Exploring LLM Agent Designs and Interaction Modalities for Scientific Visualization

Jackson E. Vonderhorst* Univ. Notre Dame
Kuangshi Ai† Univ. Notre Dame
Haichao Miao‡ LLNL
Shusen Liu§ LLNL
Chaoli Wang¶ Univ. Notre Dame

ABSTRACT

This paper examines how large language model (LLM) agents perform on scientific visualization (SciVis) tasks that require generating visualization workflows from natural-language instructions. We compare three representative agent designs—domain-specific agents with structured tool use, computer-use agents, and general-purpose coding agents—across 15 benchmark tasks, evaluating visualization quality, efficiency, robustness, computational cost, and the impact of persistent memory. We further study interaction modalities, including code scripts, model context protocol (MCP) or API calls, command-line interfaces (CLI), and graphical user interfaces (GUI). Our goal is to characterize the tradeoffs among representative SciVis agent configurations used in practice. The results reveal clear tradeoffs across agent designs and interaction modalities. General-purpose coding agents achieve the highest task success rates but incur greater computational cost, whereas domain-specific agents are more efficient and stable but less flexible. Computer-use agents perform well on individual operations but struggle with multi-step workflows. Across both CLI- and GUI-based settings, persistent memory improves performance over repeated trials, but its effectiveness depends on the interaction mode and the quality of feedback. These findings suggest that future SciVis systems should combine structured tool use, interactive capabilities, and adaptive memory mechanisms to balance performance, robustness, and flexibility.

Keywords: Scientific visualization, LLM agents, interaction modalities, evaluation

1 INTRODUCTION

Scientific visualization (SciVis) tools such as ParaView [2] are essential for analyzing complex, high-dimensional data, yet they remain difficult to use, particularly for new users. Effective visualization often requires detailed knowledge of rendering pipelines, filter semantics, and scripting APIs, making even simple exploratory tasks time-consuming and error-prone.

Recent advances in large language models (LLMs) have enabled a new generation of autonomous agents capable of translating natural language intent into visualization actions. Systems such as ChatVis [27], ParaView-MCP [21], and AI VIS co-scientist [23] demonstrate that language-driven interfaces can reduce the cognitive burden of interacting with SciVis software. More broadly, multimodal agents now exhibit the ability to perceive visual interfaces, reason about task goals, and execute actions across GUIs designed for human use [24, 35]. These developments suggest a shift from manual visualization workflows toward AI-assisted interaction.

Despite this progress, current systems remain unreliable on realistic SciVis tasks, which often require multi-step reasoning, parameter tuning, and perceptual verification. Recent work has called for community collaboration to develop benchmarks that enable meaningful evaluation of agents for scientific data analysis and visualization [4]. SciVisAgentBench [5] addresses this need by providing representative tasks and evaluation protocols, which we adopt to compare agent configurations under shared conditions.

Existing SciVis agents differ along multiple design dimensions, including interaction mechanisms, tool access, planning strategies, memory capabilities, and underlying models. In this work, we examine these systems through the lens of *agent designs* and *interaction modalities*. Agent designs describe the capabilities and architectural choices provided by a system. Interaction modalities describe how agents interface with visualization software during task execution.

To facilitate discussion, we group representative systems into three broad categories based on their primary design strategy. *Domain-specific agents* (e.g., ChatVis [27], ParaView-MCP [21]) operate directly on visualization APIs to achieve efficient and deterministic execution. *Computer-use agents* (e.g., UFO [36–38], Open Interpreter [25]) interact with existing software environments to enable flexible adaptation and error recovery. *General-purpose coding agents* (e.g., Codex CLI [26], Claude Code [9]) leverage broad programming capabilities to construct end-to-end workflows with minimal domain constraints. Additionally, some agents (e.g., Agent S [1], Letta [20]) further incorporate persistent memory and reflection to improve performance over repeated trials.

Interaction modality describes how agents execute actions, including *code scripts* and *model context protocol* (MCP) or *API calls* for structured tool use, as well as *command-line interfaces* (CLI) and *graphical user interfaces* (GUI) for more general interaction. These modalities are often associated with particular agent categories but are not themselves controlled experimental factors. More broadly, practical SciVis agents differ simultaneously in interaction mechanisms, tool access, planning strategies, memory capabilities, execution frameworks, and underlying LLM backbones. Consequently, understanding the tradeoffs among various agent configurations remains an important open problem for the SciVis community.

