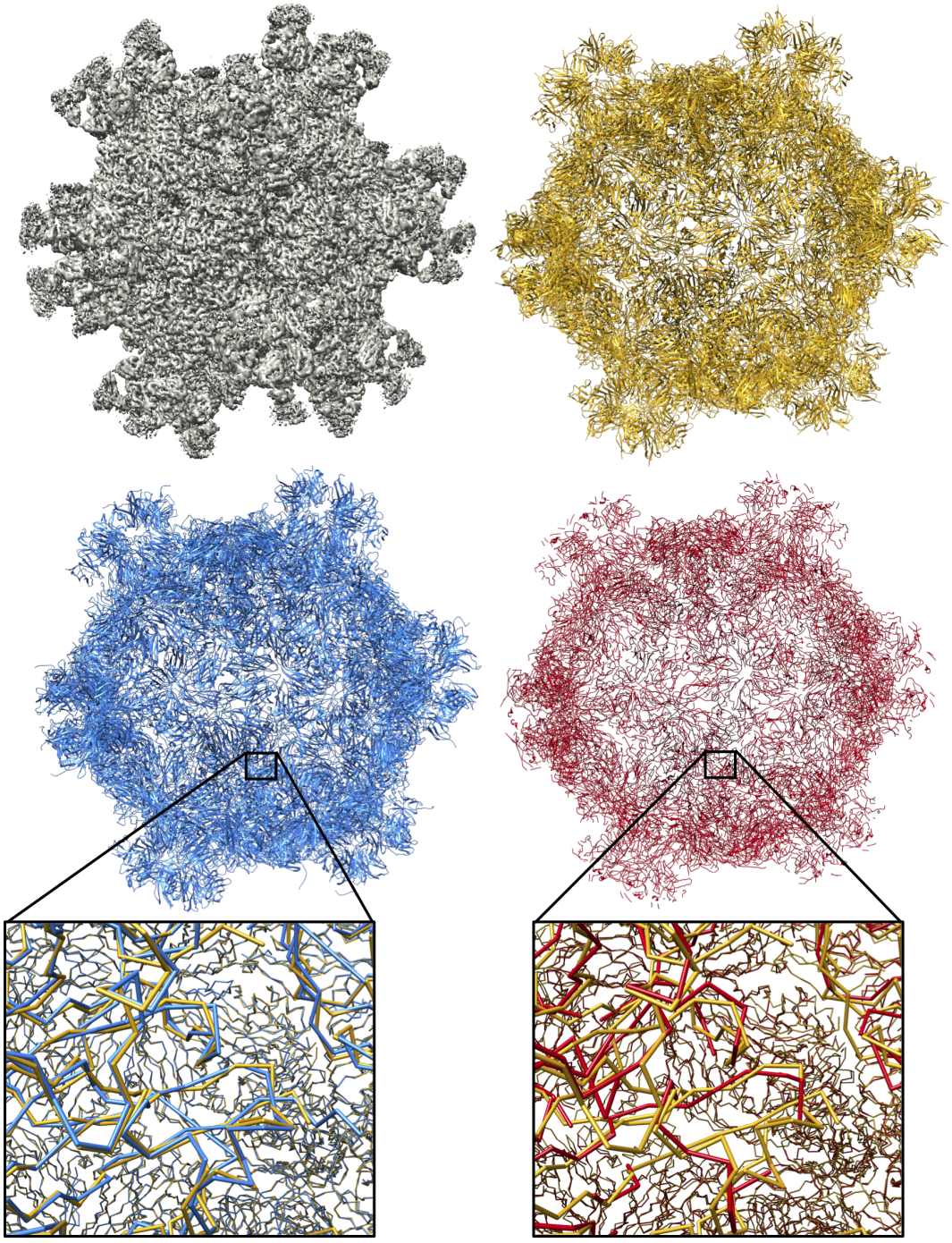

Figure 4.2: Prediction of DeepTracer (blue) and Phenix (red) next to PDB-3j9s native structure (yellow) for EMD-6572 density map.

