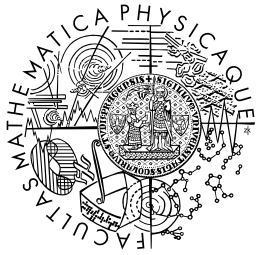

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Lukáš Polák

Prediction and visualization of cryptic binding regions

Department of Software Engineering

Supervisor of the master thesis: doc. RNDr. David Hoksza, Ph.D.

Study programme: Computer Science - Software and Data Engineering

Prague 2025

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

In ..... date ....

Author's signature

I would like to thank my supervisor, doc. RNDr. David Hoksza, Ph.D., for his guidance and help in the field of bioinformatics and during the writing of this thesis. I am also grateful to Mgr. Vít Škrhák for his valuable assistance with CryptoBench and the development of the smoothing model. My thanks extend to all colleagues from the Charles University Structural Bioinformatics Group for their insightful feedback and discussions that contributed to the improvement of my work. Above all, I am really thankful to my family and friends, whose support and encouragement have been essential throughout my studies.

This thesis was written with the assistance of artificial intelligence tools, including but not limited to large language models for text rephrasing and code development. All AI-generated content has been manually reviewed, verified, and integrated by the author. The author takes full responsibility for the accuracy and integrity of all work presented herein.

Title: Prediction and visualization of cryptic binding regions

Author: Bc. Lukáš Polák

Department: Department of Software Engineering

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

Abstract: This thesis addresses the prediction and visualization of cryptic binding sites (CBS) in protein structures, which are important for drug discovery. Building on the CryptoBench dataset and prediction model developed earlier at FMP CUNI, we introduce a methodology for clustering residue-level predictions into contiguous cryptic binding sites. The approach is integrated into CryptoShow, a newly created web application that enables users to submit protein structures, obtain CBS predictions, and visualize results interactively. The system supports both public and custom protein structures, leveraging machine learning and modern visualization tools to highlight binding regions and to show potential conformational changes. The thesis details the development of the clustering algorithm, evaluation of the methodology, and the software architecture of CryptoShow.

Keywords: bioinformatics, protein, binding sites, machine learning

Název práce: Predikce a vizualizace kryptických vazebných míst

Autor: Bc. Lukáš Polák

Katedra: Katedra softwarového inženýrství

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

Abstrakt: Tato diplomová práce se zabývá predikcí a vizualizací kryptických vazebných míst (CBS) v proteinových strukturách, která jsou důležitá při vývoji léčiv. Na základě datasetu CryptoBench a jeho modelu predikce CBS vyvinutého dříve na MFF UK představujeme metodologii pro seskupování predikcí na úrovni aminokyselin do ucelených kryptických vazebných míst. Tento přístup je integrován do nově vytvořené webové aplikace CryptoShow, která umožňuje uživatelům nahrát proteinové struktury, získávat predikce CBS a interaktivně vizualizovat výsledky. Aplikace podporuje jak veřejně dostupné, tak vlastní proteinové struktury a využívá strojové učení a moderní vizualizační nástroje k zvýraznění vazebných oblastí a zobrazení potenciálních konformačních změn mezi strukturami. Práce podrobně popisuje vývoj clusterovacího algoritmu, evaluaci metodologie a softwarovou architekturu aplikace CryptoShow.

Klíčová slova: bioinformatika, protein, vazebná místa, strojové učení

Contents

Introduction

Proteins are essential to a wide range of biological processes. Proteins consist of amino acids, which are the building blocks of these macromolecules. The amino acids form a chain that determines the protein's structure and function. Specific amino acids, referred to as binding, are more likely to interact with other molecules called ligands. This interaction is a fundamental property to drug discovery and research [1], [2], [3].

Detecting such residues, individual amino acids within the protein structure, is challenging. Various computational methods have been developed to predict potential binding residues, which are further grouped into distinct binding regions (also referred to as binding sites) based on the location of the residues [4], [5], [6], [7]. Specific binding residues may experience structural changes due to the dynamic nature of these regions, becoming cryptic binding residues (CBRs) that pose greater identification challenges. While current methods may detect cryptic binding sites (CBSs), their models are generally trained to identify any binding residues, rather than being focused explicitly on CBSs.

A dataset and methodology for detecting CBRs, CryptoBench [8], was previously developed at the Faculty of Mathematics and Physics, Charles University. This thesis has two primary objectives.

The first goal is to extend this approach by clustering individually predicted CBRs from the CryptoBench model into structurally contiguous regions called CBSs. This methodology may also apply to other prediction methods.

The second objective is to develop a user-friendly client-server web application, CryptoShow, enabling users to submit protein structures and obtain CBR predictions. These predictions would be clustered into distinct binding sites and visualized for the user, including animations that illustrate potential conformational changes between the protein's initial state and analogous structures identified using the AHoJ tool [9].

This thesis is structured in four chapters, and it is organized as follows:

Chapter 1 provides an overview of bioinformatics, introducing key terminology, concepts, and relevant tools that underpin subsequent work.

Chapter 2 details the methodological framework, beginning with predicting cryptic binding residues using the CryptoBench model. The prediction is followed by describing the clustering approach for grouping CBRs into cryptic binding sites, enhanced by a newly introduced smoothing model. It concludes with an evaluation of the overall performance.

Chapter 3 presents the design and implementation of the CryptoShow web application. This presentation includes a discussion of the technology stack, architectural and design choices, and an in-depth look at both the backend and the frontend, particularly the integration of Mol* for 3D structure visualization [10]. Deployment and maintenance considerations are also addressed.

Finally, Chapter 4 serves as user documentation, outlining the application's main functionalities and illustrating a typical use case.

1 Introduction to Bioinformatics

This chapter offers an overview of the bioinformatics field, focusing on key molecular biology concepts needed in the context of this thesis. It covers proteins, their sequences and structures, ligands, binding sites (including cryptic binding sites), major biological databases and file formats, and widely used tools for visualization, binding site prediction, and structural analysis.

1.1 Proteins, Ligands, and Amino Acids

Proteins are vital macromolecules involved in numerous biological functions, such as catalyzing biochemical reactions, offering structural support, and regulating cellular activities [11]. They consist of **amino acids** connected by **peptide bonds**, forming **polypeptide chains**. Amino acids are molecules containing an amino group ($-NH_2$), a carboxyl group ($-COOH$), and a unique side chain (R group) that determines the amino acid's properties. The individual amino acid units within the polypeptide chain are referred to as **residues**, which may be used interchangeably with amino acids throughout this thesis. The sequence of these amino acids in a polypeptide chain is known as the **primary structure** of a protein. The protein's sequence determines its function [12], [13]. There are **20 standard amino acids**, each with distinct characteristics affecting protein folding and interactions with other molecules. While many non-standard amino acids are used in drug discovery and development [14], this thesis will primarily consider the 20 standard amino acids that form the basic building blocks of proteins.

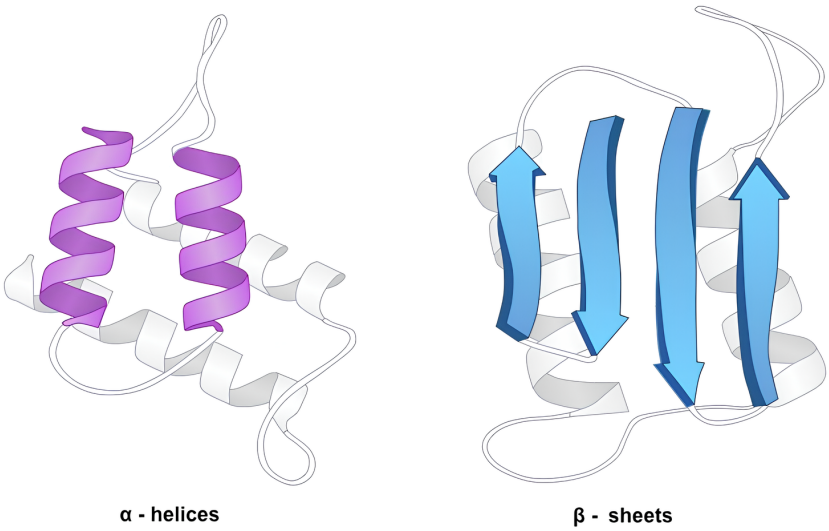

Figure 1.1 Illustration of an alpha helix and a beta sheet, the two most common types of protein secondary structure. Adapted from an image by BioNinja [15].

The second level of protein structure is the **secondary structure**, which refers to local folding patterns within the polypeptide chain. The most common secondary structures are **alpha helices** and **beta sheets**, as shown in Figure 1.1. These structures arise from hydrogen bonding between the backbone atoms of

the amino acids, stabilizing the overall protein structure. All proteins also have a **three-dimensional (3D) structure**, also known as **tertiary structure**, which is crucial for their function [12], [13].

The secondary and tertiary structures are determined by the interactions between the side chains of the amino acids, including hydrophobic interactions, hydrogen bonds, ionic bonds, and disulfide bridges. The final level of protein structure is the **quaternary structure**, which refers to assembling multiple polypeptide chains into a functional protein complex [12], [13].

The tertiary and quaternary structures of proteins are determined through experimental methods such as X-ray crystallography and nuclear magnetic resonance (NMR) spectroscopy [16]. However, with the rise of deep learning and artificial intelligence, it is now also possible to predict protein structures from their amino acid sequences with remarkable accuracy. The most notable example is AlphaFold [17], [18], a deep learning model developed by DeepMind that has achieved impressive accuracy in predicting protein structures and has been widely adopted in bioinformatics. AlphaFold also provides a per-residue confidence metric called the predicted Local Distance Difference Test (plDDT) score[1], which indicates the reliability of the predicted atomic positions. In recognition of the profound impact of these advances, the Nobel Prize in Chemistry was awarded in 2024 for developing methods for predicting protein structures using artificial intelligence [19]. Other notable tools introduced in the past months include Boltz-2 [20] and Chai-1 [21]. Today, predicted structures often closely match experimental results, as illustrated in Figure 1.2.

Protein function is primarily determined by interactions with other molecules, known as ligands. Small molecules, ions, RNA molecules, lipids, saccharides, or other molecules that bind to specific regions often result in conformational changes leading to different functions. Such interactions are crucial for enzymatic catalysis, signal transduction, and immune response. Identifying and characterizing these sites is essential in drug discovery, with computational approaches aiding both new therapeutic development and drug repurposing [22]. Additionally, de novo protein design allows for creating proteins with tailored functions, often guided by predictive tools such as AlphaFold before experimental testing [23].

1.2 Cryptic Binding Sites (CBSs)

Binding sites are distinct regions on a protein where ligands can bind and interact. These sites manifest as cavities or grooves on the protein surface and are frequently linked to conformational changes that influence protein function. Proteins with a ligand bound are termed **holo** conformations, while those lacking a ligand are called **apo** conformations. In holo conformations, the binding site is defined by the ligand's position, whereas apo conformations may contain several potential binding sites, also commonly known as **pockets**.

Cryptic binding sites (CBSs) are regions on proteins that are not visible or accessible in the unbound (apo) structure but can form and become available

[1]The predicted Local Distance Difference Test (plDDT) score is a per-residue confidence metric output by AlphaFold. It indicates the reliability of the predicted atomic positions; higher values suggest greater confidence. In the color scheme, blue indicates the most confident regions, while orange/yellow depict less confident regions.

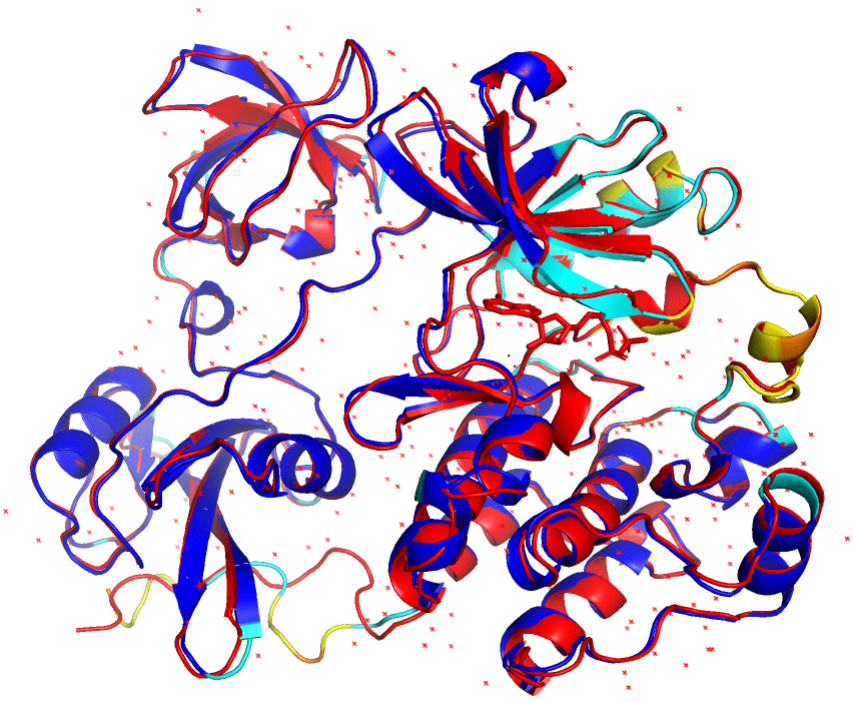

Figure 1.2 Comparison of AlphaFold predicted structure and experimental structure of the 2SRC protein structure (Crystal Structure of Human Tyrosine-Protein Kinase C-SRC, in Complex with AMP-PNP). The predicted structure is shown in blue to yellow (depicting the plDDT score), while the experimental structure is shown in red. The two structures are nearly identical after the alignment, demonstrating the accuracy of AlphaFold predictions. Image created in PyMOL.

for ligand binding following conformational changes. These sites typically appear only after ligand interaction or other molecular events that induce structural rearrangements.

A key concept in the study of CBSs is using **apo–holo protein pairs**. An **apo structure** refers to a protein in its unbound state (without a ligand), while a **holo structure** is the same protein bound to a ligand. Comparing these paired structures allows researchers to identify binding sites only present or accessible in the holo form, thus revealing cryptic sites.

Computational methods such as CryptoBench [8] have been created to predict cryptic binding sites. A significant contribution in this area is CryptoSite [24], which leverages known apo–holo protein pairs to identify CBSs, offering an alternative to time-consuming molecular dynamics simulations. Both of the methods will be covered in more detail in Section 1.4.

1.3 Databases and File Formats

This section overviews the key bioinformatics databases for protein structures and introduces the main file formats used in this field.

1.3.1 PDB Archive and RCSB PDB

The PDB (Protein Data Bank) Archive, often called the "PDB database", is a comprehensive repository of 3D structural data for experimentally determined proteins, nucleic acids, and complex assemblies. It is a foundational resource for bioinformatics, with many machine learning and deep learning models relying on its data for training and validation. The archive is maintained by the wwPDB (Worldwide PDB) organization [25], which includes the RCSB PDB (Research Collaboratory for Structural Bioinformatics Protein Data Bank) [16], PDBe (Protein Data Bank in Europe) [26], PDBj (Protein Data Bank Japan) [27], and other members.