In this paper, we present a comparative study of representative agent designs and interaction modalities for LLM-driven SciVis agents. Rather than proposing a new agent architecture, we focus on evaluating representative systems under realistic tasks derived from SciVisAgentBench. Our goal is not to isolate the effect of a single design factor, but to understand how representative SciVis agent configurations succeed or fail on realistic tasks and what tradeoffs and design principles may guide future visualization agents. Specifically, this work makes the following contributions:

- We provide one of the first empirical comparisons of representative SciVis agent configurations with an analysis of persistent memory and adaptive learning.
- We identify key behavioral tradeoffs, including determinism vs. robustness and efficiency vs. adaptability, that shape agent effectiveness.
- We adopt a task-driven evaluation that measures success, consistency, failure modes, and recovery behaviors.
- We derive design insights suggesting that hybrid agents with deterministic execution, perceptual grounding, and adaptive learning may outperform any individual agent configuration.

*e-mail: jvonder2@nd.edu
†e-mail: kai@nd.edu
‡e-mail: miao1@llnl.gov
§e-mail: liu42@llnl.gov
¶e-mail: chaoli.wang@nd.edu

2 Related Work

Recent SciVis research increasingly treats LLMs as agentic systems that execute workflows rather than serve as passive interfaces [16]. Existing approaches can be organized by agent designs, as well as by differences in interaction modality and adaptation capability.

Domain-specific agents translate natural language into structured API calls within visualization systems, e.g., ChatVis [27], ParaView-MCP [21], and InferA [29], thereby enabling efficient, reproducible execution. However, their tight coupling to predefined tools limits robustness in the face of ambiguous intent or unforeseen states.

Computer-use agents interact directly with software interfaces via perception and low-level actions. Domain-specific systems such as AVA [22], NLI4VolVis [7], TexGS-VolVis [30], and HiLSVA [3] also incorporate visual feedback for iterative refinement, while general frameworks including UFO series [36–38] and Anthropic's computer-use [8] enable flexible adaptation across applications, which improves error recovery but incurs higher computational cost and variability. Recent work further highlights that effective human oversight in such agents depends on how supervision is structured, rather than simply increasing user control [13]. Persistent memory and reflection mechanisms can improve performance over time, as in Agent-S [1] and OS-Copilot [31]. In SciVis, VizGenie [11] demonstrates reusable visualization skills. Broader studies emphasize the importance of learning in long-horizon tasks [18,28], though evaluation remains difficult due to delayed and noisy feedback.

General-purpose coding agents such as Codex [26] and Claude Code [9] leverage broad programming capabilities to construct workflows end-to-end. While flexible, they often incur higher costs and lack the structured guarantees of domain-specific systems. Recent studies further show that domain-specific agent skills can improve coding-agent performance on SciVis tasks by providing reusable procedural knowledge and tool-specific guidance [6].

Evaluating agent behavior remains challenging. Prior work considers execution success, perceptual similarity, and user studies. CoDA [15], MatPlotBench [34], and VisEval [14] propose complementary metrics, while benchmarks such as VisualWebArena [19] and WindowsAgentArena [12] extend evaluation to broader environments. Beyond outcome evaluation, recent HCI systems emphasize real-time monitoring and analysis of intermediate states. Tools such as VizCode [32] and SPARK [33] enable fine-grained tracking of multi-step programming processes, underscoring the importance of inspecting intermediate states and workflow structure rather than relying solely on final outputs. In SciVis, recent efforts call for more systematic evaluation frameworks [4]. SciVisAgentBench [5] provides realistic tasks and outcome-based protocols, while SVLAT [17] introduces a standardized assessment for scientific visualization literacy. However, existing benchmarks do not explicitly analyze how agent designs and interaction modalities influence robustness, consistency, and failure modes in complex multi-step workflows.

3 Experimental Setup

We evaluate representative SciVis agent systems through a controlled comparison of different agent designs and interaction strategies operating on shared ParaView workflows. Rather than proposing a new architecture, we compare three representative agent configurations: *domain-specific agents*, *computer-use agents*, and *general-purpose coding agents*, and additionally analyze the effect of persistent memory in selected agents. We examine differences in effectiveness, robustness, computational cost, and behavioral patterns.

Our study evaluates agents on tasks derived from SciVisAgentBench [5], which provides realistic visualization workflows representative of common ParaView usage. All agents are tested under identical task specifications and system environments to enable direct comparison. Our experiments focus not only on whether agents complete tasks, but also on how they behave during execution, including their ability to recover from errors, adapt to unexpected states, and maintain consistency across repeated trials.

