ChatMol Copilot: An Agent for Molecular Modeling and Computation Powered by LLMs

Jinyuan Sun[1], Auston Li[1,2], Yifan Deng[1], Jiabo Li[1,2]

[1]ChatMol Team [2]Wecomput Technology Co., Ltd.

Correspondence: jinyuansun@chatmol.org; jiaboli@chatmol.org

Abstract

Large Language Models (LLMs) like ChatGPT excel at diverse tasks when given explicit instructions, yet they often struggle with specialized domains such as molecular science, lacking in-depth reasoning and sophisticated planning capabilities. To address these limitations, we introduce ChatMol Copilot, a chatbot-like agent specifically engineered for protein design and small molecule computations. ChatMol Copilot employs a multi-level abstraction framework to expand the LLM's capability. At the basic level, it integrates external computational tools through function calls, thus offloading complex tasks and enabling a focus on strategic decision-making. The second level is data abstraction. Large data sets (such as a large number of molecules created by a generative model) are stored in Redis cache, and the redis keys are referenced by LLMs for data sources involved in computation. The third level of abstraction allows the LLM to orchestrate these tools, either directly or via dynamically generated Python executables. Our evaluations demonstrate that ChatMol Copilot can adeptly manage molecular modeling tasks, effectively utilizing a variety of tools as directed. By simplifying access to sophisticated molecular modeling resources, ChatMol Copilot stands to significantly accelerate drug discovery and biotechnological innovation, empowering biochemists with advanced, user-friendly AI capabilities. The open-sourced code is available at https://github.com/ChatMol/ChatMol

1 Introduction

Large Language Models (LLMs) equipped with specialized tools are catalyzing significant advancements across various scientific fields. In chemistry research, platforms like Coscientist (Boiko et al., 2023) and Chemcrow (M. Bran et al., 2024) have revolutionized lab automation and computational tasks. Furthermore, the novel CodeAct approach, which utilizes "Code as Action," leverages the coding prowess of LLMs to automate complex processes (Wang et al., 2024). Similarly, tools such as AlphaFold 3 (Abramson et al., 2024) have achieved remarkable success in predicting protein interactions and structures, underscoring the potential of computational methods in molecular biology.

Despite these strides, significant challenges persist in the molecular engineering field, particularly regarding the execution of complex modeling tasks and the interpretation of their outcomes (Greener et al., 2022). These challenges stem from a need for greater automation and a more intuitive interaction with computational tools. In response, we introduce ChatMol Copilot, a dedicated platform that enhances molecular modeling computations. ChatMol Copilot is designed with a multi-level abstraction framework to maximize automation and user-friendliness. At its foundation, it integrates external computational tools via function calls, simplifying the interface to show only inputs, outputs, and functional descriptions, thereby isolating the LLM from complex computational details. The advanced layer of this framework allows the LLM to either orchestrate these tools directly or through dynamically generated Python executables. This paper demonstrates how ChatMol Copilot effectively manages molecular modeling tasks and delivers precise, actionable responses to user inquiries, significantly streamlining the computational workflow in molecular science.

2 ChatMol Copilot Architecture

The ChatMol Copilot is designed around the capabilities of Large Language Models (LLMs). The system architecture aims to optimize workflow efficiency and precision in molecular modeling tasks.

Workflow Overview (Figure 1): The process begins with user instructions, which are interpreted by the LLM. This interpretation step determines

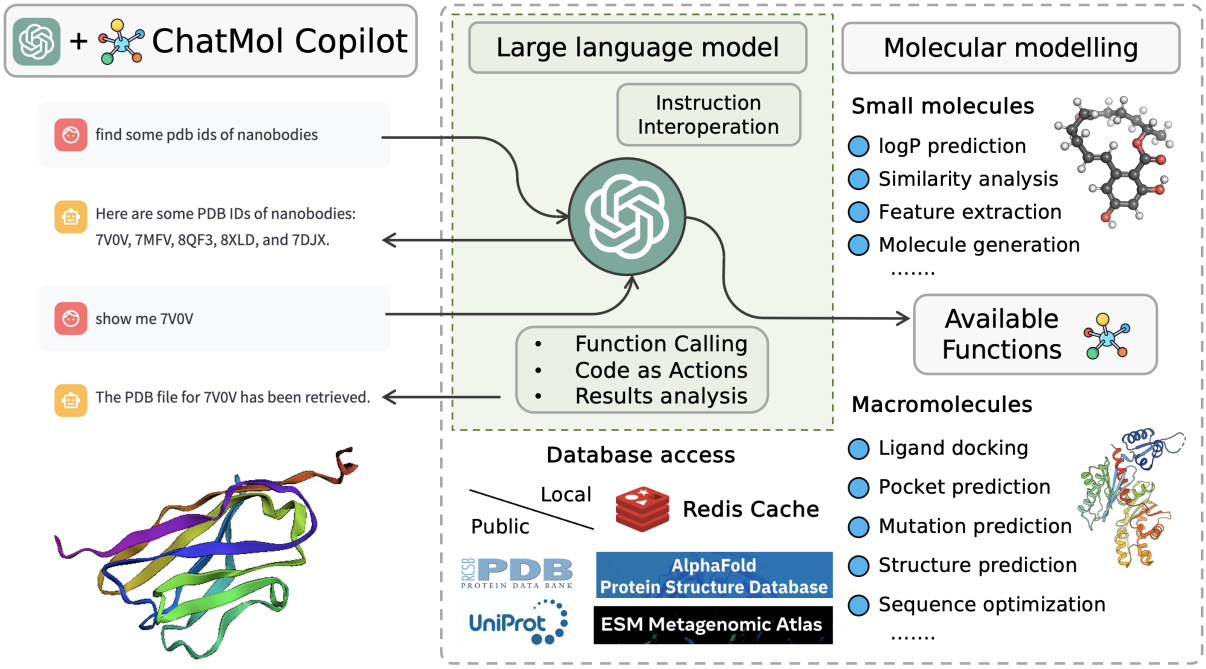


Figure 1: The general workflow of ChatMol Copilot.

