Grado en Ingeniería Biomédica

2024-2025

Trabajo Fin de Grado

“Development of computational workflows for the prediction of atomic models of biological macromolecules from their amino acid sequences.”

Estela Díez García

Tutor/es

Arrate Muñoz Barrutia

Carlos Oscar Sanchez Sorzano

Leganés, June 2025

ABSTRACT

In recent decades, biocomputational tools have become essential in researchers, particularly in the field of protein structure prediction. Since the launch of *AlphaFold2*, these tools have advanced significantly, offering increasingly accurate and sophisticated functions. However, the lack of interoperability among them has hindered seamless integration, prompting the development of the *Scipion* framework to unify various tools under a common interface. Despite its strengths, *Scipion* had yet to incorporate cutting-edge protein structure prediction software.

This thesis aims to integrate three state-of-the-art tools: *AlphaFold3*, *Boltz-1*, and *Chai- 1*, into the *Scipion* framework, adhering to the principles of traceability, reproducibility, interoperability, and integration. Two plugins were developed from scratch for *Boltz-1* and *Chai-1*, while the existing *ChimeraX* plugin was extended to support *AlphaFold3*. All plugins were implemented in Python, ensuring compatibility with the architecture of *Scipion* and automated installation procedures.

The integrated tools were tested on four macromolecules. *AlphaFold3* and *Chai-1* provided the most comprehensive outputs, including five predicted models accompanied by ranking scores. These two tools also demonstrated higher predictive accuracy than *Boltz-1*, as assessed using Root Mean Square Deviation (RMSD) analysis. These results highlight the feasibility and benefits of integrating advanced prediction tools into the *Scipion* to support robust and reproducible structural biology workflows.

This work demonstrates the feasibility and relevance of integrating advanced predictive tools into *Scipion*, setting the stage for future extensions of biocomputational workflows in structural biology.

Key words: Protein Structure Prediction, biocomputational tools, *Scipion* framework, RMSD, macromolecules, *Alphafold3*, *Chai*-1 and *Boltz*-1

ACKNOWLEDGEMENTS

I would like to begin by expressing my thanks to both of my tutors. Without their guidance and expertise, this thesis would not have been possible. Thank you, Arrate, for your valuable advice and constant support. Thank you, Carlos, for giving me the opportunity to develop this project at CNB, for welcoming me into your lab, and for being my guide throughout these past months. I could not be more grateful.

My deepest thanks also go to my family, my greatest pillar. Thank you for always believing in me, even in moments when I doubted myself.

To my friends, who have listened to my worries time and again, thank you for helping me disconnect, making me smile, and reminding me to enjoy the journey.

To my classmates, thank you for these four unforgettable years. Some of you have become more than classmates, you are family. I wouldn't have made it through this degree without you.

And finally, to my grandmother. She will not get to see me graduate, but she was my greatest motivation to pursue this path. She has been my inspiration and role model since I was a child, and she always will be, wherever I go.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCTION

1.1. Motivation

Biocomputational tools aim to design advanced biological systems that tackle critical challenges in biomedicine with machine-like precision [1].

One of the primary application areas of bioinformatics is protein structure prediction. Understanding the three-dimensional structure of proteins is crucial because it provides detailed insights into protein function. This structural knowledge is a powerful asset for accelerating drug discovery, designing new pharmaceuticals, and exploring protein interactions for a variety of biomedical and biotechnological applications [2].

Traditionally, determining the structure of a single protein experimentally could take many years, which significantly slowed down the drug discovery and research processes [3]. Although nowadays these experimental processes have been accelerated, they are still tedious and can last several months.

However, the emergence of specialized biocomputational tools has revolutionized this field by enabling researchers to predict accurate 3D protein models within minutes. This rapid prediction facilitates the analysis of protein interactions with other proteins, ligands, or small molecules, dramatically accelerating research and development cycles.

In most cases, researchers do not rely on just one program. Instead, they often combine several tools to complete their analysis, since each one offers different features and outputs. The problem is that switching between programs can be complicated. Many researchers do not have the skills to switch to another program without starting over, and in case they are able to make the switch, they face serious challenges like traceability, data, and metadata conversion, which will complicate their research and deflect their main activity: investigating.

To address the lack of interoperability between independent biocomputational programs, which typically lack shared interfaces or communication protocols, *Scipion* was developed. *Scipion* is a Python-based, open-source workflow engine that focuses on image processing. It was designed to help users combine different software tools more easily, and to ensure interoperability, traceability, reproducibility, and flexibility in their projects [4].

Due to the rapid advancement and frequent updates of biocomputational tools, *Scipion* must also evolve continuously to incorporate the latest software available in the field. Currently, the limited availability of protein structure prediction tools within the *Scipion* platform, despite their critical importance in modern research, represents a significant gap. Addressing this limitation and integrating these essential tools into *Scipion* constitutes the primary motivation of this thesis.

1.2. Objectives

The main objective of this thesis is to integrate into Scipion three programs dedicated to the prediction of molecular structures: *AlphaFold3*, *Chai-1*, and *Boltz-1*. These are all deep learning models capable of generating highly accurate predictions of the 3D structures of biomolecular complexes, including proteins, ligands, and nucleic acids.

Additionally, using *ChimeraX*, a tool already implemented in *Scipion*, RMSD analysis to compare the structures predicted by each program will be performed. This will allow us to assess the similarity between models and, therefore, evaluate the accuracy of the predictions.

As *Scipion* is built around the principles of traceability, reproducibility, interoperability, and integration [4], the final protocol must respect and support all of these aspects. The developed protocols should be user-friendly and easily combined with other tools available in *Scipion*. Moreover, the code should be clear and well-documented to allow future developers to understand, modify, or replicate it for new implementations.

Additionally, this thesis seeks to combine the technical integration of advanced molecular structure prediction tools within *Scipion* with a thorough exploration of protein structure and its crucial role in biomedical research. By addressing both the practical implementation and the scientific foundations of protein folding and function, this work aims to provide a well-rounded perspective that connects computational methods with their biological significance.

The project will be divided into three main phases, each of which must be completed to achieve the final goal:

Installation and preliminary understanding

The first step involves installing *Scipion* and gaining a basic understanding of how it works. *Scipion* requires some initial dependencies, such as *Conda*, and the installation should follow the official documentation, explained in detail in the Annex Chapter.

Once *Scipion* is properly set up, two essential plugins must be installed: the *ChimeraX* plugin and the *Xmipp* plugin.

After setting up the environment, the three prediction programs (*AlphaFold3*, *Chai-1*, and *Boltz-1*) should be investigated independently, and their usage and functionality must be understood in depth before attempting integration.

Any doubts, bugs, or issues should be addressed through the official repositories, and a solid understanding of each tool is essential before moving on to protocol development.

Program integration

This phase is divided into two parts.

First, the protocol development, which involves designing and coding the integration protocol for each tool according to *Scipion*'s architecture and coding standards. This process is described in detail in Chapter 5. The protocol must be compatible with the *Scipion* environment and should follow its workflow logic.

After the development, proceed with installation and debugging. After coding the protocol, it must be correctly installed and tested within *Scipion*. This includes solving any errors, implementing graphical elements, and ensuring proper visualization through the *Scipion* GUI.

Protocol testing

Once a fully functional protocol is ready, it must be thoroughly tested. The results obtained through *Scipion* should match those obtained when running each tool independently (outside the *Scipion* environment). This step is essential to confirm the accuracy and reliability of integration.

1.3. Content of the Document

This bachelor thesis is organized into eleven chapters. Following the introduction, which presented the project's motivation and objectives, the subsequent chapters provide a concise and comprehensive overview of the work carried out.

Chapter 2, *The Complex Structure of Proteins*, delves into the complex structure of proteins, explaining the hierarchical levels of folding and the fundamental chemical principles underlying them.

Chapter 3, *State of the Art*, explores the evolution of protein structure prediction, from traditional methods to the deep learning revolution led by *AlphaFold2* and its influence on the development of improved models such as *AlphaFold3* and *Chai-1*, *Boltz-1*.

Chapter 4, *Materials*, introduces all the materials used during the thesis and analyzes them. Including the *Scipion* framework, the three programs selected for integration, and the molecules used for the results testing.

Chapter 5, *Methodology*, explains all the steps followed to complete the development of the model.

Chapter 6, *Results*, presents the results of the validation of the model.

Chapter 7, *Discussion and Future Outlook*, analyzes the strengths and limitations of the work, based on the results of the experiments carried out, and the future research directions to follow.

Chapter 8, *Conclusion*, summarizes the key findings of the research.

Chapter 9, *Regulatory Framework*, outlines the relevant legislation applicable to the work and explores the potential for patenting the proposed method.

Chapter 10, *Socio-economic Impact*, examines the broader effects of research on society and the economy.

2. THE COMPLEX STRUCTURE OF PROTEINS

Since the latter half of the 20th century, many researchers in diverse fields have focused on the structure of proteins. This interest is driven by their complexity and functional sophistication, which are among the most structurally intricate molecules in living organisms. Their diverse roles in biological processes stem from their highly specific three dimensional conformations, making structural understanding essential for advancements in fields such as molecular biology, biochemistry, and biomedical engineering [5].

As one of the objectives of this thesis is to gain a comprehensive understanding of both the technical and biological aspects, this chapter provides an in-depth exploration of protein structure.

2.1. The Building Blocks. Amino Acids and Peptide Bonds

Proteins, also known as polypeptides, are large, complex molecules composed of amino acids linked together by covalent bonds to form long chains. There are only 20 standard amino acids involved in protein synthesis (even though more than 300 amino acids have been identified), each with distinct chemical properties, yet they can be arranged in countless combinations to produce a vast diversity of proteins [6]. Researchers have identified tens of thousands of different proteins, each with a unique sequence of amino acids and a specific function in the cell.

All amino acids share a common core structure, shown in Figure 2.1. At the center of each amino acid is a central α carbon atom, which is bonded to four different groups: a hydrogen atom, an α-carboxyl group (-COOH), an α-amino group (-NH_2), and a variable R-group, also called a side chain. The R-group is what distinguishes one amino acid from another, giving each its unique chemical and physical properties.

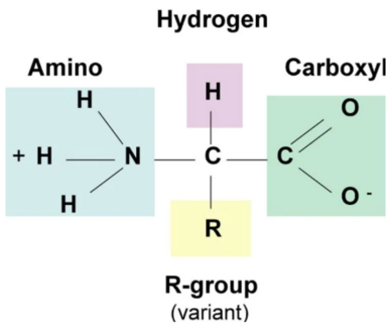


Fig. 2.1. General structure of an amino acid. All amino acids share this basic backbone, which consists of a central alpha carbon bonded to an amino group (NH3+), a carboxyl group (COO-), a hydrogen atom, and a variable side chain or R-group [7].

The covalent bond that links two amino acids together is known as a peptide bond as shown in Figure 2.2. This bond is formed through a condensation reaction, in which the carboxyl group of one amino acid reacts with the amino group of another. During this reaction, a molecule of water (H_2O) is released, and the hydroxyl (-OH) from the carboxyl group combines with a hydrogen (H) from the amino group. The resulting bond is an amide linkage (C–N), specifically referred to as a peptide bond, which forms the backbone of protein chains. However, it is the side chains that attach to this backbone structure, which makes each protein and its structure different.

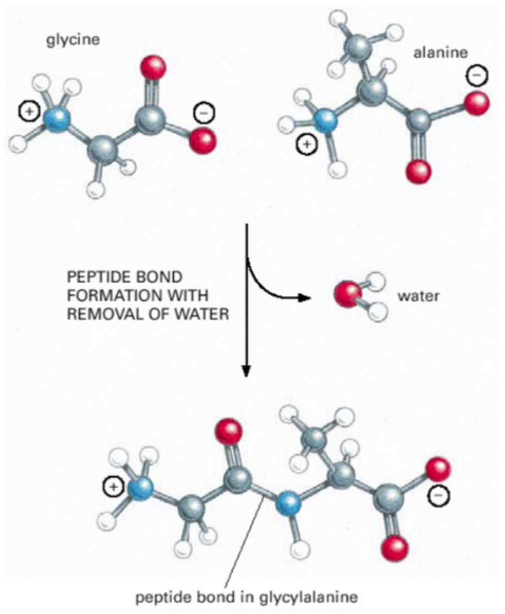


Fig. 2.2. Peptide bond formation between amino acids. The image illustrates the condensation reaction between glycine and alanine, resulting in the formation of a peptide bond and the release of a water molecule. This process links the carboxyl group of one amino acid to the amino group of another, forming a dipeptide (glycylalanine)[8].

2.2. Local Folding Patterns in Proteins

Proteins do not exist merely as simple, linear chains of polypeptides. Instead, these chains can fold in a variety of ways, giving rise to specific structural motifs. Although the number of possible conformations is vast, the folding is constrained by fundamental physical limitations, primarily due to the phenomenon of steric hindrance represented in Figure 2.3, two atoms cannot overlap. As a result, only certain folding patterns are energetically and geometrically favorable.

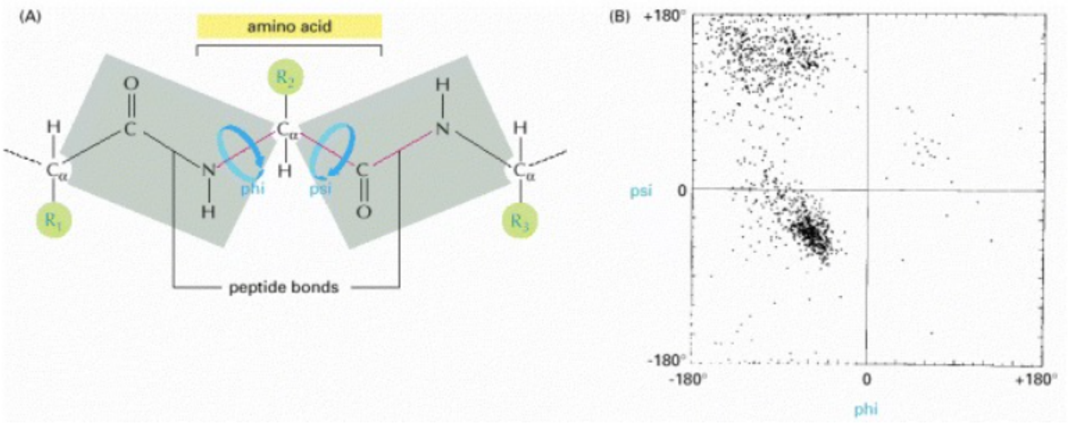


Fig. 2.3. Steric limitations on the bond angles in a polypeptide chain. (A) Each amino acid contributes three backbone bonds (red). The peptide bond is planar and rigid due to resonance (gray planes). However, the bonds adjacent to the α-carbon (C_α–C and N–C_α) allow rotation, defined by the psi (ψ) and phi (ϕ) torsion angles, respectively. These rotational freedoms are influenced by the side chains (R groups) highlighted in green. (B) A Ramachandran plot shows the allowed combinations of ϕ and ψ angles for residues in known protein structures. Due to steric hindrance, most angle combinations are not permitted, as evidenced by the clustering of observed points [8].

The folding process is further regulated by various noncovalent interactions. Although these interactions are significantly weaker than covalent bonds (each being approximately 30 to 300 times weaker), they often operate in parallel, collectively stabilizing specific regions of the polypeptide chain. There are three main types of noncovalent forces that contribute to protein folding, Figure 2.4: hydrogen bonds, van der Waals attractions, or ionic bonds.

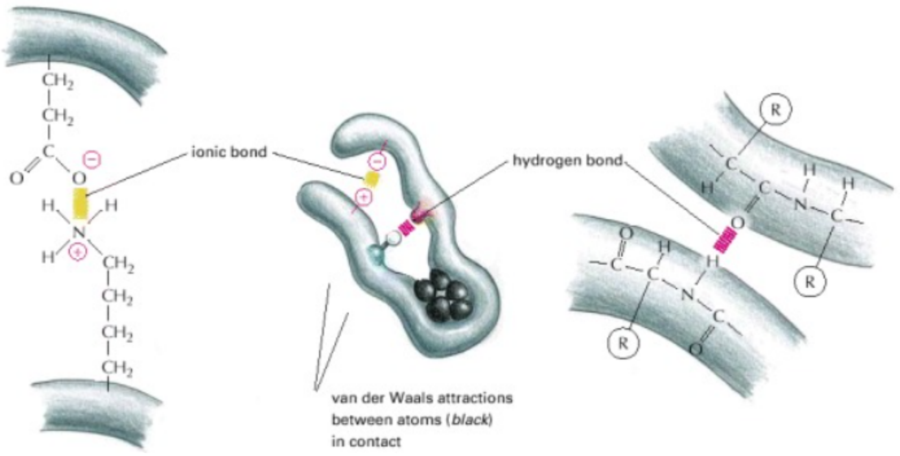


Fig. 2.4. Types of noncovalent bonds involved in protein folding. The hydrogen bond is a hydrogen atom that is weakly shared between two electronegative atoms. The ionic bond is a bond formed by the electrostatic attraction of two oppositely charged ions. The Van der Waals attractions are driven by induced electrical interactions between two or more atoms or molecules that are very close to each other [8].

Another key factor determining the final structure of a protein is the distribution of polar and non-polar amino acids. In most biological contexts, proteins exist in an aqueous environment. Consequently, nonpolar (hydrophobic) amino acids tend to cluster within the interior of the molecule to avoid contact with water, while polar (hydrophilic) amino acids are more likely to be positioned on the outer surface and interact with the surrounding solvent.

As a result of all these noncovalent interactions, each protein adopts a unique three-dimensional conformation, dictated by the specific sequence and spatial arrangement of its amino acids. Typically, the final folded structure corresponds to the lowest energy state of the molecule. However, this conformation can undergo slight changes upon interaction with other molecules, which in turn may influence the function of the protein.

2.3. α-Helix and β-Sheet Folding Patterns

When the three-dimensional structures of many different protein molecules are compared, it becomes clear that, although the overall conformation of each protein is unique, two regular folding patterns are commonly observed in certain regions, these are α-helix and the β-sheet, as shown in Figure 2.5. These structures share a key feature: they arise from hydrogen bonding between the N–H and C=O groups of the polypeptide backbone, without involving the side chains of the amino acids. As a result, a wide variety of amino acid sequences can adopt these structural motifs.

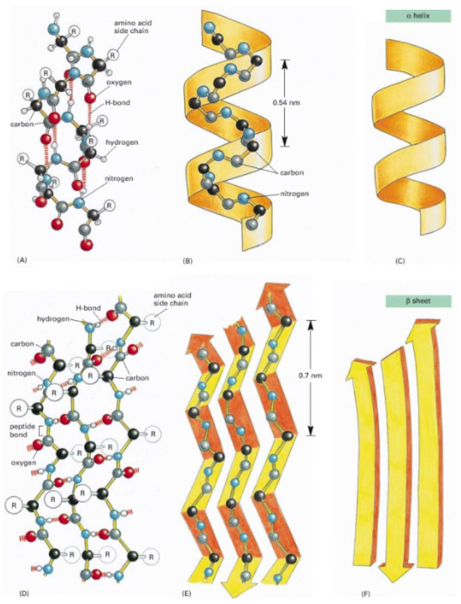


Fig. 2.5. General conformation of the polypeptide backbone observed in α-helix and β-sheet structures. (A), (B), and (C) show the α-helix, a coiled structure stabilized by hydrogen bonds every fourth peptide bond, with one turn every 3.6 amino acids. (D), (E), and (F) show the β-sheet, formed by parallel or antiparallel chains, both producing a stable, rigid structure via hydrogen bonding [8].