The RCSB PDB (Research Collaboratory for Structural Bioinformatics Protein Data Bank) group offers an REST API for programmatic data access, which is publicly accessible at `https://www.rcsb.org/`. As of June 2025, the database contains over 238,000 structures.

Many organizations also maintain local, enriched copies of the PDB archive. The open sharing of structural data is encouraged, as it supports developing and improving computational models used throughout the scientific community [28].

1.3.2 AlphaFold Database

The AlphaFold Database [29] contains over 200 million protein structures predicted by the AlphaFold model. While most of these structures have not been experimentally validated, those with high pLDDT scores are considered reliable. It is important to note that certain protein regions, known as intrinsically disordered regions, may exhibit low pLDDT scores due to their inherent flexibility and lack of stable structure. The database is publicly available at `https://alphafold.ebi.ac.uk/`.

1.3.3 UniProt Database

The UniProt database [30] is a comprehensive protein sequence and functional information resource. Each protein is assigned a unique **UniProt accession** (also called **protein ID**), which serves as a standardized identifier for that specific protein. A single UniProt accession may correspond to multiple experimental structures in the PDB database, representing different conformations or states of the same protein, while typically having one predicted structure in the AlphaFold database. UniProt accessions are commonly used to link PDB and AlphaFold entries. UniProt is available at `https://www.uniprot.org/`. The central part of the UniProt database is the UniProt Knowledgebase (UniProtKB), which contains two sections: UniProtKB/Swiss-Prot, which includes manually curated entries with high-quality annotations, and UniProtKB/TrEMBL, which contains automatically annotated entries that have not yet been reviewed [31].

1.3.4 PDB File Format

The PDB (Protein Data Bank) file format is a text-based format, representing three-dimensional structures of biological molecules, primarily proteins and nucleic acids. It contains information about atom coordinates, connectivity, b-factors, and

other structural details. Each PDB file should start with header lines providing metadata about the structure, such as the structure name, authors, methods used, and additional comments. Then, the atomic coordinates are listed in a list of ATOM and HETATM records, which specify the atom type, residue name, chain identifier, residue sequence number, and the x, y, z coordinates of each atom in the structure. The PDB format might include information about secondary structure elements, such as helices and sheets [32]. Some information might be omitted, especially in when working with structures generated by deep learning methods, such as RFDiffusion [33]. A single PDB file may contain multiple models representing distinct conformations of the same protein. It is worth noting that the PDB format is considered deprecated in favor of the more modern mmCIF format (covered in 1.3.5), though it remains widely used due to its simplicity and broad tool support.

```
1  ATOM      1  N   ASN A   1      -9.281  -0.356  10.387  1.00  1.00
2  ATOM      2  CA  ASN A   1      -9.739   0.620  11.367  1.00  1.00
3  ATOM      3  C   ASN A   1      -9.686   0.050  12.779  1.00  1.00
4  ATOM      4  O   ASN A   1      -9.577   0.792  13.755  1.00  1.00
5  ATOM      5  N   LEU A   2      -9.786  -1.133  12.898  1.00  1.00
6  ATOM      6  CA  LEU A   2      -9.557  -1.787  14.181  1.00  1.00
7  ATOM      7  C   LEU A   2      -8.101  -2.218  14.328  1.00  1.00
8  ATOM      8  O   LEU A   2      -7.570  -2.940  13.485  1.00  1.00
9  ATOM      9  N   TRP A   3      -7.476  -1.724  15.307  1.00  1.00
10 ATOM     10  CA  TRP A   3      -6.033  -1.854  15.472  1.00  1.00
11 ATOM     11  C   TRP A   3      -5.622  -3.315  15.593  1.00  1.00
12 ATOM     12  O   TRP A   3      -4.590  -3.726  15.063  1.00  1.00
13...
14 ATOM    201  N   LEU A  51       7.711  11.573  -4.452  1.00  1.00
15 ATOM    202  CA  LEU A  51       8.073  11.629  -3.040  1.00  1.00
16 ATOM    203  C   LEU A  51       6.893  11.249  -2.153  1.00  1.00
17 ATOM    204  O   LEU A  51       5.780  11.743  -2.338  1.00  1.00
18 ATOM    205  N   TYR A  52       7.039  10.233  -1.420  1.00  1.00
19 ATOM    206  CA  TYR A  52       5.980   9.756  -0.539  1.00  1.00
20 ATOM    207  C   TYR A  52       6.430   9.763   0.917  1.00  1.00
21 ATOM    208  O   TYR A  52       7.618   9.629   1.210  1.00  1.00
```

Listing 1.1 An edited example of a PDB file containing information about a protein structure generated by RFDiffusion.

1.3.5 mmCIF File Format

The mmCIF (Macromolecular Crystallographic Information File, also called PDBx) format is a more modern and flexible alternative to the PDB format, designed to overcome the limitations of the PDB format, especially for large structures [34]. The PDB database, mentioned in Section 1.3.1, has been transitioning to the mmCIF format for new entries. Although it is encouraged to use mmCIF for new structures, many existing tools and databases still rely on the PDB format thanks to its simpler format. Like PDB files, mmCIF files can also represent multiple conformations of the same structure.

1.3.6 FASTA File Format

The FASTA file format is a widely used, text-based format for representing biological sequences, such as proteins or nucleic acids. Each sequence entry begins with a header line that starts with a ">" character, followed by a description or identifier. The sequence itself is written on the following lines, typically using single-letter codes. Multiple sequences can be included in a single FASTA file, each with its own header. While the header line is commonly present, it is technically optional, which allows easy creation of FASTA files [35].

```
1 >6A5J_1|Chain A|ILE-LYS-LYS-ILE-LEU-SER-LYS-ILE-LYS-LYS-LEU-LEU-LYS|Rana
    chensinensis (79015)
2 IKKILSKIKKLLK
```

Listing 1.2 An example of a FASTA file containing information about the 6A5J sequence (containing only 13 amino acids).

1.3.7 XTC File Format

The XTC (eXtended Trajectory Coordinate) file format is a binary format commonly used to store molecular dynamics simulation trajectories, especially in the GROMACS software package [36]. It is widely used for efficiently storing large amounts of trajectory data, including atomic coordinates, velocities, and box dimensions, in a compressed binary format. In the context of bioinformatics, XTC files are often used to capture and analyze protein trajectory changes over time, enabling researchers to study conformational dynamics and structural transitions during simulations.

1.4 Related Tools and Projects

This section is dedicated to the most relevant tools and projects necessary for the work presented in this thesis.

1.4.1 P2Rank and PrankWeb

P2Rank is a Groovy/Java-based standalone tool for predicting binding sites in protein structures. The P2Rank algorithm is based on machine learning. The structure is covered with a grid of points on the protein's solvent accessible surface, and each point is assigned a score based on the likelihood of being a binding site based on the local environment of the point. Then, the points are clustered and ranked based on their scores. The pre-trained tool does not rely on any external databases or templates. The method is widely adopted due to its reliability and accurate predictions and speed [4]. Source codes for P2Rank are available at `https://github.com/rdk/p2rank`.

PrankWeb is a web-based interface for P2Rank, allowing simple access to the P2Rank method without the need to run and install the tool locally. It is available at `https://prankweb.cz/`. The web interface allows users to upload

a PDB/mmCIF file and receive a visualization of the prediction of the binding sites, with the predicted binding sites colored according to their scores. Moreover, PrankWeb allows the users to perform molecular docking using the predicted binding sites, which is a common post-processing step in drug discovery [37], [38], [39]. The docking is performed using the AutoDock Vina [40] software, a widely used molecular docking tool.

The PrankWeb interface was a huge inspiration for the design of the web interface presented in this thesis as the goal is to provide a similar user experience, but with a focus on cryptic binding sites and their specific challenges.

1.4.2 AHoJ

AHoJ (Apo–Holo Juxtaposition) is a web application for retreiving and visualizing apo-holo pairs of proteins (See Section 1.2). This functionality is especially useful for cryptic binding site predictions, as it allows the users to find similar structures with possibly similar properties based on the predicted CBS [9]. The application is available at `https://apoholo.cz/`. Currently, AHoJ supports searching for apo-holo pairs only by PDB ID.

AHoJ is also integrated within the CryptoShow interface as well. Section 3.2.3 will cover this in more detail.

1.4.3 PyMOL

PyMOL is a powerful molecular visualization tool written in Python. Users might use it either as a standalone application or a Python library. Thanks to its feature-rich API and UI, PyMOL is a common choice for researchers in structural biology and cheminformatics [41].

1.4.4 Mol*

Mol* (also MolStar) is a TypeScript library for visualizing molecular structures directly in web browsers. Thanks to its modern architecture, use of WebGL and rich feature set, Mol* is nowadays a very popular choice for web-based molecular visualization tools [42]. Mol* also provides an online version of a molecular viewer, which is available at `https://molstar.org/viewer/` [10]. A highly customized version of the viewer is also used in the CryptoShow interface, which will be covered in more detail in Section 3.3.

1.4.5 CryptoBench

CryptoBench is a comprehensive benchmark dataset for training and evaluating CBSs' prediction methods, built from a large set of apo–holo protein pairs and grouped by UniProt ID with predefined cross-validation splits. A part of the publication is dedicated to baseline evaluations, which show that a sequence-based neural network using protein language model embeddings outperforms state-of-the-art structure-based methods like PocketMiner [43] and P2Rank [4] across key metrics in terms of CBSs. The source codes for CryptoBench are available at `https://github.com/skrhakv/CryptoBench/` [8].

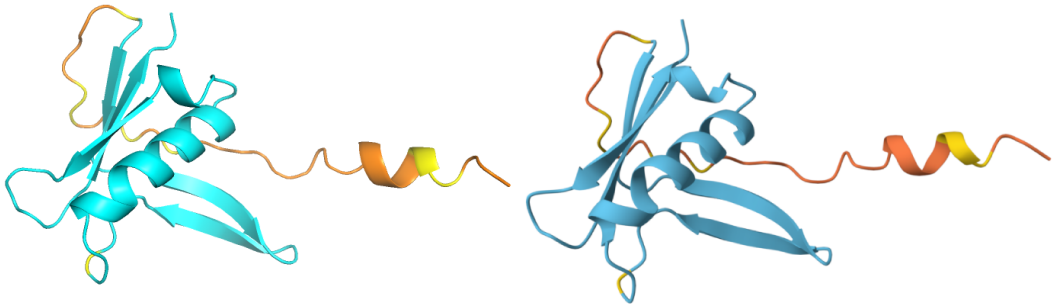

Figure 1.3 A comparison of the PyMOL (left) and Mol* (right) molecular viewers showing the AlphaFold predicted structure of the protein AF_AFQ9XWU9F1.

The model used in CryptoBench (referred to as **CB-Model** in the context of this thesis) is able to predict cryptic binding sites when provided with ESM-2 embeddings [44] of the protein sequence. The prediction capability represents one of the core functionalities of CryptoShow. More details about the CryptoBench method will be provided in Section 2.3.

1.4.6 CryptoSite

CryptoSite is another tool designed to identify CBSs. CryptoSite characterizes cryptic sites based on sequence, structure, and dynamics by constructing a structurally defined apo-holo pairs dataset. Leveraging these insights, CryptoSite uses machine learning to predict cryptic sites with high accuracy, expanding the set of potentially druggable proteins. The tool has been applied to the human proteome, increasing the fraction of disease-associated proteins considered druggable. Experimental validation further demonstrates the practical use of CryptoSite. The web server is accessible at `https://modbase.compbio.ucsf.edu/cryptosite` [24].

While CryptoSite is a valuable resource for cryptic binding site analysis and its web server remains available, its computational processes often take several days to complete, limiting its suitability for interactive use. In contrast, a primary goal of CryptoShow is to provide a fast and interactive experience, allowing users to obtain results within minutes. The relatively low usage of CryptoSite underscores this need; as of June 2025, only 10 jobs were submitted in the last 7 days. Also, the results of the CryptoSite jobs are available only for 7 days after the job completion.

2 Methodology

This chapter covers the methodology used in this work. Section 3.2 provides more details on the implementation in the application.

2.1 Pipeline Overview

The CryptoShow pipeline consists of several steps that work together to predict cryptic binding sites in protein structures and visualize them. The pipeline follows the steps outlined below, supported by Figure 2.1. The clustering, pocket smoothing, and visualization steps are newly implemented in this thesis.

1. **Structure Input** - The pipeline starts with a protein structure, which can be provided in PDB or mmCIF format. The structure can be obtained from the RCSB PDB database (see Section 1.3.1), the AlphaFold database (see Section 1.3.2), or a custom structure file.

2. **Sequence Extraction** - The sequence of the protein is extracted from the provided structure file. This step is essential as the subsequent prediction model operates on sequence data.

3. **Prediction** - The model used in CryptoBench (Section 2.3), referred to as **CB-Model**, is used to assign scores to individual residues on the protein sequence. The model generates residue-level predictions, which are then processed to identify potential binding site candidates.

4. **Clustering** - The predicted residues are clustered (Section 2.4) to form potential cryptic binding sites (pockets). This is crucial for visualization, as coloring the clusters instead of just individual residues increases the interpretability of the results.

5. **Pocket smoothing** - The pockets are then smoothed using a machine learning model (Section 2.4). This enhances the predictions by adding residues that are likely part of the cryptic binding site, even if their scores did not exceed the original high-scoring threshold.

6. **Visualization** - The predicted cryptic binding sites are visualized in the frontend application (Section 3.3). The visualization includes the predicted binding sites and their associated scores. Also, the user can run an AHoJ query to retrieve apo-holo pairs of protein structures and visualize the trajectory animation of the conformational changes between the apo and holo structures.