whether the user's request can be directly answered or if it necessitates the use of specialized tools. ChatMol Copilot supports a broad range of molecular modeling applications, encompassing both small molecules (such as pharmaceuticals) and macromolecules (such as proteins and their interactions). In addition to pre-defined tools, the system can create new tools by writing and executing Python code. This significantly expands its capabilities beyond the predefined action space. Data abstraction is employed to alleviate the burden of data processing from LLMs. Besides internal data usage, the system has access to significant biological databases, enabling it to retrieve and utilize publicly available data as needed. As conversations progress, the integration of user demands with the computational capabilities of ChatMol Copilot facilitates the completion of increasingly complex tasks.

2.1 Equipped Tools

Numerous specialized neural networks have been developed for various molecular property prediction tasks (Wu et al., 2018). ChatMol Copilot integrates a range of tools to meet diverse computational needs, enabling comprehensive and efficient analysis.

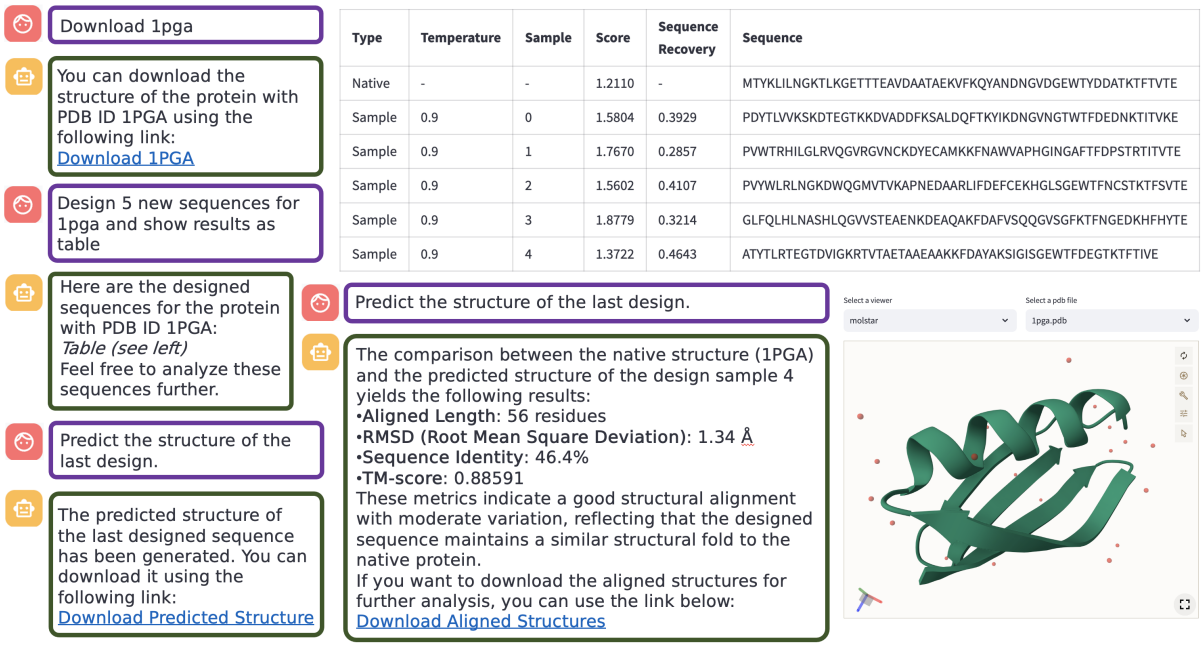
Neural Network-Based Tools: For tools that utilize neural network inference, such as ESMFold (Lin et al., 2023) and ProteinMPNN (Dauparas et al., 2022), we have implemented publicly accessible APIs. This ensures minimal hardware requirements for users.

Local Execution Tools: For faster, Python-based tools, execution is handled locally on the user's computer. Examples include RDKit (Landrum et al., 2013) and TM-align (Zhang and Skolnick, 2005). Table 1 lists the primary tools integrated into the system for both small molecule and macromolecule analysis.

2.2 Integration with Microservices

Microservices are a staple in modern cloud computing architectures due to their scalability and modularity. Each microservice operates independently with a well-defined API and service description. In the ChatMol framework, we have developed a generic method for integrating these microservices into the ChatMol toolbox see Table 1.

For each microservice in our registry, Python code is generated based on the input parameter descriptions. This code is then wrapped into a standard function call, compiled on-the-fly, and added to our function calling list. As new microservices are registered, the function list is automatically updated, greatly enhancing the toolbox's capabilities while simplifying ongoing maintenance.