Agent settings. We evaluate eight representative agents spanning different agent design strategies and interaction modalities. Domain-specific agents include ChatVis [27] and ParaView-MCP [21], which operate through scripts or structured API calls. Computer-use agents include the Microsoft UFO series [36–38] and Open Interpreter [25], which interact with software through GUIs. General-purpose coding agents include Claude Code [9] and Codex [26], which primarily operate via CLI-based code generation. In addition, Agent S [1] and Letta [20] are evaluated in both learning-enabled and learning-disabled configurations to isolate the effect of persistent memory across GUI- and CLI-based settings. For Agent-S, memory consists of previously observed task trajectories and execution experiences that can be retrieved to guide future actions. For Letta, memory consists of persistent records of prior interactions, execution outcomes, and planning context that can be recalled during subsequent tasks.

Task design. All agents are evaluated on a shared subset of 15 ParaView tasks derived from SciVisAgentBench (Figure 1), selected for a representative coverage of the SciVisAgentBench taxonomy across application domains, data types, visualization operations, and complexity levels [5], while keeping the experimental cost manageable. These tasks represent common ParaView workflows and span multiple levels of complexity, ranging from single-step operations (e.g., applying a filter or loading a dataset) to multi-step visualization pipelines involving dependent operations such as filter composition, parameter tuning, camera configuration, and output generation. Each of the eight evaluated agents is tested across the full set of tasks, with each task executed 10 times per agent to capture performance variability and assess consistency across repeated trials.

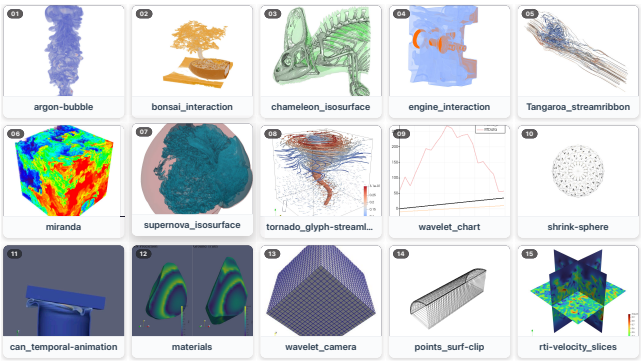

Figure 1: The 15 representative ParaView visualization tasks from SciVisAgentBench.

Stepwise evaluation. For agents operating through GUIs (UFO, Open Interpreter, and Agent-S), we additionally evaluate step-by-step task decomposition. Each task is manually broken into a sequence of intermediate steps, and a corresponding gold-standard state is defined for each step. Agents execute each step independently, and outputs are compared against the gold-standard state using the same rubric criteria as in full-task evaluation. A step is marked as a pass if the generated state sufficiently matches the expected intermediate result, and a failure otherwise. To control for compounding errors, each step begins from the gold-standard state of the previous step rather than the agent's prior output. This isolates the agent's ability to perform individual operations correctly. Final performance is reported as the proportion of correctly executed steps across all tasks and trials.

Analysis goals. Our goal is not to identify a universally superior agent. Domain-specific agents are expected to excel in efficiency, while computer-use agents may offer greater adaptability. Persistent memory and adaptive learning may further improve some agents with repeated experience. Instead, we aim to understand which capabilities are most beneficial for SciVis and how different agent configurations succeed or struggle in realistic workflows. Specifically,

Table 1: Benchmark performance and resource usage on the 15 representative ParaView visualization tasks. The overall score is reported on a 0-100 scale as mean±std across 10 trials. Token counts and time are per-task statistics, averaged over all 15 tasks across 10 trials (mean±std). Completion rate measures the proportion of runs that finish without explicit execution errors. In contrast, pass metrics are based on the LLM-judge (Claude-Opus-4.6) rubric, in which a trial passes if its score is at least 50%. See Figure 2 for pass@k and pass^k curves.