2.2 Protein Structure Input and Sequence Extraction

The first step in the pipeline is to provide a protein structure in either PDB or mmCIF format. The structure can be obtained from the PDB database [16] or

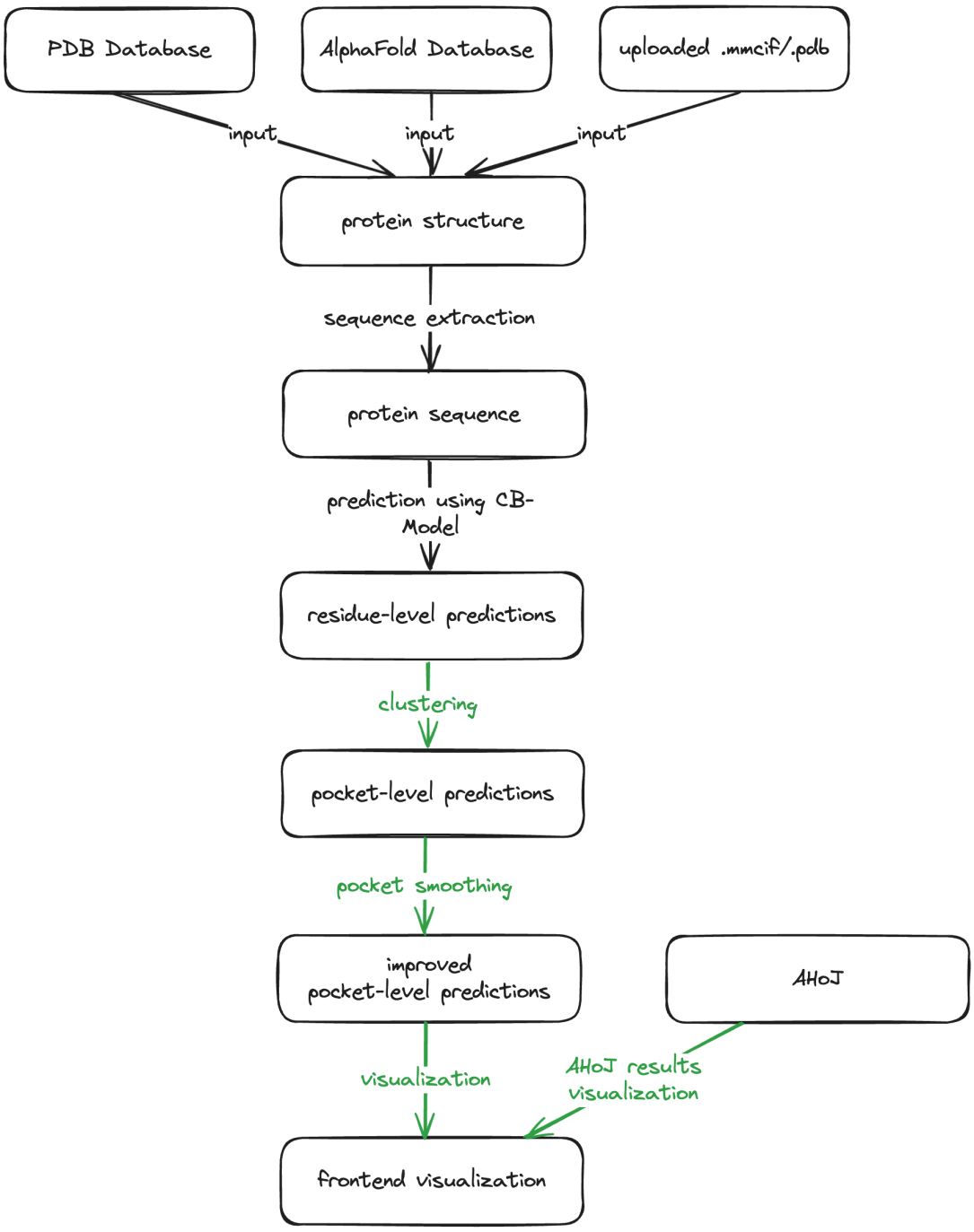

Figure 2.1 Overview of the pipeline. The pipeline consists of several steps, starting with the extraction of the sequence from a provided protein structure, followed by the prediction of cryptic binding sites using the CB-Model, followed by clustering and smoothing of the predictions, and finally querying AHoJ to retrieve relevant protein structures for trajectory animation. Steps marked with green arrows are newly implemented in this thesis.

the AlphaFold database [17], or it can be a custom structure file. The structure is then parsed to extract the protein sequence for the CB-Model prediction.

2.3 Prediction Using the CB-Model

The initial step in our methodology involves utilizing the CryptoBench [8] model to generate residue-level predictions for protein sequences. It is important to note that this model works solely with sequence information, without incorporating any structural (tertiary or quaternary) data. Specifically, we employ the tiny variant of the CryptoBench[1] model, which is trained on a reduced version of the fine-tuned ESM-2 embeddings [44] with 650 million parameters, as opposed to the full 3 billion. This choice is motivated by the observation that prediction accuracy remains comparable (according to the authors). At the same time, computational requirements are significantly reduced, enabling the pipeline to run efficiently on standard personal computers without the need for a GPU.

The CryptoBench dataset was created by filtering the AHoJ-DB [45] dataset, which contains 57.8 million binding pockets (both apo and holo) [46]. The filtering steps involved resolution filtering, removing redundant structures, geometric quality assurance, crypticity constraints, ligand filtering, and sequence clustering. The final dataset consists of 1107 apo-holo pairs with 5493 cryptic binding sites. This dataset is used to train the model that has been benchmarked against PocketMiner [43] and P2Rank [4], and has shown better performance in terms of cryptic binding site prediction.

2.4 Clustering and Smoothing

The next step in our methodology is to take the residue-level predictions generated by the CB-Model and cluster the high-scoring residues to form potential binding site candidates.

Many algorithms are available for clustering, namely DBSCAN [47], single-linkage clustering, hierarchical clustering [48], and many more. In our case, we opted for the DBSCAN algorithm, a density-based clustering algorithm that groups points that are closely packed together. This approach is particularly suitable for our use case, as it allows us to identify clusters of high-scoring residues without requiring a predefined number of clusters. It also allows the specification of minimum residue distance and minimum number of residues in a cluster.

Our method relies on three parameters to control the clustering process:

- **EPS (ϵ)** - This parameter defines the maximum distance between two residues for them to be considered part of the same cluster. In our case, we opted for a value of 5.0 Å, which is a reasonable distance for residues when considering their spatial proximity in a protein structure as the average distance between the Cα atoms[2] in a protein structure is 3.8 Å [49].

[1] Available at `https://github.com/skrhakv/TinyCryptobench`

[2] The Cα atom is the central carbon atom in the backbone of a protein, to which the amino group, carboxyl group, hydrogen atom, and side chain are attached.

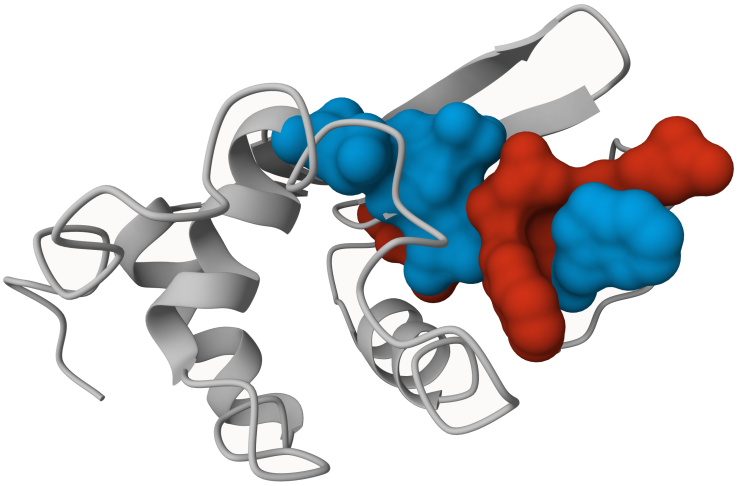

Figure 2.2 Difference between the predicted cryptic binding site before and after the smoothing step in the 1LYZ protein structure (Real-space refinement of the structure of hen egg-white lysozyme). The residues colored in blue depict the residues predicted as binding by the CB-Model, while the residues colored in red are the residues added by the smoothing model. Visualized using Mol*.

- **Minimum cluster size** - This parameter defines the minimum number of residues required to form a cluster. We set this value to 3, as two residues are insufficient to form a binding site candidate, as the spatial complexity of a binding site is usually higher. This is also supported by the fact that less than 0.2% of cryptic binding sites in the CryptoBench dataset have fewer than three residues.
- **Minimum (CB-Model) prediction score** - This parameter defines the minimum score for a residue to be marked as a high-scoring residue. We opted for a value of 0.7 (inspired by the CryptoBench publication [8]), as this covers most of the dataset's binding residues, while also not marking the entire protein as a binding site candidate.

After the clustering, we need to smooth the clusters to enhance the predictions. This is caused by the fact that the clustering is performed only on the high-scoring residues, which can lead to missing residues in the binding site (as shown in Section 2.5 and Figure 2.2). We collaborated with Vít Škrhák to develop an additional machine learning model to address this. This model incorporates the protein sequence, the predicted clusters from the clustering step (along with sequence information), precomputed ESM-2 embeddings, and the residue distance matrix derived from the structure file. This **smoothing model** is then trained on a modified version of the CryptoBench dataset, using the annotated cryptic binding sites as the training and test sets.

The initial step involves preparing the dataset to train the smoothing model. The dataset, consisting of 5493 cryptic binding sites, is split into training and testing sets using an 80:20 ratio. The sets follow the same structure as the CryptoBench dataset to prevent data leakage, i.e., the train set from CryptoBench is used as the train set for the smoothing model, and the test set from CryptoBench

is used as the test set for the smoothing model. The following criteria further filter the test set:

- Pockets spanning multiple structure chains are omitted from the evaluation, as the process would be much more complicated, and the prediction would require complex approaches.
- Many cryptic binding sites in the test set are similar, which can lead to heavily inflated results. To address this, we have used the Szymkiewicz–Simpson coefficient (overlap), which measures similarity between sets A and B. The coefficient is defined in Equation 2.1.

$$\text{Szymkiewicz–Simpson}(A, B) = \frac{|A \cap B|}{\min(|A|, |B|)} \tag{2.1}$$

  After computing the overlap between each pair of cryptic binding sites in one structure, we keep only the CBSs with the overlaps under 0.5.

After the filtering, we have a test set of 193 apo conformations and 230 cryptic binding sites (out of the original 570 before the filtering).

Before training, it is essential to ensure that the precomputed embeddings from the previous step are available. Subsequently, the distance matrix for the residue coordinates must be calculated, which is accomplished using Equation 2.2.

$$D_{ij} = \|\mathbf{c}_i - \mathbf{c}_j\|_2 = \sqrt{\sum_{k=1}^{3} (c_{ik} - c_{jk})^2} \tag{2.2}$$

where D_{ij} denotes the Euclidean distance between residues i and j, with $\mathbf{c}_i$ and $\mathbf{c}_j$ representing their respective 3D coordinates.

After the computation of the distance matrices, the next step involves generating positive and negative training examples for the dataset. For the positive examples, residues are identified as candidates if their distance to any binding residue is less than a specified positive threshold (set to 15 Å). For each such residue, a feature vector is constructed by combining its embedding with the mean embedding of its nearby binding neighbors. These feature vectors are then labeled as positive examples.

For negative examples, we identify residues located within a smaller threshold of 10 Å from binding residues but are not annotated as binding residues. These residues are labeled as negative examples. This strategy helps the model learn to differentiate between actual binding residues and nearby residues that do not participate in binding.

We begin training by calculating the class weights and loading the dataset. The neural network, which uses the same architecture described in Section 2.3, is then trained. Cross-validation was not performed here primarily because the CryptoBench dataset is already partitioned into well-defined train and test sets, minimizing the risk of data leakage. Additionally, cross-validation would not provide a significant additional benefit given the dataset's size and the existing split. Therefore, a straightforward train-test split was used, following the original CryptoBench partitioning. Evaluation of the model performance is described in Section 2.5.

The complete source code for the smoothing methodology can be found in the `cryptoshow-benchmark/smoothing` directory. This includes scripts for model training, prepared training and testing datasets, inference, and a utility script for distance matrix computation.

2.5 Pipeline Evaluation

After setting up the pipeline, we need to know if the clustering and smoothing steps perform well, i.e., if the final clusters correspond to actual binding sites. To address this, we have compared the clusters with the known binding sites from the CryptoBench test set, which contains 193 apo conformations and 230 cryptic binding sites (after filtering). The evaluation was performed in two steps: first, we benchmarked the clustering method without the smoothing model, and then we evaluated clustering including the smoothing model.

The evaluation follows the same procedure as the CryptoShow pipeline - first, the predictions are generated for test set structure, and then clustering is performed. After creating the clusters, we compute the following metrics for each cluster, i.e., a predicted binding site, and compare them with the known binding sites:

- **Distance between the predicted CBS center and the true CBS center (DCC)** - This metric is computed as the distance between the predicted CBS center and the center of the true CBS. Both centers are calculated as the average of the 3D coordinates of the residues in the predicted CBS and the true CBS, respectively. The distance is computed in Å. This standard metric is used to evaluate pocket prediction methods [50].
- **Percentage of true CBS residues covered by the predicted CBS** - This metric is computed as the percentage of residues in the true CBS that are also part of the predicted CBS.
- **Dice coefficient** - This metric is computed as the ratio of the number of residues in the intersection of the predicted CBS and the true CBS to the average number of residues in both sets A and B. It is defined in Equation 2.3.

$$\text{Dice}(A, B) = \frac{2|A \cap B|}{|A| + |B|} \tag{2.3}$$

After calculating the metrics against all predicted CBSs for each true CBS, the predicted CBS with the highest metric score is selected as the best match. After performing this procedure for all true CBSs and structures, we have plotted histograms to illustrate the performance of the clustering method. The results are visualized in Figure 2.3.

The results show that the clustering method without smoothing performs poorly, with the average DCC being around 10 Å, and the percentage of true CBS residues covered by the predicted CBSs being around 25%. This might not be a bad result, as one missing residue in the binding site can significantly increase the distance, but the percentage of residues covered is relatively low. This is caused by the fact that the clustering is performed only on the high-scoring residues. But,

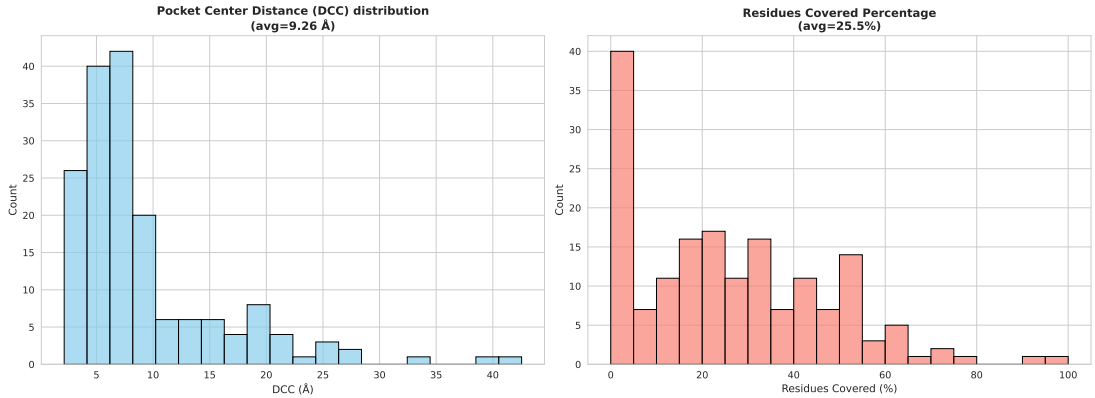

Figure 2.3 Clustering benchmark results before pocket smoothing. The left plot shows the distance between the predicted CBS center and the true CBS center (DCC), and the right plot shows the percentage of residues covered by the predicted CBSs.

some of the true CBSs have residues with low scores (shown later in the section in Figure 2.6), which are still part of the binding site.

Now, we will evaluate the clustering method with the smoothing model included. The smoothing model is trained and tested on a filtered version of the CryptoBench dataset, as described in Section 2.4. The smoothing model is evaluated on the test set, and the following metrics - F1 score, AUC, and accuracy (described in Equations 2.4, 2.5, and 2.6) - are computed for each threshold. Table 2.1 and Figure 2.4 illustrate the model's performance across various thresholds.