Type	Temperature	Sample	Score	Sequence Recovery	Sequence
Native	-	-	1.2110	-	MTYKLILNGKTLKGETTTEAVDAATAEKVFKQYANDNGVDGEWTYDDATKTFTVTE
Sample	0.9	0	1.5804	0.3929	PDYTLVVKSKDTEGTKKDVADDFKSALDQFTKYIKDNGVNGTWTFDEDNKTITVKE
Sample	0.9	1	1.7670	0.2857	PVWTRHILGLRVQGVRGVNCKDYECAMKKFNAWVAPHGINGAFTFDPSTRTITVTE
Sample	0.9	2	1.5602	0.4107	PVYWLRLNGKDWQGMVTVKAPNEDAARLIFDEFCEKHGLSGEWTFNCSTKTFSVTE
Sample	0.9	3	1.8779	0.3214	GLFQLHLNASHLQGVVSTEAENKDEAQAKFDAFVSQQGVSGFKTFNGEDKHFHYTE
Sample	0.9	4	1.3722	0.4643	ATYTLRTEGTDVIGKRTVTAETAAEAAKKFDAYAKSIGISGEWTFDEGTKTFTIVE

Figure 2: General protein design task. Text within purple boxes are instructions from users, and text within green boxes are answers given by ChatMol Copilot. Texts colored blue with underlines are hyperlinks for download files. The table and cartoon represented protein are real screen shots from the GUI of ChatMol Copilot.

2.3 Code as Actions and Redis Cache

Expanding the system's capabilities can be achieved through the automatic generation and execution of code, known as CodeAct. In ChatMol Copilot, we implemented a generic Python code executor and a universal data object access mechanism using Redis cache. Code generation can be based on task descriptions and knowledge from documents. Redis cache and generic data object read/write operations in code enabling the LLM to manage tasks and data flow much simpler by referencing data with their keys.

3 Use Cases of ChatMol Copilot for Molecular Modeling

This section showcases four examples demonstrating the wide-ranging capabilities of ChatMol Copilot in molecular modeling tasks. These use cases illustrate how ChatMol Copilot adheres to user instructions, utilizing appropriate tools from its equipped toolkit and microservices to meet the demands of biochemists, from protein modeling to small molecule *de novo* synthesis.

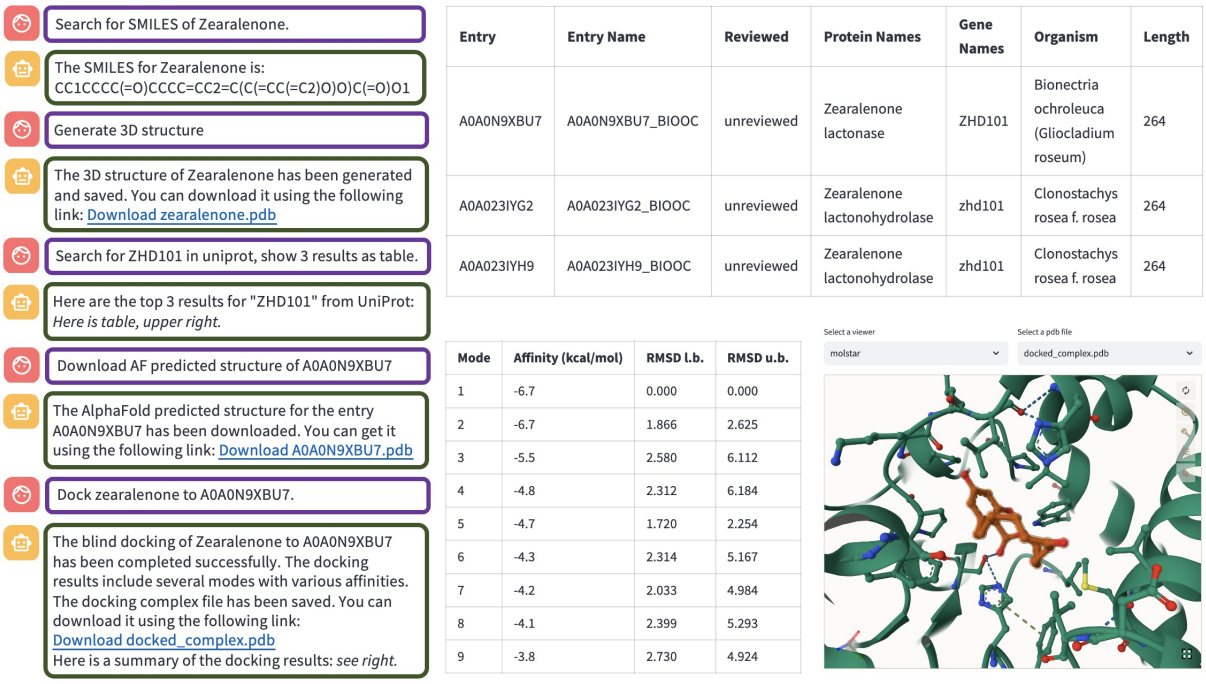
3.1 General Protein Design Task

Proteins, essential macromolecules in cells, perform various biological functions, including DNA duplication, metabolic reaction catalysis, and cell cycle regulation. They are also pivotal in healthcare as therapeutic agents like insulin and antibodies, and in various industries as catalysts for cleaner energy and chemicals (Huang et al., 2016).

A fundamental challenge in protein design is to find a sequence that folds into a desired structure (Dauparas et al., 2022). This task is complicated by epistasis, where residue-residue interactions can lead to misfolding and loss of function. To address this, we utilized ProteinMPNN (Dauparas et al., 2022) for sequence design and ESMFold (Lin et al., 2023) to predict the fold of the designed sequences. The effectiveness of the designs was validated by comparing their structures to the initial templates, with ChatMol Copilot presenting the results in a well-organized table format. Key metrics such as Root Mean Squared Error (RMSD) and TM-score are highlighted to assess the structural integrity of the designed sequences Figure 2.