Setting	Overall Score ↑	Completion Rate ↑	Input Tokens ↓	Output Tokens ↓	Time ↓	Agent Configuration
ChatVis+GPT-5.2	37.25±4.92	45.0%±8.5%	8.17K±842	10.81K±816	1m 35s±16s	Domain-Specific (Code Script)
ParaView-MCP+Claude Sonnet-4.6	30.49±7.08	50.7%±8.9%	146.67K±17.89K	7.82K±739	2m 11s±15s	Domain-Specific (MCP)
Codex CLI+GPT-5.2	68.99±1.92	100.0%±0.0%	774.52K±623.29K	7.48K±4.54K	2m 15s±1m 27s	General-Purpose Coding (CLI)
Claude Code+Claude-Sonnet-4.6	66.35±3.92	96.0%±4.7%	346.44K±340.56K	7.78K±7.44K	2m 49s±2m 47s	General-Purpose Coding (CLI)
UFO+GPT-5.2	15.71±4.52	27.1%±10.7%	310.96K±29.46K	5.84K±422	6m 13s±1m 3s	Computer-Use (GUI)
Open Interpreter+Claude-Sonnet-4.6	14.04±6.03	27.9%±11.2%	211.37K±15.14K	4.26K±443	5m 31s±59s	Computer-Use (GUI)
Agent-S (Learning Enabled)+GPT-5.2	18.31±8.97	40.0%±12.6%	242.00K±18.53K	3.74K±318	5m 22s±52s	Memory+Learning (GUI)
Agent-S (Learning Disabled)+GPT-5.2	10.75±6.11	28.6%±9.8%	276.90K±20.10K	5.07K±480	5m 58s±1m 8s	Memory+Learning (GUI)
Letta (Learning Enabled)+Claude-Sonnet-4.6	30.78±9.44	42.9%±10.3%	11.51K±801	8.77K±841	1m 47s±10s	Memory+Learning (CLI)
Letta (Learning Disabled)+Claude-Sonnet-4.6	19.09±6.42	32.9%±6.8%	18.17K±1.88K	10.24K±914	2m 35s±26s	Memory+Learning (CLI)

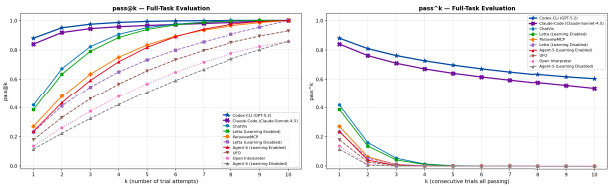

Figure 2: pass@k (top) and pass^k (bottom) curves for full-task evaluation. ChatVis and Letta (Learning Enabled) lead, with all coding agents converging to pass@k ≈ 1.0 by k=10. pass^k decays sharply, with only the top performers retaining non-zero values past k=4.

Table 2: Stepwise evaluation of GUI-based agents. Each of the 15 tasks is decomposed into 5 steps, yielding 15×5 step instances per trial. At each step, the agent starts from the ground-truth state of the previous step and executes the next. We run 3 trials. Step Pass Rate is computed per trial as #successful steps$/(15 \times 5)$ and averaged across trials (mean±std). Token counts and time are per-task statistics, averaged over 15 tasks across 3 trials (mean±std).

Setting	Input Tokens ↓	Output Tokens ↓	Time ↓	Step Pass Rate ↑
UFO+GPT-5.2	296.93K±50.69K	7.36K±596	5m 44s±37s	63.9%±5.9%
Open Interpreter+Claude-Sonnet-4.6	199.46K±43.90K	5.42K±420	5m 12s±32s	60.7%±6.7%
Agent-S (Learning Enabled)+GPT-5.2	221.54K±38.23K	5.38K±461	5m 3s±31s	65.2%±8.9%
Agent-S (Learning Disabled)+GPT-5.2	221.54K±38.23K	5.38K±461	5m 3s±31s	59.5%±8.1%

this study investigates: (1) the reliability of each agent configuration across repeated trials; (2) the behavioral factors that distinguish successful from unsuccessful executions; (3) the efficiency trade-offs including runtime and token usage; (4) the impact of persistent memory and learning on multi-step task performance; and (5) the extent to which complementary strengths may inform the design of future SciVis agents.

4 Results and Discussion

Consistent patterns emerge across all experiments. General-purpose coding agents achieve the highest task completion rates and pass@k performance, but at significantly higher token usage and runtime compared to all other configurations. Domain-specific agents exhibit substantially lower computational cost and more stable execution patterns, though they remain limited in flexibility and overall performance ceiling. Computer-use agents underperform on full-task workflows but achieve substantially higher success rates under step-wise decomposition, indicating that multi-step planning, rather than interface perception, is their primary limitation. Stepwise decomposition improves both execution quality and computational efficiency by constraining planning complexity and reducing token usage and runtime. Finally, in auxiliary comparisons on Agent-S and Letta, persistent memory and learning improve both performance and efficiency by reducing redundant exploration of unsuccessful strategies.