As mentioned before, these are the most common conformations observed in nature, but many other structures are also possible, such as triple helices, turns, or random coils [8].

2.4. Higher Levels of Protein Folding

Proteins exhibit four hierarchical levels of structural organization. The first level, known as the primary structure, refers to the linear sequence of amino acids linked by peptide bonds. The second level, or secondary structure, includes regular, repeating elements such as α-helices and β-sheets, stabilized by hydrogen bonds between backbone atoms.

The tertiary structure describes the overall three-dimensional folding of a single polypeptide chain, driven by interactions among side chains (R-groups), including hydrophobic interactions, hydrogen bonds, charge-charge interactions, and disulfide bridges. Finally, the quaternary structure applies to proteins composed of multiple polypeptide chains, where these subunits assemble into a functional complex through various intermolecular forces, which are the same as the interactions that stabilize the tertiary structure.

Each of these structural levels contributes to the final shape and biological function of the protein [9].

3. STATE OF ART

3.1. The Revolution of Protein Structure Prediction. AlphaFold2

The prediction of protein structures has evolved significantly over recent decades. Early computational approaches, such as homology modeling, relied on the alignment of protein sequences to known templates and performed well when suitable reference structures were available [10]. However, these methods struggled with novel protein folds. To address these limitations, free modeling techniques emerged, including tools like *Rosetta*, which used fragment assembly and energy minimization strategies. The real breakthrough came with the application of deep learning, which led to remarkable improvements in prediction accuracy, culminating in the development of *AlphaFold2* [11].

Such was its importance that the developers of *AlphaFold*, David Baker, Demis Hassabis, and John Jumper, were awarded the Nobel Prize in Chemistry in 2024[12].

AlphaFold2 revolutionized protein structure prediction by directly estimating the 3D coordinates of all heavy atoms from a protein's amino acid sequence, supported by multiple sequence alignments (MSAs).

Its architecture, visually represented in Figure 3.1, consists of two main components: the Evoformer, which processes sequence and evolutionary data using novel attention-based mechanisms to infer residue relationships, and the Structure Module, which refines a spatial representation of the protein through iterative 3D modeling. This second process includes rotation and translation operations for each residue and uses an innovative recycling technique to enhance structural accuracy through repeated refinement [13].

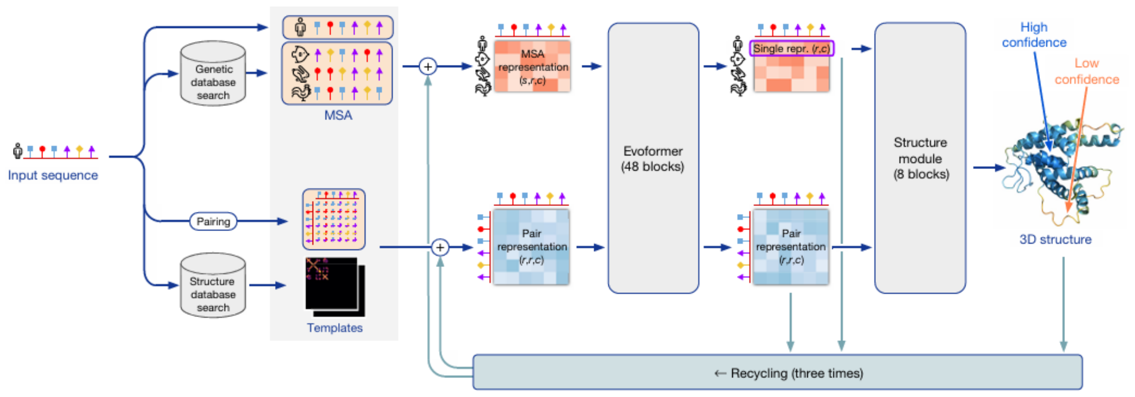

Input sequence
Genetic database search
Pairing
Structure database search
MSA
Templates
MSA representation (s,r,c)
Pair representation (r,r,c)
Evoformer (48 blocks)
Single repr. (r,c)
Pair representation (r,r,c)
Structure module (8 blocks)
High confidence
Low confidence
3D structure
← Recycling (three times)

Fig. 3.1. *Alphafold2* model architecture. Arrows indicate the flow of information between components of the *AlphaFold2* pipeline, from input sequence through database searches, representation learning, and structure prediction. The model combines MSAs and template information to build sequence and pair representations, which are iteratively refined and used to predict the 3D structure. Array shapes are shown in parentheses, where s is the number of sequences, r the number of residues, and c the number of channels [13].

As with all neural networks, *AlphaFold2* has certain limitations. The capabilities and constraints of neural networks are largely determined by the data used during training. In the case of *AlphaFold2*, only the protein components of Protein Data Bank (PDB) structures were included, omitting other biologically relevant molecules such as small ligands, nucleic acids, and cofactors.

The original version of *AlphaFold2* was specifically designed to predict the three-dimensional structures of individual protein chains. This functionality was later expanded with the development of *AlphaFold-Multimer*, an extension trained to predict the structures of protein-protein complexes.

In particular, *AlphaFold2* does not reproduce only known protein structures. Independent research has shown that the model is capable of accurately predicting novel protein folds, three-dimensional conformations not previously observed in the PDB.

Despite its remarkable performance, *AlphaFold2* exhibits several limitations. It is relatively insensitive to point mutations, where a single amino acid is altered due to a change in the DNA sequence. It also performs poorly with so-called .ºrphan"proteins, which lack homologous sequences in existing databases. Moreover, the model does not account for conformational changes that proteins may undergo as part of their functional mechanisms.

In addition, *AlphaFold2* does not consider the presence or influence of non-protein molecules that interact with proteins, such as nucleic acids, small molecules, ions, or other cofactors, which may play crucial roles in protein structure and function [14].

3.2. Deep Learning Models for Protein Structure Prediction

Alphafold3

In May 2024, *AlphaFold3* was released, representing a substantial advancement over *AlphaFold2* in both accuracy and scope. While it continues to improve the prediction of protein structures, its main strengths lie in the more precise modeling of protein complexes and its expanded applicability to a wider range of biomolecules, encompassing nearly all molecular entities found in the Protein Data Bank (PDB) [15].

AlphaFold3 also offers improved predictions of the structural impact of covalent modifications, including bonded ligands, glycosylation, chemically modified residues, and nucleic acid bases, across proteins, RNA, and DNA. Notably, the model shows significant progress in predicting protein-protein interactions, particularly enhancing the accuracy of antibody, antigen interface modeling when compared to *AlphaFold-Multimer*.

Unlike *AlphaFold2*, however, *AlphaFold3* is not fully open source nor available for commercial use, prompting criticism from the scientific community and sparking a global race to develop a commercially accessible alternative[16].

As illustrated in Figure 3.2, *AlphaFold3* retains a structural architecture similar to that of *AlphaFold2*, with several critical improvements. Template and genetic searches are still employed, but the MSA module is reduced in size. The Evoformer has been replaced by a Pairformer module, which processes only single- and pair-representations.

The structure module has been replaced with a generative diffusion model, which predicts a distribution of structures rather than a single conformation. This approach yields accurate results without requiring parametrization or physics-based minimization, such as AMBER (Assisted Model Building with Energy Refinement). To prevent implausible structures in disordered regions, the model incorporates cross-distillation using AlphaFold-Multimer training data that includes flexible loop regions.Furthermore

Finally, a newly introduced confidence module provides error estimations at the atomic and pairwise levels, improving the reliability of the predictions [17].

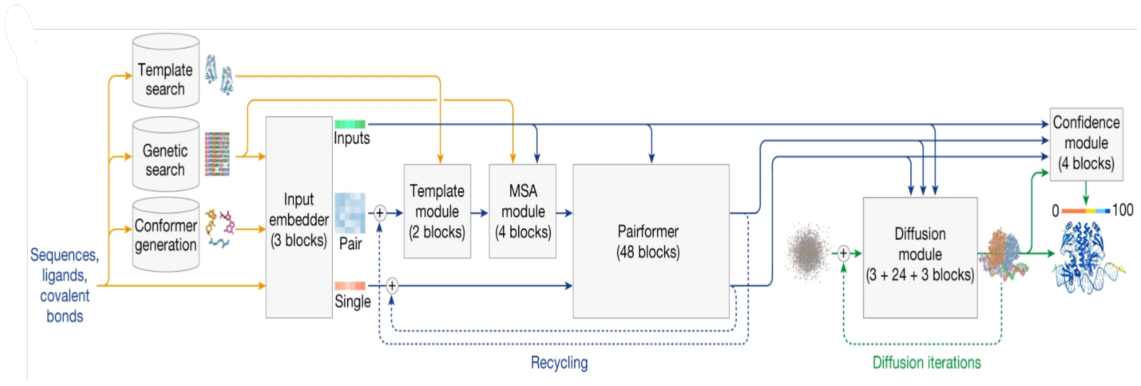

Template search
Genetic search
Conformer generation
Sequences, ligands, covalent bonds
Input embedder (3 blocks)
Inputs
Pair
Single
Template module (2 blocks)
MSA module (4 blocks)
Pairformer (48 blocks)
Recycling
Diffusion module (3 + 24 + 3 blocks)
Diffusion iterations
Confidence module (4 blocks)
0
100

Fig. 3.2. *Alphafold3* model architecture. Its model architecture consists of a pipeline of embedding, alignment (template and MSA), deep transformer processing (Pairformer), and iterative 3D refinement via diffusion. Confidence scoring and recycling loops improve accuracy [17].

Chai-1

The release of *AlphaFold3* served as a starting point for the creation of new protein structure prediction programs. One of these is *Chai-1*. In September 2024, *Chai-1* was introduced as an openly accessible foundation model, this means closed-sourced but publicly available, for the prediction of biomolecular structures. The model weights and inference code are available for non-commercial use. Additionally, a web server is provided to facilitate interaction with the model, which is accessible for commercial applications, including drug discovery tasks.

Chai-1 enables unified prediction across a diverse range of biomolecules, including proteins, small molecules, DNA, RNA, and covalently modified structures. Although *Chai-1* achieves its highest performance when supplied with multiple sequence alignments (MSA), it is also capable of generating strong predictions in single-sequence mode, without relying on MSAs.

Despite its capabilities, *Chai-1* presents two primary limitations. First, while it often predicts the structures of individual protein chains with high accuracy, it may fail to assemble these chains correctly within a complex when additional inter-chain contact information is lacking. Second, *Chai-1* displays high sensitivity to post-translational modifications; modifications such as the removal or substitution of modified residues can lead to significant deviations in the predicted structures. This sensitivity probably stems from the dependence of *Chai-1* on the presence of modifications during training to accurately predict molecular structures [18].

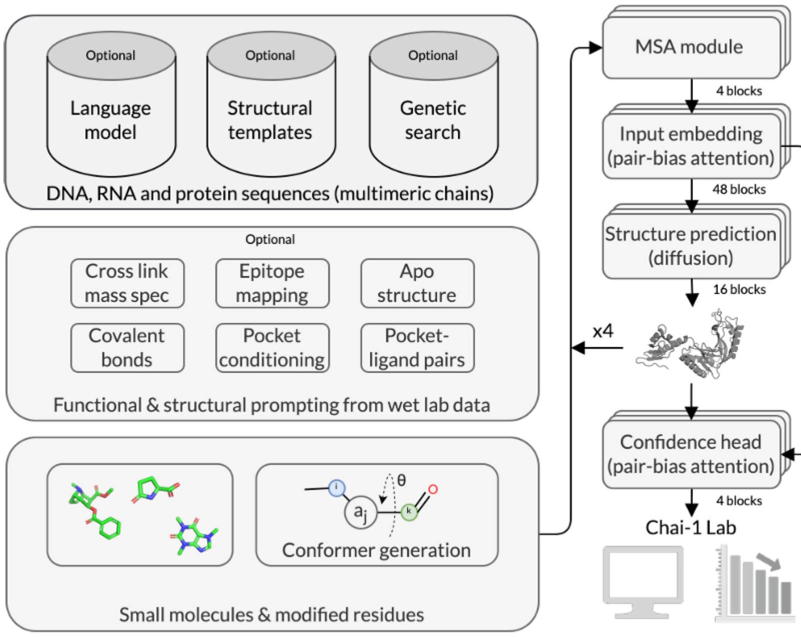


Fig. 3.3. *Chai-1* model architecture. *Chai-1* accepts DNA, RNA, and protein sequences, optionally enhanced with language model embeddings, structural templates, genetic search, and functional wet-lab data such as cross-linking or epitope mapping. The model incorporates residue-level embeddings to improve single-sequence performance and includes constraint features to simulate experimental data, aiding complex assembly. Structure prediction is performed through an MSA-based diffusion process with iterative refinement and confidence estimation [18].

Boltz-1

Boltz-1, released in November 2024, represents a significant advancement in biomolecular structure prediction, standing out as the first fully open-source and commercially accessible model to achieve predictive accuracy on par with *AlphaFold3*. In contrast to models with restricted access, *Boltz-1* openly provides its training code, inference code, model weights, and datasets under the permissive MIT license.

While *AlphaFold3* leverages template structures to guide its predictions, *Boltz-1* operates independently of such templates. Instead, it directly processes raw inputs consisting of proteins, ligands, and nucleic acids. To enhance its predictive capabilities, *Boltz-1* augments these inputs by integrating multiple sequence alignments (MSAs) and predicted molecular conformations.

Architecturally, *Boltz-1* draws conceptual inspiration from *AlphaFold3* but introduces several novel innovations that collectively distinguish its design and performance. As illustrated in Figure 3.4, one major innovation includes the development of more efficient algorithms for MSA pairing and structure cropping during training, as well as the ability to condition predictions based on user-defined binding pockets. Furthermore, *Boltz-1* introduces substantial modifications to the internal flow of molecular representations within the network, accompanied by improvements to its diffusion-based training and inference processes. Another key advancement involves the redefinition of the confidence estimation mechanism of the model, integrating architectural revisions that improve both accuracy and robustness [19].

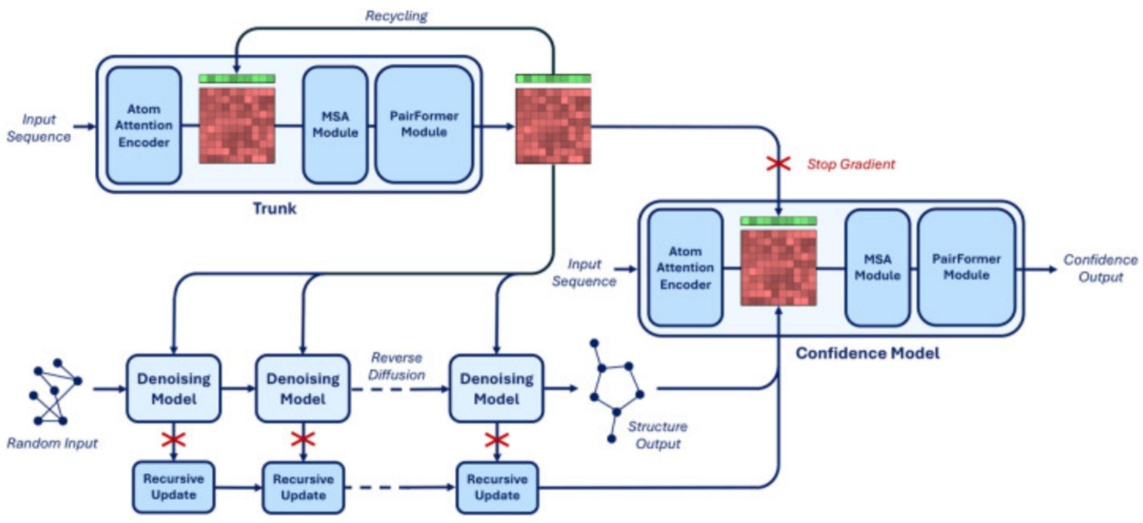


Fig. 3.4. *Boltz-1* model architecture. *Boltz-1* predicts molecular structures using a two-part architecture: a trunk model processes input sequences via attention, MSA, and PairFormer modules; then, a denoising module performs reverse diffusion steps to generate 3D structures. A separate confidence model evaluates structureal reliability. Gradients are stopped between the trunk and confidence outputs to stabilize learning [19].

3.3. Protein Structure Prediction in *Scipion*

Although *Scipion* was initially developed to support image processing in electron microscopy by integrating various 3DEM software packages under a unified interface for both biologists and developers [4], it has gradually expanded to incorporate additional functionalities. Among these, *Scipion* includes some protocols related to protein structure prediction, notably through the integration of *AlphaFold2*.

This functionality is currently available within the *ChimeraX* plugin, which enables users to search and retrieve existing models from the *AlphaFold* Database, execute new *AlphaFold* predictions using Google Colab, or run local *AlphaFold2* predictions when the necessary environment is configured.

However, the current implementation is limited to *AlphaFold2*, which, as previously discussed, presents several constraints, such as difficulties handling non-protein components. These limitations were addressed in the more advanced *AlphaFold3* version, which is not yet implemented in *Scipion*.

Furthermore, other promising tools for structure prediction, such as *Chai-1*, and *Boltz-1*, have not been integrated either. This lack of diversity in predictive tools represents a significant limitation for users seeking to perform flexible, up-to-date, and comprehensive structural modeling workflows within the *Scipion* framework.

4. MATERIALS

This chapter outlines the materials required to successfully achieve the objectives of this thesis. The main materials include the entire *Scipion* framework, the software used for integration, and the molecules necessary for the analysis presented in the Results section.

In addition to these core components, two additional tools are introduced due to their essential roles in the development and visualization processes: *PyCharm* and *ChimeraX*.

PyCharm is an integrated development environment (IDE) specifically designed for Python programming. It plays a crucial role in this project as the development environment where all Python scripts, explained in detail in the Methodology chapter (Chapter 5), are created and edited to function within the *Scipion* framework. The use of *PyCharm* ensures efficient code management, debugging, and testing throughout the development process.

ChimeraX is a molecular visualization program developed by the Resource for Biocomputing, Visualization, and Informatics (RBVI). It is integrated within the *Scipion* environment and plays a vital role during the visualization phase of this project. Beyond visualization, *ChimeraX*'s powerful scripting capabilities allow it to execute multiple functions that modify molecular structures, perform sequence alignments, and other specialized functions. In this thesis, particularly the alignment of molecular models will be used.

Together, these materials provide a comprehensive platform for the development and analysis of the project.

4.1. *Scipion* Framework Analysis

As explained in Chapter 1, *Scipion* is a cryo-electron microscopy (cryo-EM) image processing framework developed by the CNB-CSIC, designed to support and streamline biological research.