3.2 Peptide/MHC-II Binding Affinity Prediction

The prediction of binding affinity between peptides and MHC-II complexes is critical for assessing the immunogenic potential of newly designed proteins (Jensen et al., 2018). In this use case, ChatMol Copilot was tasked with mutating a peptide sequence five times randomly, then calculating and tabulating the binding affinities of these variants



Entry	Entry Name	Reviewed	Protein Names	Gene Names	Organism	Length
A0A0N9XBU7	A0A0N9XBU7_BIOOC	unreviewed	Zearalenone lactonase	ZHD101	Bionectria ochroleuca (Gliocladium roseum)	264
A0A023IYG2	A0A023IYG2_BIOOC	unreviewed	Zearalenone lactonohydrolase	zhd101	Clonostachys rosea f. rosea	264
A0A023IYH9	A0A023IYH9_BIOOC	unreviewed	Zearalenone lactonohydrolase	zhd101	Clonostachys rosea f. rosea	264

Mode	Affinity (kcal/mol)	RMSD l.b.	RMSD u.b.
1	-6.7	0.000	0.000
2	-6.7	1.866	2.625
3	-5.5	2.580	6.112
4	-4.8	2.312	6.184
5	-4.7	1.720	2.254
6	-4.3	2.314	5.167
7	-4.2	2.033	4.984
8	-4.1	2.399	5.293
9	-3.8	2.730	4.924

Figure 3: Protein ligand docking task.

with the HLA-DPA10103-DPB10201 allele. This multi-step process, handled efficiently by a single user prompt, showcases ChatMol Copilot's ability to manage complex, multi-stage computational tasks effectively Figure 4.

3.3 Molecular docking task

In both designing of drugs or enzymes, molecular docking is commonly involved to determine the intermolecular interactions (Meng et al., 2011). A common molecular docking process requires input of: (1) receptor structure, (2) ligand conformer, (3) docking parameters including centre of box and the size of the box. We show that ChatMol Copilot will facilitate this multi-step task by using a set of related tools. With the name of a ligand provided, the copilot used the tool to search for SMILES and another tool to generate a conformer. After downloading the structure file of the receptor from the RCSB PDB database, the docking parameters were automatically determined under the help of the pocket prediction tool. Finally, the docked complex will be presented to the user Figure 3.

3.4 Molecule generation and filtering with generated Python code

Generating novel molecules with desired properties and structures is very important in drug discovery. A recently large molecule generation model SAFE (Noutahi et al., 2024) is open-sourced. There are 6 different modes for molecule generation, each is provided as an API service, and all the 6 APIs are integrated into ChatMol. In the following example, 200 molecules are requested to be generated with a common core. The molecules are stored in Redis cache with key 'SuperStructure_smiles'. Figure 5.

Molecular properties were calculated using a functional call, and the results were stored in Redis cache. To apply filtering with Lipinski's rule of 5 (Lipinski et al., 2012) to the generated molecules a Python function is created by GPT-4o: Figure 6.

In this code, the generated molecules with their properties were read from Redis, and the then Lipinski's rule of 5 is applied to remove molecules that violate the rules. The remaining molecules are saved into Redis. At the end of the code, the total number of the resulting molecules and the first 5 samples are returned.

4 Discussion and Conclusions

In this work, we present a practical solution how to leverage large language models to assist molecular design and computation, particularly for proteins. We also propose the architecture with multi-level abstraction so as to achieve a higher level of automation, which combines multiple steps in one shot. The automatic code generation and execution expands the systems capabilities beyond the predefined action space. The data abstraction with Redis

cache makes the "Code as Actions" (Wang et al., 2024) more practical for molecular modeling and computation. Similar to the basic concept of "Code as Actions," we use LLMs to generate PyMOL commands in ChatMol based on user instructions, performing relatively complex molecular visualization tasks. Closed-source commercial models like GPT-4 and the Claude series can write PyMOL commands with high accuracy based on user instructions. Smaller open-source models, when fine-tuned with specific instructions, can also perform this task. As the capabilities of relatively smaller LLMs like phi-3 (Abdin et al., 2024) continue to improve, we can expect future open-source, affordable models to replace current commercial models for ChatMol Copilot needs, further democratizing this field. Even though our current experiments are primitive, we believe that the multi-level abstraction approach is a promising direction to achieve even higher intelligent for molecular design and computation.

References

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J Ballard, Joshua Bambrick, et al. 2024. Accurate structure prediction of biomolecular interactions with alphafold 3. *Nature*, pages 1–3.

Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. 2023. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578.

Justas Dauparas, Ivan Anishchenko, Nathaniel Bennett, Hua Bai, Robert J Ragotte, Lukas F Milles, Basile IM Wicky, Alexis Courbet, Rob J de Haas, Neville Bethel, et al. 2022. Robust deep learning–based protein sequence design using proteinmpnn. *Science*, 378(6615):49–56.

Warren L DeLano et al. 2002. Pymol: An open-source molecular graphics tool. *CCP4 Newsl. Protein Crystallogr*, 40(1):82–92.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR.

Joe G Greener, Shaun M Kandathil, Lewis Moffat, and David T Jones. 2022. A guide to machine learning for biologists. *Nature reviews Molecular cell biology*, 23(1):40–55.