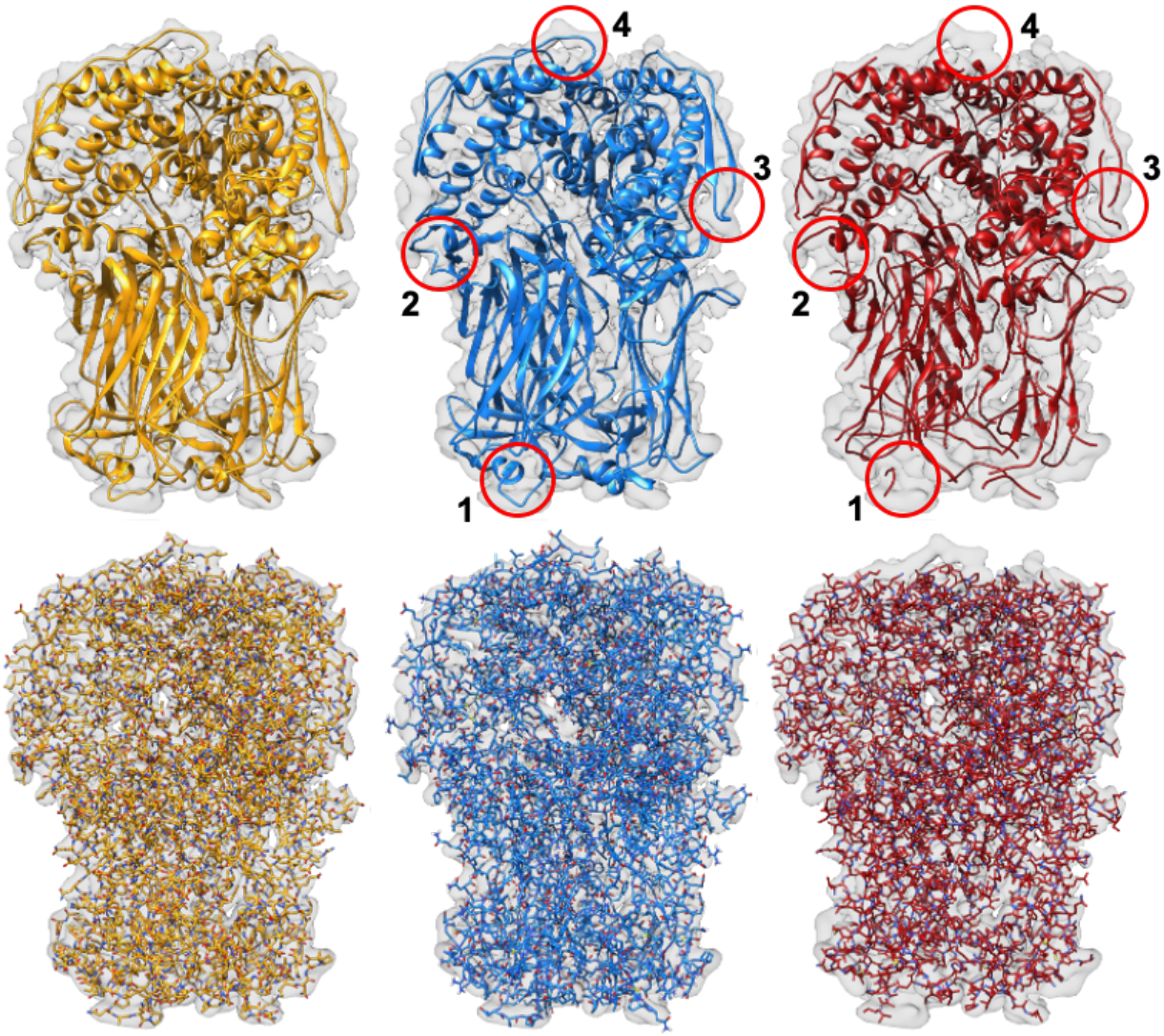


Figure 4.3: Predictions of the DeepTracer (blue) and Phenix (red) compared to PDB-3j9s native structure (yellow) for EMD-6272 density map. Top row shows structures in ribbon view and lower row in all-atom view.

### *4.3 Coronavirus-Related Results*

In the search for an effective COVID-19 vaccine and medicine, structural information about the viral protein is crucial. Therefore, we applied DeepTracer on a set of coronavirus-related density maps to demonstrate how it can aid researchers in obtaining such structural information. To create a point of comparison, we applied Phenix on the same set of density maps. The dataset was aggregated by the EMDataResource and contained 62 high-resolution density maps, 52 of which have a deposited model PDB structure [35]. The dataset as well as the determined models will be actively updated at DeepTracer's website as more and more data is deposited to EMDR. To our knowledge, this is the first CoV-related 3D cryo-EM modeling test dataset.

The scatter plots in Figure 4.4 show the evaluation results for the metrics calculated by Phenix's `chain_comparison` tool, for the 52 coronavirus-related density maps that have a deposited model structure. The average percentage of matched model residues is 84% for DeepTracer and 49.8% for Phenix. This means that, on average, around 34% more residues were correctly placed by DeepTracer than by Phenix. The RMSD metric calculated an average value of 1.37Å for Phenix compared to 0.93Å with DeepTracer. Thus, DeepTracer not only determines more residues correctly than Phenix, but the correctly determined residues were also closer to the residues of the deposited model by around 0.4Å. For the sequence matching results, Phenix scored 24.95%, while DeepTracer achieved a sequence matching percentage of 63.08%. Finally, the mean length of consecutively matched residues in the modelled and deposited structure increased from 8.9 with Phenix to 20 with DeepTracer.