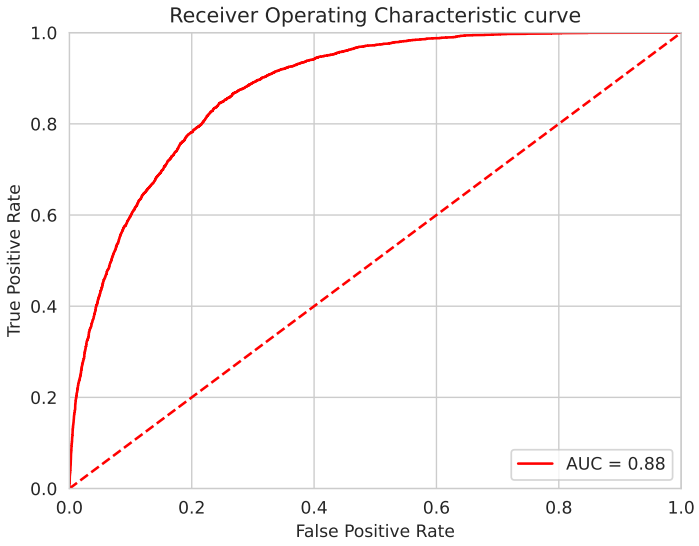

Figure 2.4 Receiver Operating Characteristic (ROC) curve for the smoothing model.

$$\text{F1 score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}, \quad \text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.4}$$

$$\text{AUC} = \int_0^1 \frac{\text{TP}(t)}{\text{TP}(t) + \text{FN}(t)} \, d\left(\frac{\text{FP}(t)}{\text{FP}(t) + \text{TN}(t)}\right) \tag{2.5}$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2.6}$$

where TP, TN, FP, and FN denote true positives, true negatives, false positives, and false negatives.

Threshold	F1 Score	Test Accuracy
0.1	0.5875	55.09%
0.2	0.6787	64.16%
0.3	0.7369	70.51%
0.4	0.7777	75.24%
0.5	0.8051	78.64%
0.6	0.8266	81.55%
0.7	0.8417	83.93%
0.8	0.8413	85.04%
0.9	0.8163	84.57%

Table 2.1 Model performance metrics across different thresholds for the smoothing model.

During the smoothing step, the trained model is loaded and a threshold is set (0.7 was chosen as the optimum based on the F1 score from the smoothing model evaluation, see Table 2.1). The relevant structure or sequence data is then prepared for inference. After running the model, the resulting smoothed pocket includes additional residues that are likely part of the cryptic binding site, even if their scores did not exceed the original high-scoring threshold.

The results demonstrate improved performance. As shown in Figures 2.5 and 2.6, the smoothing approach yields an average DCC of 5.12 Å (a substantial improvement from 9.26 Å without smoothing), an average residue coverage of 77.8% (compared to the previous 25.5%), and an average Dice coefficient of 0.51 (versus the earlier 0.32). These findings demonstrate that the smoothing methodology enhances prediction performance. While the predictions are not flawless, the majority of predicted pockets exhibit DCC within the 0 to 5 Å range, representing a satisfactory outcome.

The evaluation notebooks, including the whole pipeline and other source codes, are available in the `cryptoshow-benchmark/benchmark_experiments` directory.

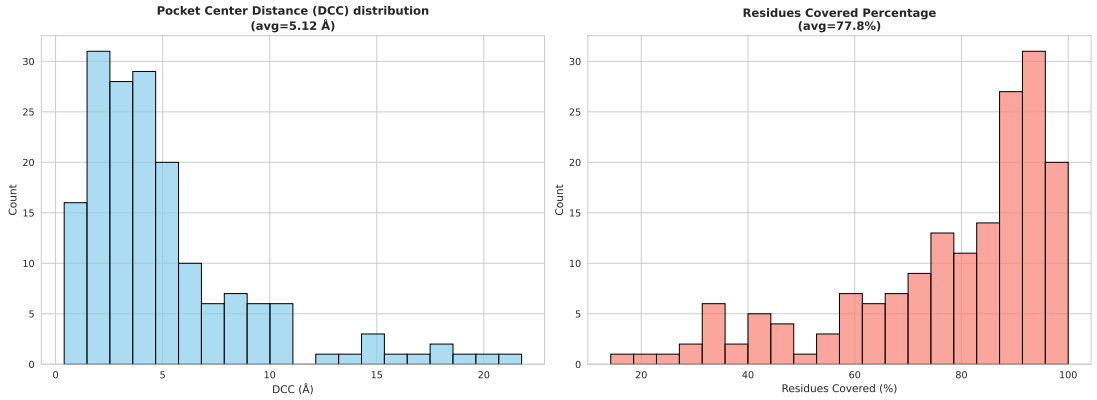

Figure 2.5 Clustering benchmark results after pocket smoothing. The left plot shows the distance between the top-scoring predicted CBS center and the true CBS center (DCC), and the right plot shows the percentage of residues covered by the predicted CBSs.

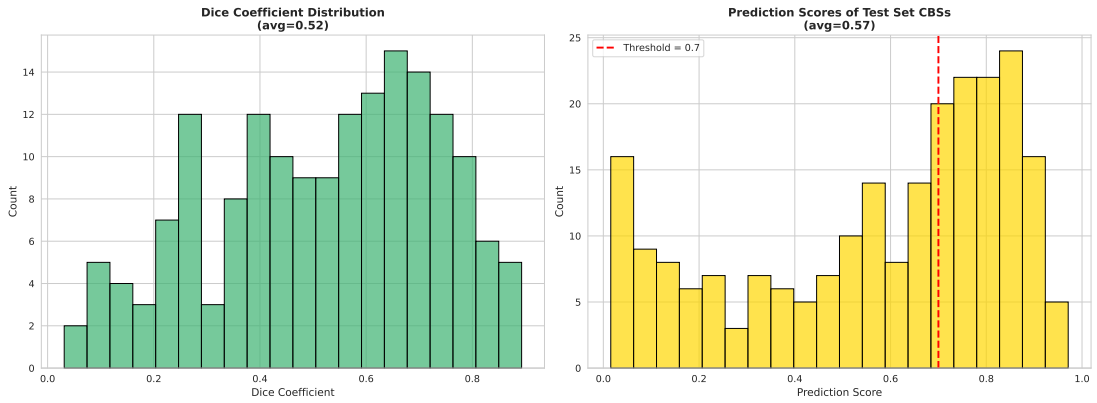

Figure 2.6 Clustering benchmark results after pocket smoothing. The left plot shows the Dice coefficient between the top-scoring predicted CBS and true CBS. The right plot shows the distribution of prediction scores for all residues in the true CBSs.

3 Software

The third chapter of this thesis describes the web application, CryptoShow, which is developed to provide a user-friendly interface for the methodology described in Chapter 2. This chapter covers the architecture, technologies used, backend and frontend development, testing, monitoring, and application deployment.

3.1 Architecture and Used Technologies

This section outlines the architecture of the CryptoShow application and the technologies employed in its development. The application is designed to be modular and scalable, allowing for easy integration of new features and improvements. For this purpose, we have chosen a service-oriented architecture (SOA) that separates the components, enabling independent development and deployment. The architecture is defined by several Docker containers[1], each responsible for a specific part of the application. The services are orchestrated using Docker Compose, described in the `docker-compose.yml` file, which specifies the services, networks, and volumes required for the application to run. The service architecture is as follows:

- **backend** - Acts as the central API service, facilitating communication between most components. It communicates with Redis[2], manages API requests from the frontend, and delegates asynchronous tasks to Celery workers[3]. The FastAPI server[4] serves as its entry point. Further details are provided in Section 3.2.
- **worker-cpu / worker-gpu** - These two services execute asynchronous, resource-intensive computational tasks (i.e., the prediction, smoothing, and trajectory animation) without blocking the main FastAPI event loop. `worker-cpu` runs on the CPU, while `worker-gpu` leverages GPU acceleration via CUDA when available. More details are provided in Section 3.2.
- **frontend** - Delivers the user interface. NginX[5] is the gateway to the defined services in Docker Compose and serves static content from a specified directory. Additional information is available in Section 3.3.
- **redis** - Functions as the message broker for Celery workers, enabling efficient task distribution.
- **monitoring-flag** - A minimal service that generates a flag file to activate the monitoring proxy endpoint. Additional information about this service and those that follow is available in Section 3.4.
- **remove-monitoring-flag** - A lightweight service that removes the monitoring flag file, disabling the monitoring endpoint.

[1] Available at `https://www.docker.com/`.
[2] Available at `https://redis.io/`.
[3] Available at `https://github.com/celery/celery`.
[4] Available at `https://github.com/fastapi/fastapi`.
[5] Available at `https://nginx.org/`.

- **flower**[6] - Provides real-time monitoring and statistics for Celery tasks and queues.
- **celery-exporter**[7] - Exposes Celery metrics to Prometheus, similarly to the Flower service.
- **prometheus**[8] - Collects and stores metrics exported from the Celery workers.
- **grafana**[9] - Offers a user-friendly dashboard for visualizing metrics collected by Prometheus.

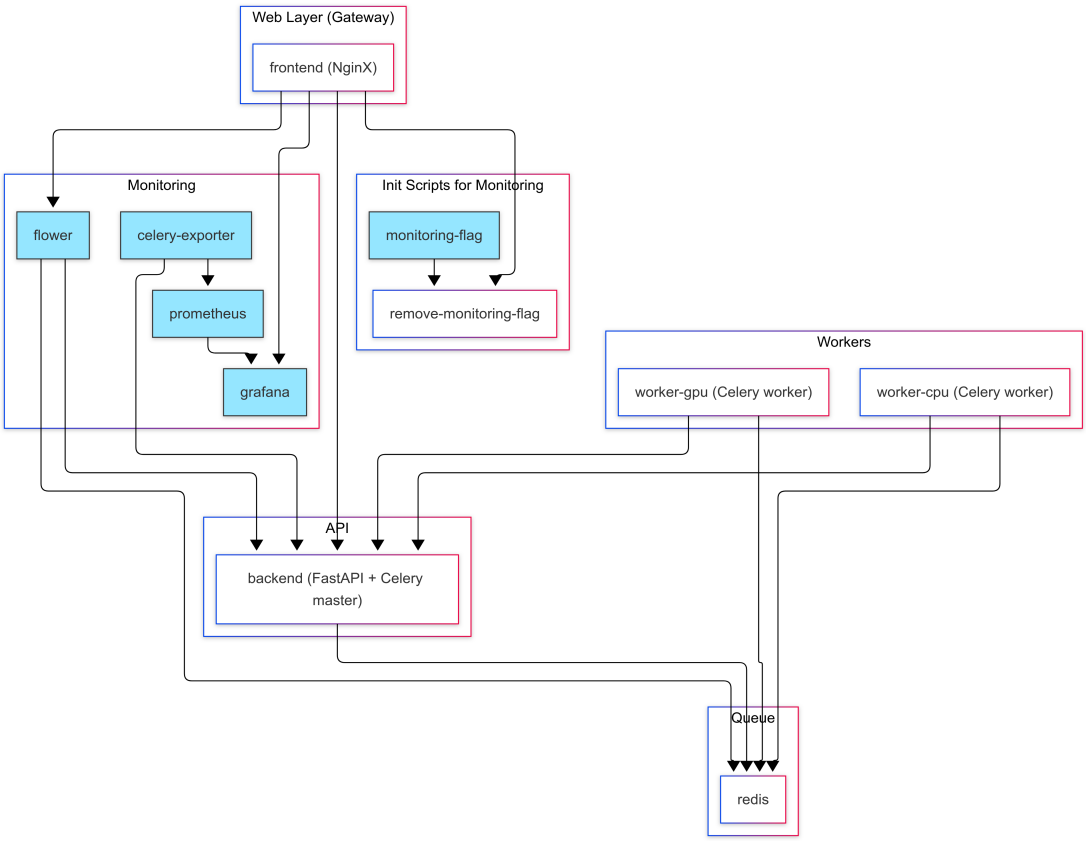

Figure 3.1 Service architecture of the CryptoShow application. Each small box represents a Docker containerized service. Light blue services are optional, as monitoring is not always required. Arrows indicate communication paths between services. Generated using the Mermaid Chart tool, available at `https://www.mermaidchart.com/`.

Figure 3.1 provides an overview of the service architecture. This type of architecture offers a scalable solution with the potential for easy deployment and development.

Let's also focus on the technology stack for some of the services:

- **backend, worker-cpu / worker-gpu** - Python, notable libraries and tools: Celery (asynchronous tasks in Python), FastAPI, Flower, PyTorch, BioPython, Biotite, Scikit-Learn, MDAnalysis, Gemmi, Redis, uv

[6]Available at `https://github.com/mher/flower`.
[7]Available at `https://github.com/danihodovic/celery-exporter`.
[8]Available at `https://prometheus.io/`.
[9]Available at `https://grafana.com/`.

- **frontend** - TypeScript, notable libraries, tools, and frameworks: React, NginX, Mol*, Bun, Vite

The remaining services utilize standard technology stacks commonly associated with their respective functionalities.

3.2 Backend

The backend of the CryptoShow application is responsible for the core functionalities, such as calculating the predictions, clustering and smoothing the results, generating trajectory animations for AHoJ results, and serving the API endpoints for the frontend. This section covers the **backend, worker-cpu, worker-gpu** and **redis** services. All source codes can be found in the `backend` directory.

All backend services utilize the same initial Dockerfile with one Python environment. This ensures consistency and speeds up the build process. We use the `pyproject.toml` file to install the Python environment, as the file contains all necessary dependencies for the backend services, and is installed using the `uv` tool[10]. After installing the Python environment, the entry point for the backend service is run, which first checks for the presence of the machine learning models and downloads them if they are unavailable. Subsequently, the FastAPI server is started running on port 5000.

3.2.1 FastAPI

The backend service is built using FastAPI, a modern web framework for building APIs with Python. It serves as the application's main entry point, handling API requests and delegating tasks to Celery workers.

FastAPI is chosen for its performance and ease of use, including the option to automatically generate OpenAPI documentation (available via the `/api/docs` endpoint). The server includes configuration for CORS (Cross-Origin Resource Sharing), logging, and error handling. The API contains several endpoints, including endpoints for health checks, prediction for a given PDB ID, prediction for a custom protein structure, Celery task status checks, and trajectory animation generation. To improve performance, WebSockets are used to check the status of the tasks, allowing the frontend to receive updates without polling the server. The API endpoints are described in Appendix A.3.

The backend service uses Celery workers to support asynchronous task execution without blocking the main event loop. These workers are responsible for executing resource-intensive tasks, such as running the CB model, smoothing the model, and generating trajectory animations. Depending on the availability of resources (covered in detail in Section 3.5), the workers can run on either CPU or GPU. The backend service communicates with Redis, which acts as a message broker for Celery, allowing efficient task distribution and management.

The logic for the API endpoints is implemented in the `backend/main.py` file.

[10]Available at `https://github.com/astral-sh/uv`.