Po-Ssu Huang, Scott E Boyken, and David Baker. 2016. The coming of age of de novo protein design. *Nature*, 537(7620):320–327.

Kamilla Kjaergaard Jensen, Massimo Andreatta, Paolo Marcatili, Søren Buus, Jason A Greenbaum, Zhen Yan, Alessandro Sette, Bjoern Peters, and Morten Nielsen. 2018. Improved methods for predicting peptide binding affinity to mhc class ii molecules. *Immunology*, 154(3):394–406.

Greg Landrum et al. 2013. Rdkit: A software suite for cheminformatics, computational chemistry, and predictive modeling. *Greg Landrum*, 8(31.10):5281.

Jiabo Li, Tedman Ehlers, Jon Sutter, Shikha Varma-O'Brien, and Johannes Kirchmair. 2007. Caesar: a new conformer generation algorithm based on recursive buildup and local rotational symmetry consideration. *Journal of chemical information and modeling*, 47(5):1923–1932.

Jiabo Li and Roy McWeeny. 2002. Vb2000: Pushing valence bond theory to new limits. *International journal of quantum chemistry*, 89(4):208–216.

Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, et al. 2023. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130.

Christopher A Lipinski, Franco Lombardo, Beryl W Dominy, and Paul J Feeney. 2012. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Advanced drug delivery reviews*, 64:4–17.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. 2024. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, pages 1–11.

Xuan-Yu Meng, Hong-Xing Zhang, Mihaly Mezei, and Meng Cui. 2011. Molecular docking: a powerful approach for structure-based drug discovery. *Current computer-aided drug design*, 7(2):146–157.

Emmanuel Noutahi, Cristian Gabellini, Michael Craig, Jonathan SC Lim, and Prudencio Tossou. 2024. Gotta be safe: a new framework for molecular design. *Digital Discovery*, 3(4):796–804.

Nicholas Rego and David Koes. 2015. 3dmol. js: molecular visualization with webgl. *Bioinformatics*, 31(8):1322–1324.

David Sehnal, Sebastian Bittrich, Mandar Deshpande, Radka Svobodová, Karel Berka, Václav Bazgier, Sameer Velankar, Stephen K Burley, Jaroslav Koča,

and Alexander S Rose. 2021. Mol* viewer: modern web app for 3d visualization and analysis of large biomolecular structures. *Nucleic acids research*, 49(W1):W431–W437.

Jinyuan Sun, Tong Zhu, Yinglu Cui, and Bian Wu. 2023. Structure-based self-supervised learning enables ultrafast prediction of stability changes upon mutation at the protein universe scale. *bioRxiv*, pages 2023–08.

Renxiao Wang, Xueliang Fang, Yipin Lu, Chao-Yie Yang, and Shaomeng Wang. 2005. The pdbbind database: methodologies and updates. *Journal of medicinal chemistry*, 48(12):4111–4119.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*.

Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. 2018. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530.

Yang Zhang and Jeffrey Skolnick. 2005. Tm-align: a protein structure alignment algorithm based on the tm-score. *Nucleic acids research*, 33(7):2302–2309.

A Cases of using ChatMol Copilot

A.1 Protein stability engineering task

Enzyme stability engineering plays a crucial role in various biotechnological applications by enhancing the resilience of enzymes to environmental conditions and enabling them to maintain their catalytic activity over extended periods. This process involves modifying specific amino acid residues within the enzyme structure to improve its thermal stability, pH tolerance, resistance to proteolytic degradation, and overall performance under varying conditions.

In the process copilot performed, it searches the RCSB PDB database for the LinB enzyme and download it. Subsequently, stabilizing mutations are recommended based on the energy values calculated for each mutation in the provided protein structure according users instructions. These mutations represent amino acid substitutions that are predicted to increase the stability of the enzyme. By introducing these mutations, the enzyme's structural integrity can be enhanced, leading to improved enzymatic activity and potential applications in biocatalysis, drug development, and other biotechnological processes.

A.2 Generate a set of molecules, compute the molecular properties and display the results in a table

In this case, the *de novo* generation method is used to create a set of molecules. A set of molecular properties are computed for each molecule, and the results are collected for all molecules and a table is created. All these steps are accomplished with just one prompt.

B All tools

B.1 Ligand binding pocket prediction

A message passing nerual network (Gilmer et al., 2017) based pocket prediction tool was developed named PocketMPNN. Although many pocket prediction methods were available, a residue-level prediction tool was still in the absence. However, it is of significant importance to facilitate the molecular docking process. Therefore, we developed a neural network trained on the PDB-Bind database (Wang et al., 2005) for pocket residue prediction and a publicly available API was provided. We only took this as a demonstration due to it not being computation extensive and still having satisfactory accuracy.

B.2 Protein structure prediction

The public API provided by the ESM Metagenomic Atlas was used for structure prediction. The ESMFold is of good prediction accuracy and fast response compared with MSA-based prediction such AlphaFold2. Within the length of 400 aa, this API usually responds within 20 seconds. Additionally, ESMFold's reliance on evolutionary information enables it to handle diverse protein sequences and structural motifs with high fidelity.

B.3 Mutation effect prediction

The public API of Pythia (Sun et al., 2023) was used for mutation effect prediction. The Pythia is a ultra fast mutation effect predictor with good accuracy.

B.4 Protein structure visualisation

During the conversation, py3Dmol (Rego and Koes, 2015) is used to show a cartoon representation of a protein. For more interactive and general visualisation and interaction, the streamlit plugin of Mol* (Sehnal et al., 2021) was used.