The SARS-CoV-2 results from Table 4.1 show a similar pattern as the results of all coronavirus-related maps. DeepTracer outperformed Phenix in every metric with the most significant differences in the matching percentage and sequence matching. Additionally, the DeepTracer achieved a GDC score almost three times that of the Phenix method.

In Figure 4.5, we can see the structures modelled by DeepTracer for the EMD-30044 density map, which captures the human receptor angiotensin-converting enzyme 2 (ACE2)

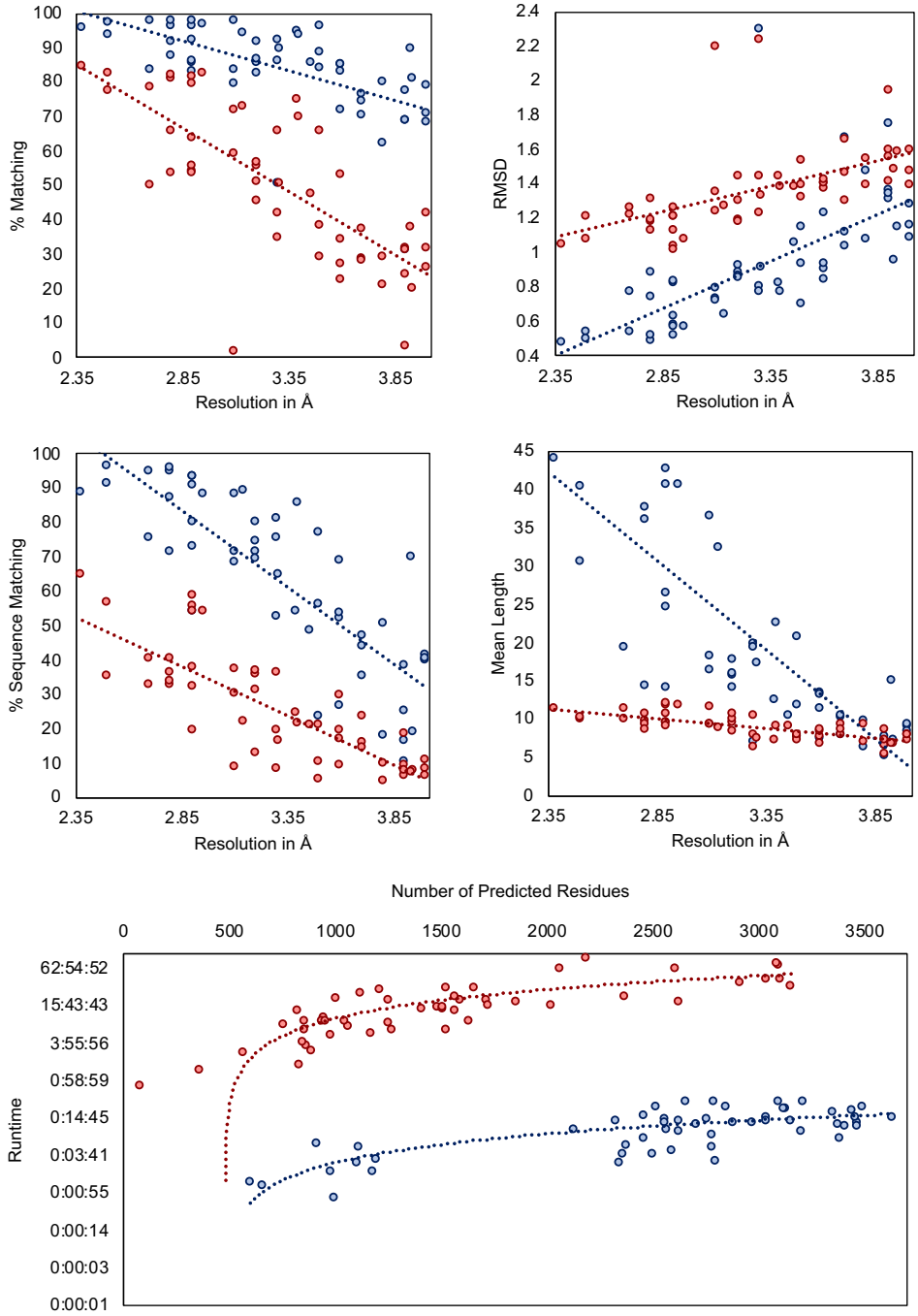


Figure 4.4: Results for coronavirus-related density maps. Evaluation of models built by DeepTracer (blue) and Phenix (red) for 52 coronavirus-related high-resolution density maps. The dotted lines represent the trend for each method. Computation times are shown on a logarithmic scale.

to which the spike protein of the SARS-CoV-2 virus binds to [36] and the EMD-21374 density map of a SARS-CoV-2 spike glycoprotein. No model structure has been deposited to the EMDR for either density map. This represents an ideal opportunity to showcase the potential of DeepTracer. Without any other parameters or manual processing steps, DeepTracer can determine detailed models based on the density maps. Researchers can use these models to develop therapeutics targeting the binding process between the spike protein and the human enzyme.