4.1 Overall Performance and Cost-Accuracy Tradeoffs

We evaluate all eight agents on 15 SciVisAgentBench tasks, with results summarized in Table 1 and pass@k / pass^k curves shown in Figure 2. General-purpose coding agents operating via CLI achieve the strongest overall performance. Codex CLI attains the highest overall score (68.99±1.92) and a 100.0% completion rate, followed closely by Claude Code (66.35±3.92, 96.0% completion). Both models also dominate pass@k, approaching saturation by $k = 10$. However, this performance comes at a substantial computational cost. Codex CLI requires an average of 774.52K input tokens per task, with high variance, while Claude Code similarly incurs large token overhead. These results indicate that CLI-based agents achieve high reliability through extensive code generation and iterative refinement, rather than solely through efficient task execution. In contrast, domain-specific agents (ChatVis and ParaView-MCP) exhibit substantially lower cost profiles, with input token usage reduced by one to two orders of magnitude (e.g., 8.17K for ChatVis). Although their completion rates (45.0%–50.7%) remain below those of general-purpose coding agents, they demonstrate greater efficiency per successful execution, reflecting the advantages of structured tool access and constrained action spaces. Taken together, these results establish a clear tradeoff: general-purpose coding agents maximize task completion, whereas domain-specific agents optimize efficiency.

4.2 Consistency and Pass@k Behavior

The pass@k and pass^k curves in Figure 2 provide additional insight into execution reliability. While several agents achieve high pass@k values, particularly at larger k, pass^k declines rapidly across all models, indicating that consistent success across repeated trials remains limited. Even the strongest general-purpose coding agents exhibit variability in execution trajectories, suggesting that their success often depends on favorable intermediate decisions rather than fully stable reasoning strategies. Domain-specific agents and Letta (learning enabled) demonstrate stronger early pass@k performance, reflecting more deterministic execution patterns. However, their asymptotic performance remains lower, highlighting a tradeoff between consistency and ultimate task-completion capability.

4.3 Limitations of GUI-Based Interaction

Computer-use agents (UFO, Open Interpreter, and Agent-S) underperform in the full-task setting, with completion rates below 40%, substantially higher runtime (5–6 minutes per task), and token usage (200K–300K range) relative to other configurations (Table 1). This performance gap is largely attributable to the demands of GUI-based interaction. Unlike script- or API-driven agents, these models must repeatedly interpret high-dimensional visual input, maintain execution context across multiple steps, and execute fine-grained interface actions. These requirements introduce compounding sources of error, particularly in multi-step workflows where early inaccuracies propagate through later actions.

However, these results should not be interpreted as a fundamental limitation of GUI-based agents. SciVis pipelines often require iterative visual verification, parameter-sensitive operations, and maintenance of intermediate visualization states. Errors in transfer functions, camera placement, pipeline ordering, or rendering parameters may remain visually ambiguous until later stages of the workflow, causing failures to propagate across operations. As a result, multi-step SciVis workflows place disproportionate demands on perceptual

grounding and sequential reasoning, both of which remain challenging for current multimodal agents.

4.4 Stepwise Decomposition and Interaction Granularity

To isolate the effect of multi-step workflow execution, we evaluate GUI-based agents under stepwise task decomposition (Table 2). Under this setting, step-level pass rates increase to approximately 60%–65% across all models. In addition to improving completion rates, stepwise decomposition also yields consistent reductions in token usage and runtime. Because each step constrains the action space and reduces planning depth, agents avoid redundant exploration and repeated correction cycles that are common in full-task execution. This suggests that a significant portion of the inefficiency observed in GUI-based agents arises from the overhead of multi-step reasoning rather than the cost of individual actions. GUI-based agents are capable of accurate localized reasoning, but struggle to maintain coherence across extended action sequences. In other words, the primary limitation of computer-use agents is not interface understanding itself, but planning depth. When task complexity is reduced through decomposition, their performance becomes substantially more competitive at the step level. GUI-based agents may be better suited for interactive or human-in-the-loop settings, where task structure can be incrementally specified, rather than for fully autonomous execution of long SciVis workflows.

More broadly, these results suggest that future SciVis agents may benefit from hierarchical workflow decomposition and intermediate-state supervision. Visualization workflows naturally expose semantically meaningful intermediate states that can be inspected, validated, and corrected before execution continues. This creates opportunities for mixed-initiative interaction and stepwise evaluation strategies that are particularly well suited to SciVis environments.