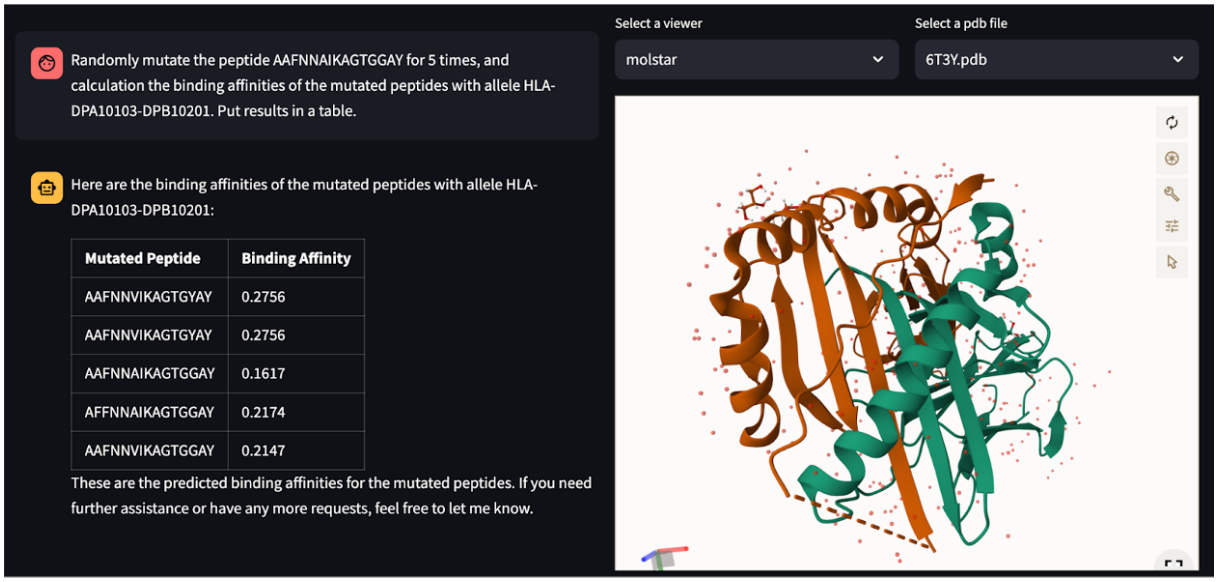


Figure 4: Using MHC binding affinity prediction tool

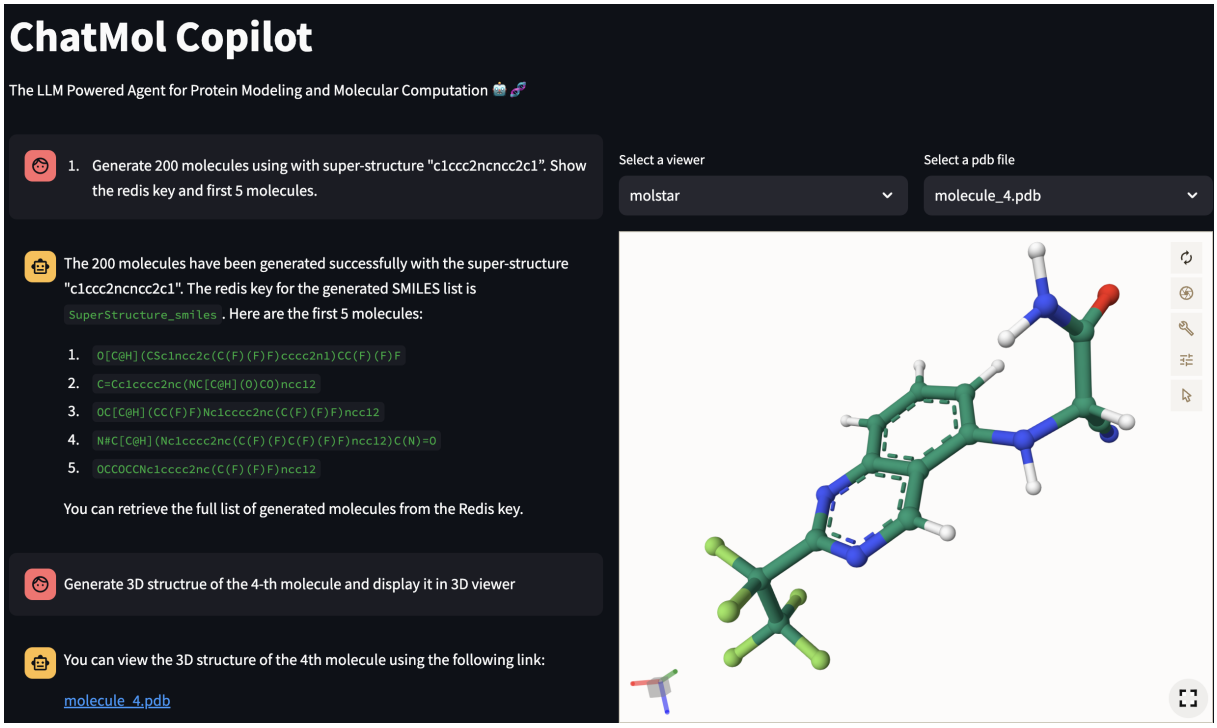


Figure 5: Using SAFE for molecular generation

B.5 Docking

The AutoDock Vina is a fast and widely applied docking tool. We implemented a RESTful API to make it adaptable in the form of a function calling for LLM to use this tool.