Table 4.1: Comparison of DeepTracer (DT) and Phenix (P) for SARS-CoV-2 dataset.

| **EMDB** | **PDB** | **Residues** | **% Matching** | | **RMSD** | | **% Seq ID** | | **GDC** | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | DT | P | DT | P | DT | P | DT | P |
| 21375 | 6vsb | 2905 | 84.90 | 48.60 | 1.14 | 1.40 | 45.90 | 20.90 | 17.88 | 5.39 |
| 21452 | 6vxx | 2916 | 91.40 | 53.80 | 0.96 | 1.18 | 61.30 | 40.00 | - | - |
| 30039 | 6m17 | 3072 | 80.30 | 53.10 | 1.72 | 1.72 | 69.80 | 54.60 | 11.84 | 8.31 |
| 30127 | 6m71 | 1077 | 91.70 | 54.20 | 1.02 | 1.20 | 58.60 | 16.60 | 20.74 | 8.89 |
| 30178 | 7btf | 1227 | 94.90 | 81.00 | 0.83 | 1.09 | 85.80 | 51.80 | 65.57 | 23.06 |
| 30209 | 7bv1 | 1102 | 87.60 | 67.00 | 0.84 | 1.29 | 87.50 | 30.50 | 55.62 | 18.15 |
| 30210 | 7bv2 | 1006 | 92.40 | 78.20 | 0.78 | 1.08 | 88.90 | 53.70 | 40.90 | 13.32 |
| Avg. | | | 89.03 | 62.27 | 1.04 | 1.28 | 71.11 | 38.30 | 35.42 | 12.85 |

GDC score could not be calculated for the EMD-21452 density map as the LGA web service could not process the modelled structures due to their size.

