Agentic PyMOL: Structural analysis in PyMOL through natural language

Agentic PyMOL is an MCP server that gives language-model agents discoverable, two-way access to a user's live PyMOL session, letting agents both manipulate molecular visualizations and read structured analytical results back out.

Published Jun 3, 2026

Arcadia Science

DOI: 10.57844/arcadia-gsbx-phd2

Purpose

We built Agentic PyMOL, a model context protocol (MCP) server that gives language-model agents discoverable, two-way access to a user's live PyMOL session. Through MCP, an agent can fetch structures, make selections, align objects, measure geometry, render images, and read structured results back out of PyMOL. This lets researchers use natural language to carry out interactive PyMOL analysis from within their existing agentic workspace. We're sharing it for scientists and developers who want to incorporate PyMOL into agentic workflows.

Statement of need

PyMOL [1] is one of the most widely used tools for molecular visualization and interactive structural analysis. A PyMOL session contains a rich, queryable state: molecular objects, selections, chains, residues, atoms, coordinates, views, alignments, distances, sequences, scenes, and rendered images. In everyday scientific work, researchers use PyMOL not only to look at structures but to ask structural questions: Which chains are present? Which residues contact a ligand or

nucleic acid? How well do two structures align? What atoms are near a binding site? What view best communicates a structural feature?

Answering these questions requires fluency in PyMOL's command-driven scripting language, which has a unique syntax. PyMOL also exposes a parallel Python API, which is powerful but adds a second interface to learn. For newcomers and for experienced users working quickly, the gap between a structural question and the PyMOL incantation that answers it can be significant. The volume of community-produced learning resources (wikis, tutorials, blogs, presentation slides) is itself evidence of how much effort goes into bridging that gap.

Large language model (LLM) agents are increasingly capable of coordinating multi-step work across a user's broader computational workspace, and PyMOL integrations are no exception. Existing approaches generally follow one of two design patterns: embedding a conversational agent inside PyMOL, or exposing PyMOL as a command target that an external agent can drive.

Chatbot- and copilot-style systems place the agent experience inside PyMOL itself or inside a dedicated molecular-assistant interface. ChatMol [2] provides a GUI, a PyMOL plugin, and copilot-style workflows as part of a broader molecular-design assistant. PyMolAI takes this further by forking the open-source version of PyMOL with a chatbot panel. These approaches make molecular visualization feel conversational, but they're insular: The agent isn't operating from the same general-purpose workspace where the user is reading papers, editing code, inspecting data, and coordinating other tools.

Model context protocol (MCP) integrations solve this by exposing PyMOL as a set of tools that an external agent can discover and call. pymol-mcp was an early demonstrator of the core value of this approach, but is underdeveloped and now unmaintained. MCPymol offers workflows for generating beautiful visualizations, but many of its tools confirm actions without surfacing the data an agent would need to build or reason upon. Furthermore, it includes extraneous add-ons that provide support for bespoke tasks like 3D printing, which prevent it from serving as a lightweight bridge. pymol-claude-code takes a less opinionated, more lightweight approach by supporting just three tools (`run_command` , `run_python` , `pymol_get`). However, this “raw passthrough” approach doesn’t surface

discoverable tools that let an agent understand what PyMOL can do without relying on model memorization of PyMOL's command language. Additionally, as the name suggests, it's coupled to a single agent rather than being available to any MCP-compatible client.

Together, these integrations give agents useful reach into PyMOL, but they generally treat it as a system to be driven rather than as a source of structured, queryable state. An agent can issue commands, but it often can't read typed results back out of PyMOL in a form it can inspect, reason over, or pass into downstream work.

What these MCP integrations share is a common limitation: They can issue commands to PyMOL, but have inconsistent/unstructured support for querying state and forwarding data in a reliable format that an agent can build on. To our knowledge, no existing solution provides lightweight, discoverable, two-way communication between a general-purpose LLM agent and a user's live PyMOL session, where the agent can both take action in PyMOL and read structured, PyMOL-native results back out. Agentic PyMOL fills this gap.