B.6 Blind Docking

During the docking process, it is necessary for the geometric centre of a pocket to be assigned. However, this inspection of a structure can be challenging without an experimentally determined protein-ligand complex. Here, we combined the pocket prediction with the Autodock Vina, using the geometric centre of predicted pocket residues as a hint for docking.

B.7 Protein sequence design

We use ProteinMPNN for protein sequence design. It is a neural network based on the message passing neural network, trained on protein structure to generate the native protein sequences and has been experimentally verified to be a robust tool. We also implemented a public accessible API for this copilot.

C Other details of ChatMol Copilot

C.1 Visualisation Components (Mol*, PyMOL and py3Dmol)

Visualisation is one of the most important components for the interactions between a user and ChatMol system. In ChatMol Copilot, three different visualisation components can be used. In addition to traditional interactions via the mouse, one important new way of using computers is to communicate with human natural language. This is made possible via LLMs, such as ChatGPT. The advantages of the three visualisation components are listed below:

PyMOL (DeLano et al., 2002) has high visual quality, and widely adopted by science communities.

MolStar (Mol*) serves as a basis for the next-generation data delivery and analysis tools for (not only) macromolecular structure data.

py3Dmol is a python package can be integrated easily in other python code.

We provide three options there so that users have choices according to their personal preferences.

C.2 Registry for computational services

To improve the interoperability of various computational services, all backend services are wrapped with FastAPI. For the convenience of usage and management of these services, a simple registry system for all FastAPI services is implemented. The registry itself is also a FastAPI service, which provides registration for new services, a map for finding and query the services, and for load balancing and routing. Each registry record contains a brief description of the service, the service name, the endpoint URL and the description of input/output parameters.

C.3 Function Calling and Agentic Approach

Agentic approach is the new trend of workflow automation and more deeply the road map to artificial general intelligence (AGI) as pointed out by Andrew Ng in his very recent talk at here.

As our initial approach in this new paradigm, we have implemented tool use and self reflection in our system design. In additional third party tools, all our internal computational tools which are already wrapped into FastAPI calls are further integrated into ChatMol as function calling services that can be orchestrated using LLMs, such as ChatGPT.

C.4 Registered Services

Registry This is the first service of the registry system. The main function of this service is to register other services. To register a server, the following information must be provided: service name, a brief description of the service, the URL for the service endpoint, a list of input parameter names, and the description of the parameters. **AlphaConf** A super-fast 3D conformation generation method developed by ChemXAI. The input is a file of molecules in SDF format, and the output is a file of generated 3D conformations. It takes less than 30 minutes to generate conformations for all ChEMBL database molecules on a 16-core linux machine. The conformation quality as measured by the coverage of bioactive conformers is comparable or even better than the best commercially available products, such as Omega or ConfGenX. AlphaConf follows a divide-n-conquer and build-up strategy similar to CAESAR algorithm (Li et al., 2007). A highly efficient 3D conformation storage technology is used to compress storage by factor up to 3 orders of magnitude. 100,000 conformations/second (16 core machine). 142M confs of ChEMBL storage:

**Python function to filter molecules based on Lipinski's rule of five:

```
def filter_by_lipinski():
    from chatmol_fn import redis_reader, redis_writer
    import pandas as pd

    # Read the molecule properties data from Redis
    mol_property_table = redis_reader("mol_property_table")

    # Filter according to Lipinski's rule of five
    filtered_data = [mol for mol in mol_property_table if
                        mol['HBD (Hydrogen Bond Donors)'] <= 5 and
                        mol['HBA (Hydrogen Bond Acceptors)'] <= 10
                        mol['Molecular Weight'] < 500 and
                        mol['LOGP'] <= 5]

    # Write the filtered data to Redis
    redis_writer("filtered_lipinski_molecules", filtered_data)

    # Return the total number of molecules and the first 5 molecu
    return {"total_filtered_molecules": len(filtered_data),
            "first_5_molecules": filtered_data[:5]}

# Note: Do not execute this function yet, pending approval.
```

Please review the Python function code above. If it looks good to you, I will proceed with executing it.

Figure 6: Python code generated for Lipinski's rule of five

Table 1: Integrated Tools in ChatMol Copilot

Macromolecules	Description	Small Molecules	Description
PocketMPNN	Ligand binding pocket prediction	SAFE	Molecule generation
ESM Atlas	Protein structure prediction	generate 3D conformation	3D conformation by RDKit
Pythia	Mutation effect prediction	get smiles feature	Calculate features of molecules
py3Dmol, Mol*	Visualizer	predict logp from smiles	Prediction logP for molecules
Autodock	Docking simulation	smiles similarity	Compare molecular similarity
ProteinMPNN	Protein sequence design	AlphaConf	Fast conformation generation
BAPrediction	Peptide-MHC-II binding affinity	AlphaShape	Shape based virtual screening
search rcsb, query uniprot, fetch asked pdb, get smiles from name	Query databases	VB2000	Ab initio valence bond calculation

2.7GB.

AlphaShape Shape and pharmacophore based virtual screening with GPU acceleration. 1000,000 molecule shape comparison/second on a 2-RTX4090 GPU machine.

VB2000 3.0 This is a completely new implementation of early work VB2000 (Li and McWeeny, 2002). A modern *ab initial* valence bond calculation program. The first version was released in year 2000, and the current version is 3.0. More information of VB2000 from the official website at here.