After the installation of *Scipion*, the software can be launched by executing the command ./scipion3 in the Ubuntu terminal. Upon execution, the *Scipion* graphical user interface (GUI) will open, represented in Figure 4.1, allowing the user to begin data processing workflows.

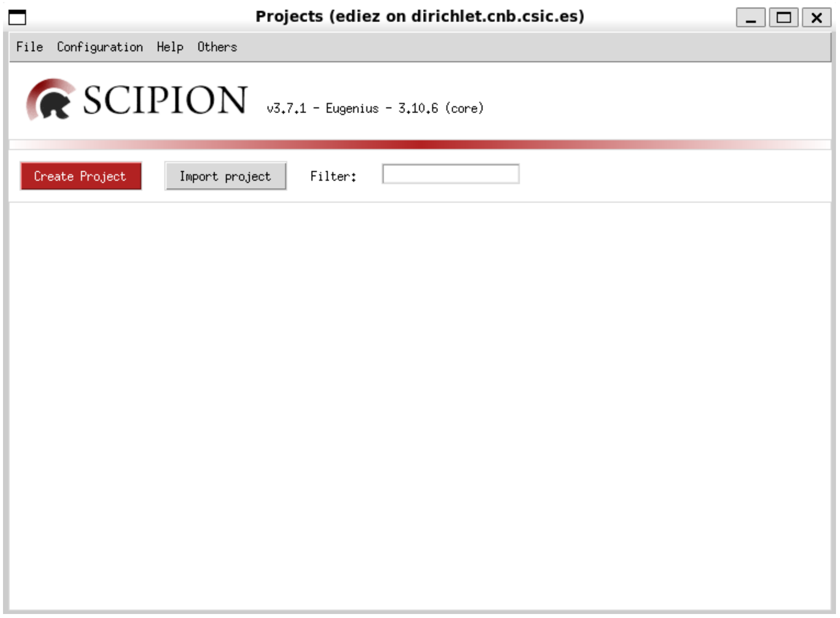
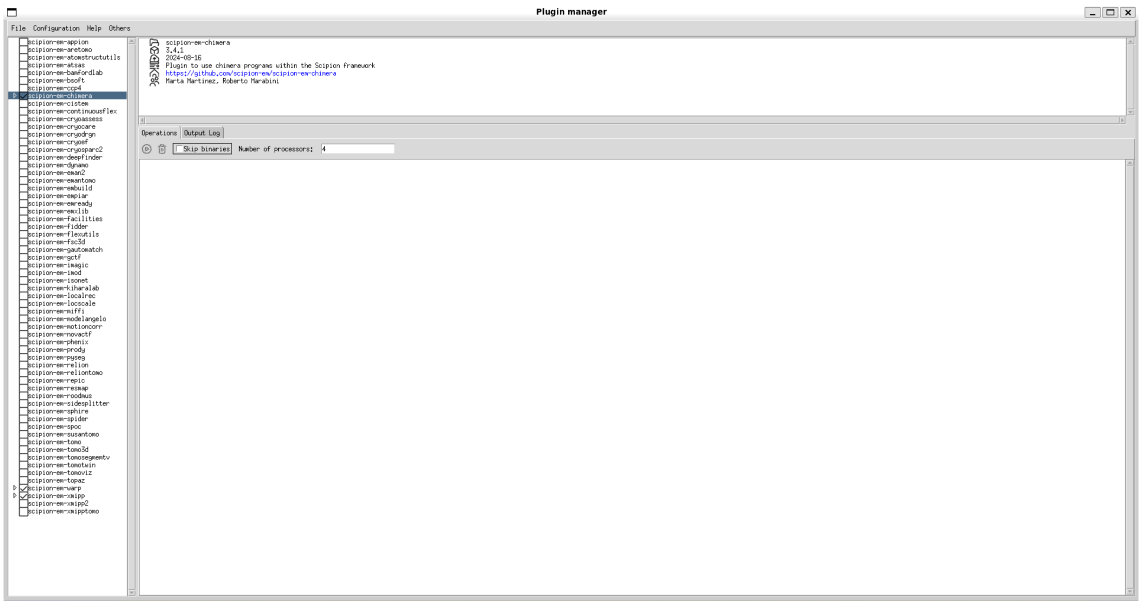

Projects (ediez on dirichlet.cnb.csic.es)
File Configuration Help Others
SCIPION v3.7.1 - Eugenius - 3.10.6 (core)
Create Project
Import project
Filter:

Fig. 4.1. *Scipion* home screen interface. We can find several options on the home screen. Create Project allows the user to start a new project from scratch, while Import Project enables the import of an existing project file, allowing for further editing or additions. Additionally, the Filter option allows users to search for a project simply by typing its name or part of it. At the top of the *Scipion* window, there is a dropdown menu bar with several sections. File allows access to the data folder; Configuration provides access to general, host, and protocol settings; Help links to support info; and Others includes key tools like the Plugin Manager, which is essential for installing additional plugins.

The Plugin Manager tab, visible in Figure 4.2, is a highly useful tool that allows for the easy installation and uninstallation of plugins.



- [ ] scipion-em-appion
- [ ] scipion-em-aretomo
- [ ] scipion-em-atomstructutils
- [ ] scipion-em-atsas
- [ ] scipion-em-bamfordlab
- [ ] scipion-em-bsoft
- [ ] scipion-em-ccp4
- [✓] scipion-em-chimera
- [ ] scipion-em-cistem
- [ ] scipion-em-continuousflex
- [ ] scipion-em-cryoassess
- [ ] scipion-em-cryocare
- [ ] scipion-em-cryodrgn
- [ ] scipion-em-cryoef
- [ ] scipion-em-cryosparc2
- [ ] scipion-em-deepfinder
- [ ] scipion-em-dynamo
- [ ] scipion-em-eman2
- [ ] scipion-em-emantomo
- [ ] scipion-em-embuild
- [ ] scipion-em-empiar
- [ ] scipion-em-emready
- [ ] scipion-em-emxlib
- [ ] scipion-em-facilities
- [ ] scipion-em-fidder
- [ ] scipion-em-flexutils
- [ ] scipion-em-fsc3d
- [ ] scipion-em-gautomatch
- [ ] scipion-em-gctf
- [ ] scipion-em-imagic
- [ ] scipion-em-imod
- [ ] scipion-em-isonet
- [ ] scipion-em-kiharalab
- [ ] scipion-em-localrec
- [ ] scipion-em-locscale
- [ ] scipion-em-miffi
- [ ] scipion-em-modelangelo
- [ ] scipion-em-motioncorr
- [ ] scipion-em-novactf
- [ ] scipion-em-phenix
- [ ] scipion-em-prody
- [ ] scipion-em-pyseg
- [ ] scipion-em-relion
- [ ] scipion-em-reliontomo
- [ ] scipion-em-repic
- [ ] scipion-em-resmap
- [ ] scipion-em-roodmus
- [ ] scipion-em-sidesplitter
- [ ] scipion-em-sphire
- [ ] scipion-em-spider
- [ ] scipion-em-spoc
- [ ] scipion-em-susantomo
- [ ] scipion-em-tomo
- [ ] scipion-em-tomo3d
- [ ] scipion-em-tomosegmemtv
- [ ] scipion-em-tomotwin
- [ ] scipion-em-tomoviz
- [ ] scipion-em-topaz
- [✓] scipion-em-warp
- [✓] scipion-em-xmipp
- [ ] scipion-em-xmipp2
- [ ] scipion-em-xmipptomo

```
scipion-em-chimera
3.4.1
2024-08-16
Plugin to use chimera programs within the Scipion framework
https://github.com/scipion-em/scipion-em-chimera
Marta Martinez, Roberto Marabini
```

Skip binaries [ ] Number of processors: 4

Fig. 4.2. Plugin manager. On the left side, a list shows all available plugins, each marked with a checkbox indicating whether it is installed. Plugins can be installed or uninstalled by simply checking or unchecking the boxes, and clicking the Run button executes all pending tasks. On the right, the upper panel displays information about the selected plugin, while the lower section contains two tabs: Operations, showing the task queue, and Output Log, which records terminal output during execution.

Once a new project is created, the project window is displayed, shown in Figure 4.3, providing access to all available plugins.

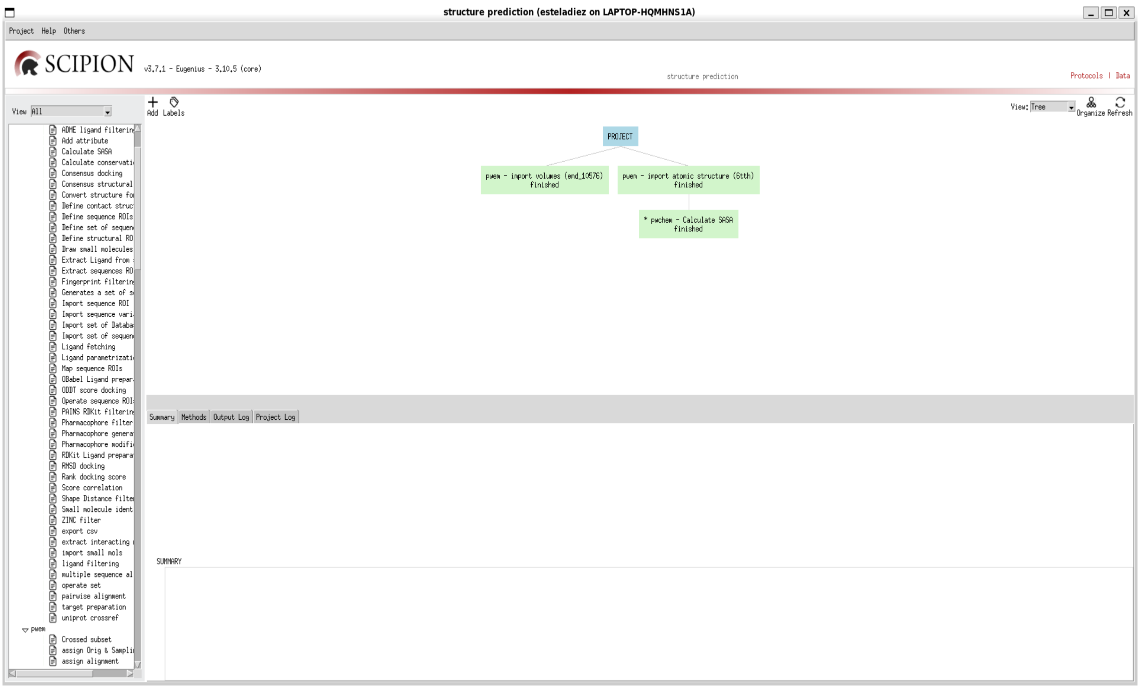
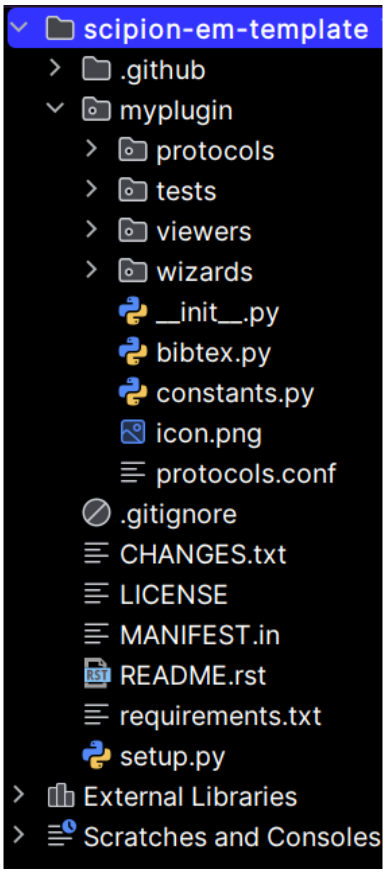


Fig. 4.3. *Scipion* project window interface. This screen provides a schematic view of the launched projects. With the main title 'PROJECT', arrows organize the protocols followed. A green box indicates that the protocol ran successfully without errors, while a red box signifies that an error occurred. On the left side, there is a list of all available plugins in *Scipion*, organized by function. At the bottom, four tabs provide detailed information about the executed protocol. Summary offers a brief overview of the execution or possible errors; Methods summarizes the protocols used and includes references if available; Output Log includes two sub-tabs, run.stdout, which provides a detailed step-by-step execution log, and run.stderr, which lists any detected errors and the corresponding algorithm lines (if empty, no errors were found); and, finally, Project Log, which displays terminal output from the task scheduler managing protocol execution.

4.1.1. Plugin Architecture

To have a better understanding of how *Scipion* works, it is important to know its plugin architecture. *Scipion*'s developers have created a template plugin, that serves as a starting point to create a new one. It contains the basic structure of a plugin, and in this section, it will be analyzed and explained in detail.



```
scipion-em-template
  .github
  myplugin
    protocols
    tests
    viewers
    wizards
    __init__.py
    bibtex.py
    constants.py
    icon.png
    protocols.conf
  .gitignore
  CHANGES.txt
  LICENSE
  MANIFEST.in
  README.rst
  requirements.txt
  setup.py
External Libraries
Scratches and Consoles
```

Fig. 4.4. Directory structure. Directory structure of the scipion-em-template plugin, highlighting key components such as the myplugin folder with its protocols, viewers, and wizards subdirectories, as well as essential configuration files (constants.py, protocols.conf, requirements.txt) required for integration within the *Scipion* framework.

Each *Scipion* plugin is customizable, and developers modify the files within the template to adapt them to the specific functions required by their plugin. The core structure of a plugin involves modifying key files to ensure that it correctly integrates into the *Scipion* framework.

The file containing all the information about the plugin and its functionality is the README.rst file. This file can be viewed on GitHub, not within *Scipion*, and is crucial for users to understand. Any further questions can also be addressed through GitHub.

One important file is setup.py, traditionally used for packaging and distributing Python projects. However, in the latest version of *Scipion*, the use of setup.py has led to integration issues. To address these challenges, *Scipion* now favors the use of pyproject.toml, a more modern approach for configuring and managing Python projects. Therefore, when creating new plugins for *Scipion*, pyproject.toml will be utilized instead of the older setup.py.

Another key file is requirements.txt. This file outlines the external Python libraries, packages, and pre-existing plugins required for the plugin's functionality. By listing these dependencies, the requirements.txt file ensures that all the necessary components are available for the plugin to function properly within the *Scipion* environment.

Nevertheless, the core of the plugin is found within the myplugin folder: the protocol, the viewers, the wizards, the tests, and the installer.

The __init__.py file contains all the installation information, creating the environment within *Scipion* to activate while using the plugin, along with the programs installed within it. This file utilizes constants set in the constants.py file, such as the plugin version, name, environment name, and others.

Inside this folder, there are several subdirectories. First, and most importantly, is the protocols directory, which contains all the plugin's protocols (in our case, there will only be one), as well as the __init__.py file, which imports the protocols. The structure of these files will be analyzed in the next subsection.

There is also the wizards directory, which is used for creating wizards in the *Scipion* interface. These wizards guide the user through specific tasks or workflows within the plugin.

Additionally, the viewers directory determines how the "Analyze Results" tab in *Scipion* will appear. This allows the output to be visualized in applications like *ChimeraX* or *PyMOL*, for example. This directory also includes an __init__.py file for importing.

Lastly, the tests directory contains a file for each protocol with tests that perform runs with example cases and check that the results are correct. This is done to confirm that the installation has been completed successfully and that the installed plugin is ready for use.

To determine where the plugin will appear in the *Scipion* Project Window Interface (Figure 4.3), the protocols.conf file must be modified to include data specifying where the plugin should be categorized. In *Scipion*, it can be classified under categories such as: Protocol SPA, Random Conical Tilt, Model Building, Virtual Screening, or Tomography.

4.1.2. Protocol Architecture

Every protocol follows a specific structure, which is organized as outlined below:

As with any Python script, the first step is to import the necessary Python libraries or any functions and constants that might have been defined in other plugins.

Following this, some initial variables are defined at the beginning of the protocol, such as _label, which is used to specify the protocol's name.

The variables are defined within the _defineParams function. In this function, several parameters from the pyworkflow library are imported, including EnumParam to accept inputs from a predefined list, FileParam to accept a file as input, IntParam for defining parameters that only accept integer values, StringParam to define text-based parameters, BoolParam for accepting boolean inputs (True/False), and FloatParam, which works similarly to IntParam, but for floating-point values.

Once the protocol's parameters have been defined, the next step is to establish the steps the protocol will follow, using function names. These steps are specified in the _insertAllSteps function. The structure of this part can vary significantly from one protocol to another, so there is no universal format that applies to all. In the Methods section, it will be explained how this part was generated for each specific protocol, depending on the parameters and functions involved.

Afterward, all functions defined within _insertAllSteps will be scheduled for execution. Additional functions, not defined in _insertAllSteps, can still be included and used during the protocol. However, these functions will not be executed unless they are referenced elsewhere within the protocol.

At the final stage, there is always a section dedicated to information functions. This includes the _validate function, which checks that all necessary conditions are met before the protocol begins running. For example, if the protocol expects an input file in .cif format and the file provided is in .pdb format, the protocol will not start. In addition, the _summary function is included in this section. This function generates a brief summary of what happened during the protocol's execution, which appears in *Scipion*; if any errors occur, they will also be reported in this section. The _methods function explains the methods used during the protocol, and finally, the _citations function is included to reference any external programs used in the protocol. For example, if an external program was installed in __init__.py, it should be cited here.

4.2. Analysis of the Software Selected for Integration

This section is crucial for the subsequent development of the protocols. In the State of the Art section, it has already been discussed in detail the strengths and limitations of each program, as well as their model architecture, providing a foundational understanding of the internal workings of the software. While gaining a complete and in-depth knowledge of their architectures could be extensive, this thesis is not focused on fully understanding these models but rather on their implementation.

In this section, the main focus will be analyzing the servers of these programs to understand their input requirements, which is essential to determine the parameters of the protocol. In addition, their output will be examined, as it is crucial to define how the protocol will interact with and process the results of each program.

After analyzing the servers of *Chai-1* and *Alphafold3*, since *Boltz-1* does not have a web server and their GitHub repositories, Table 4.1. represents all the relevant information regarding the input parameters.

TABLA 4.1. INPUT PARAMETERS OF THE SOFTWARE

Software	**INPUT PARAMETER**			
	Entity type	**Input sequence**	**MSA**	**Input File**
Alphafold3	Protein DNA RNA Ligand Ion	YES	NO	JSON File
Chai-1	Protein DNA RNA Ligand Ion	YES	YES	FASTA File PDB id
Boltz-1	Protein DNA RNA Ligand Ion	NO	YES	FASTA File PDB id

The table correlates each software with the types of parameters it uses. Entity type refers to the type of molecules the program accepts for structure prediction. Input sequence indicates whether the program allows manual input of the sequence, in addition to accepting it through an input file. MSA, Multiple Sequence Alignment, specify whether the program supports their activation and use to enhance prediction accuracy. Finally, the input file section lists the file formats accepted by the software, which will be the ones that the created protocol will accept.

It is equally important to understand the output generated by each program based on the inputs provided. This is explained and summarized in Table 4.2.