3.2.2 Prediction and Clustering

To begin with the prediction, the necessary dependencies for the CryptoBench model must be installed. Once the Python environment is set up (this is done automatically in Docker), the Celery task is added to the queue. The task is defined in the `backend/tasks.py` file, which is responsible for executing the prediction logic.

First, the structure provided by the user (either in the form of an ID or a custom structure) is processed and parsed using specialized parsers for the PDB/mmCIF formats (see Section 1.3.4 and Section 1.3.5). Only the first model of the structure is considered, as the CryptoBench model is designed to work with single-model structures. After parsing, the structure is converted to a protein sequence and saved to FASTA files (see Section 1.3.6) for each chain, which is required for the prediction.

Afterwards, the fine-tuned ESM-2 model (`facebook/esm2_t33_650M_UR50D`[11]) is loaded, followed by the corresponding weight file from the Tiny-CryptoBench repository. The protein sequence is then tokenized using the ESM-2 tokenizer, segmented into chunks of 1022 tokens (1024 minus two special tokens), and processed by the CryptoBench model to obtain predictions. These predictions are subsequently concatenated to produce a single prediction vector representing the entire sequence. The model consists of three linear layers, two dropout layers, and a ReLU activation function. Then, the predictions are generated by applying the sigmoid activation function to the output of the final linear layer, resulting in a vector of probabilities for each residue in the sequence. The implementation for this procedure is provided in the `backend/prediction/compute_score.py` file. The code is intended for individual protein sequences, requiring parsing and splitting the sequence by chain to ensure accurate predictions. Concatenating all chains into a single sequence would provide incorrect context to the model and result in inaccurate predictions.

After receiving the predictions for individual chains, the 3D coordinates of all residues are extracted from the input structure. This step is required for the clustering and smoothing processes. The predictions are first clustered using the methodology described in Section 2.4. An example code snippet of the clustering process is available in Listing 3.1. After clustering, the clusters are further smoothed. All source codes for the clustering and smoothing processes can be found in the `backend/clustering` directory and follow the described methodology.

Listing 3.1 Clustering function from `code/clustering.py` (modified).

```
import numpy as np
from sklearn.cluster import DBSCAN


def compute_clusters(
    points: list[list[float]],
    prediction_scores: list[float],
):
    points_array = np.array(points)
    scores_array = np.array(prediction_scores).reshape(-1, 1)
    stacked = np.hstack((points_array, scores_array))
```

[11] More information available at `https://github.com/facebookresearch/esm`.

```
    HIGH_SCORE_THRESHOLD = 0.7

    high_score_mask = stacked[:, 3] > HIGH_SCORE_THRESHOLD
    high_score_points = stacked[high_score_mask][:, :3]

    EPS = 5.0
    MIN_SAMPLES = 3

    if len(high_score_points) < MIN_SAMPLES:
        return -1 * np.ones(len(points), dtype=int)

    dbscan = DBSCAN(eps=EPS, min_samples=MIN_SAMPLES)
    labels = dbscan.fit_predict(high_score_points)

    all_labels = -1 * np.ones(len(points), dtype=int)
    all_labels[high_score_mask] = labels
    labels = all_labels

    return labels
```

Following this refinement process, all collected data is combined into a single JSON object, which is stored, returned, and used by the frontend for result visualization.

3.2.3 Trajectory Animation

CryptoShow aims not only to predict and visualize cryptic binding sites, but also to demonstrate potential conformational changes through animated trajectories. To achieve this, we have developed a trajectory animation feature that illustrates structural transitions between related protein conformations.

Let us begin by introducing AHoJ [9] and AHoJ-DB [45] (briefly covered in Section 1.4.2). Apo–Holo Juxtaposition (AHoJ) is a web-based tool designed to identify apo-holo pairs of protein structures within the PDB database (Section 1.3.1); currently, querying is limited to this database only.

AHoJ identifies binding residues by spatially marking user-defined ligands using PyMOL (Section 1.4.3). The typical workflow involves the user specifying a binding spot, either by selecting a ligand, or defining a set of residues of interest in the query structure. The tool then searches for other structures of the same protein (i.e., structures sharing the same UniProt accession number, see Section 1.3.3) that contain the same binding spot. It does this by mapping the binding residues onto the UniProt sequence and examining each candidate chain to determine whether the mapped binding residues are present above a minimum threshold. Successful candidates are aligned to the query chain using TM-align [51], and the area around the superimposed query ligand is examined for ligands. The process classifies each candidate chain as apo or holo based on ligand presence or absence in the defined binding sites. The results are visualized in the browser and downloadable for PyMOL analysis. AHoJ-DB is a database of precomputed apo-holo pairs of protein structures.

For CryptoShow's trajectory animation functionality, we make use of the AHoJ public API to identify apo-holo structural pairs corresponding to the input

structure. The query process requires the PDB ID of the input structure[12], the target chain containing the predicted cryptic binding site, and the central residue of the identified CBS. Listing 3.2 demonstrates the query format. The API response includes all available paired structures, from which users can select any structure to visualize potential conformational transitions. Upon selection, the CryptoShow API initiates the trajectory computation.

Listing 3.2 Sample query format for the AHoJ tool, specifically the 2SRC protein structure (Crystal Structure of Human Tyrosine-Protein Kinase C-SRC, in Complex with AMP-PNP), chain A, aspartic acid residue at position 404

```
1    2src A ASP 404
```

First, the target structure for animation is downloaded from AHoJ. Then, combined with the original structure, both structure files are converted to PDB format to ensure each contains only a single model and to allow easier manipulation compared to the mmCIF format (see Section 1.3.4 and Section 1.3.5). Subsequently, the PDB file is processed using the MDAnalysis library [52], from which the sequence and corresponding residues are extracted. At this point, we have obtained the necessary structural information.

Next, we must determine which residues should be included in the animation. We employ a straightforward approach to achieve this by computing the longest common subsequence of both protein sequences. This approach ensures that only residues present in both structures are animated. The subsequence length must be greater than zero due to the inherent logic of AHoJ. Once the common subsequence is identified, the corresponding residues from both structures are extracted. The MDAnalysis library performs this operation at the atomic level, as PDB files store coordinates for individual atoms rather than residues. Additionally, all ligands are extracted from both structures to include them in the final animation.

Ultimately, the trajectory is generated using cubic spline interpolation [53] implemented in the SciPy library [54]. A trimmed PDB file containing only the relevant residues and ligands is created, along with a trajectory file in the XTC format (see Section 1.3.7) featuring interpolated coordinates across a predetermined number of frames (set arbitrarily to 50). The final visualization presents the original structure at 50% opacity and the target holo structure at full opacity, as illustrated in Figure 3.2.

Although this process generates a trajectory, it is essential to note that this represents only a simple interpolation of residue coordinates. The trajectory does not depict actual conformational changes but provides a smooth transition between the two structures. To capture genuine conformational dynamics, one must perform molecular dynamics simulations, which are computationally intensive and require significant time investment [55].

The source code for the trajectory animation functionality can be found in the `backend/trajectory_generator` directory. As with the prediction and clustering components, the implementation details in the actual pipeline will be covered in

[12]As mentioned above, AlphaFold structures and custom structures are not supported by AHoJ. Additionally, the input structure may be in holo form. The type of input is not validated in any way. Apo structures are generally more appropriate for this analysis, but we leave this decision to the user, as it does not affect the functionality of CryptoShow.

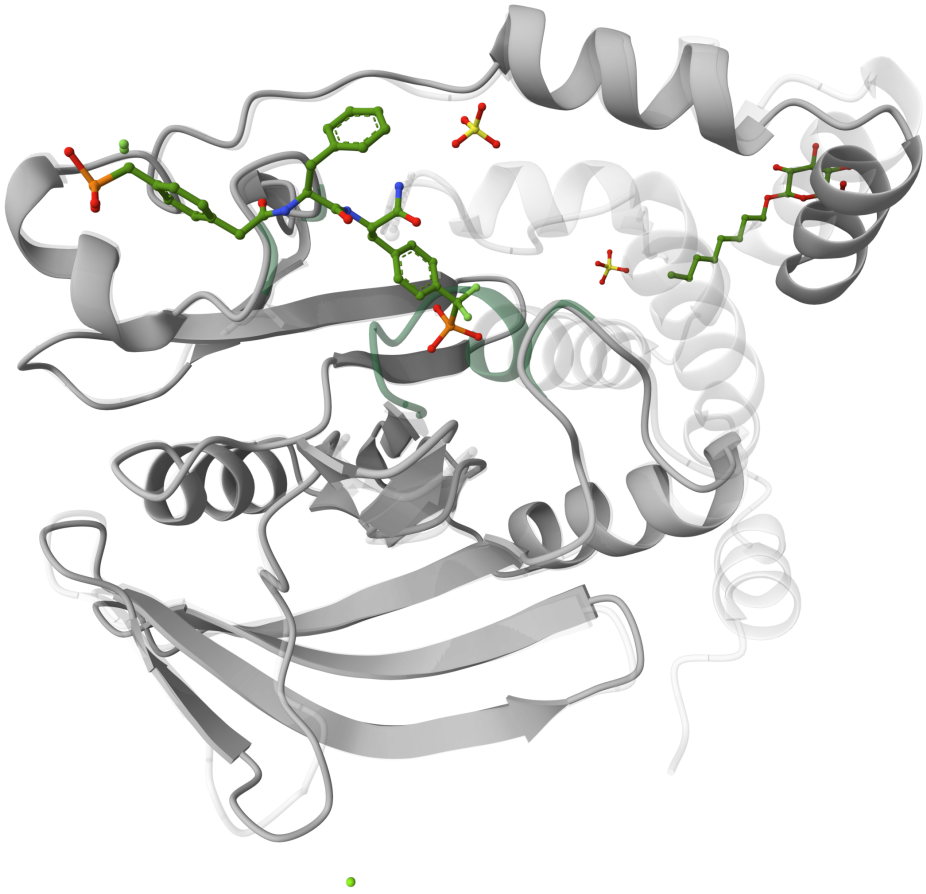

Figure 3.2 Example of trajectory animation demonstrating conformational changes between the input structure 1PTY (Crystal Structure of Protein Tyrosine Phosphatase 1B Complexed with Two Phosphotyrosine Molecules) and 2CNE (Structural Insights into the Design of Nonpeptidic Isothiazolidinone - Containing Inhibitors of Protein Tyrosine Phosphatase 1B). The input structure is displayed with 50% transparency, while the target holo structure is rendered at full opacity. The ligand is represented in ball-and-stick format. Visualized using Mol* in CryptoShow.

3.3 Frontend

The frontend of the CryptoShow application is designed to provide an intuitive and user-friendly interface for the pipeline described in Chapter 2. The main technologies used for the frontend development include TypeScript, React, and Mol* (see Section 1.4.4), which is a powerful molecular visualization library.

3.3.1 NginX

The main component is NginX [56], which serves as a reverse proxy to many services and as one endpoint for the frontend. There are multiple configurations for NginX, which are defined in the `frontend/nginx` files. We want to serve the frontend application in all cases, described by the default configuration. However, to improve user experience, we also provide configurations for TLS/SSL encryption, which is recommended for production environments. Moreover, we provide a configuration for the monitoring proxy endpoint, which is monitors the Celery tasks and queues (more details in Section 3.4). The NginX server listens on port 80 for HTTP requests and port 443 for HTTPS requests. The configuration files specify the root directory for static files, the API endpoints' location, and the backend service's proxy settings.

NginX does not support different behaviors for different environment variable values, so a simple Bash script generates the final configuration file on the fly. This script is executed during the Docker container startup, allowing for dynamic configuration based on the environment variables.

3.3.2 React and TypeScript

The application uses React[13], a popular TypeScript library for creating web applications. This decision was made as React remains state-of-the-art, and TypeScript provides strong typing and a better developer experience than JavaScript. The frontend libraries are installed using the Bun[14] package manager, which is known for its speed and efficiency (compared to Node.js). The source codes are transpiled and bundled using Vite[15], a modern build tool for TypeScript. The bundled files are then served by NginX, which is configured to serve static files from the `frontend/dist` directory.

The component structure follows a modular approach, with each component defined in its own file and CSS style. The main entry point for the React application is the `frontend/src/main.tsx` file, which renders the main `App` component. The application is organized into several `src` subdirectories:

- **components** - Contains reusable components, such as tables, forms, and other UI elements.

[13]Available at `https://react.dev/`.
[14]Available at `https://bun.sh/`.
[15]Available at `https://vitejs.dev/`.

- **pages** - Defines the page layouts, such as the home page, results page, about page, and error page.
- **hooks** - Includes custom React hooks for managing state and side effects.
- **contexts** - Provides React contexts for managing global state variables that need to be accessed by multiple components.
- **providers** - Contains React providers for managing global state.

At the same level as the `src` directory, there is a `public` directory, containing static files, such as images, icons, and HTML files. The `index.html` file serves as the main entry point for the application, where the React application is mounted.

The React architecture follows the standard practices, beginning with the `HomePage` component. This page serves as the landing page of the application, providing the user with an input table to enter the PDB ID or upload a custom protein structure. This logic is implemented in the aforementioned file and the `InputTable` component. After computing the prediction, the user is presented with the `Visualization` page. This page is split into several main components, such as the `ResultTable`, `MolstarControls`, and Mol* visualization component (handled in the `MolstarComponent` file), which is covered in Section 3.3.3.

Some of the component state is managed directly by the components themselves, while state that needs to be shared across multiple components is managed using React contexts. This approach allows better code organization and avoids prop drilling.

An example of a React component is shown in Listing 3.3.

Listing 3.3 Example of the PocketHeader React component from `frontend/src/components/PocketHeader.tsx`.

```tsx
import React from "react";
import { Pocket } from "../types";
import { getColorString } from "../utils";
import "./PocketHeader.css";

interface PocketHeaderProps {
    pocket: Pocket;
    isExpanded: boolean;
    toggleExpand: (e: React.MouseEvent) => void;
}

const PocketHeader = ({ pocket, isExpanded, toggleExpand }: PocketHeaderProps) => (
    <div className="pocket-header" onClick={toggleExpand}>
        <div className="pocket-header-left">
            <span className="toggle-icon">{isExpanded ? "v" : ">"}</span>
            <span className="pocket-id" style={{ color: getColorString(
                pocket.pocket_id) }}>
                Pocket {pocket.pocket_id}
            </span>
        </div>
        <span className="prediction-score">
            Score: {pocket.average_prediction.toFixed(3)}
        </span>
```

```
23     </div>
24 );

26 export default PocketHeader;
```

3.3.3 Mol*

In contrast to other frontend components, the Mol* library requires a more detailed explanation, as it is a complex library with many features, but limited documentation. As described in Section 1.4.4, Mol* is a powerful molecular visualization library that provides advanced features for visualizing and interacting with molecular structures. It is used in the CryptoShow application to visualize the predicted cryptic binding sites and the trajectory animations.

Mol* does not work as a standard React component, but rather is rendered in a separate div element within the React component tree. This requires a different approach to state management. The div element is defined in the `Visualization` component, and the Mol* viewer is initialized in the `MolstarComponent` component via the `initializePlugin` method.