BAPrediction Binding affinity prediction of peptide-MHC-II molecules. The prediction model is trained with the latest data sets, which include both binding affinity data (BA) and eluted ligand binding data. A combination of XGBoost and a novel feature engineering method has been used to improve the prediction accuracy. It provides better results than the published results in literature.

Molecule Generation SAFE is a very recently released open-source molecular generation model is used. The model has 87M parameters and is trained with 1.1 billion compounds in SAFE representations. The SAFE model provides 4 modes for molecule generation: 1) DenovoGen (*de novo* molecular generation). Random generation of molecules with no constraints. The output is a set of SMILES strings of the generated molecules. The input parameter is the number of molecules to be generated. 2) SuperStructure. In super structure generation, new molecules are generated based on a starting core. A smiles of the starting core need to be provided. 3) MotifExtend. In motif extension, we are interested in generating a molecule containing a given motif as a starting point. The extension point of the motif need to be labelled. 4) LinkerGen. Linker generation for linking two fragments. The smiles of two terminal fragments need to be provided in the inputs.

C.5 ChatMol in PyMOL

As an example of "code as action" and the utilization of open-source LLMs, we demonstrate a case where LLMs are directly used to generate PyMOL command lines and perform corresponding molecular visualization tasks in PyMOL. This case involves the use of two LLMs: GPT-4o and a fine-tuned Llama-3-8B-instruct. Both models correctly execute the commands "download 1pga" and "remove waters." However, GPT-4o produced an incorrect response Figure 7 when handling the command "color it by secondary structures".

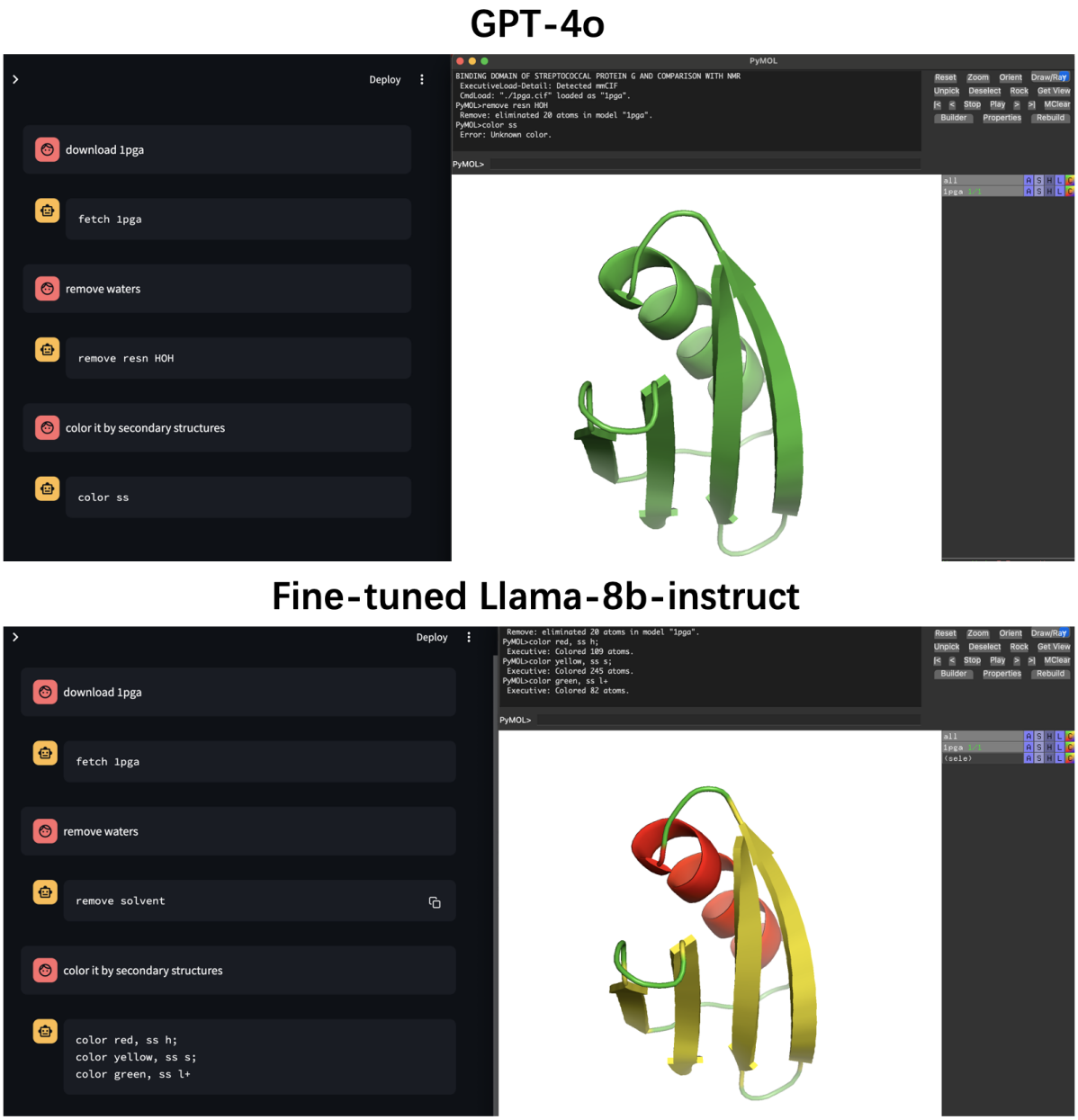


Figure 7: Performing same task using GPT-4o and fine-tuned llama-3-8b instruct