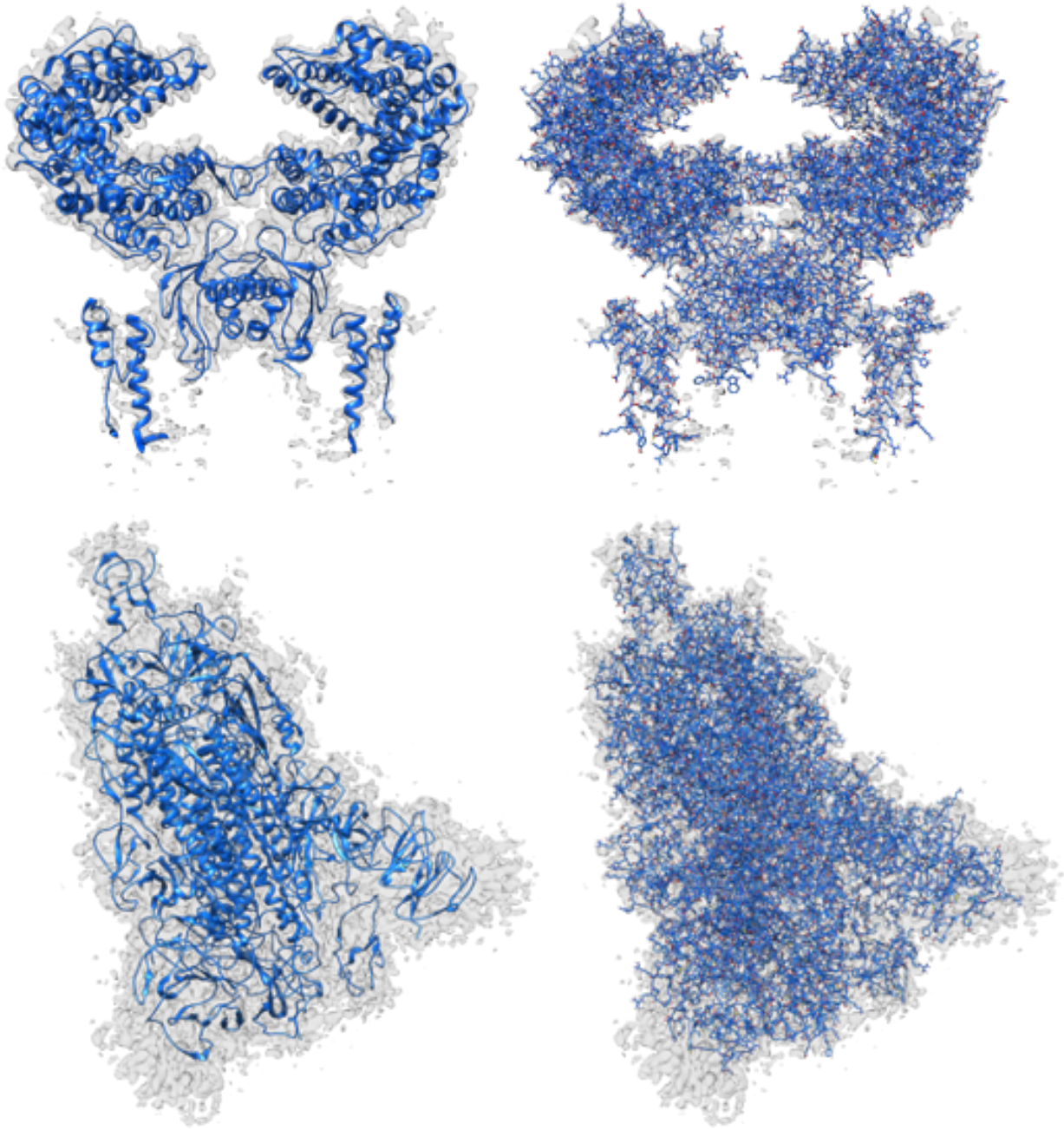

Figure 4.5: Models built from SARS-CoV-2 density maps, which do not have deposited model structures in the EMDR. DeepTracer model for the EMD-30044 density map (top) showing a human receptor angiotensin-converting enzyme 2 (ACE2) to which spike proteins of the SARS-CoV-2 virus bind to and the EMD-21374 depicting a SARS-CoV-2 spike glycoprotein. No model structure has been deposited to the EMDataResource for the density maps as of the date this paper is announced.

### 4.4 Comparison with MAINMAST and Rosetta

In addition to Phenix, MAINMAST and Rosetta are two further established cryo-EM prediction methods. We conducted a brief analysis of their performances compared to DeepTracer based on a test set of nine density maps taken from the previous papers [37, 38]. Note that the density maps were cropped such that they captured only a single protein chain. This cropping was necessary as both methods can only perform single-chain predictions. To evaluate predictions we utilize Phenix's `chain_comparison` tool. The results of this analysis can be seen in Table 4.2. We can note that DeepTracer outperforms Rosetta in all four metrics with particularly significant improvements in the percentage of matched residues as well as false-positive predictions. Compared to the MAINMAST method DeepTracer performed worse in three of the four metrics. However, predictions of DeepTracer were much more complete with an average matching percentage of 93.4% compared to only 36.4% with MAINMAST. That means that MAINMAST correctly predicted only around 1/3 of the protein structure.

### 4.5 Computation Time

A major bottleneck of many prediction methods is their computational complexity which renders them unable to predict larger protein structures. Therefore, we conduct an analysis of the DeepTracer's computation time which is presented in the plot diagram in Figure 4.6. It shows that smaller structure predictions take less than a minute to finish and structures with up to around 5,000 residues can usually be predicted in less than 10 minutes. Even very large structures with more than 100,000 residues were predicted by the DeepTracer in under 10 hours. All predictions were run on a machine with an Nvidia GeForce GTX 1080 Ti GPU, 8 processors, and 64 GB of memory. Although a comparison with the Phenix method provides limited insight as the predictions were conducted by the authors of the method on a different machine, it is still worth mentioning that according to [18] the prediction of the EMD-6630 density map took 3 hours to complete while the EMD-9565 map took 129 hours

Table 4.2: Comparison of DeepTracer with MAINMAST and Rosetta on a dataset of 9 density maps.

| Method | Protein | % Matching | RMSD | % Seq Matching | % FP |
|---|---|---|---|---|---|
| DeepTracer | BPP1 | 93.30 | 0.82 | 66.20 | 2.56 |
| | FrhB | 98.20 | 0.67 | 68.40 | 3.17 |
| | T20S | 86.40 | 1.19 | 33.50 | 4.55 |
| | VP6 | 93.20 | 0.98 | 48.10 | 2.38 |
| | TRPV1 | 87.70 | 0.85 | 71.30 | 3.20 |
| | FrhA | 97.90 | 0.60 | 98.10 | 1.31 |
| | FrhG | 96.10 | 0.86 | 90.40 | 5.63 |
| | STIV | 90.40 | 0.88 | 70.10 | 0.32 |
| | TMV | 97.40 | 0.84 | 58.30 | 3.21 |
| | **Avg.** | **93.40** | **0.85** | **67.16** | **2.93** |
| Rosetta | BPP1 | 73.10 | 1.51 | 58.20 | 28.44 |
| | FrhB | 90.00 | 1.07 | 93.30 | 10.68 |
| | T20S | 77.40 | 1.61 | 62.60 | 24.89 |
| | VP6 | 72.00 | 1.37 | 48.60 | 28.72 |
| | TRPV1 | 84.20 | 1.22 | 69.70 | 18.10 |
| | FrhA | 92.50 | 0.90 | 93.30 | 8.29 |
| | FrhG | 63.20 | 1.79 | 37.50 | 41.23 |
| | STIV | 47.70 | 1.65 | 46.30 | 52.75 |
| | TMV | 91.00 | 1.20 | 88.70 | 10.97 |
| | **Avg.** | **76.79** | **1.37** | **66.47** | **24.89** |
| MAINMAST | BPP1 | 17.40 | 0.68 | 100.00 | 0.00 |
| | FrhB | 59.60 | 0.77 | 98.80 | 2.34 |
| | T20S | 23.10 | 1.01 | 100.00 | 1.96 |
| | VP6 | 26.20 | 0.73 | 99.00 | 0.00 |
| | TRPV1 | 33.20 | 0.72 | 99.00 | 0.00 |
| | FrhA | 57.70 | 0.62 | 97.70 | 0.89 |
| | FrhG | 31.10 | 0.72 | 100.00 | 1.39 |
| | STIV | 14.80 | 0.68 | 100.00 | 0.00 |
| | TMV | 64.50 | 0.78 | 98.00 | 0.99 |
| | **Avg.** | **36.40** | **0.75** | **99.17** | **0.84** |