Once the viewer has been initialized, the structure data is loaded into the viewer. Several protein representations are created at this stage, allowing users to switch between them via dedicated methods. The loading functionality is implemented in the `loadStructure` method. Optionally, a trajectory file can also be loaded alongside the structure to enable the trajectory animation described in Section 3.2.3.

The predicted binding sites can be visualized after the structure is loaded into the viewer. For each pocket found in the JSON object returned by the backend (see Section 3.2.2), new, colored representations are generated for the corresponding predicted cryptic binding sites. This approach allows users to easily distinguish the CBSs and select their preferred representation type alongside the main structure.

The file also provides additional methods for the visualization process, including adjusting structure transparency, retrieving residue selections, obtaining detailed residue information with coordinates, and overpainting the structure.

For the trajectory animation, the original structure is displayed with an opacity of 0.5, making it visually distinct from the apo-holo pair returned by AHoJ. The aligned structure and the generated trajectory file from the backend are both loaded into the Mol* viewer. The trajectory file contains the intermediate states of the aligned structure. Mol* then animates these states, visually illustrating the protein's conformational transition.

Screenshots demonstrating the Mol* viewer in use are presented in Chapter 4.

An example of the Mol* code snippet is shown in Listing 3.4. The code changes the transparency of a given structure.

Listing 3.4 Code snippet from `frontend/src/components/MolstarComponent.ts`.

```
1 /**
2  * Sets the transparency of a structure in the Mol* plugin.
3  * @param plugin Mol* plugin instance
4  * @param alpha Transparency value (0-1)
```

```
 * @param representations Array of representations
 * @param structure Structure object (Mol*)
 */
export const setStructureTransparency = async (plugin: PluginUIContext, alpha: number, representations: RepresentationWithRef<PocketRepresentationType | PolymerRepresentationType>[], structure: StateObjectSelector) => {
    type TransparencyParams = {
        bundle: Bundle;
        value: number;
    };

    const params: TransparencyParams[] = [];

    const query = MS.struct.generator.all;
    const sel = Script.getStructureSelection(query, structure.cell!.obj!.data);
    const bundle = Bundle.fromSelection(sel);

    params.push({
        bundle: bundle,
        value: 1 - alpha
    });

    for (const element of representations) {
        const builder = plugin.state.data.build();
        if (element.transparentObjectRef) {
            builder.to(element.object.ref).delete(element.transparentObjectRef);
        }
        await builder.commit();

        const r = await plugin.state.data.build().to(element.object.ref).apply(StateTransforms.Representation
            .TransparencyStructureRepresentation3DFromBundle, { layers: params }).commit();
        element.transparentObjectRef = r.ref;
    }
};
```

3.4 Tests and Monitoring

Multiple monitoring and testing tools have been set up to improve the reliability and stability of the CryptoShow application. These tools help to ensure the application is running, and that the Celery tasks are executed correctly. The monitoring tools also provide insights into the application's performance and the Celery workers.

First, basic health checks are implemented in the Docker Compose file. Docker periodically checks the health of the services and restarts them if they are unhealthy. The health checks are defined in the **`docker-compose.yml`** file, where each service has a **`healthcheck`** section that specifies the command to run to check the health of the service. For example, the backend service checks if the FastAPI server is running by sending a request to the **`/health`** endpoint. These health checks run in every case, regardless of the environment variable values. If a Docker healthcheck

fails, Docker marks the container as unhealthy. This information can be used to restart the container.

To enhance the monitoring capabilities, Flower is used to monitor the Celery tasks and queues. Flower provides a web-based interface that allows users to view the status of the Celery workers, the tasks in the queue, and the task execution times. It is configured to run on port 5555 and can be accessed via the `/flower` endpoint thanks to NginX. The Flower service communicates with the Redis service to retrieve task information.

Moreover, Grafana, Prometheus, and Celery Exporter collaborate to provide a comprehensive monitoring solution with a user-friendly dashboard. Prometheus serves as a data source for Grafana, receiving metrics from the Celery workers via the Celery Exporter. The Celery Exporter is a lightweight service that exposes Celery metrics to Prometheus, allowing it to scrape and store these metrics. Grafana then visualizes the metrics in a dashboard. The Grafana service is configured to run on port 3001 and can be accessed via the `/grafana` endpoint. Prometheus is configured to run on port 9090 and can be accessed via the `/prometheus` endpoint. The configuration files for Prometheus and Grafana are located in the `monitoring` directory. Figure 3.3 shows an example of the Grafana dashboard.

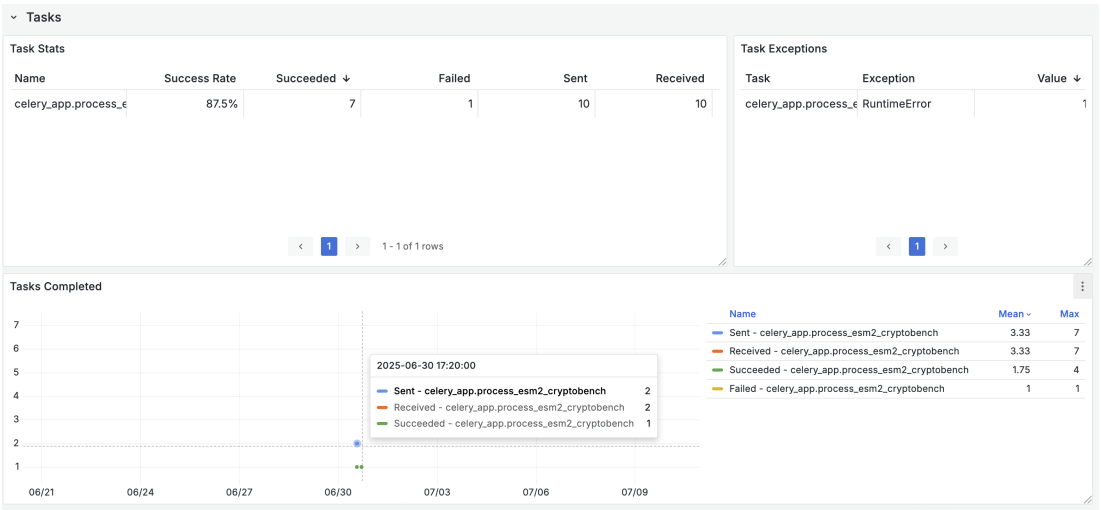

Figure 3.3 Screenshot of the Grafana dashboard displaying Prometheus statistics for the CryptoShow application.

Although the monitoring services are helpful, they are not always necessary for the application to function. This is the reason why a monitoring flag mechanism has been introduced. The `monitoring-flag` service creates a flag file that enables the monitoring services, while the `remove-monitoring-flag` service removes the flag file, disabling the monitoring services. The container to remove the flag file is run every time the application is started, so the monitoring services are disabled by default. If the developer wants to enable the monitoring services, they run the Docker Compose command with the `-profile monitoring` option, which turns on the monitoring profile. This activates the `monitoring-flag` service, creating the flag file and enabling the monitoring services. The monitoring services are then accessible via the respective endpoints thanks to the flexible configuration of NginX. Otherwise, the endpoints are unavailable, and the containers are not

started to save RAM and CPU resources.

As most of the functionality of the frontend is visual, the frontend is tested manually. Still, some bugs can be caught using Sentry[16], a tool for error tracking and monitoring. Sentry is configured to capture errors in the frontend, providing detailed information about the errors, including stack traces. This allows quicker identification of application bugs. The Sentry service is integrated using the environment variables in the `frontend` directory (see the example environment file `frontend/.env.template`).

3.5 Deployment

CryptoShow is distributed as a fully dockerized application, enabling simple deployment and scalability. It can be deployed on any server with Docker Compose support, requiring at least 32 GB of RAM and 8 CPU cores.

Before starting, the `.env` file must be created in the root folder to specify required environment variables. The variables are listed in the `.env.template` file, and include the following:

- **ENABLE_SSL**: `<true/false>` - Enable or disable SSL. Recommended for production.
- **SSL_PATH**: `<path to SSL certificate and key folder>` - Path to SSL certificate and key (not required if SSL is disabled).
- **DOMAIN**: `<domain name>` - Domain name for the deployment.
- **UID**: `<user id (default: 2727)>` - Linux user ID for running containers. Make sure that the user has sufficient permissions to the filesystem.
- **GID**: `<group id (default: 2727)>` - Linux group ID for running containers.
- **HTPASSWD_PATH**: `<path to htpasswd file>` - Path to HTTP basic auth password file for monitoring access.
- **HTTP_PROXY**: `<http proxy URL>` - HTTP proxy URL for development behind a proxy (optional).
- **HTTPS_PROXY**: `<https proxy URL>` - HTTPS proxy URL (optional).

Once the `.env` file has been created, the next step is to build the Docker images for all services specified in the `docker-compose.yml` file. This can be accomplished using the standard `docker compose build` command. Alternatively, Docker Bake[17] may be used, a tool that enables parallel building of multiple Docker images. To use Docker Bake, run `docker buildx bake`.

CryptoShow supports accelerated computations using the CUDA architecture. To specify whether the application should utilize GPU or CPU resources, the developer must select the appropriate mode by passing the `-profile <cpu/gpu>` flag when building and running the application with Docker Compose.

[16] Available at `https://sentry.io/`

[17] Available at `https://docs.docker.com/build/bake/`.

As described in Section 3.4, monitoring features can be enabled by including the monitoring profile when running Docker Compose: `-profile monitoring`.

Before deploying the application, make sure to enable incoming and outgoing traffic to ports 80 and 443 (if using SSL). Also, note that multiple ports are exposed:

- **80/tcp** - HTTP (frontend, NginX)
- **443/tcp** - HTTPS (frontend, NginX)
- **3001/tcp** - Grafana (monitoring dashboard, optional)
- **5000/tcp** - Backend API (FastAPI)
- **5555/tcp** - Flower (Celery monitoring, optional)
- **6379/tcp** - Redis (message broker)
- **9090/tcp** - Prometheus (monitoring, optional)
- **9808/tcp** - Celery Exporter (monitoring, optional)

Blocking direct access to the other ports is not strictly required, but leaving the ports open can lead to unauthorized access to the monitoring.

Once the infrastructure is built, the application can be started using the `docker compose up` command, along with any relevant profile flags as described above. The application will then be accessible on port 80 or 443, depending on the value of the ENABLE_SSL environment variable.

Detailed step-by-step instructions, including SSL certificate generation and file permission configuration, are provided in the `README` file.

3.5.1 Local Frontend Development

For faster development, the frontend application can be run locally. First, install the required dependencies using the Bun package manager with `bun install` . Then, start the development server using `bun run dev`. The frontend will be accessible at port 3000.

3.5.2 Local Backend Development

While running the backend services locally is technically possible, it requires installing multiple tools and frameworks, complicating the process. Therefore, it is strongly recommended that Docker is used for backend development. Developers may choose to modify the Dockerfile and Docker Compose configurations to enable hot reloading if desired, but this is not provided by default. For local service development, consult the relevant Dockerfiles and the Docker Compose file.

4 User Documentation

This chapter provides user documentation for the CryptoShow application, including a brief introduction on how to use the application.

4.1 How to Use

To use the CryptoShow application, either deploy it locally (see Section 3.5) or access the live version at `https://cryptoshow.cz`.

Upon opening the main page, an input table is displayed. Provide either a structure identifier (PDB or AlphaFold ID) or upload a PDB/mmCIF file. Once a valid structure is provided, the application will begin processing and displays the computation status. Processing time varies based on the structure size and server load.

Once the computation is finished, the user is presented with a link to the calculation results. After clicking the link, the user is redirected to the visualization page. The following elements are shown:

- An interactive 3D visualization of the structure, allowing users to rotate and zoom. Residues that are part of detected cryptic binding sites are highlighted using distinct colors.
- A toolbox providing options to adjust visualization styles.
- A table listing the identified cryptic binding sites with expandable rows. Each entry includes details about the involved residues, their scores, and a PyMOL selection string for visualizing the site in PyMOL. For each pocket (in PDB structures), users can initiate an AHoJ query to obtain apo-holo pairs and visualize possible conformational changes. Results can also be downloaded as a ZIP archive.

The 3D visualization in Mol* with the toolbox is shown in Figure 4.1. The results table with expandable rows is shown in Figure 4.2.

4.2 Use Case: Predicting Cryptic Binding Sites in DHFR

To illustrate the intended use of the CryptoShow application, we examine a real-world example from computational drug discovery. This scenario demonstrates how structural analysis and simulation tools can uncover cryptic binding sites in proteins—sites that are not evident in ligand-free structures but emerge in ligand-bound forms or during dynamic conformational changes. This example is inspired by the work of Meller et al. [43].

Figure 4.1 Mol* 3D visualization of the protein structure of Calcium-free Calmodulin (1CFD), along with the Crystal Structure of Myosin-1c tail in complex with Calmodulin (4R8G), with the visualization toolbox underneath.

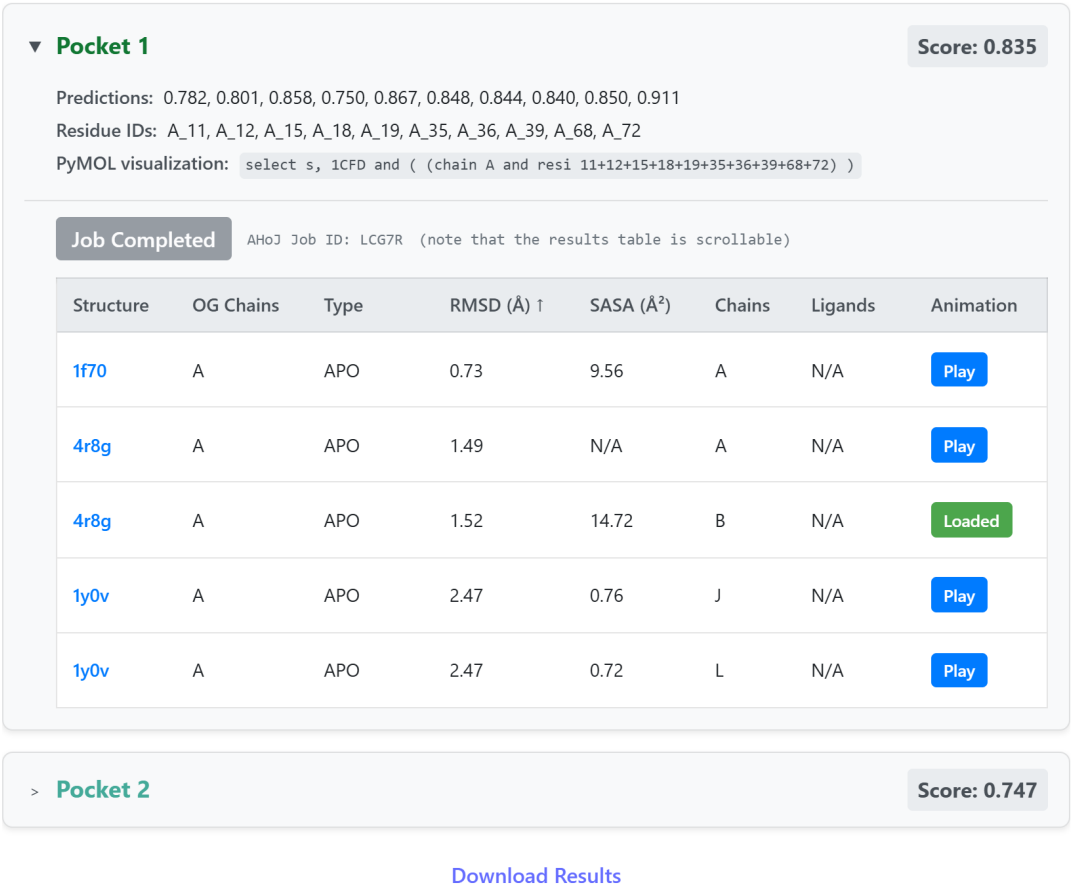

Figure 4.2 Table of cryptic binding sites for the Calcium-free Calmodulin (1CFD) protein structure with expandable rows, showing details about the involved residues, their scores, and a PyMOL selection string for visualizing the site in PyMOL.