TABLA 4.2. OUTPUT OF THE SOFTWARE

Software	**OUTPUT OBTAINED**
Alphafold3	A .zip folder composed of 5 .CIF files and 10 .JSON files
Chai-1	A .zip folder composed of 5 .CIF files, 5 .JSON files and 5 .NPY files
Boltz-1	A folder composed of four folders: *Lightning_Logs* with a .YAML file (parameters information), *MSA* with a .CSV file, *Process* folder with one .json and two .NPZ files and the *Predicitions* folder, composed of one .CIF file (with the predicted structure information) one .son and one .NPZ file.

4.3. Molecule Analysis

In this section, the molecules used to test the integrated programs are introduced along with their scientific background.

Although their scientific background is not the main focus of this thesis, it is important to understand how the integration of protein structure prediction programs has significant potential in research and how it can impact drug discovery and pharmacological outcomes.

The proteins analyzed during this thesis, and consequently in the Results section, are hemoglobin, 2,3-bisphosphoglycerate (2,3-BPG), and sphingosine-1-phosphate.

Human hemoglobin molecules consist of closely related proteins formed by the symmetric pairing of two dimers of polypeptide chains, the structurally similar and similarly sized globins, α and β, into a tetrameric structural and functional unit. The two dimers, referred to as $\alpha_1\beta_1$ and $\alpha_2\beta_2$, are arranged around a two-fold axis of symmetry, resulting in a large central water cavity in the T (tense) state and a narrower cavity in the R (relaxed) state. The α-cleft and β-cleft serve as the two entry points into this central water cavity. Since the cavity is larger in the T state, these clefts are also larger compared to the R state. Another significant difference is that the T state contains more salt bridges and hydrogen-bond interactions than the R state.

The α and β subunits each comprise 7 and 8 helices (named A through H), connected by non-helical segments (corners). Each subunit contains a binding pocket for heme, formed by the E and F helices. The heme group, shown in Figure 4.5, consists of a ferrous ion coordinated at the center of a porphyrin ring by its four nitrogen atoms. The iron ion is covalently anchored to hemoglobin within the heme proximal pocket by the imidazole group of a histidine residue located on the F helix (known as the proximal histidine or His(F8)).

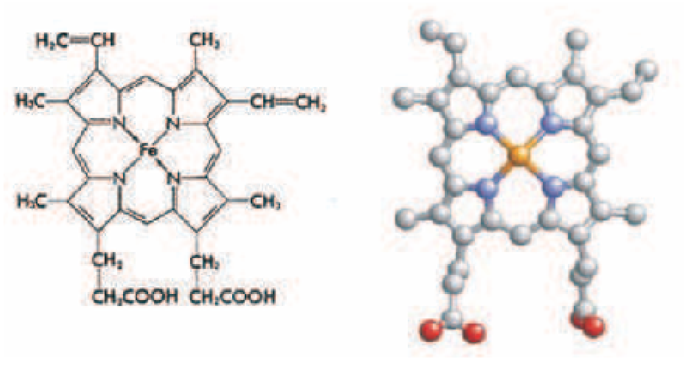


Fig. 4.5. Heme group structure. On the left, the developed chemical formula is represented, while on the right, the structure adopted by heme within the β-chain of hemoglobin is shown. This figure was created using the *RasMol* program and data from Fermi and Perutz (1984). Colors represent elements: yellow for Fe, red for O_2, gray for carbon, and blue for nitrogen [20].

This arrangement allows the iron (Fe) to bind oxygen (O_2) via a covalent bond, completing an octahedral coordination with six ligands. The imidazole of a histidine residue at the distal pocket (His E7) stabilizes the bound oxygen through hydrogen bonding. In deoxygenated hemoglobin, this site is instead occupied by a weakly bound water molecule, forming a distorted octahedron [21].

Hemoglobin's primary function is to transport oxygen from the lungs to the tissues by reversibly binding O_2. It also specifically interacts with other biologically important gases, including carbon monoxide (CO), nitric oxide (NO), and carbon dioxide (CO_2) [22].

The oxygen transport function of hemoglobin can be described in terms of an equilibrium between two states: the tense (T) state and the relaxed (R) state. In the T state, hemoglobin exhibits low oxygen affinity, whereas in the R state, affinity is high.

This equilibrium is modulated by endogenous heterotropic ligands, among which 2,3-bisphosphoglycerate (2,3-BPG) plays a key role.

2,3-BPG is an allosteric effector that modulates hemoglobin's oxygen-binding and releasing properties by binding to the central cavity formed by the spatial arrangement of the four globin subunits. Its binding decreases hemoglobin's oxygen affinity because 2,3-BPG binds exclusively to deoxyhemoglobin [23].

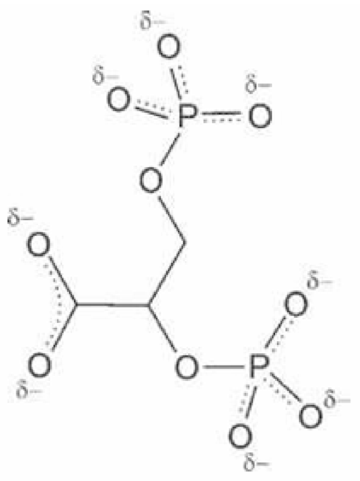


Fig. 4.6. Structure of 2,3-bisphosphoglycerate (2,3-BPG). This compound is characterized by a high negative charge density [20].

Abnormal concentrations of 2,3-BPG are linked to various inherited diseases and altered oxygen transport. For example, increased levels of 2,3-BPG facilitate oxygen release in patients suffering from anemia and hypoxemia. However, elevated 2,3-BPG levels can be detrimental in conditions such as sickle cell anemia, where it promotes the polymerization of sickle hemoglobin and thus contributes to disease pathology [23].

Controlling the physiological concentration of 2,3-BPG is a promising therapeutic target. Consequently, researchers are investigating drugs that modulate its levels. Understanding the three-dimensional structures of hemoglobin and 2,3-BPG is crucial for these efforts.

This is where computational tools that enhance the prediction of macromolecular structures become essential.

In this thesis, the accuracy of such tools in predicting these structures will be analyzed.

For this purpose, the Protein Data Bank will be used to obtain the amino acid sequences of the molecules studied.

The following structures will be analyzed:

1. 1A3N: T state deoxy human hemoglobin [24].
2. 2DN2: R state deoxy human hemoglobin [25].
3. 2DN1: R state oxy human hemoglobin [26].
4. 1B86: Human deoxyhemoglobin–2,3-bisphosphoglycerate complex. [27].

5. METHODOLOGY

With all the necessary materials already introduced, this Methodology section will detail how the plugins were developed and the specific steps followed throughout the integration process.

A visual representation of the steps followed during this methodological process is shown in Figure 5.1, starting from the creation of an empty plugin and culminating with the predicted structure displayed in the *ChimeraX* window.

As explained in Chapter 4, Section 4.1.1, the structure followed by all the integrated programs will be based on the scipion-em-template, which will guide the general structure of all the plugins, although it must be completed with the corresponding commands to enable their correct and specialized functioning.

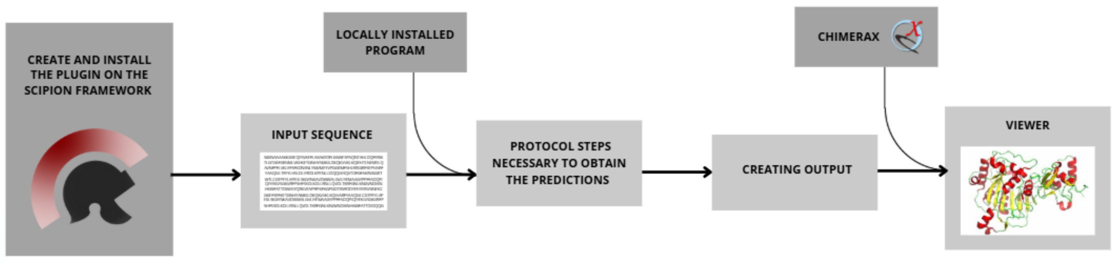

CREATE AND INSTALL THE PLUGIN ON THE SCIPION FRAMEWORK
LOCALLY INSTALLED PROGRAM
INPUT SEQUENCE
PROTOCOL STEPS NECESSARY TO OBTAIN THE PREDICTIONS
CHIMERAX
CREATING OUTPUT
VIEWER

Fig. 5.1. Methodology diagram. Workflow diagram illustrating the integration of a protein structure prediction tool into the *Scipion* framework. The process begins with plugin creation and installation, followed by input sequence processing through a locally installed program. After executing the necessary protocol steps, the output is generated and visualized using *ChimeraX* through a dedicated *Scipion* viewer.

5.1. Creating the Plugin

To create the plugin, the instructions described on GitHub will be followed. To access and edit the plugin files, copy all the files included in the plugin folder to your device.

At this moment, the plugin name is assigned. In this thesis, two plugins are created following the previously described steps, with the names: scipion-em-boltz1 and scipion-em-chai1. The alphafold3 protocol will be included in the scipion-em-chimerax plugin, which already has a protocol named alphafold, including only the *AlphaFold2* functions.

The plugin is already named, however it cannot function only with that, it has to be modified and completed to assign all the parameter inputs, functions, outputs and viewers and to be recognized by *Scipion* it must follow certain rules.

The pyproject.toml file, used to package and distribute Python projects, as mentioned before-file, contains the plugin author names, plugin name, and provide a brief description of the plugin's function in one line. Additionally, the link where the plugin's repository is located is specified.

As explained in Chapter 4, the requirements.txt file establishes the prerequisites for the program. It must include all the necessary programs to be installed via pip.

For the three created plugins, this file is exactly the same. It is only composed of two prerequisites:

- scipion-pyworkflow
- scipion-em

These dependencies ensure that the necessary *Scipion* environment and related libraries are available for the plugins to function properly. When the plugin is installed, these dependencies will be automatically installed by pip to ensure compatibility with the *Scipion* framework.

One of the key parts of this thesis is that every program will be installed locally on the *Scipion* environment. To do this, an automatic installer must be created, together with an environment to install al the dependencies.

5.1.1. Creating an automatic installer

The automatic installer is located in the protocol file, which is the most important file of the plugin, as it contains all the necessary information for the plugin's functionality.

To be more concrete, the installer is in the __init__.py file. This file is essential because it creates the environment and installs all the required software for the plugin to operate. Since two programs will be locally installed, *Boltz-1* and *Chai-1*, each used as their respective plugins, this file plays a central role in ensuring everything is set up correctly.

The first step is to create the environment for these programs. This environment will house all the necessary components and dependencies, ensuring that the plugins function correctly within the overall system. Before creating this environment, some constants must be set.

To ensure reproducibility and traceability, key objectives of this thesis, a separate file named constants.py is created to store several parameters, such as the program name, plugin environment name, and the versions. Two versions will be set, the plugin version (for instance, version 0.0.1 for the initial release; future upgrades will require updating this version) and the version of the program its being installed (which is determined from the repository, and while it is not mandatory to install the latest version, it is recommended).

This approach simplifies the process of updating parameters since any changes can be made in just one place, rather than having to modify every line where the parameter is used. Thus, this file is created before starting to develop the functions for each plugin. To use these constants in the __init__.py file (or any other Python file), they must be imported, along with any libraries or parameters used during the program.

With this in place, the algorithm to install the program can be constructed.

First, the activation of the Conda environment is defined, ensuring that the environment is set up and ready to be used.

Next, set up the environment variables required for the program to run. It ensures that Python virtual environments function properly by removing any existing Python path from the environment. Additionally, it handles GPU settings if a gpuID is provided. This allows for the use of GPU resources, if available, and ensures proper execution on compatible hardware.

Once the environment is configured, the installation command for the program is generated. This function combines the Conda environment activation with the plugin-specific environment activation and adds the program's execution command, formatted correctly for running within the Conda environment. This results in a single command that activates both environments and runs the program. These commands include:

1. Creating a new Conda environment with the specified name and Python version.
2. Installing the program from the Conda repository, using the version specified in the constants file.
3. Marking the installation as complete by creating a file named according to the user's specifications. The file naming convention typically follows the pattern of the plugin name followed by installed (pluginname_installed).

Finally, a function is created to execute the program. This function will be imported into each protocol (or every time it wants to run locally the program) to execute the installed program using the determined inputs.

Having already programmed the installer file for the plugin, additional files must be edited to enable the installation and recognition of the plugin, such as the wizard and protocol folders.

In this thesis, wizards are not created or used, as they are not necessary for the current implementation. However, it is important to create and configure the wizard folder for future development versions. Even though the wizard is not being utilized at this stage, it is crucial for maintaining the structure of the plugin, as future developers may need to use it. Therefore, the wizard folder must be created and assigned a name to it.

Additionally, an __init__.py file is created within this folder to define and name the wizard properly. This file ensures that the wizard is recognized as part of the plugin structure, providing a foundation for future modifications or additions.

In this section, the protocol folder will not be explained in detail, as it consists of several steps that together construct the full protocol. However, to create the plugin and ensure it can be installed within the *Scipion* framework, a name has to be assigned to the protocol and edit the __init__.py file, so that it can be properly recognized. The names assigned to the protocols are Chai1Protocol and BoltzProtocol, chosen according to the program names for ease of recognition.

At this point, *Scipion* is capable of detecting and installing the protocol, but it still needs to be classified under the correct group. As previously mentioned, there are four major groups: Protocol SPA, Random Conical Tilt, Model Building, Virtual Screening, and Tomography. Our three plugins are placed under the Model Building section.

To ensure the protocol appears in the correct section of *Scipion*, the protocols.conf file is modified. In this file, the group name is specified (in this case, Model Building) and the corresponding protocol class. The protocol class must match the one assigned in the protocol's algorithm (for example, Chai1Protocol and BoltzProtocol). This configuration allows *Scipion* to follow the instructions and display the plugin in the appropriate section.

5.2. Creating the Protocol

Although the plugin is already recognized by *Scipion* and can be installed, initially an empty plugin without defined parameters or functions is installed. To complete the plugin, the protocol file must be created and implemented by following the steps described in the subsequent sections.

5.2.1. Protocol Parameters

Defining parameters is a fundamental part of the protocol creation process.

The construction of the parameter commands is based on the information presented in Chapter 4.2.

The *Chai-1* protocol includes two selection tabs to determine the Run Format: the first, named 'Run locally installed *Chai-1*', is selected by default; the second is called 'Import predictions run on *Chai-1* Server'.

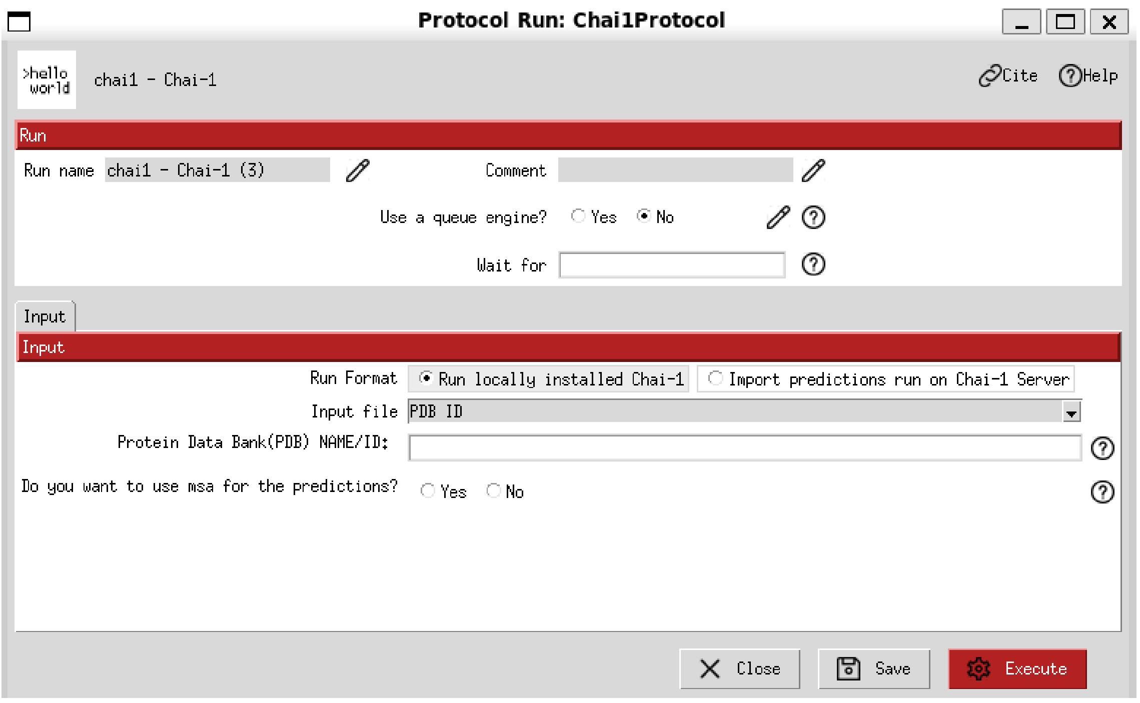
The 'Run locally installed *Chai-1*' option is used when the user wants to input only the amino acid sequence (it can also accept nucleic acid or ligand sequences as input) and receive predicted models as output. All processing occurs within the *Scipion* framework, allowing users to avoid interacting directly with the *Chai-1* server or understanding its internal workings.

As represented in Table 4.1, the input must be either a PDB ID or a FASTA file. In the Input File section, the user can select between 'PDB ID' or 'FASTA FILE', depending on the type of input they want to provide.

The PDB ID entered must be valid and exist in the Protein Data Bank (PDB); otherwise, the protocol will not run.

Regarding FASTA files, *Chai-1* requires a specific format to correctly read the sequence. If the file does not meet these criteria, the user will receive an error. However, the protocol is designed to accept any FASTA file and adapt it into the format required by *Chai-1* , allowing the import of any FASTA sequence as input.

In contrast, the 'Import predictions run on *Chai-1* Server' option accepts only one type of input: a .zip file downloaded from the *Chai-1* server containing precomputed predictions. This parameter is important because researchers may already have predictions generated on the *Chai-1* server but want to import them into the *Scipion* framework to continue their analyses using its capabilities. Providing this option ensures interoperability between the stand alone *Chai-1* 1 server and the integrated *Scipion* environment.