to finish. The computation times of the DeepTracer for the same density maps were 1 minute and 9 hours, respectively.

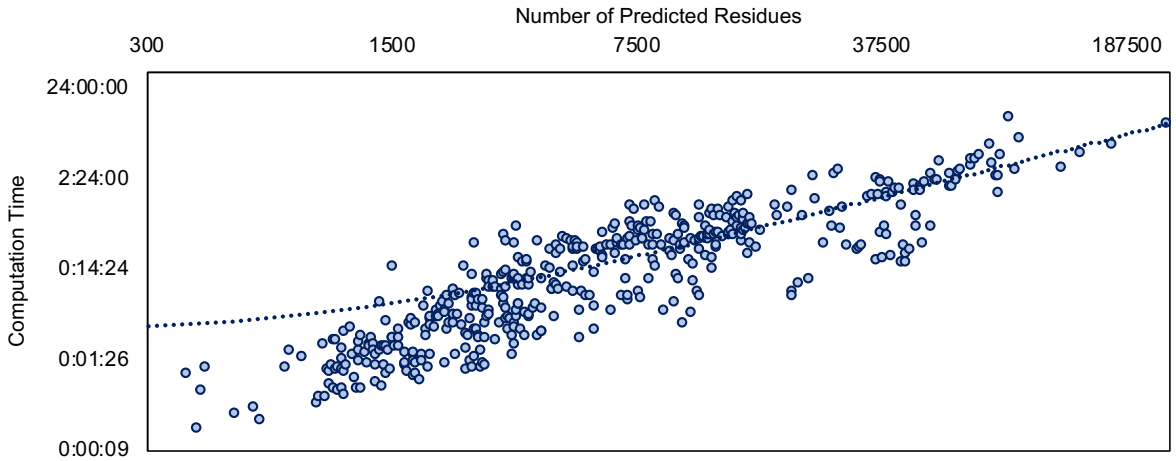


Figure 4.6: Computation times of the DeepTracer for the predictions of the Phenix test dataset. Both axes have a logarithmic scale.

# Chapter 5

# CONCLUSION

In this thesis, we presented DeepTracer, a fully-automatic tool that determines the all-atom structures of protein complexes based on their cryo-EM density maps, using a tailored deep convolutional neural network and a set of computational methods. We applied this novel software on a set of coronavirus-related density maps and compared the results to Phenix, the state of the art cryo-EM model determination method [18]. We found that DeepTracer correctly placed, on average, around 30% more residues than Phenix with an average RMSD improvement of 0.11Å, from 1.29Å to 1.18Å. We also applied DeepTracer on a dataset of coronavirus-related density maps and calculated a coverage of 84% compared to 49.8% with Phenix and an average RMSD value of 0.93Å for DeepTracer and 1.37Å for Phenix. Furthermore, we compared DeepTracer with Rosetta and MAINMAST on a previously published set of nine density maps and observed significant RMSD improvements in comparison with Rosetta from 1.37Å to 0.85Å and much more complete models compared to MAINMAST with a coverage increase of 57%, from 36.4% to 93.4%. These results represent a significant accuracy boost, resulting in more complete protein structures. Particularly, for large protein complexes, DeepTracer built models much faster than other methods, tracing tens of thousands of residues with million of atoms within only a few hours. We achieved the results without any manual pre-processing steps, such as zoning or cutting of the density map using a deposited model structure. This means we can determine models without any prior knowledge about the cryo-EM map, and the users do not need to tune any parameters in order to obtain an accurate structure.