4.2.1 Case Study: Dihydrofolate Reductase (DHFR)

As a case study, we use the protein dihydrofolate reductase (DHFR) from *Escherichia coli*, an important target in antimicrobial drug development [57]. We begin with the apo structure, PDB ID **2W9T**, which shows DHFR in its unbound state with no visible deep binding pockets beyond the known active site.

Using the CryptoShow application, we predict potential cryptic pockets based solely on the apo structure. One binding site is identified near the so-called Met20 loop[1], a flexible region of the protein known to undergo conformational changes. Figure 4.3 illustrates this prediction, with the Met20 loop highlighted in light yellow.

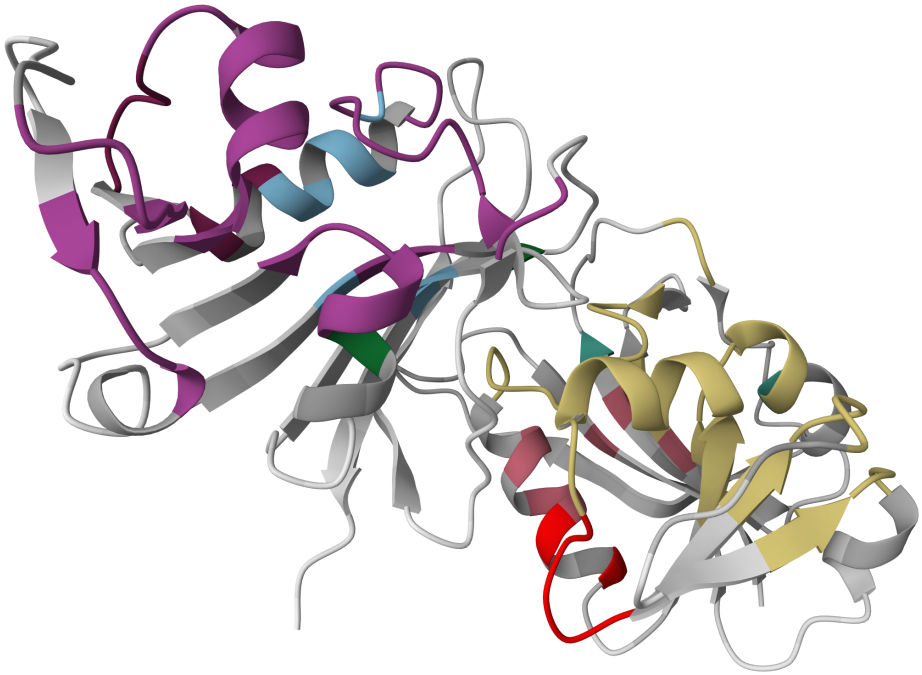

Figure 4.3 Predicted cryptic binding sites in the apo structure of DHFR (PDB ID 2W9T) using the CryptoShow application. The pockets are highlighted in different colors, with the Met20 binding site shown in light yellow. Visualized using Mol*.

4.2.2 Validation Using Holo Structure

To evaluate the accuracy of the prediction, we consult the AHoJ database by clicking the button in the pocket details window (see Figure 4.4) to identify known holo structures of DHFR. Indeed, a holo form is available: PDB ID **2W9S**, in which the protein is bound to the antimicrobial drug Trimethoprim. In this structure, the Met20 loop shifts position, exposing a pocket that matches the one predicted by the CryptoShow application from the apo form. The conformational

[1]The Met20 loop is a flexible segment of DHFR, named after the methionine amino acid at position 20 (residue ID 42 in this structure), and is known for its role in modulating access to the active site through conformational changes.

change of the Met20 loop can be visualized in the application. Figure 4.5 presents this structural comparison.

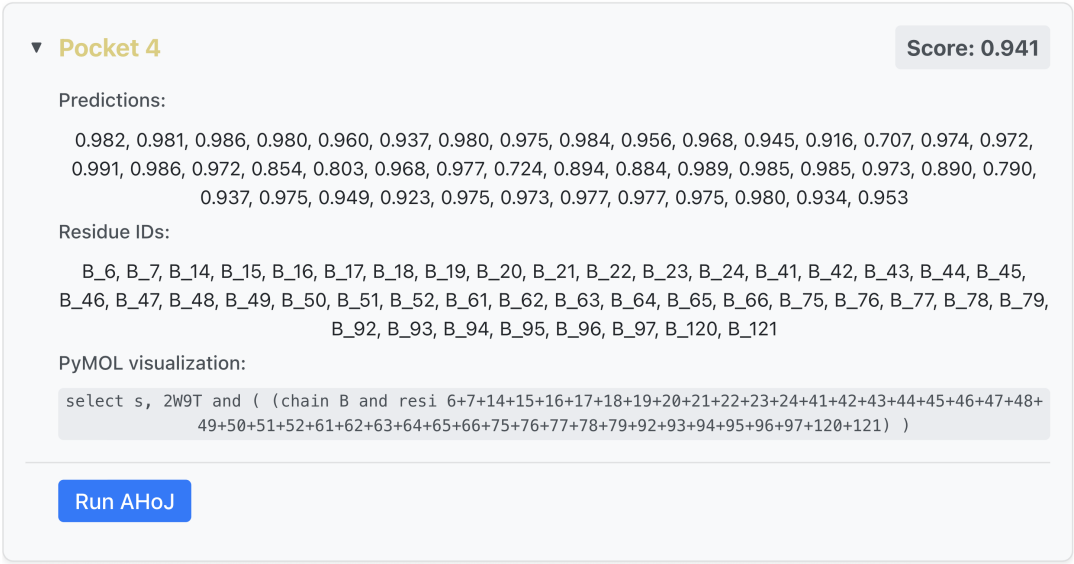

Figure 4.4 Details window for the Met20 loop CBS in the 2W9T protein structure. The AHoJ query can be initiated by clicking the button.

This comparison between 2W9T (apo) and 2W9S (holo) provides a validation of the cryptic pocket prediction. The application was able to identify a site that is not visible in the apo structure but becomes occupied by a ligand in the holo structure.

4.2.3 Further Exploration via Molecular Dynamics

To further investigate the dynamic opening of the predicted pocket, one can perform molecular dynamics (MD) simulations using tools such as GROMACS [36], starting from the 2W9T structure. While CryptoShow provides only a brief, illustrative visualization of the conformational change, MD simulations offer a detailed and realistic exploration of the protein's dynamics. During the simulation, the Met20 loop displays conformational flexibility, and the predicted pocket transiently appears. Pocket detection tools such as Fpocket [5] and TRAPP [58] can be used to monitor the pocket's volume and accessibility throughout the simulation.

The results support the idea that the site is not an artifact of ligand binding, but a real conformational feature that may be exploited for inhibitor design and further studies.

4.2.4 Use Case Conclusion

This use case demonstrates how the CryptoShow application can be used to predict cryptic binding sites from an apo structure, and how known holo structures, such as 2W9S, can be used to validate the accuracy of those predictions. It highlights the practical application of structural bioinformatics and simulation

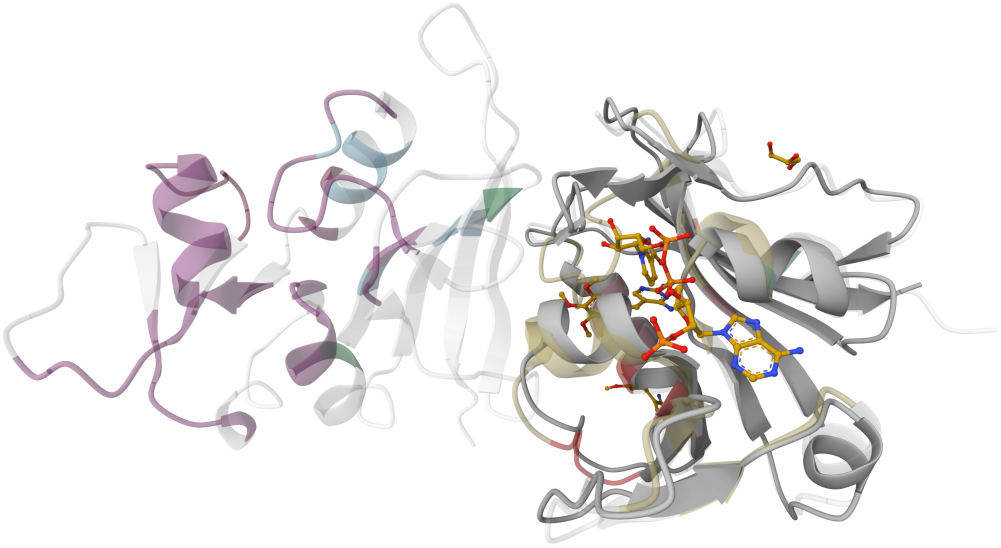

Figure 4.5 Comparison of DHFR structures: apo form (2W9T) and holo form (2W9S) complexed with trimethoprim, illustrating the conformational rearrangement of the Met20 loop region.

tools in supporting early-stage drug discovery efforts, demonstrating the real-world utility of the developed application in computational structural biology research.

Conclusion

Cryptic binding sites play a crucial role in understanding protein function and facilitating drug discovery. In this thesis, a methodology was introduced for clustering residue-level predictions into coherent cryptic binding sites, incorporating a novel smoothing step using a dedicated machine learning model. The approach was evaluated on the CryptoBench dataset, which comprises protein structures with annotated cryptic binding sites. Results indicate that integrating the clustering technique with the smoothing model enhances performance compared to the original method.

As a next step, the CryptoShow web application was created to offer an intuitive platform for detecting and visualizing cryptic binding sites in protein structures, leveraging the CryptoBench model and the clustering methodology introduced earlier. The application enables users to conveniently examine prediction outcomes and investigate identified cryptic sites. Additionally, integration with the AHoJ tool was implemented, allowing users to search for similar protein structures and visualize possible conformational changes. During the development of the CryptoShow application, we focused on a modular, service-oriented architecture, which allows for easy integration of new features and models in the future.

Potential future enhancements include enabling users to select other structure models of the uploaded protein, as opposed to being limited to the first one, integrating additional predictive models next to the CryptoBench model, and providing the capability to predict cryptic binding sites directly from protein sequences. Furthermore, the clustering approach could be refined, as the current methodology tends to generate a larger number of smaller pockets rather than consolidating them into larger ones, though this task is inherently challenging due to the nature of cryptic binding sites.

We believe that the CryptoShow application will be a valuable tool for researchers in structural bioinformatics and computational drug discovery.

Bibliography

1. Trainor, George L. The importance of plasma protein binding in drug discovery. *Expert opinion on drug discovery*. 2007, vol. 2, no. 1, pp. 51–64.
2. Ballante, Flavio. *Protein-ligand interactions and drug design*. Springer, 2021.
3. Mannhold, Raimund; Kubinyi, Hugo; Folkers, Gerd. *Protein-ligand interactions: from molecular recognition to drug design*. John Wiley & Sons, 2006.
4. Krivák, Radoslav; Hoksza, David. P2Rank: machine learning based tool for rapid and accurate prediction of ligand binding sites from protein structure. *Journal of cheminformatics*. 2018, vol. 10, pp. 1–12.
5. Le Guilloux, Vincent; Schmidtke, Peter; Tuffery, Pierre. Fpocket: an open source platform for ligand pocket detection. *BMC bioinformatics*. 2009, vol. 10, pp. 1–11.
6. Aggarwal, Rishal; Gupta, Akash; Chelur, Vineeth; Jawahar, CV; Priyakumar, U Deva. DeepPocket: ligand binding site detection and segmentation using 3D convolutional neural networks. *Journal of Chemical Information and Modeling*. 2021, vol. 62, no. 21, pp. 5069–5079.
7. Smith, Zachary; Strobel, Michael; Vani, Bodhi P; Tiwary, Pratyush. Graph attention site prediction (GrASP): identifying druggable binding sites using graph neural networks with attention. *Journal of chemical information and modeling*. 2024, vol. 64, no. 7, pp. 2637–2644.
8. Škrhák, Vit; Novotný, Marian; Feidakis, Christos P; Krivák, Radoslav; Hoksza, David. CryptoBench: cryptic protein–ligand binding sites dataset and benchmark. *Bioinformatics*. 2025, vol. 41, no. 1, btae745.
9. Feidakis, Christos P; Krivak, Radoslav; Hoksza, David; Novotny, Marian. AHoJ: rapid, tailored search and retrieval of apo and holo protein structures for user-defined ligands. *Bioinformatics*. 2022, vol. 38, no. 24, pp. 5452–5453.
10. Sehnal, David; Bittrich, Sebastian; Deshpande, Mandar; Svobodová, Radka; Berka, Karel; Bazgier, Václav; Velankar, Sameer; Burley, Stephen K; Koča, Jaroslav; Rose, Alexander S. Mol* Viewer: modern web app for 3D visualization and analysis of large biomolecular structures. *Nucleic Acids Research*. 2021, vol. 49, no. W1, W431–W437.
11. Cooper, Geoffrey M; Adams, Kenneth. *The cell: a molecular approach*. Oxford University Press, 2022.
12. Nelson, David L; Lehninger, Albert L; Cox, Michael M. *Lehninger principles of biochemistry*. Macmillan, 2008.
13. Voet, Donald; Voet, Judith G. *Biochemistry*. John Wiley & Sons, 2010.
14. Dumas, Anaëlle; Lercher, Lukas; Spicer, Christopher D; Davis, Benjamin G. Designing logical codon reassignment–Expanding the chemistry in biology. *Chemical science*. 2015, vol. 6, no. 1, pp. 50–69.

15. BioNinja. *Secondary Structure – Alpha Helices versus Beta Pleated Sheets*. Available also from: `https://old-ib.bioninja.com.au/`. Accessed: 2025-06-30.

16. Berman, Helen M; Westbrook, John; Feng, Zukang; Gilliland, Gary; Bhat, Talapady N; Weissig, Helge; Shindyalov, Ilya N; Bourne, Philip E. The protein data bank. *Nucleic acids research*. 2000, vol. 28, no. 1, pp. 235–242.