Protocol Run: Chai1Protocol

chai1 - Chai-1

Run

Run name	chai1 - Chai-1 (3)	Comment	
		Use a queue engine?	Yes [✓ No]
		Wait for	

Input

Run Format	[✓ Run locally installed Chai-1] Import predictions run on Chai-1 Server
Input file	PDB ID
Protein Data Bank(PDB) NAME/ID:	
Do you want to use msa for the predictions?	Yes No

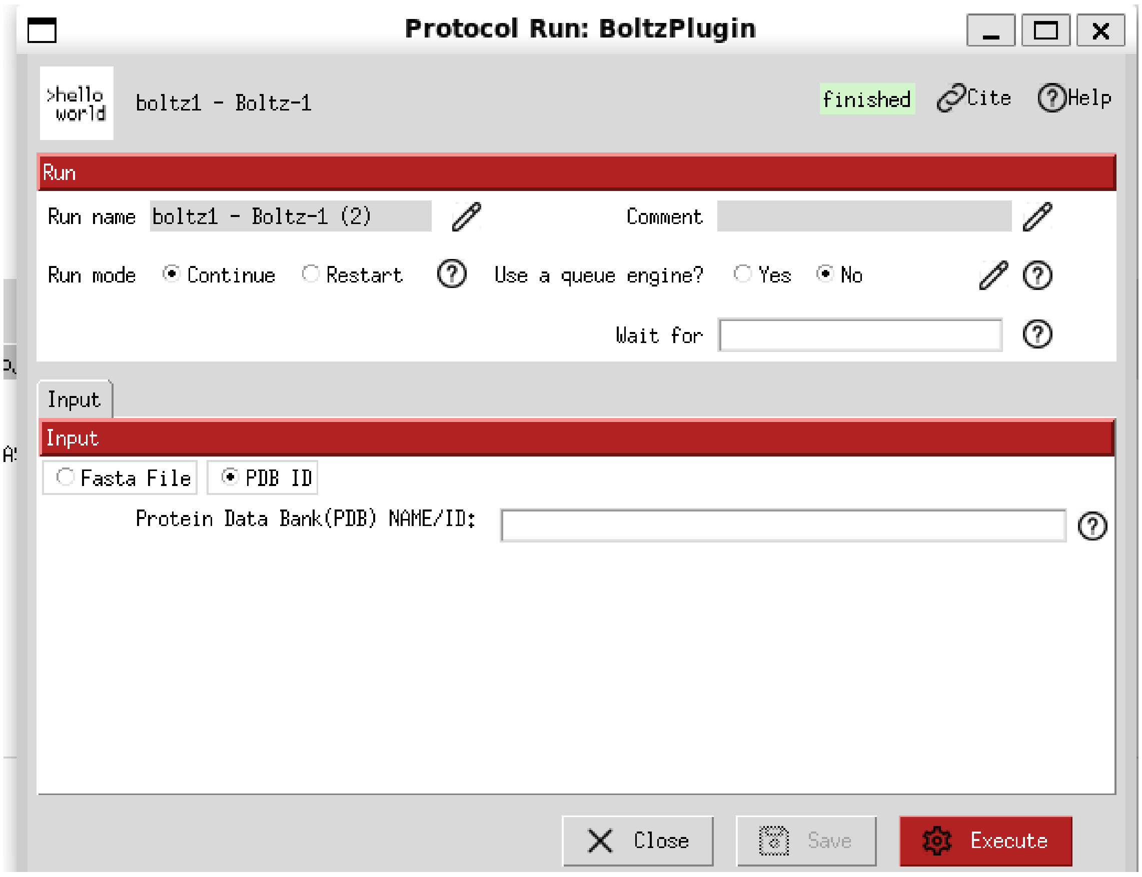
Fig. 5.2. *Chai-1* protocol interface within the *Scipion* framework. The user can select between running a locally installed *Chai-1* instance or importing predictions from the *Chai-1* server. Input options include specifying a PDB ID or uploading a FASTA file. Additional parameters allow users to enable or disable the use of MSASs (multiple sequence alignments) during prediction.

The next plugin integrates *Boltz-1*, which, unlike *Chai-1*, does not provide a web server interface. Instead, it must be installed locally on the user's computer and operated through specific commands, which will be detailed in the following section. Therefore, the input parameters for this plugin are designed exclusively for local execution, without options for remote server use.

Similarly to *Chai-1*, the accepted input files for *Boltz-1* include a valid PDB id or a fasta file. Users can select between these two input types via a dedicated selection tab, depending on their preferred method of providing the sequence.

As with *Chai-1*, the PDB id must correspond to an existing entry in the Protein Data Bank (PDB); otherwise, the protocol will not proceed.

Regarding fasta files, *Boltz-1* requires sequences to be in a specific format for proper processing. If the input file does not meet these criteria, the protocol is designed to pre-process and adapt most fasta files into a compatible format, facilitating their use as input for the program.


Protocol Run: BoltzPlugin

boltz1 - Boltz-1 finished

Run

Run name: boltz1 - Boltz-1 (2)

Comment:

Run mode: [✓ Continue] Restart

Use a queue engine? Yes [✓ No]

Wait for:

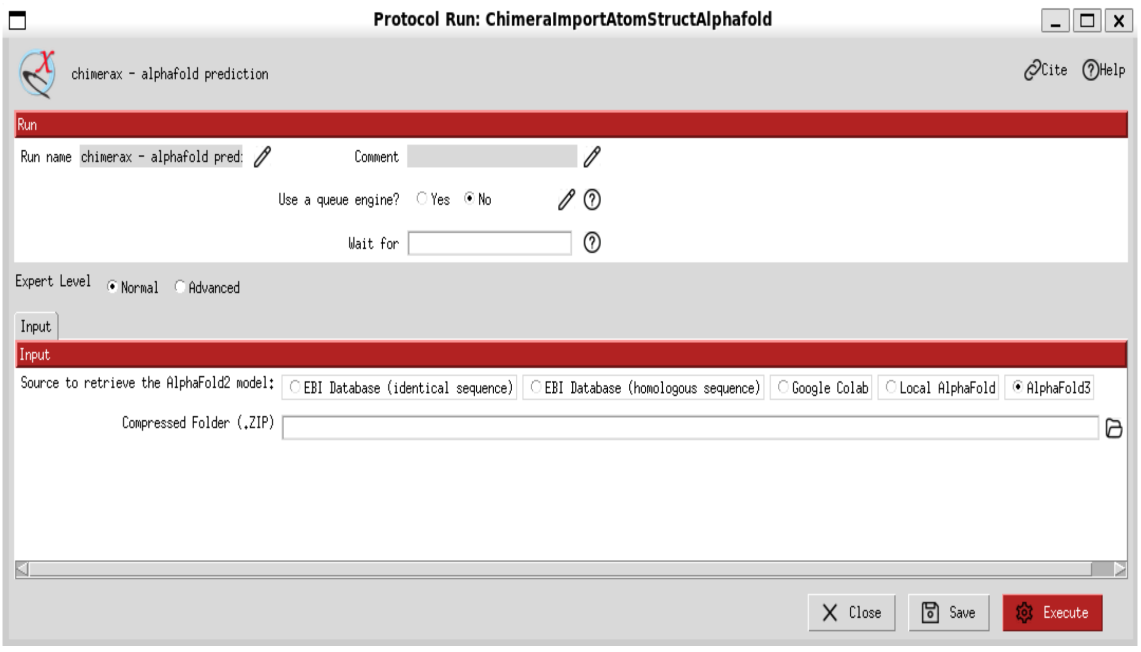
Input

Fasta File [✓ PDB ID]

Protein Data Bank(PDB) NAME/ID:

Fig. 5.3. *Boltz-1* protocol interface integrated into the *Scipion* framework. The interfaceallows the user to select the input format (either FASTA file or a PDB ID) to initiate structure prediction. Additional options include setting the run mode and configuring queue execution settings. Once configured, theprotocol can be launched by clicking the “Execute” button.

Finally, the parameters for *AlphaFold3* are defined. As explained in previous sections, *AlphaFold3* cannot be installed locally, which presents a significant limitation. However, due to its importance in protein structure prediction, it is essential to include it in this thesis. Consequently, the only way to integrate *AlphaFold3* into *Scipion* is by creating a protocol that imports the results folder downloaded from the web server.


Protocol Run: ChimeraImportAtomStructAlphafold

chimerax - alphafold prediction

Run

Run name: chimerax - alphafold pred:

Comment:

Use a queue engine? Yes [✓ No]

Wait for:

Expert Level [✓ Normal] Advanced

Input

Source to retrieve the AlphaFold2 model: EBI Database (identical sequence) EBI Database (homologous sequence) Google Colab Local AlphaFold [✓ AlphaFold3]

Compressed Folder (.ZIP):

Fig. 5.4. *Alphafold3* protocol interface within the modified *ChimeraX* plugin in *Scipion*. Users can specify the input source by selecting a prediction run previously performed on the *AlphaFold3* server and provide the corresponding .zip file for import. The interface enables integration of *AlphaFold3* predictions into the *Scipion* workflow for further visualization and analysis.

5.2.2. Protocol Steps

Once all the necessary parameters for executing the protocol have been set, the sequence of internal steps the protocol will follow must be defined. This section explains how, starting from a simple amino acid sequence or PDB id, a complex and accurate three-dimensional structure is generated.

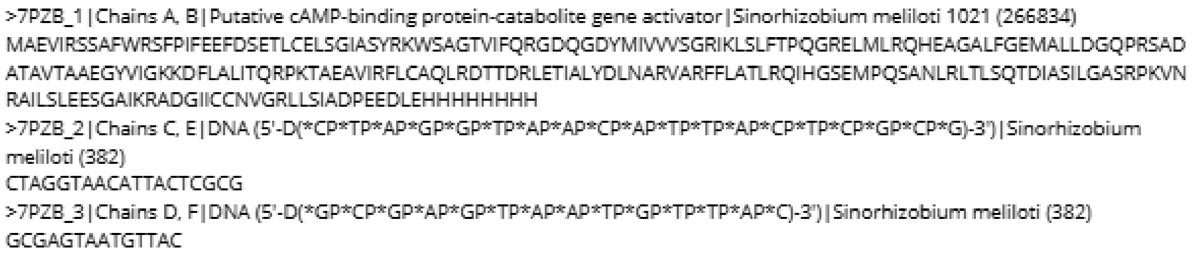
Both *Chai-1* and *Boltz-1* have the same input parameter types, namely 'PDB ID' or 'FASTA file'. Therefore, the initial steps for both programs will overlap. Depending on the input format, the path followed will differ.

If a PDB id is provided, the program will download the corresponding fasta file from the Protein Data Bank (PDB) for that specific ID. The obtained FASTA file will be saved in a temporary directory within the project environment, and the path will be stored in a variable.

In contrast, if the user provides a FASTA file, it will be directly saved in the project directory.

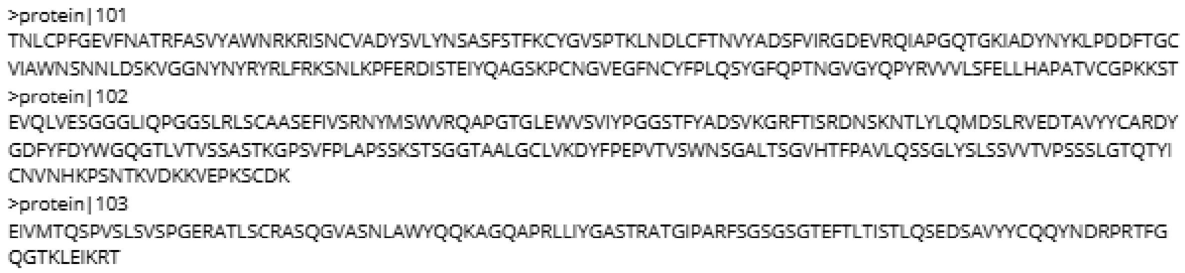
In both cases, a file in FASTA format is obtained and stored in the project environment. As discussed in previous sections, both *Chai-1* and *Boltz-1* are very selective regarding the FASTA file format, and the content must comply with specific rules to be properly read and processed by each program. Therefore, the next step is to read and adapt the file according to the individual requirements of each program.

In the case of *Chai-1*, to fully understand why the FASTA file must be adapted, let us consider an example FASTA file downloaded from the Protein Data Bank (PDB), shown in Figure 5.4, and compare it with the example FASTA format provided in *Chai-1*'s repository, represented in Figure 5.5.



```
>7PZB_1|Chains A, B|Putative cAMP-binding protein-catabolite gene activator|Sinorhizobium meliloti 1021 (266834)
MAEVIRSSAFWRSFPIFEEFDSETLCELSGIASYRKWSAGTVIFQRGDQGDYMIVVVSGRIKLSLFTPQGRELMLRQHEAGALFGEMALLDGQPRSAD
ATAVTAAEGYVIGKKDFLALITQRPKTAEAVIRFLCAQLRDTTDRLETIALYDLNARVARFFLATLRQIHGSEMPQSANLRLTLSQTDIASILGASRPKVN
RAILSLEESGAIKRADGIICCNVGRLLSIADPEEDLEHHHHHHHH
>7PZB_2|Chains C, E|DNA (5'-D(*CP*TP*AP*GP*GP*TP*AP*AP*CP*AP*TP*TP*AP*CP*TP*CP*GP*CP*G)-3')|Sinorhizobium
meliloti (382)
CTAGGTAACATTACTCGCG
>7PZB_3|Chains D, F|DNA (5'-D(*GP*CP*GP*AP*GP*TP*AP*AP*TP*GP*TP*TP*AP*C)-3')|Sinorhizobium meliloti (382)
GCGAGTAATGTTAC
```

Fig. 5.5. 7PZB FASTA file. This is an exmple of a random protein's FASTA file downloaded from PDB.[28]

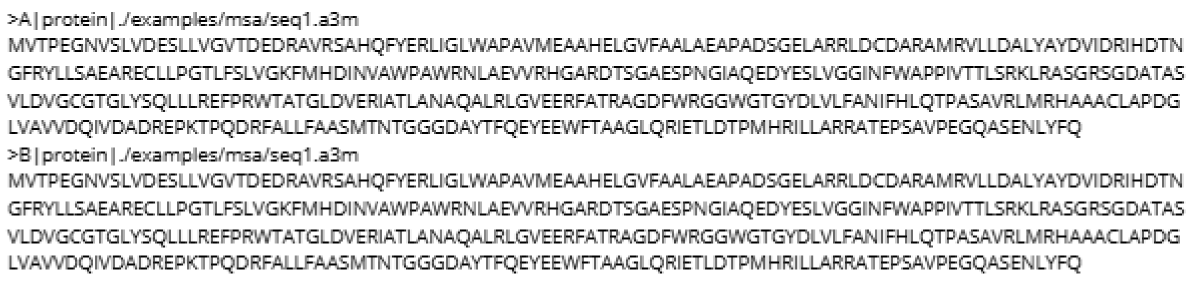


```
>protein|101
TNLCPFGEVFNATRFASVYAWNRKRISNCVADYSVLYNSASFSTFKCYGVSPTKLNDLCFTNVYADSFVIRGDEVRQIAPGQTGKIADYNYKLPDDFTGC
VIAWNSNNLDSKVGGNYNYRYRLFRKSNLKPFERDISTEIYQAGSKPCNGVEGFNCYFPLQSYGFQPTNGVGYQPYRVVVLSFELLHAPATVCGPKKST
>protein|102
EVQLVESGGGLIQPGGSLRLSCAASEFIVSRNYMSWVRQAPGTGLEWVSVIYPGGSTFYADSVKGRFTISRDNSKNTLYLQMDSLRVEDTAVYYCARDY
GDFYFDYWGQGTLVTVSSASTKGPSVFPLAPSSKSTSGGTAALGCLVKDYFPEPVTVSWNSGALTSGVHTFPAVLQSSGLYSLSSVVTVPSSSLGTQTYI
CNVNHKPSNTKVDKKVEPKSCDK
>protein|103
EIVMTQSPVSLSVSPGERATLSCRASQGVASNLAWYQQKAGQAPRLLIYGASTRATGIPARFSGSGSGTEFTLTISTLQSEDSAVYYCQQYNDRPRTFG
QGTKLEIKRT
```

Fig. 5.6. Example *Chai-1*'s FASTA file format. FASTA format example given in the Github repository.[29]

To adapt the file to the correct format, certain information from the header lines must be removed. Only the macromolecule type and its name should remain.

Regarding *Boltz-1*, using the same example macromolecule as in the previous case (Figure 5.4), the fasta file format required is shown in Figure 5.7.



```
>A|protein|./examples/msa/seq1.a3m
MVTPEGNVSLVDESLLVGVTDEDRAVRSAHQFYERLIGLWAPAVMEAAHELGVFAALAEAPADSGELARRLDCDARAMRVLLDALYAYDVIDRIHDTN
GFRYLLSAEARECLLPGTLFSLVGKFMHDINVAWPAWRNLAEVVRHGARDTSGAESPNGIAQEDYESLVGGINFWAPPIVTTLSRKLRASGRSGDATAS
VLDVGCGTGLYSQLLLREFPRWTATGLDVERIATLANAQALRLGVEERFATRAGDFWRGGWGTGYDLVLFANIFHLQTPASAVRLMRHAAACLAPDG
LVAVVDQIVDADREPKTPQDRFALLFAASMTNTGGGDAYTFQEYEEWFTAAGLQRIETLDTPMHRILLARRATEPSAVPEGQASENLYFQ
>B|protein|./examples/msa/seq1.a3m
MVTPEGNVSLVDESLLVGVTDEDRAVRSAHQFYERLIGLWAPAVMEAAHELGVFAALAEAPADSGELARRLDCDARAMRVLLDALYAYDVIDRIHDTN
GFRYLLSAEARECLLPGTLFSLVGKFMHDINVAWPAWRNLAEVVRHGARDTSGAESPNGIAQEDYESLVGGINFWAPPIVTTLSRKLRASGRSGDATAS
VLDVGCGTGLYSQLLLREFPRWTATGLDVERIATLANAQALRLGVEERFATRAGDFWRGGWGTGYDLVLFANIFHLQTPASAVRLMRHAAACLAPDG
LVAVVDQIVDADREPKTPQDRFALLFAASMTNTGGGDAYTFQEYEEWFTAAGLQRIETLDTPMHRILLARRATEPSAVPEGQASENLYFQ
```

Fig. 5.7. Example of the FASTA file format required by *Boltz-1*, as provided in the GitHub repository.[19]

This format is very similar to that of *Chai-1*; however, the header line in *Boltz-1* includes three sections instead of two. The first section contains a letter used to enumerate each molecule, starting from “A”.

Once the file is in the correct format, the predictions can be generated.

During the creation of the automatic installer, it was mentioned that a function to execute the program was also developed. Now, it is time to use this function. This function is coded to open the program environment directly in Conda and execute the specified commands. The commands required to run each program are listed in the respective GitHub repositories.

For *Chai-1*, the command needed to run the program is as follows: *chai-lab fold input.fasta output_folder*.

By default, the model generates five sample predictions and uses embeddings without MSAs or templates.

If the user selects to use MSA in the input window, a different command will be executed. This command, which is recommended for improved performance, is: *chai-lab fold –use-msa-server –use-templates-server input.fasta output_folder*.

In all cases, the command must be adapted to include the corresponding FASTA file (the one already modified to meet the program’s requirements) and specify the output directory where the folder with the predictions will be saved. It is important to determine this output directory, as it will be used for both the output and the viewers. [29]

For *Boltz-1*, the command specified in the repository to perform the predictions is: *boltz predict input_path –use_msa_server*, where input_path refers to the path of the .fasta file. In this case, the output directory is not specified, so it will be saved by default in the protocol’s folder.

In contrast to *Chai-1*, and as shown in the command line to run the program, *Boltz-1* automatically performs the prediction using MSA; there is no option to run it without MSA. [19]

Once this step is completed, the predictions are generated and saved within the *Scipion* environment, specifically in the project directory.