As the cryo-EM technology becomes more readily available, the number of captured density maps, especially larger protein complexes, is rising rapidly. DeepTracer allows for a

greater throughput of cryo-EM as it can automatically and accurately infer structural information from density maps of macromolecules. This ultimately accelerates the scientific discovery process, which is particularly urgent today, given the ongoing coronavirus pandemic. Coronavirus-related density maps are deposited to the EMDR on a daily basis. Our efficient and automated method to model these maps is an important tool for researchers to resolve the structural information of the virus-related macromolecules.

## *5.1 Future Work*

To further improve the prediction accuracy and efficiency of DeepTracer there are several promising areas for future work. Firstly, the method does not yet take full advantage of the the breadth of information contained in the input amino acid sequence. With its current implementation, DeepTracer only uses the sequence to update the type of predicted amino acids. However, the sequence also describes exactly how many amino acids the protein complex consists of and how they are connected. This information could be used to add and remove predicted amino acids as well as to fix incorrectly connected chains. Future work could also include a refinement of the neural network architecture. Currently, the neural network consist of four parallel U-Nets that share only a single input layer. However, much of the prediction work performed by each U-Net is similar. Therefore, it might be promising to connect the separate U-Nets such that they share more layers which would decrease the overall size of the neural network boosting its efficiency. Lastly, the prediction accuracy is likely to improve if we filter faulty entries from the training dataset. An entry is considered faulty if the deposited model structure does not align with the density map. Removing such entries will prevent the neural network from learning incorrect patterns which is expected to improve its prediction accuracy.

# BIBLIOGRAPHY

[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[2] Carl Ivar Branden and John Tooze. *Introduction to protein structure*. Garland Science, 2012.

[3] Fredric S Cohen. How viruses invade cells. *Biophysical journal*, 110(5):1028–1032, 2016.

[4] Stefania Bambini and Rino Rappuoli. The use of genomics in microbial vaccine development. *Drug discovery today*, 14(5-6):252–260, 2009.

[5] Xiao-Chen Bai, Greg McMullan, and Sjors HW Scheres. How cryo-em is revolutionizing structural biology. *Trends in biochemical sciences*, 40(1):49–57, 2015.

[6] Yuanchen Dong, Shuwen Zhang, Zhaolong Wu, Xuemei Li, Wei Li Wang, Yanan Zhu, Svetla Stoilova-McPhie, Ying Lu, Daniel Finley, and Youdong Mao. Cryo-em structures and dynamics of substrate-engaged human 26s proteasome. *Nature*, 565(7737):49–55, 2019.

[7] Kaiming Zhang, Huawei Zhang, Shanshan Li, Grigore D Pintilie, Tung-Chung Mou, Yuanzhu Gao, Qinfen Zhang, Henry van den Bedem, Michael F Schmid, Shannon Wing Ngor Au, et al. Cryo-em structures of helicobacter pylori vacuolating cytotoxin a oligomeric assemblies at near-atomic resolution. *Proceedings of the National Academy of Sciences*, 116(14):6800–6805, 2019.

[8] Jacques Dubochet, Marc Adrian, Jiin-Ju Chang, Jean-Claude Homo, Jean Lepault, Alasdair W McDowall, and Patrick Schultz. Cryo-electron microscopy of vitrified specimens. *Quarterly reviews of biophysics*, 21(2):129–228, 1988.

[9] Anchi Cheng, Richard Henderson, David Mastronarde, Steven J. Ludtke, Remco H.M. Schoenmakers, Judith Short, Roberto Marabini, Sargis Dallakyan, David Agard, and Martyn Winn. Mrc2014: Extensions to the mrc format header for electron cryo-microscopy and tomography. *Journal of Structural Biology*, 192(2):146 – 150, 2015. Recent Advances in Detector Technologies and Applications for Molecular TEM.

[10] Emdataresource. https://www.emdataresource.org/. (Accessed on 03/04/2020).

[11] Rcsb pdb. https://www.rcsb.org/. (Accessed on 03/04/2020).

[12] A Senior, J Jumper, and D Hassabis. Alphafold: Using ai for scientific discovery. *DeepMind. Recuperado de: https://deepmind. com/blog/alphafold*, 2018.

[13] Alexis Lamiable, Pierre Thévenet, Julien Rey, Marek Vavrusa, Philippe Derreumaux, and Pierre Tufféry. Pep-fold3: faster de novo structure prediction for linear peptides in solution and in complex. *Nucleic acids research*, 44(W1):W449–W454, 2016.

[14] Philip Bradley, Kira MS Misura, and David Baker. Toward high-resolution de novo structure prediction for small proteins. *Science*, 309(5742):1868–1871, 2005.

[15] W Gordon Crewther and Adam S Inglis. Automatic procedures for determining amino acid sequences of peptides. *Analytical biochemistry*, 68(2):572–585, 1975.