Design philosophy

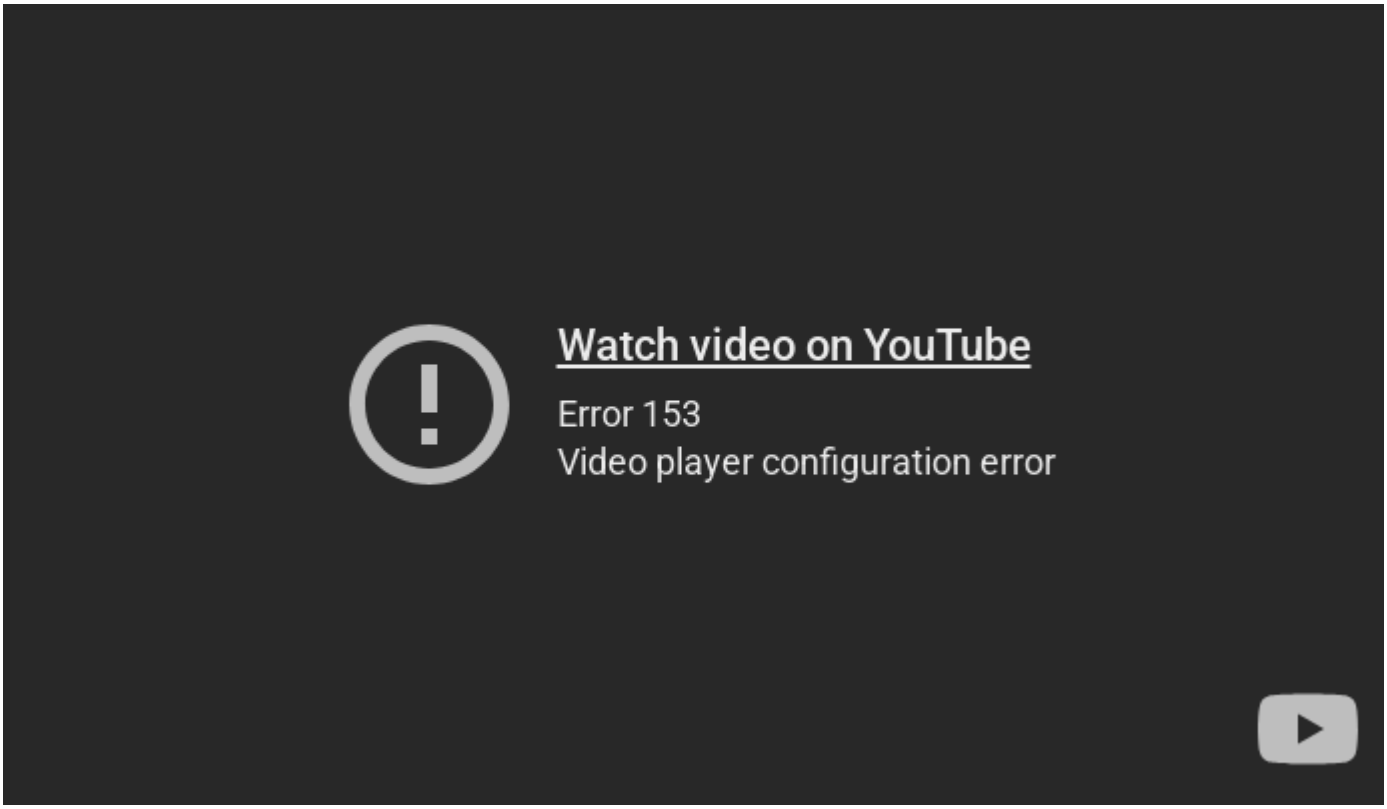
Agentic PyMOL treats MCP not merely as a transport layer for sending PyMOL commands, but as the discoverable interface between a general-purpose agent and PyMOL's native visualization and analysis capabilities. An agent discovers what PyMOL can do, invokes operations with structured inputs, and receives structured results back. This means the agent can take action in PyMOL while also reading structured information back out.

This readback is central to the design. Instead of reporting that a command succeeded, Agentic PyMOL returns data such as loaded objects, chains, atom counts, coordinates, distances, alignments, views, sequences, rendered images, and typed errors. This allows the agent to answer structural-biology questions, not just execute visualization instructions. For example, an agent can use PyMOL to determine which residues in a DNA-binding protein contact DNA, highlight the

residues in the live PyMOL session, and export the data as a table for downstream tasks, all as part of a single coordinated workflow.

Agentic PyMOL is intentionally small. It isn't a docking suite, conservation-analysis server, sequence-search tool, structure-prediction platform, or all-in-one molecular chatbot. It focuses on PyMOL-native operations: visualization, selection logic, structural inspection, geometry, alignment, rendering, and session state. The tool surface tracks these operations. Tools are organized around what PyMOL itself can do.