In cases where output predictions obtained from the *Chai-1* or *AlphaFold3* servers are uploaded, the input folder (which must be a .zip file in both cases) is saved in the project directory.

To summarize, regardless of the path followed or the type of input provided, whether a direct amino acid sequence or a predictions folder, the files containing the three-dimensional structure predictions are saved in the appropriate project directory.

In the case of *Chai-1* and *AlphaFold3*, the workflow from this point onward is very similar. Both programs generate five model predictions in .cif format, along with five corresponding .json files (one per predicted model) that contain additional important information about the predictions, such as a ranking score indicating the accuracy of each model.

The protocol selects the five .cif files. One of the objectives of this protocol is to automatically align all the model predictions and produce an output containing the aligned .cif files. To achieve this, the alignment is performed internally using the *ChimeraX* program, and the resulting aligned .cif files are saved as the final prediction outputs.

To carry out this process, command lines are first written into a file with the .cxc extension, which can be read by *ChimeraX*. This file is then executed within *ChimeraX* to generate and save the aligned models.

The .cxc file contains the following commands (where *input_file_path* and *output_file_path* are replaced with the actual paths of the predicted models):

1. *open input_file_path*
2. *matchmaker #2-5 to #1*
3. *save output_file_path*

Once the predicted models are aligned and saved, their file paths are stored in a variable, which is later passed to the output function. This output process will be detailed in the following section.

Additionally, the protocol selects the .json files associated with each predicted model and analyzes their contents to extract the ranking scores. These scores provide quantitative measures of the confidence or accuracy of each prediction. The extracted information is stored in a variable.

To enhance user interaction and facilitate decision-making, a print function is implemented that displays the ranking scores for all predicted models. This feedback allows users to quickly identify the most reliable models based on their confidence levels.

Boltz-1 does not present the output results in the same format as *Chai-1* and *AlphaFold3*. Instead, it consists on a single folder with all the information.

This folder is way more complex and organized than the other two. It consists of a folder named Predictions that contains all the predictions information, together with the predicted model in .cif format. As there is only one predicted model, the alignment process is not necessary, as there are no structures to align it with. So, this .cif file is just saved in a variable to introduce it in the output function.

5.2.3. Protocol Output

Scipion, in the project window interface, has a space to show the output of the protocol to the user.

The output is later the one supposed to be shown when clicking the 'Analyze Results' tab, although this can be modified if correctly programmed.

To assign the .cif files as the output, a function must be created.

This function simply iterates over the resulting structure files and creates corresponding *Scipion* objects that are linked to those files. Each object is then assigned a unique keyword based on the filename, ensuring it can be properly registered within the *Scipion* framework. These keyword-object pairs are collected into a dictionary and passed as keyword arguments to the method self._defineOutputs(). This method is essential, as it informs *Scipion* which objects should be considered outputs of the protocol. Once defined, these outputs become accessible through the interface and can be reused as inputs in subsequent steps of a workflow, ensuring interoperability and enabling researchers to easily switch between programs or reuse the results obtained in one protocol with other *Scipion* functions, a key objective of this thesis.

5.3. Create the Protocol Viewer

This is one of the most essential parts of the protocols developed in this thesis, as the core function of the protocol is to enable the visualization of the three-dimensional predicted structures. The viewer is based on the creation of a *ChimeraX* .cxc file containing the necessary commands for visualization.

Similar to the alignment process using *ChimeraX*, this step will follow a comparable procedure.

The first task is to create the axis around which the structure will be centered. This step is crucial to ensure the structure is correctly positioned when opened in the *ChimeraX* environment. To do so, the axis in three-dimensional space and position the structure at the coordinates (0, 0, 0) are defined.

Next, *ChimeraX* is instructed to open the five .cif files. The alignment method is no longer necessary, as the alignment step has already been completed and the .cif files have been modified to reflect the aligned structures. Therefore, when opening the .cif files, they should appear already aligned.

The final step is to apply color coding, which will help the user visually identify which parts of the predicted structure are most accurately predicted and which parts are less reliable. The viewer will be set up to color the regions that are more consistent across the five models in blue, while regions that exhibit more variation will be colored red. Intermediate colors, such as yellow or orange, will also be assigned to represent the degrees of variation. This color scale will provide a clear visual representation of prediction confidence.

6. RESULTS

After developing all the necessary plugins and successfully installing them within the *Scipion* framework, the next step is to test their functionality.

As described in Chapter 4, the macromolecules used to test these plugins include hemoglobin in both the T and R states, as well as hemoglobin bound to 2,3-BPG. These selections represent a real-world scenario where the prediction of protein structures serves as a powerful research tool.

After running all the plugins within *Scipion*, the results obtained are shown in Figure 6.1.

The results were tested using the PDB ID, rather than FASTA files (as the output would be the same), and in the case of Chai-1, only the local run method was tested, not the web server option.

In the following sections, the results of each plugin will be analyzed in detail, along with a comparison between them using the RMSD concept.

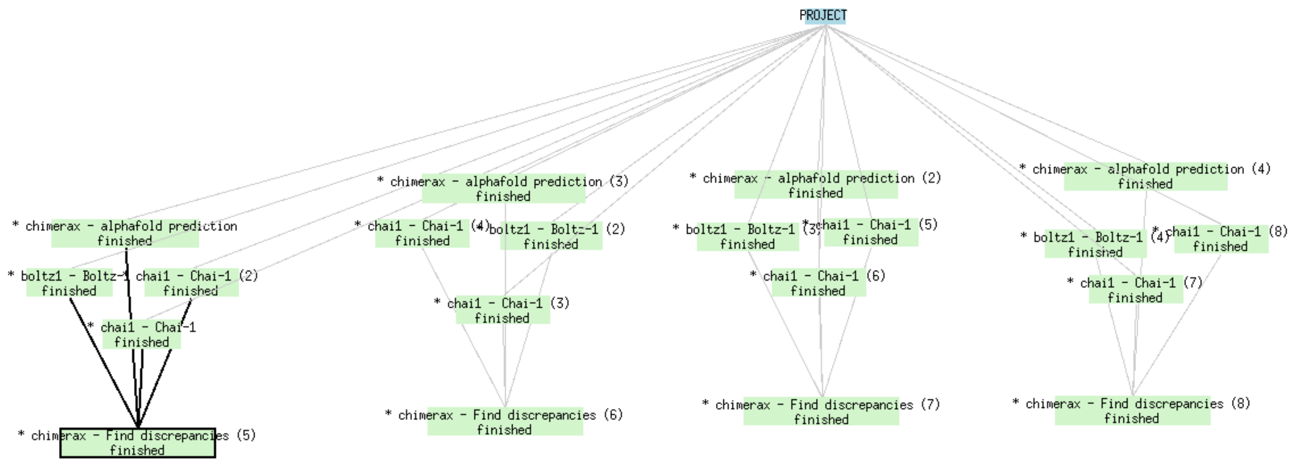

PROJECT
* chimerax - alphafold prediction finished
* boltz1 - Boltz-1 finished
* chai1 - Chai-1 (2) finished
* chai1 - Chai-1 finished
* chimerax - Find discrepancies (5) finished
* chimerax - alphafold prediction (3) finished
* chai1 - Chai-1 (4) finished
* boltz1 - Boltz-1 (2) finished
* chai1 - Chai-1 (3) finished
* chimerax - Find discrepancies (6) finished
* chimerax - alphafold prediction (2) finished
* boltz1 - Boltz-1 (3) finished
* chai1 - Chai-1 (5) finished
* chai1 - Chai-1 (6) finished
* chimerax - Find discrepancies (7) finished
* chimerax - alphafold prediction (4) finished
* boltz1 - Boltz-1 (4) finished
* chai1 - Chai-1 (8) finished
* chai1 - Chai-1 (7) finished
* chimerax - Find discrepancies (8) finished

Fig. 6.1. Final results window . *Scipion* project workflow view showing the complete execution pipeline for protein structure prediction using *AlphaFold3*, Boltz-1, and *Chai-1*. Each green box represents a successfully completed protocol, including sequence input, prediction runs, and subsequent structure alignment using *ChimeraX*. The visual layout demonstrates the interoperability of the integrated tools and the traceability of the analysis steps.

6.1. Chai-1 Plugin

The first test was conducted using the Chai-1 plugin with the T state deoxy human hemoglobin (1A3N). Both methods were tested: one using MSA and the other without MSA, to evaluate how the accuracy changes with the use of this tool.

Five model predictions were obtained for each protocol (one protocol using MSA and the other without it).

The ranking scores of each predicted model are shown in Tables 6.1 - 6.4.

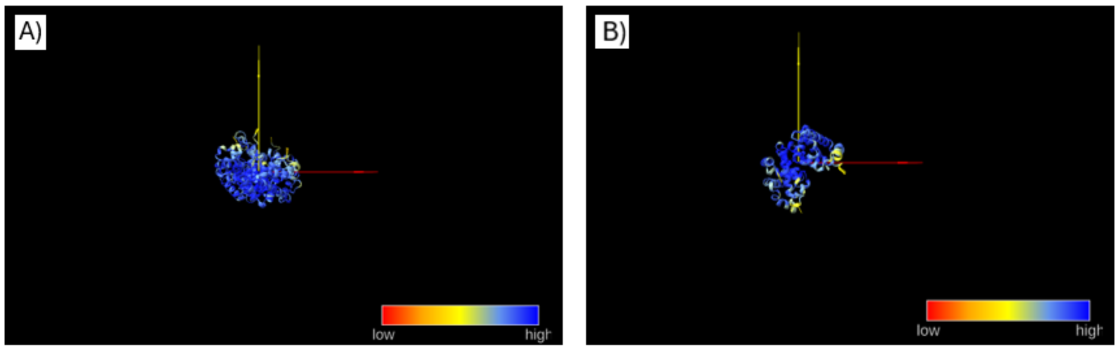


Fig. 6.2. *Chai-1* predicted 1A3N structure. A) Structure computed without using MSA. B) Structure computed using MSA.

TABLA 6.1. RANKING SCORES OF *CHAI-1* PREDICTED MODELS 1A3N

Input Parameter	**With MSA**	**Without MSA**
Model 0	0.9220	0.9056
Model 1	0.9212	0.9038
Model 2	0.9209	0.9042
Model 3	0.9210	0.9043
Model 4	0.9207	0.9043

Then with R state deoxy human hemoglobin (2DN2)

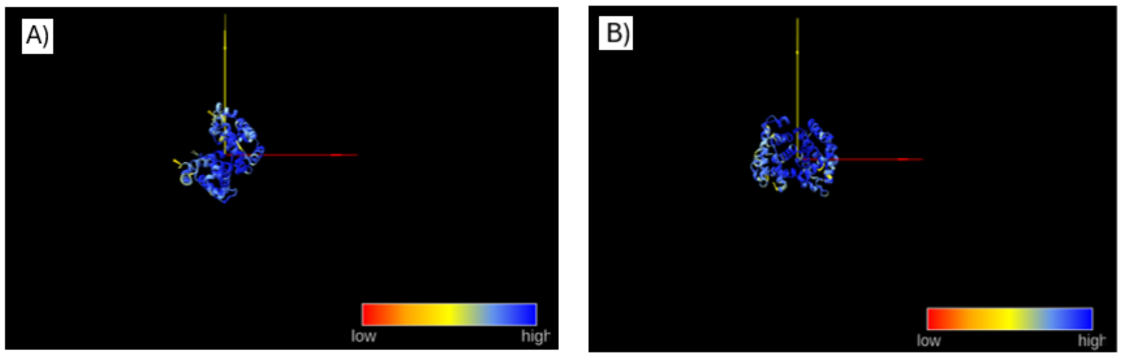


Fig. 6.3. *Chai-1* predicted 2DN2 structure. A) Structure computed without using MSA. B) Structure computed using MSA.

TABLA 6.2. RANKING SCORES OF *CHAI-1* PREDICTED MODELS
2DN2

Input Parameter	**With MSA**	**Without MSA**
Model 0	0.9207	0.9061
Model 1	0.9214	0.9045
Model 2	0.9225	0.9040
Model 3	0.9220	0.9045
Model 4	0.9214	0.9044

Then with R state oxy human hemoglobin (2DN1)

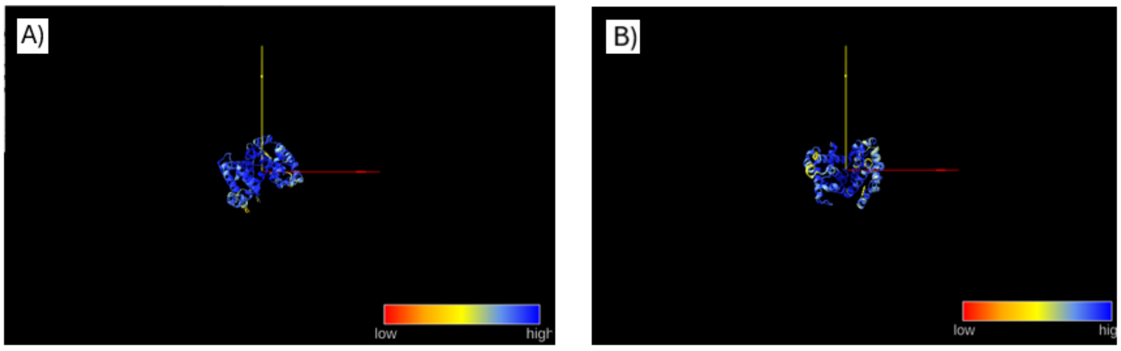


Fig. 6.4. *Chai-1* predicted 2DN1 structure. A) Structure computed without using MSA. B) Structure computed using MSA.

TABLA 6.3. RANKING SCORES OF *CHAI-1* PREDICTED MODELS
2DN1

Input Parameter	**With MSA**	**Without MSA**
Model 0	0.9215	0.9037
Model 1	0.9212	0.9047
Model 2	0.9215	0.9037
Model 3	0.9220	0.9043
Model 4	0.9210	0.9040

Finally, with Human deoxyhemoglobin–2,3-bisphosphoglycerate complex (1B86).

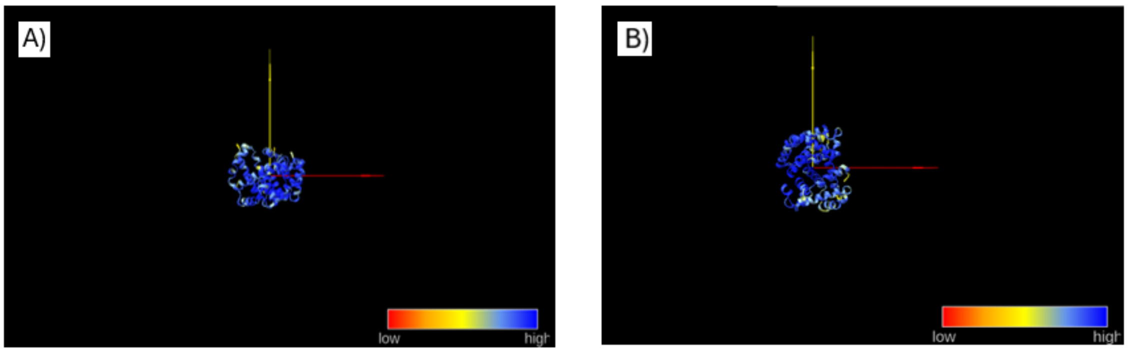


Fig. 6.5. *Chai-1* predicted 1B86 structure. A) Structure computed without using MSA. B) Structure computed using MSA.

TABLA 6.4. RANKING SCORES OF *CHAI-1* PREDICTED MODELS 1B86

Input Parameter	**With MSA**	**Without MSA**
Model 0	0.9221	0.9044
Model 1	0.9220	0.954
Model 2	0.9223	0.9038
Model 3	0.9222	0.9055
Model 4	0.9218	0.9046

As shown in the images and explained in the Methodology chapter, the viewer window displays the structures already aligned and color-coded based on the similarity of the predictions.

Blue-colored regions of the structures indicate high similarity across the five models, suggesting that these parts are more accurate and closely resemble the true structure. Conversely, red and orange-colored regions represent areas with greater variability among the models, implying that these parts are less accurately predicted, as they differ in their structural conformation across the models.

6.2. AlphaFold3 Plugin

AlphaFold3 generates five predicted models, similar to the *Chai-1* plugin; however, unlike *Chai-1*, no MSA is used to perform the predictions in this program.

Each predicted structure and model of the macromolecules is accompanied by its ranking score information, which can be found in Tables 6.5 - 6.8.

The structures visualized in the *ChimeraX* framework are displayed alongside their respective tables, which include the ranking score information.

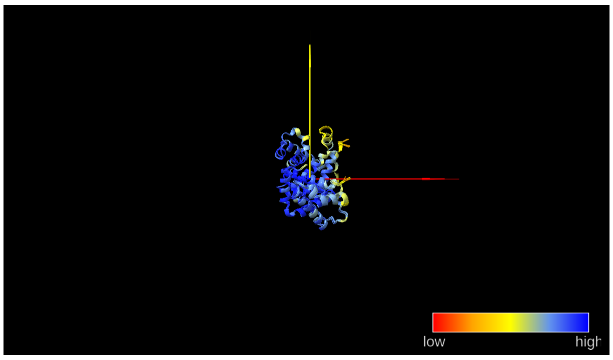


Fig. 6.6. *Alphafold3* predicted 1A3N structure.

TABLA 6.5. RANKING SCORES OF *ALPHAFOLD3* PREDICTED MODELS 1A3N

Input Parameter	**Ranking Score**
Model 0	0.91
Model 1	0.91
Model 2	0.91
Model 3	0.91
Model 4	0.90

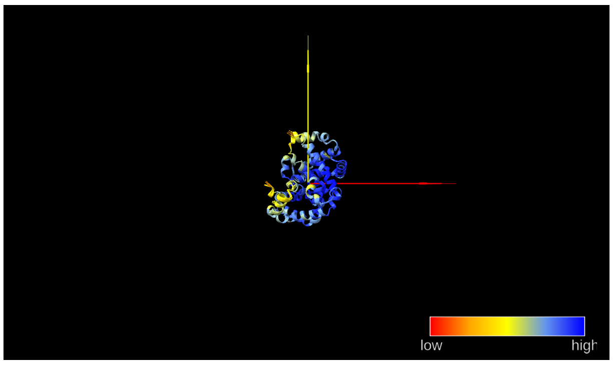


Fig. 6.7. *Alphafold3* predicted 2DN2 structure.

TABLA 6.6. RANKING SCORES OF *ALPHAFOLD3* PREDICTED MODELS 2DN2

Input Parameter	**Ranking Score**
Model 0	0.91
Model 1	0.91
Model 2	0.91
Model 3	0.91
Model 4	0.90

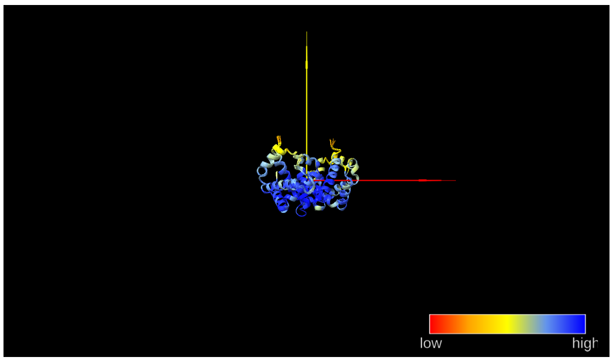


Fig. 6.8. *Alphafold3* predicted 2DN1 structure.