[16] David E Kim, Ben Blum, Philip Bradley, and David Baker. Sampling bottlenecks in de novo protein structure prediction. *Journal of molecular biology*, 393(1):249–260, 2009.

[17] Dorothee Liebschner, Pavel V Afonine, Matthew L Baker, Gábor Bunkóczi, Vincent B Chen, Tristan I Croll, Bradley Hintze, L-W Hung, Swati Jain, Airlie J McCoy, et al. Macromolecular structure determination using x-rays, neutrons and electrons: recent developments in phenix. *Acta Crystallographica Section D: Structural Biology*, 75(10):861–877, 2019.

[18] Thomas C Terwilliger, Paul D Adams, Pavel V Afonine, and Oleg V Sobolev. A fully automatic method yielding initial models from high-resolution cryo-electron microscopy maps. *Nature methods*, 15(11):905–908, 2018.

[19] pickle — Python object serialization — Python 3.8.2rc2 documentation.

[20] Eric F Pettersen, Thomas D Goddard, Conrad C Huang, Gregory S Couch, Daniel M Greenblatt, Elaine C Meng, and Thomas E Ferrin. Ucsf chimera—a visualization system for exploratory research and analysis. *Journal of computational chemistry*, 25(13):1605–1612, 2004.

[21] vop. https://www.cgl.ucsf.edu/chimera/docs/UsersGuide/midas/vop.html\#resample. (Accessed on 03/01/2020).

[22] Spencer Moritz, Jonas Pfab, Tianqi Wu, Jie Hou, Jianlin Cheng, Renzhi Cao, Liguo Wang, and Dong Si. Cascaded-cnn: deep learning to predict protein backbone structure from high-resolution cryo-em density maps. *BioRxiv*, page 572990, 2019.

[23] Hdf5 acknowledgments. `https://support.hdfgroup.org/HDF5/acknowledge5.html`. (Accessed on 03/06/2020).

[24] Lida Abdi and Sattar Hashemi. To combat multi-class imbalanced problems by means of over-sampling techniques. *IEEE transactions on Knowledge and Data Engineering*, 28(1):238–251, 2015.

[25] Xu-Ying Liu, Jianxin Wu, and Zhi-Hua Zhou. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(2):539–550, 2008.

[26] Yuri Sousa Aurelio, Gustavo Matheus de Almeida, Cristiano Leite de Castro, and Antonio Padua Braga. Learning from imbalanced data sets with weighted cross-entropy function. *Neural Processing Letters*, 50(2):1937–1949, 2019.

[27] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[28] Frances C Bernstein, Thomas F Koetzle, Graheme JB Williams, Edgar F Meyer Jr, Michael D Brice, John R Rodgers, Olga Kennard, Takehiko Shimanouchi, and Mitsuo Tasumi. The protein data bank: A computer-based archival file for macromolecular structures. *European journal of biochemistry*, 80(2):319–324, 1977.

[29] Sandeep Chakraborty, Ravindra Venkatramani, Basuthkar J Rao, Bjarni Asgeirsson, and Abhaya M Dandekar. Protein structure quality assessment based on the distance profiles of consecutive backbone c$\alpha$ atoms. *F1000Research*, 2, 2013.

[30] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.

[31] Siavash Mirarab, Nam Nguyen, Sheng Guo, Li-San Wang, Junhyong Kim, and Tandy Warnow. Pasta: ultra-large multiple sequence alignment for nucleotide and amino-acid sequences. *Journal of Computational Biology*, 22(5):377–386, 2015.

[32] Desmond G Higgins and Paul M Sharp. Clustal: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1):237–244, 1988.

[33] Chuong B Do, Mahathi SP Mahabhashyam, Michael Brudno, and Serafim Batzoglou. Probcons: Probabilistic consistency-based multiple sequence alignment. *Genome research*, 15(2):330–340, 2005.

[34] David B Wagner. Dynamic programming. *The Mathematica Journal*, 5(4):42–51, 1995.

[35] Coronavirus: Emdataresource. `https://www.emdataresource.org/news/coronavirus\_resources.html`. (Accessed on 05/09/2020).

[36] Renhong Yan, Yuanyuan Zhang, Yaning Li, Lu Xia, Yingying Guo, and Qiang Zhou. Structural basis for the recognition of sars-cov-2 by full-length human ace2. *Science*, 367(6485):1444–1448, 2020.

[37] Brandon Frenz, Alexandra C Walls, Edward H Egelman, David Veesler, and Frank DiMaio. Rosettaes: a sampling strategy enabling automated interpretation of difficult cryo-em maps. *Nature methods*, 14(8):797–800, 2017.

[38] Genki Terashi and Daisuke Kihara. De novo main-chain modeling for em maps using mainmast. *Nature communications*, 9(1):1–11, 2018.