17. Jumper, John; Evans, Richard; Pritzel, Alexander; Green, Tim; Figurnov, Michael; Ronneberger, Olaf; Tunyasuvunakool, Kathryn; Bates, Russ; Žídek, Augustin; Potapenko, Anna, et al. Highly accurate protein structure prediction with AlphaFold. *nature*. 2021, vol. 596, no. 7873, pp. 583–589.

18. Abramson, Josh; Adler, Jonas; Dunger, Jack; Evans, Richard; Green, Tim; Pritzel, Alexander; Ronneberger, Olaf; Willmore, Lindsay; Ballard, Andrew J; Bambrick, Joshua, et al. Accurate structure prediction of biomolecular interactions with AlphaFold 3. *Nature*. 2024, vol. 630, no. 8016, pp. 493–500.

19. Abriata, Luciano A. The Nobel Prize in Chemistry: past, present, and future of AI in biology. *Communications Biology*. 2024, vol. 7, no. 1, p. 1409.

20. Passaro, Saro; Corso, Gabriele; Wohlwend, Jeremy; Reveiz, Mateo; Thaler, Stephan; Somnath, Vignesh Ram; Getz, Noah; Portnoi, Tally; Roy, Julien; Stark, Hannes; Kwabi-Addo, David; Beaini, Dominique; Jaakkola, Tommi; Barzilay, Regina. Boltz-2: Towards Accurate and Efficient Binding Affinity Prediction. 2025.

21. team, Chai Discovery; Boitreaud, Jacques; Dent, Jack; McPartlon, Matthew; Meier, Joshua; Reis, Vinicius; Rogozhonikov, Alex; Wu, Kevin. Chai-1: Decoding the molecular interactions of life. *BioRxiv*. 2024, pp. 2024–10.

22. Konc, Janez. Binding site comparisons for target-centered drug discovery. *Expert opinion on drug discovery*. 2019, vol. 14, no. 5, pp. 445–454.

23. Huang, Po-Ssu; Boyken, Scott E; Baker, David. The coming of age of de novo protein design. *Nature*. 2016, vol. 537, no. 7620, pp. 320–327.

24. Cimermancic, Peter; Weinkam, Patrick; Rettenmaier, T Justin; Bichmann, Leon; Keedy, Daniel A; Woldeyes, Rahel A; Schneidman-Duhovny, Dina; Demerdash, Omar N; Mitchell, Julie C; Wells, James A, et al. CryptoSite: expanding the druggable proteome by characterization and prediction of cryptic binding sites. *Journal of molecular biology*. 2016, vol. 428, no. 4, pp. 709–719.

25. Berman, Helen; Henrick, Kim; Nakamura, Haruki. Announcing the worldwide protein data bank. *Nature structural & molecular biology*. 2003, vol. 10, no. 12, pp. 980–980.

26. Velankar, Sameer; Best, Christoph; Beuth, Barbara; Boutselakis, CH; Cobley, Norman; Sousa Da Silva, AW; Dimitropoulos, Dimitris; Golovin, Adel; Hirshberg, Miriam; John, Melford, et al. PDBe: protein data bank in Europe. *Nucleic acids research*. 2010, vol. 38, no. suppl_1, pp. D308–D317.

27. Kinjo, Akira R; Suzuki, Hirofumi; Yamashita, Reiko; Ikegawa, Yasuyo; Kudou, Takahiro; Igarashi, Reiko; Kengaku, Yumiko; Cho, Hasumi; Standley, Daron M; Nakagawa, Atsushi, et al. Protein Data Bank Japan (PDBj): maintaining a structural data archive and resource description framework format. *Nucleic acids research*. 2012, vol. 40, no. D1, pp. D453–D460.

28. Callaway, Ewen. AlphaFold is running out of data—so drug firms are building their own version. *Nature*. 2025, vol. 640, no. 8058, pp. 297–298.

29. Varadi, Mihaly; Bertoni, Damian; Magana, Paulyna; Paramval, Urmila; Pidruchna, Ivanna; Radhakrishnan, Malarvizhi; Tsenkov, Maxim; Nair, Sreenath; Mirdita, Milot; Yeo, Jingi, et al. AlphaFold Protein Structure Database in 2024: providing structure coverage for over 214 million protein sequences. *Nucleic acids research*. 2024, vol. 52, no. D1, pp. D368–D375.

30. UniProt: the Universal protein knowledgebase in 2025. *Nucleic Acids Research*. 2025, vol. 53, no. D1, pp. D609–D617.

31. Boutet, Emmanuel; Lieberherr, Damien; Tognolli, Michael; Schneider, Michel; Bansal, Parit; Bridge, Alan J; Poux, Sylvain; Bougueleret, Lydie; Xenarios, Ioannis. UniProtKB/Swiss-Prot, the manually annotated section of the UniProt KnowledgeBase: how to use the entry view. *Plant bioinformatics: methods and protocols*. 2016, pp. 23–54.

32. Westbrook, John D; Fitzgerald, PM. The PDB format, mmCIF, and other data formats. *Methods Biochem Anal*. 2003, vol. 44, pp. 161–179.

33. Watson, Joseph L; Juergens, David; Bennett, Nathaniel R; Trippe, Brian L; Yim, Jason; Eisenach, Helen E; Ahern, Woody; Borst, Andrew J; Ragotte, Robert J; Milles, Lukas F, et al. De novo design of protein structure and function with RFdiffusion. *Nature*. 2023, vol. 620, no. 7976, pp. 1089–1100.

34. Bourne, Philip E; Berman, Helen M; McMahon, Brian; Watenpaugh, Keith D; Westbrook, John D; Fitzgerald, Paula MD. [30] Macromolecular crystallographic information file. In: *Methods in enzymology*. Elsevier, 1997, vol. 277, pp. 571–590.

35. Lipman, David J; Pearson, William R. Rapid and sensitive protein similarity searches. *Science*. 1985, vol. 227, no. 4693, pp. 1435–1441.

36. Van Der Spoel, David; Lindahl, Erik; Hess, Berk; Groenhof, Gerrit; Mark, Alan E; Berendsen, Herman JC. GROMACS: fast, flexible, and free. *Journal of computational chemistry*. 2005, vol. 26, no. 16, pp. 1701–1718.

37. Polák, Lukáš; Škoda, Petr; Riedlová, Kamila; Krivák, Radoslav; Novotný, Marian; Hoksza, David. PrankWeb 4: a modular web server for protein–ligand binding site prediction and downstream analysis. *Nucleic Acids Research*. 2025, gkaf421.

38. Jakubec, David; Skoda, Petr; Krivak, Radoslav; Novotny, Marian; Hoksza, David. PrankWeb 3: accelerated ligand-binding site predictions for experimental and modelled protein structures. *Nucleic Acids Research*. 2022, vol. 50, no. W1, W593–W597.

39. Jendele, Lukas; Krivak, Radoslav; Skoda, Petr; Novotny, Marian; Hoksza, David. PrankWeb: a web server for ligand binding site prediction and visualization. *Nucleic acids research*. 2019, vol. 47, no. W1, W345–W349.

40. Trott, Oleg; Olson, Arthur J. AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of computational chemistry*. 2010, vol. 31, no. 2, pp. 455–461.

41. DeLano, Warren L et al. Pymol: An open-source molecular graphics tool. *CCP4 Newsl. Protein Crystallogr*. 2002, vol. 40, no. 1, pp. 82–92.

42. Sehnal, David; Rose, Alexander S; Koca, Jaroslav; Burley, Stephen K; Velankar, Sameer. Mol*: Towards a Common Library and Tools for Web Molecular Graphics. In: *MolVa@ EuroVis*. 2018, pp. 29–33.

43. Meller, Artur; Ward, Michael D; Borowsky, Jonathan H; Lotthammer, Jeffrey M; Kshirsagar, Meghana; Oviedo, Felipe; Ferres, Juan Lavista; Bowman, Gregory. Predicting the locations of cryptic pockets from single protein structures using the PocketMiner graph neural network. *Biophysical journal*. 2023, vol. 122, no. 3, 445a.

44. Lin, Zeming; Akin, Halil; Rao, Roshan; Hie, Brian; Zhu, Zhongkai; Lu, Wenting; Smetanin, Nikita; Santos Costa, Allan dos; Fazel-Zarandi, Maryam; Sercu, Tom; Candido, Sal, et al. Language models of protein sequences at the scale of evolution enable accurate structure prediction. *bioRxiv*. 2022.

45. Feidakis, Christos P; Krivak, Radoslav; Hoksza, David; Novotny, Marian. AHoJ-DB: A PDB-Wide Assignment of Apo & Holo Relationships Based on Individual Protein–Ligand Interactions. *Journal of Molecular Biology*. 2024, vol. 436, no. 17, p. 168545.

46. Feidakis, Christos P; Krivak, Radoslav; Hoksza, David; Novotny, Marian. *AHoJ-DB Statistics*. Available also from: `https://apoholo.cz/api/db/archive/v2c/db_stats.json`. Accessed: 2025-07-01.

47. Schubert, Erich; Sander, Jörg; Ester, Martin; Kriegel, Hans Peter; Xu, Xiaowei. DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems (TODS)*. 2017, vol. 42, no. 3, pp. 1–21.

48. Jarman, Angur Mahmud. Hierarchical cluster analysis: Comparison of single linkage, complete linkage, average linkage and centroid linkage method. *Georgia Southern University*. 2020, vol. 29, p. 32.

49. Creighton, Thomas E. *Proteins: structures and molecular properties*. Macmillan, 1993.

50. Kandel, Jeevan; Tayara, Hilal; Chong, Kil To. PUResNet: prediction of protein-ligand binding sites using deep residual neural network. *Journal of cheminformatics*. 2021, vol. 13, pp. 1–14.

51. Zhang, Yang; Skolnick, Jeffrey. TM-align: a protein structure alignment algorithm based on the TM-score. *Nucleic acids research*. 2005, vol. 33, no. 7, pp. 2302–2309.

52. Gowers, Richard J; Linke, Max; Barnoud, Jonathan; Reddy, Tyler John Edward; Melo, Manuel N; Seyler, Sean L; Domanski, Jan; Dotson, David L; Buchoux, Sébastien; Kenney, Ian M, et al. *MDAnalysis: a Python package for the rapid analysis of molecular dynamics simulations*. 2019. Tech. rep. Los Alamos National Laboratory (LANL), Los Alamos, NM (United States).

53. McKinley, Sky; Levine, Megan. Cubic spline interpolation. *College of the Redwoods*. 1998, vol. 45, no. 1, pp. 1049–1060.

54. Virtanen, Pauli; Gommers, Ralf; Oliphant, Travis E; Haberland, Matt; Reddy, Tyler; Cournapeau, David; Burovski, Evgeni; Peterson, Pearu; Weckesser, Warren; Bright, Jonathan, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods*. 2020, vol. 17, no. 3, pp. 261–272.

55. Schlitter, J; Engels, M; Krüger, P; Jacoby, E; Wollmer, A. Targeted molecular dynamics simulation of conformational change-application to the T $\leftrightarrow$ R transition in insulin. *Molecular Simulation*. 1993, vol. 10, no. 2-6, pp. 291–308.

56. Reese, Will. Nginx: the high-performance web server and reverse proxy. *Linux Journal*. 2008, vol. 2008, no. 173, p. 2.

57. Heaslet, Holly; Harris, Melissa; Fahnoe, Kelly; Sarver, Ronald; Putz, Henry; Chang, Jeanne; Subramanyam, Chakrapani; Barreiro, Gabriela; Miller, J Richard. Structural comparison of chromosomal and exogenous dihydrofolate reductase from Staphylococcus aureus in complex with the potent inhibitor trimethoprim. *Proteins: Structure, Function, and Bioinformatics*. 2009, vol. 76, no. 3, pp. 706–717.

58. Kokh, Daria B; Richter, Stefan; Henrich, Stefan; Czodrowski, Paul; Rippmann, Friedrich; Wade, Rebecca C. *Trapp: a tool for analysis of transient binding pockets in proteins*. ACS Publications, 2013.

List of Figures

List of Tables

Listings

List of Abbreviations

3D Three-Dimensional

AFDB AlphaFold Database

AHoJ Apo–Holo Juxtaposition

AI Artificial Intelligence

Å Ångstrom

API Application Programming Interface

CBS Cryptic Binding Site

CBR Cryptic Binding Residue

CORS Cross-Origin Resource Sharing

CPU Central Processing Unit

CSV Comma-Separated Values

CUDA Compute Unified Device Architecture

DHFR Dihydrofolate Reductase

FASTA Fast Alignment Search Tool - Any

GPU Graphics Processing Unit

HTTP HyperText Transfer Protocol

JSON JavaScript Object Notation

ML Machine Learning

mmCIF Macromolecular Crystallographic Information File

NMR Nuclear Magnetic Resonance

PDB Protein Data Bank

PDF Portable Document Format

plDDT predicted Local Distance Difference Test

PNG Portable Network Graphics

PyMOL Python Molecular Graphics System

RCSB Research Collaboratory for Structural Bioinformatics

REST Representational State Transfer

SOA Service-Oriented Architecture

WS WebSocket protocol

XTC eXtended Trajectory Coordinate

ZIP ZIP archive format

A Attachments

A.1 Source Codes

The application source code is publicly available on GitHub at `https://github.com/luk27official/cryptoshow` under tag `v1.0.0` and included in this thesis attachment. The code is licensed under the MIT License.

Additionally, the `cryptoshow-benchmark` folder contains benchmarking notebooks referenced in Section 2.5. This folder is also accessible on GitHub at `https://github.com/luk27official/cryptoshow-benchmark`.

A.2 Web Application

The CryptoShow web application is available at `https://cryptoshow.cz/`.

A.3 API Endpoints

The following list summarizes the API endpoints exposed by the FastAPI server.

- **GET /** - Root endpoint, returns a hello message for debugging.
- **GET /openapi** - Returns the generated OpenAPI schema for the API.
- **GET /gpu-status** - Checks if CUDA is available; returns a task ID for the CUDA Celery test task.
- **GET /health** - Returns the health status of the server (`status: healthy`).
- **POST /calculate** - Calculates the prediction for a given PDB/AF ID.
- **POST /calculate-custom** - Calculates the prediction for an uploaded PDB/mmCIF file.
- **GET /task-status/{task_id}** - Checks the status of a Celery task. Returns status and result if available.
- **GET /file/{task_hash}/{filename}** - Downloads a file (e.g., results) for a given task.
- **GET /animate/{task_hash}/{aligned_structure_filename}/ {target_chains}** - Creates an animated trajectory file for selected chains and a given structure file.
- **WS /ws/task-status/{task_id}** - WebSocket endpoint for real-time Celery task status updates.
- **POST /proxy/ahoj/job** - Proxies a POST request to `apoholo.cz` for job submission.

- **GET /proxy/ahoj/{task_hash}/{path}** - Proxies a GET request to `apoholo.cz` for file retrieval.