TABLA 6.7. RANKING SCORES OF *ALPHAFOLD3* PREDICTED MODELS 2DN1

Input Parameter	**Ranking Score**
Model 0	0.91
Model 1	0.91
Model 2	0.91
Model 3	0.91
Model 4	0.90

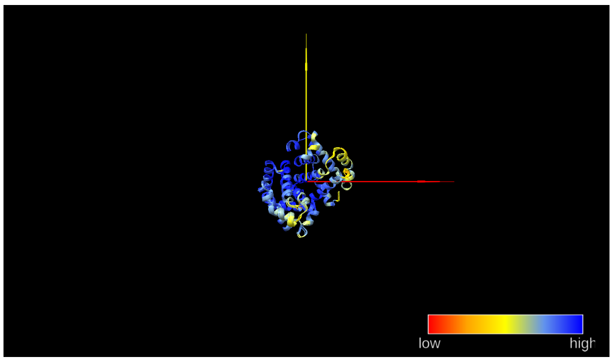


Fig. 6.9. *Alphafold3* predicted 1B86 structure.

TABLA 6.8. RANKING SCORES OF *ALPHAFOLD3* PREDICTED MODELS 1B86

Input Parameter	**Ranking Score**
Model 0	0.91
Model 1	0.91
Model 2	0.91
Model 3	0.91
Model 4	0.90

Same as what happened with *Chai-1* results the viewer window displays the structures already aligned and color-coded based on the similarity of the predictions.

Blue-colored regions of the structures indicate high similarity across the five models and red or orange-colored regions represent areas with greater variability among the models.

6.3. Boltz-1 Plugin

Boltz-1 produces a single predicted model as output, which is generated using MSA data. Therefore, it is not possible to compare multiple model predictions, as only one model is provided.

Additionally, unlike the other programs, *Boltz-1* does not output a ranking score for the predicted structure.

The results obtained from this program are visualized in the *ChimeraX* framework and will be analyzed accordingly.

The predicted structures obtained are displayed in Figure 6.5, Figure 6.6, Figure 6.7, and Figure 6.8.

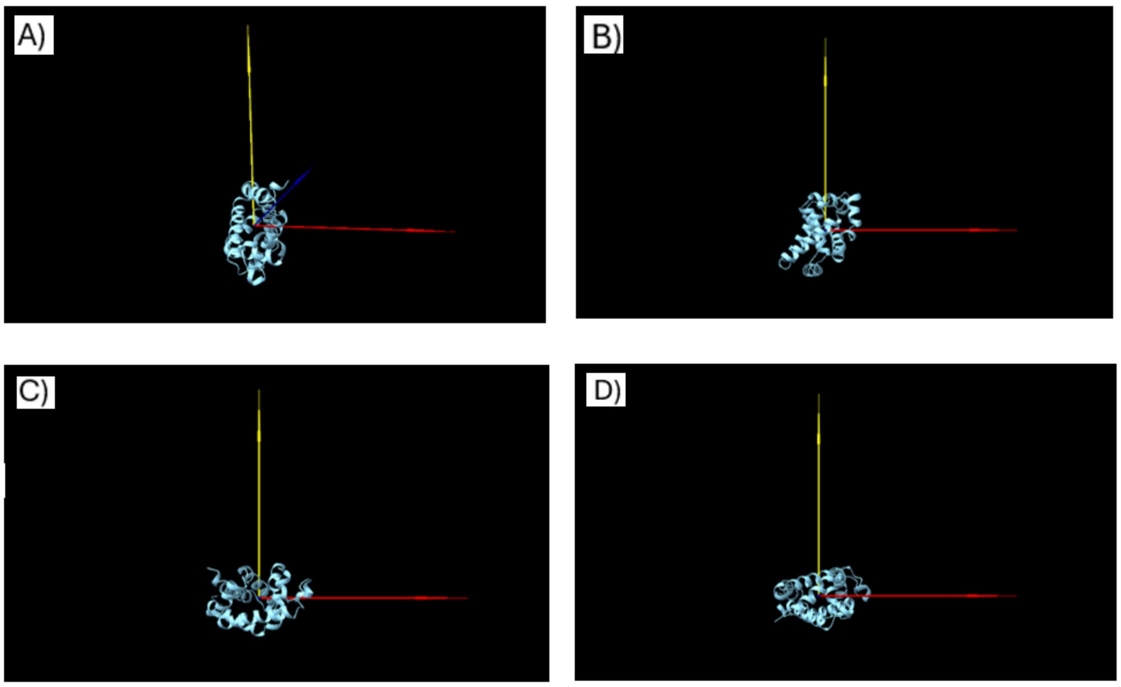


Fig. 6.10. *Boltz-1* predicted structures. A) *Boltz-1* predicted 1A3N structure. B) *Boltz-1* predicted 2DN2 structure. C) *Boltz-1* predicted 2DN1 structure. D) *Boltz-1* predicted 1B86 structure.

Unlike *AlphaFold3* and *Chai-1*, *Boltz-1* does not align multiple models, as it only generates a single prediction. As a result, the color information in the image generated in

ChimeraX does not provide any comparative data, since no other models are available for comparison.

6.4. Comparison Between Integrated Software using RMSD.

Root Mean Square Deviation (RMSD) is one of the most common methods to determine the accuracy of algorithms in charge of predicting structures.

There are two ways of calculating RMSD, absolute RMSD and relative RMSD. Absolute RMSD, measures the distance between corresponding atom pairs of two conformers without coordinate translation or rotation and is used mainly for docking evaluations. Relative RMSD implies an additional alignment step of the molecules before the actual RMSD calculation and is the one used to determine the accuracy of conformational model generators, which is this thesis case [30].

A protocol already implemented in *Scipion* is used. These protocols have to be run once per macromolecule, so in this thesis case, four times. Each run protocol contains all the predicted structures created with the three programs for each macromolecule.

The output results obtained are the files containing all the RMSD value information per atom of all the predicted models of each molecule, aligned.

With this information, three graphs per molecule are created to compare the software.

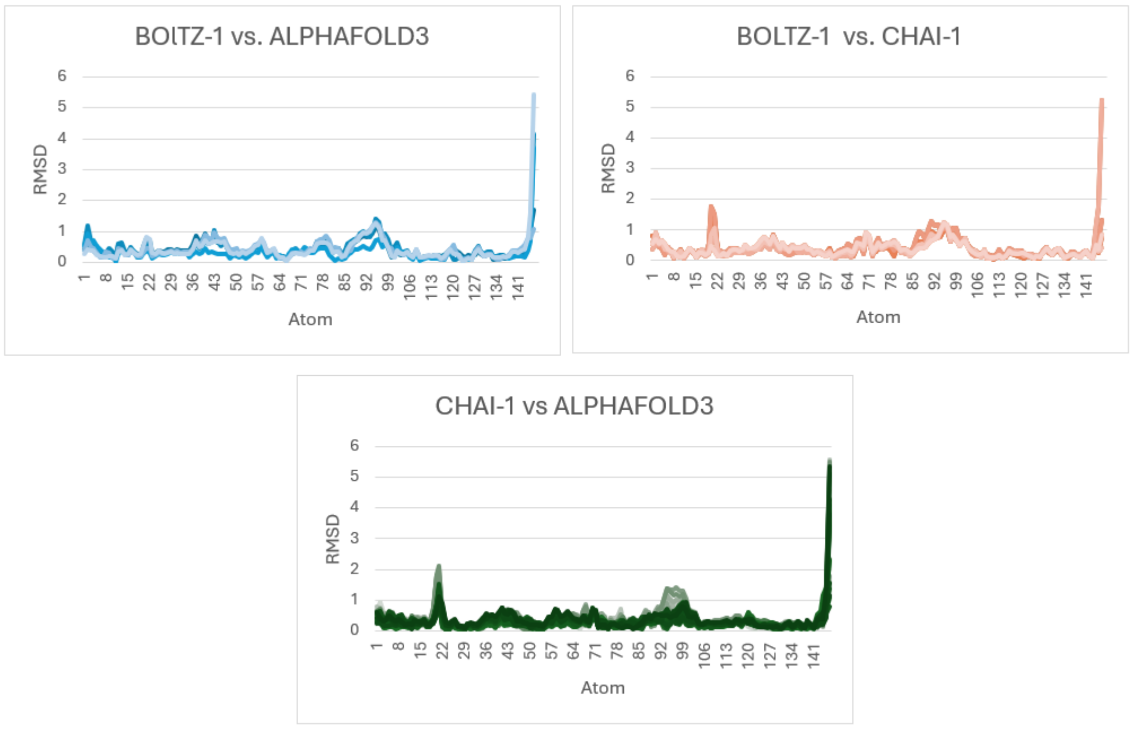


Fig. 6.11. 1A3N RMSD comparison between software graphs.

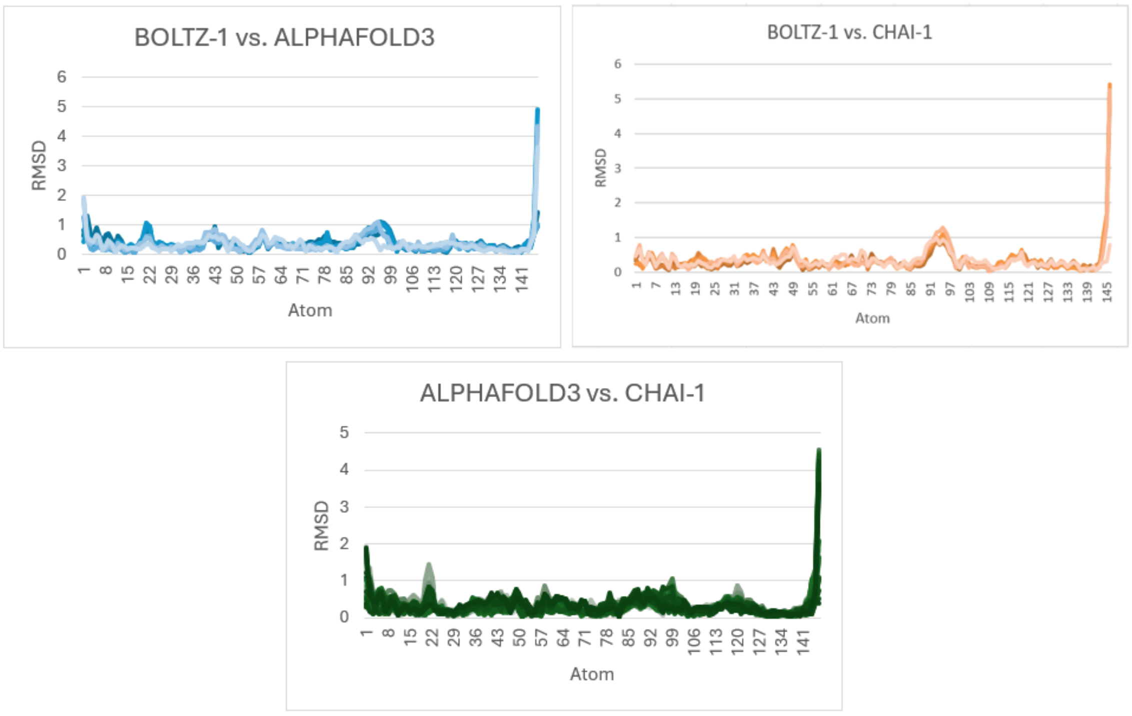


Fig. 6.12. 2DN2 RMSD comparison between software graphs.

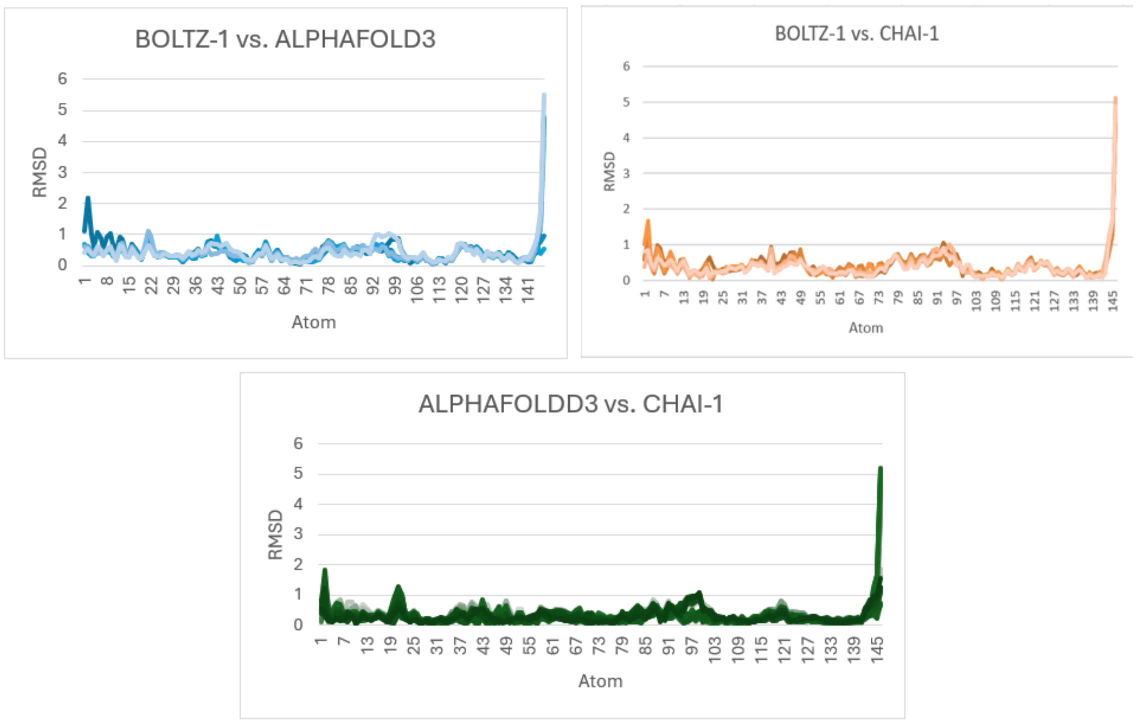


Fig. 6.13. 2DN1 RMSD comparison between software graphs.

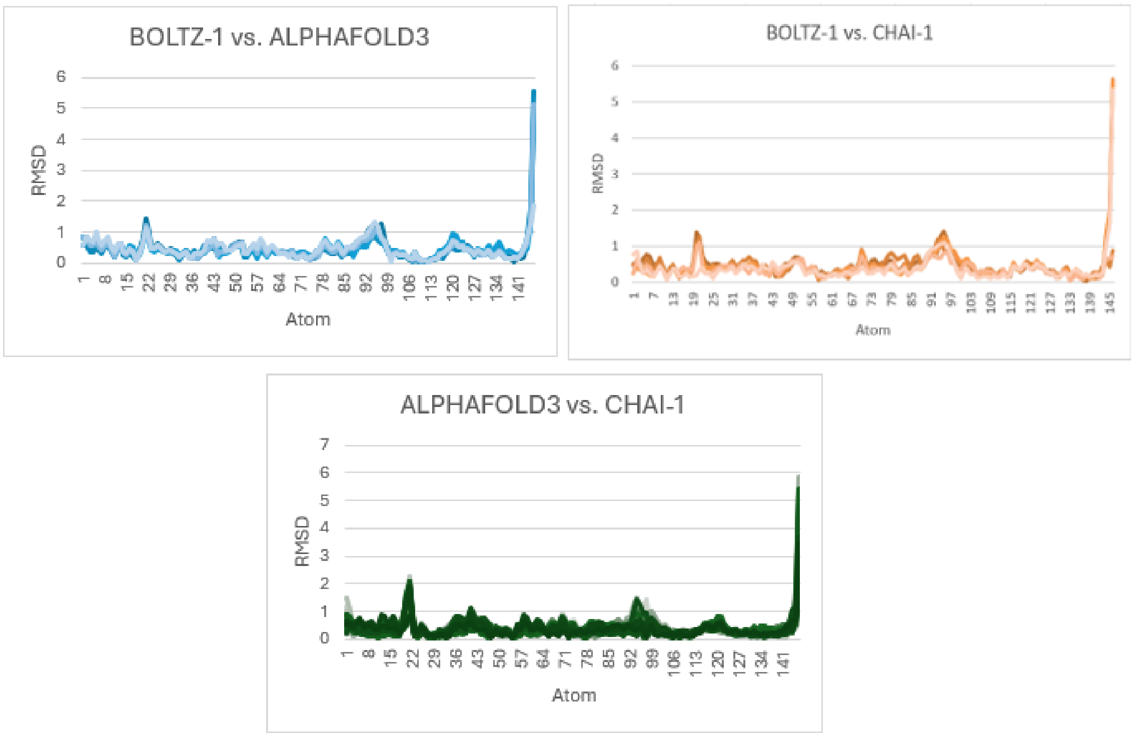


Fig. 6.14. 1B86 RMSD comparison between software graphs.

7. DISCUSSION AND FUTURE OUTLOOK

After computing the predicted structures for all the macromolecules and using the three integrated programs, the results obtained provide valuable information regarding their accuracy. However, to gain a deeper understanding and determine the best program, the RMSD (Root Mean Square Deviation) comparison method is applied.

AlphaFold3 and *Chai-1* offer more detailed information, as they both allow for the extraction of ranking scores for each predicted model, along with a color-coded visualization.

For both programs, most regions of the predicted macromolecule structures are primarily blue or yellow, with very few red areas. This is a positive indication of prediction accuracy, as it suggests that most parts of the five predicted models are similar. In turn, this high degree of similarity increases the likelihood that the structures are predicted correctly.

Regarding the ranking scores, they are generally very high, around 90 %.

However, in the case of *AlphaFold3*, it is observed that the scores are identical for all predictions and models. This could be due to the similarity between the predicted structures, as all models correspond to hemoglobin in different states or when bound to a ligand.

In the case of *Chai-1*, as expected, the ranking scores obtained using MSA are higher, indicating that predictions made with MSA provide more accurate results compared to those made without it.

On the other hand, the *Boltz-1* plugin provides more limited information for each prediction compared to the other two programs. While the core function, structure prediction, is performed, the lack of additional data limits its overall utility for researchers. More information would enhance the tool's capability and assist in providing deeper insights.

However, the ranking scores alone are not sufficient to determine the most accurate predictive program. To achieve this, the RMSD comparison method was employed.

In general, lower RMSD values are associated with protein structure pairs that have better resolution. The RMSD values tend to increase when comparing proteins refined at different resolutions [31]. This means that higher RMSD values, as seen in the graphs, indicate worse alignment, with predicted structures being more distant from one another.