Rather than asking scientists to adopt a custom PyMOL fork, move their work into a specialized chat interface, or rely on untyped command strings, Agentic PyMOL works with the PyMOL installation they already have and the LLM they already use. PyMOL remains PyMOL; the agent gains a discoverable tool surface for interacting with it. This makes the system useful both for interactive structural exploration, where the user can watch PyMOL update live, and for agentic analysis, where PyMOL-derived results flow into downstream computational work.



A demonstration of how you can use Agentic PyMOL to conduct meaningful structural analysis with natural language.

In this example, the agent fetches a structure, identifies binding-site contacts, labels them in the live PyMOL session, and summarizes its findings — all through conversation. The source code is available on GitHub.

Architecture

Agentic PyMOL consists of two components: an MCP server and a small companion PyMOL plugin. The MCP server is launched by an MCP-compatible client over standard input/output and exposes typed tools for interacting with PyMOL. The plugin runs inside the user's existing PyMOL installation and dispatches requests to `pymol.cmd`, allowing the agent to manipulate the live PyMOL session while the user watches.

The MCP server and PyMOL plugin communicate over a local TCP/JSON bridge. The plugin binds to localhost and requires a shared-secret token on every request. This matters because the bridge accepts arbitrary `pymol.cmd` calls and unsandboxed Python execution: Without the token, any local process could connect and run code in the user's PyMOL session — and, through it, on the host.

On first launch, the plugin generates the token locally; the MCP server reads the same path or accepts an explicit token override through configuration.

The tool surface includes tools for session inspection, loading and saving structures, rendering and screenshots, alignment and RMSD, object and chain introspection, geometry, coordinates, atom-level data, views, and controlled escape hatches for raw PyMOL commands or Python execution. Tool failures are surfaced as structured errors with an error type, message, and original PyMOL traceback. Long-running calls are bound by a configurable timeout; when a call exceeds that timeout, the server sends an interrupt request to PyMOL so the interactive session can recover.

This architecture preserves the strengths of both systems. PyMOL remains the molecular visualization and analysis engine. The MCP client remains the general-purpose agentic workspace. Agentic PyMOL is the narrow bridge between them: small enough to remain understandable, discoverable enough to be employed by the agent, typed enough for agents to reason over, and interactive enough for scientists to keep PyMOL in the loop.

Community

The number of independent PyMOL/LLM integrations that have emerged in a short period suggests a shared need for reliable, discoverable, and typed access to PyMOL from agentic workflows. Agentic PyMOL aims to serve as shared infrastructure for this need: a stable, well-maintained PyMOL layer that the community can build on rather than rebuild independently.

To that end, Agentic PyMOL is deliberately open-source and unopinionated about higher-level workflows. By staying close to what PyMOL itself can do, Agentic PyMOL can serve as a dependency for projects that are opinionated about those workflows, without imposing its own. The primary direction for growth is increasing parity between PyMOL's native operations and the MCP tool surface. Contributions that strengthen that layer are welcome.

Additional methods

We used Claude (Opus 4.7) to help write code and to review our code and selectively incorporated its feedback.

Key takeaways

PyMOL is a powerful tool for both molecular visualization and structural analysis, but most LLM integrations treat it as either a chat environment or a one-way command target. Agentic PyMOL is a small MCP server that connects a general-purpose LLM agent to the user's existing PyMOL session with discoverable, typed, two-way communication. The agent can drive PyMOL (load structures, align objects, change representations, render images) and read structured results back (coordinates, distances, chains, atom counts, sequences, alignments, typed errors). This means the agent can answer structural-biology questions using PyMOL, not just send it commands.

Agentic PyMOL works with whatever PyMOL installation you already have. It doesn't require a custom fork, a specialized chat interface, or a particular LLM provider. It's deliberately small and unopinionated about higher-level workflows so other tools/workflows can build on its shared infrastructure.

> The **codebase** is hosted on GitHub (DOI: 10.5281/zenodo.20531917).

Contributors (A–Z)

- **James R. Golden**: Supervision, Validation
- **Evan Kiefl**: Conceptualization, Software, Visualization, Writing

References

1. https://pymol.org
2. Sun J, Li A, Deng Y, Li J. (2024). ChatMol Copilot: An Agent for Molecular Modeling and Computation Powered by LLMs.

https://doi.org/10.18653/v1/2024.langmol-1.7