Upon analyzing the resulting graphs, no clear distinction emerges between the programs to decisively determine the best alignment. A common feature across all the graphs is the high RMSD values observed in the last ten atoms. This region corresponds to the C-terminal end, and, along with the N-terminal region (corresponding to the first few atoms), it tends to be more disorganized and structurally flexible than the central region.

It is encouraging that most of the central regions present RMSD values between 2 and 0, with some occasional peaks in specific regions. This suggests that the three programs generated highly accurate predictions overall.

However, in all three cases, the data comparing *AlphaFold3* and *Chai-1* models show better results compared to comparisons with *Boltz-1*.

In the case of the 2DN2 molecule graph, this is the graph that reaches the lowest RMSD values, not exceeding a value of 5, which is not observed in any of the other graphs. Moreover, despite some occasional peaks, it is the graph that maintains values close to 0 for most of the atoms, indicating less variability overall.

Based on these findings, we can conclude that *Boltz-1* is the least accurate of the three programs, as it produces structures that are more distant from the other predictions. Nevertheless, it is important to note that the results from all three programs were highly accurate, with no significant discrepancies, and all produced consistent and reliable predictions.

This thesis highlights the importance of accurate predictive biocomputational tools and the need to improve their accessibility for researchers. However, this is just the beginning, marking a starting point for future advancements.

The application of bioinformatics to biology has reached a pivotal moment, greatly facilitating the entire research process.

In the case of protein structure prediction, future approaches will expand to include the prediction of protein-protein interactions, protein-macromolecule interactions (such as those with lipids, nucleic acids, or polysaccharides), and protein-small molecule interactions.

Moreover, the accuracy of these predictions will continue to improve over time, with model architectures being refined to enable predictions that are even closer to real-life scenarios. As technology advances and more data becomes available, these tools will become essential for advancing not only structural biology but also drug discovery and other fields within the life sciences.

Not only will the existing programs continue to improve, but new ones are already emerging. Programs such as *ABCFold* and *ProteniX* are now available to further advance the field. *ABCFold* simplifies the use of *AlphaFold3*, *Boltz-1*, and *Chai-1* by providing a standardized input format for predicting atomic structures, with *Boltz-1* and *Chai-1* being installed during runtime [32]. *ProteniX* is a comprehensive implementation of *AlphaFold3*, designed to push forward the field of biomolecular structure prediction [33]. This programs, could also be implemented on *Scipion*, to enable researchers use their functionalities.

8. CONCLUSIONS

This thesis has addressed the integration of cutting-edge protein structure prediction tools, *AlphaFold3*, *Chai-1*, and *Boltz-1*, into the *Scipion* framework, a platform known for its interoperability, reproducibility, and traceability. The work involved the development of two plugins from scratch and the modification of an existing one, ensuring that each integrated program could be executed within *Scipion* with clear, adaptable, and user-friendly protocols.

Through the development of automated installers, parameterized protocols, and *ChimeraX*-based specialized viewers, integration was not only technically successful but also functionally robust. The output of each tool could be visualized and analyzed within the same environment, facilitating comparative studies between predictive models.

The experimental validation of these integrations, using four different macromolecules, confirmed the correct operation of the plugins and highlighted their predictive capabilities. *Chai-1* and *AlphaFold3* were particularly effective, offering multiple model predictions and confidence scores that allowed for a deeper understanding of the variability and reliability of the results. *Boltz-1*, while functional and accurate, provided fewer outputs and lacked scoring metrics, limiting its comparative analysis.

A detailed RMSD-based comparison showed that all three tools performed well, with *AlphaFold3* and *Chai-1* producing more consistent and closely aligned structures. These findings validate their potential for reliable and reproducible protein modeling.

Ultimately, this thesis demonstrates not only the feasibility, but also the value of expanding *Scipion* capabilities in the domain of protein structure prediction. It sets the foundation for future developments, including the integration of additional models such as *ABCFold* or *ProteniX*, and reinforces the critical role of bioinformatics in the acceleration of biomedical research. Integrating such tools into accessible platforms like *Scipion* can democratize structural biology, streamline workflows, and enhance research impact in both academic and clinical settings.

9. REGULATORY FRAMEWORK

The techniques introduced along this work are not regulated by any legislation or subjected to any intellectual property and they do not break any code of professional ethics. However, several packages and programming languages were used for the development of the aforementioned techniques with their own regulations.

Scipion is under the GNU General Public License, which is a free, copyleft license for software. Moreover, *Scipion* releases are distributed for Linux OS (Operative System) which is also an open source software under the GNU license.

In the case of the programming languages used for the development of the scripts, Python is open source, although the IDE (Integrated Development Environment) used may be subject to regulatory frameworks. The IDE used is *PyCharm*, which has its own agreement that approves its usage for academic research.

The molecules used during Chapter 6 were obtained from the Protein Data Bank (PDB). PDB is an open-access repository governed by Worldwide Protein Data Bank (wwPDB), data is freely available for academic research but it must always be properly cited.

Regarding visualization tools, *ChimeraX* is licenced for non-commercial uses only, such as academic or research ones. The *Scipion* framework incorporates *ChimeraX* as an external tool, which means *ChimeraX* is independently installed and governed by its own academic license.

Regarding the three softwares implementations, *Alphafold3*, *Chai-1* and *Boltz-1*, as happened with *ChimeraX*, are incorporated as external tools to the *Scipion* framework.

10. SOCIO-ECONOMIC IMPACT

The computational prediction of macromolecule structures has a profound impact on both industry and research. It enables a better understanding of the human body and biological processes, which in turn enhances the pharmaceutical industry, drug discovery, and scientific advancement.

In the case of drug discovery, survey data from 1983 to 2000 estimated that manufacturers' costs for launching a new drug were approximately \$802 million. More recent estimates (1997–2001) suggest that the cost is closer to \$1.7 billion [34]. This process is divided into several phases, and understanding protein structures is crucial in the early stages.

Since most drug targets are proteins, it is essential to know their three-dimensional structures in detail.

Traditionally, solving protein structures was a process that involved experimental techniques such as X-ray crystallography, nuclear magnetic resonance (NMR), and, more recently, cryo-electron microscopy. These techniques are highly effective but are labor-intensive and time-consuming [35].

The use of these time-consuming techniques results in long research periods, leading to significantly higher research costs.

However, with the advent of biocomputational tools that can predict protein structures within hours or even minutes, the process of drug discovery, such as finding potential drugs that inhibit a specific protein, has been greatly accelerated. As a result, research costs are often reduced.

Not only are costs reduced, but the efficiency and speed of these investigations allow for the discovery and release of treatments to the market more quickly. This accelerated timeline is particularly crucial in cases like the rapid development of the COVID-19 vaccine [36], where fast action was needed to address a global health crisis.

Despite the advantages, it is important to note that these biocomputational tools, like all artificially created or computationally generated outputs, should always be experimentally validated and monitored by human experts.

These tools are designed to enhance and support researchers, but they should never replace the human aspect of research. Experimental research remains crucial, although these computational tools can serve as valuable complements, providing insights and accelerating the discovery process.

It is crucial to exercise caution when interpreting predicted structures, keeping in mind that they may not be perfectly accurate. Traditional experimental techniques involve direct testing of the protein to determine its structure, a step that does not occur in computational predictions. Therefore, while computational tools offer valuable insights, they should be used as a complement to, rather than a replacement for, experimental methods.

Project Budget

The costs associated with creating and using the model are divided into two categories: human resources and technical support. The first category, human resources, includes the salaries of the personnel involved in the project. The second category, technical support, encompasses all the necessary equipment for developing and using the project.

The costs for human resources are detailed in Table 10.1, which includes the salaries of the CNB and UC3M tutors, as well as the student researcher.

Position	**Wage/hour**	**Working time**	**Cost**
CSIC tutor	50 €	60 h	3,000 €
UC3M tutor	40 €	20 h	800 €
student researcher	15 €	450 h	6,750 €
Total cost			**10,550 €**

TABLA 10.1. HUMAN RESOURCES COST BREAKDOWN

For technical costs, shown in Table 10.2, this includes the student's personal computer, along with the CNB computer equipped with a GPU, which was used to run the programs and obtain the results presented in the Results section.

Equipment	**Price**	**Usage time**	**Amortization**	**Cost**
CNB server virtual machine	5,997.87 €	6 months	36 months	999.65€
Student computer	900 €	6 months	36 months	150€
Total cost				**1,149.65 €**

TABLA 10.2. TECHNICAL EQUIPMENT COST BREAKDOWN

The total budget required for the project is summarized in Table 10.3.

Type of cost	**Cost**
Human resources	10,550 €
Technical equipment	1,149.65 €
Total cost	**11,699.65 €**

TABLA 10.3. TOTAL COST

BIBLIOGRAFÍA

[1] Academia Lab. “Bioinformática estructural.” Revisado el 15 de mayo del 2025, Enciclopedia. (2025), [En línea]. Disponible en: https://academia-lab.com/enciclopedia/bioinformatica-estructural/.

[2] T. Ogunjobi et al., “Bioinformatics tools in protein analysis: Structure prediction, interaction modelling, and function relationship,” *European Journal of Sustainable Development Research*, vol. 8, n.º 1, 2024.

[3] J. P. Hughes, S. Rees, S. B. Kalindjian y K. L. Philpott, “Principles of early drug discovery,” *British Journal of Pharmacology*, vol. 162, n.º 6, pp. 1239-1249, 2011.

[4] P. Conesa et al., “Scipion3: A workflow engine for cryo-electron microscopy image processing and structural biology,” *Biological Imaging*, vol. 3, e13, 2023.

[5] H. Deng, Y. Jia e Y. Zhang, “Protein structure prediction,” *International Journal of Modern Physics B*, vol. 32, n.º 18, p. 1 840 009, 2018.

[6] C. Kamble, R. Chavan y V. Kamble, “A Review on Amino Acids,” *STM Journals*, vol. 8, p. 2021, ene. de 2022.

[7] S. Aryal. “Amino Acids - Properties, Structure, Classification, Functions.” (jul. de 2022).

[8] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts y P. Walter, *Molecular Biology of the Cell*, 4th. New York: Garland Science, 2002, cap. The Shape and Structure of Proteins, Available from: NCBI Bookshelf.

[9] D. Whitford, *Proteins: structure and function*. John Wiley & Sons, 2013.

[10] V. Vyas, R. Ukawala, M. Ghate y C. Chintha, “Homology modeling a fast tool for drug discovery: current perspectives,” *Indian journal of pharmaceutical sciences*, vol. 74, n.º 1, p. 1, 2012.

[11] N. Fatima, S. Khan y S. Zahid, “A critical address to advancements and challenges in computational strategies for structural prediction of protein in recent past,” *Computational Biology and Chemistry*, p. 108 430, 2025.

[12] L. A. Abriata, “The Nobel Prize in Chemistry: past, present, and future of AI in biology,” *Communications Biology*, vol. 7, n.º 1, p. 1409, 2024.

[13] J. Jumper et al., “Highly accurate protein structure prediction with AlphaFold,” *nature*, vol. 596, n.º 7873, pp. 583-589, 2021.

[14] European Bioinformatics Institute (EMBL-EBI), *Strengths and limitations of AlphaFold*, https://www.ebi.ac.uk/training/online/courses/alphafold/an-introductory-guide-to-its-strengths-and-limitations/strengths-and-limitations-of-alphafold/, 2024.

[15] J. Abramson et al., “Accurate structure prediction of biomolecular interactions with AlphaFold 3,” *Nature*, vol. 630, n.º 8016, pp. 493-500, 2024.

[16] N. Editorial, “AlphaFold3 improves protein structure prediction,” *Nature*, vol. 629, p. 728, mayo de 2024.

[17] F. Hoffmann, *AlphaFold3 and its improvements in comparison to AlphaFold2*, https://medium.com/@falk_hoffmann/alphafold3-and-its-improvements-in-comparison-to-alphafold2-96815ffbb044, 2024.

[18] Chai Assets Team, “Chai Technical Report Version 1,” Chai Assets, inf. téc., sep. de 2024.

[19] J. Wohlwend et al., “Boltz-1: Democratizing Biomolecular Interaction Modeling,” *bioRxiv*, 2024. DOI: 10.1101/2024.11.19.624167.

[20] L. F. VERA, “La hemoglobina: una molécula prodigiosa,” *Revista de la Real Academia de Ciencias Exactas Físicas y Naturales*, vol. 104, n.º 1, pp. 213-232, 2010.

[21] M. H. Ahmed, M. S. Ghatge y M. K. Safo, “Hemoglobin: structure, function and allostery,” *Vertebrate and invertebrate respiratory proteins, lipoproteins and other body fluid proteins*, pp. 345-382, 2020.

[22] A. N. Schechter, “Hemoglobin research and the origins of molecular medicine,” *Blood, The Journal of the American Society of Hematology*, vol. 112, n.º 10, pp. 3927-3938, 2008.

[23] Z. Zhong y E. V. Anslyn, “Controlling the Oxygenation Level of Hemoglobin by Using a Synthetic Receptor for 2, 3-Bisphosphoglycerate,” *Angewandte Chemie International Edition*, vol. 42, n.º 26, pp. 3005-3008, 2003.

[24] J. Tame y B. Vallone, *Deoxy human hemoglobin*, Protein Data Bank entry 1A3N, 1998.

[25] S.-Y. Park, T. Yokoyama, N. Shibayama, Y. Shiro y J. Tame, *1.25 Å resolution crystal structures of human haemoglobin in the oxy, deoxy and carbonmonoxy forms*, Protein Data Bank entry 2DN2, 2006.

[26] S.-Y. Park, T. Yokoyama, N. Shibayama, Y. Shiro y J. R. Tame, *1.25 Å resolution crystal structure of human hemoglobin in the oxy form*, Protein Data Bank entry 2DN1, 2006.

[27] V. Richard, G. G. Dodson e Y. Mauguen, *Human deoxyhaemoglobin–2,3-diphosphoglycerate complex*, Protein Data Bank entry 1B86, 1999.

[28] L. Werel y L.-O. Essen, *Structure of the Clr-cAMP-DNA complex*, Protein Data Bank entry 7PZB, 2022.

[29] Chai Discovery, “Chai-1: Decoding the molecular interactions of life,” *bioRxiv*, 2024. DOI: 10.1101/2024.10.10.615955. eprint: https://www.biorxiv.org/content/early/2024/10/11/2024.10.10.615955.full.pdf. [En línea]. Disponible en: https://www.biorxiv.org/content/early/2024/10/11/2024.10.10.615955.

[30] J. Kirchmair, P. Markt, S. Distinto, G. Wolber y T. Langer, “Evaluation of the performance of 3D virtual screening protocols: RMSD comparisons, enrichment assessments, and decoy selection—what can we learn from earlier mistakes?” *Journal of computer-aided molecular design*, vol. 22, pp. 213-228, 2008.

[31] O. Carugo, “How root-mean-square distance (rmsd) values depend on the resolution of protein structures that are compared,” *Applied Crystallography*, vol. 36, n.º 1, pp. 125-128, 2003.

[32] L. G. Elliott, A. J. Simpkin y D. J. Rigden, “ABCFold: easier running and comparison of AlphaFold 3, Boltz-1 and Chai-1,” *bioRxiv*, pp. 2025-03, 2025.

[33] B. A. A. Team et al., “Protenix-advancing structure prediction through a comprehensive AlphaFold3 reproduction,” *bioRxiv*, pp. 2025-01, 2025.

[34] J. Sollano, J. Kirsch, M. Bala, M. Chambers y L. Harpole, “The economics of drug discovery and the ultimate valuation of pharmacotherapies in the marketplace,” *Clinical Pharmacology & Therapeutics*, vol. 84, n.º 2, pp. 263-266, 2008.

[35] R. P. Montfort, “Sobre el premio nobel de química 2024 en diseño y estructura de las proteínas,” *Revista de Educación Bioquímica*, vol. 43, n.º 4, pp. 210-212, 2025.

[36] V. C. Osamor, E. Ikeakanam, J. U. Bishung, T. N. Abiodun y R. H. Ekpo, “COVID-19 vaccines: computational tools and development,” *Informatics in Medicine Unlocked*, vol. 37, p. 101 164, 2023.

DECLARATION OF USE OF GENERATIVE IA IN BACHELOR THESIS (TFG)

I have used Generative AI in this work

Check all that apply:

[✓ **YES**]	**NO**

If you have ticked YES, please complete the following 3 parts of this document:

Part 1: Reflection on ethical and responsible behaviour

Please be aware that the use of Generative AI carries some risks and may generate a series of consequences that affect the moral integrity of your performance with it. Therefore, we ask you to answer the following questions honestly (*please tick all that apply*):

Question		
1. In my interaction with Generative AI tools, I have submitted **sensitive data** with the consent of the data subjects.		
YES, I have used this data with permission	NO, I have used this data without authorisation	[✓ NO, I have not used sensitive data]
2. In my interaction with Generative AI tools, I have submitted **copyrighted materials** with the permission of those concerned.		
YES, I have used these materials with permission	NO, I have used these materials without permission	[✓ NO, I have not used protected materials]
3. In my interaction with Generative AI tools, I have submitted **personal data** with the consent of the data subjects.		
YES, I have used this data with permission	NO, I have used this data without authorisation	[✓ NO, I have not used personal data]

4. My use of the Generative AI tool has **respected its terms of use**, as well as the essential ethical principles, not being maliciously oriented to obtain an inappropriate result for the work presented, that is to say, one that produces an impression or knowledge contrary to the reality of the results obtained, that supplants my own work or that could harm people.	
[✓ YES]	NO

If you **did NOT** have the permission of those concerned in any of questions 1, 2 or 3, briefly explain why *(e.g. "the materials were protected but permitted use for this purpose" or "the terms of use, which can be found at this address (...), prevent the use I have made, but it was essential given the nature of the work".*

Part 2: Declaration of technical use

Use the following model statement as many times as necessary, in order to reflect all types of iteration you have had with Generative AI tools. Include one example for each type of use where indicated: *[Add an example].*

I declare that I have made use of the Generative AI system (Name of AI system/tool and version: ChatGPT, Gemini, Copilot...) **for:**

Documentation and drafting:

- *Revision or rewriting of previously drafted paragraphs*

I have requested corrections for texts I have written, to ensure they were gramatically correct.

I have also sought help from AI to paraphrase certain phrases that were very repetitive or that I didn't know how to write correctly.

Develop specific content

Generative AI has been used as a support tool for the development of the specific content of the dissertation, including:

- *Assistance in the development of lines of code (programming)*

I have asked for help with programming in Python, mainly regarding the syntax, as I was taught to program in Matlab, and there are some syntax differences between both languages.

Part 3: Reflection on utility

Please provide a personal assessment (free format) of the strengths and weaknesses you have identified in the use of Generative AI tools in the development of your work. Mention if it has helped you in the learning process, or in the development or drawing conclusions from your work.

The use of generative AI has allowed me to ensure that the text was well-written and free of grammatical errors, facilitating the writing process and improving the flow of the text.

However, generative AI, although highly efficient in improving the writing, cannot replace the critical thinking and creativity required to develop the main arguments or interpret the data and results of the research. Throughout the work, I made sure to maintain strict control over the direction and content of the text, and the AI was used only as a complementary tool to refine style and formatting.