4.5 Effects of Persistent Memory and Adaptive Learning

We further evaluate the impact of persistent memory by comparing learning-enabled and learning-disabled configurations for Letta and Agent-S (Table 1). The evaluated memory mechanisms primarily store and retrieve prior execution experiences, enabling agents to reuse successful strategies and avoid repeating previously unsuccessful behaviors. Across both Letta and Agent-S, enabling memory yields consistent improvements in completion rate, overall score, and efficiency. For example, Letta improves from 19.09 to 30.78 in overall score, while Agent-S improves from 10.75 to 18.31. Notably, the benefits of persistent memory extend beyond improvements in success rate. Across both Letta and Agent-S, enabling memory also reduces token usage and execution time. This indicates that persistent memory enables agents to avoid revisiting previously unsuccessful strategies, leading to more efficient trajectories through the solution space. In this sense, memory improves effectiveness and efficiency without introducing additional computational overhead. These gains indicate that persistent memory reduces the occurrence of repeated failures and enables more efficient exploration of the solution space. In the case of Letta, the improvements are particularly pronounced, as memory reduces redundant script generation and repeated execution errors. For Agent-S, the gains are more modest, since perceptual and interaction overhead remain dominant even when experience is retained. At the same time, the coding-based memory setting reveals an important limitation for SciVis tasks. An agent may converge to workflows that are syntactically valid and executable, yet semantically incorrect from a visualization perspective. Correctness in SciVis often depends on perceptual properties of rendered outputs, such as spatial structure, feature visibility, and camera configuration. Persistent memory alone is insufficient and should be complemented by visual validation and SciVis-specific feedback mechanisms.

4.6 Synthesis, Implications, and Limitations

Taken together, these results suggest that future SciVis agents should not be designed as purely autonomous code generators or GUI operators. Effective SciVis systems may instead require support for visualization-state reasoning, perceptual verification, and hierarchical workflow decomposition. SciVis workflows frequently contain semantically incorrect yet executable intermediate states, making visual feedback and intermediate validation particularly important. Our findings further suggest that evaluation protocols for SciVis agents should extend beyond final task completion to include intermediate-state correctness, perceptual consistency, and robustness across complex multi-step workflows. More broadly, the results indicate that no single configurations is sufficient, while hybrid systems combining structured tool use, perceptual grounding, and adaptive memory mechanisms may provide the most promising direction for future SciVis agents. Meanwhile, the evaluated systems differ along multiple dimensions, including backbone models, planning strategies, memory mechanisms, tool access, and interaction interfaces. Consequently, the observed performance differences should be interpreted as tradeoffs among representative SciVis agent configurations rather than the causal effect of any single design factor. Future work could conduct controlled ablation studies to isolate the contribution of individual components.

5 Conclusions and Future Work

We present a comparative study of agent designs and interaction modalities for LLM-based SciVis agents, evaluating domain-specific, computer-use, and general-purpose coding agents on realistic multi-step SciVis tasks, together with an auxiliary analysis of persistent memory in selected agents. Our results show that while general-purpose coding agents achieve the highest task completion rates, they incur significantly higher computational costs. Domain-specific agents provide more efficient and stable execution but lack flexibility. In contrast, computer-use agents reveal strong step-level reasoning capabilities but struggle to maintain coherent execution across multi-step workflows. Persistent memory improves both effectiveness and efficiency by reducing redundant exploration across repeated trials.

Looking forward, two primary directions emerge for future SciVis agent design. First, continued improvements in general-purpose coding agents suggest that CLI-based frameworks may become increasingly powerful for fully automated workflows. Their key advantage lies in the flexibility of operating within a full programming environment rather than through predefined tool interfaces. As foundation models become more capable, the reliability gap between coding agents and structured tool-use agents may continue to narrow. At the same time, introducing domain-specific constraints through procedural guidelines, such as agent skills [10], could substantially improve efficiency and reliability by reducing redundant exploration. Second, hybrid approaches that combine complementary agent configurations offer a promising solution. For example, future agents may integrate deterministic API-based execution for reliable low-level operations, CLI-based reasoning for complex workflow generation, GUI interaction for perceptual grounding, and persistent memory mechanisms to adapt execution strategies over time.

A key challenge for future work is enabling reliable SciVis-specific self-evaluation mechanisms. Unlike conventional programming tasks, visualization correctness often depends on perceptual and semantic properties that cannot be fully verified through code execution alone. In particular, integrating visual feedback into learning loops could allow agents to assess intermediate visualization states and detect issues such as incorrect transfer functions, camera configurations, and feature visibility, improving both accuracy and consistency. Additionally, incorporating selective elements of GUI-based interaction may provide useful grounding for debugging and recovery in complex workflows. More broadly, future research should explore how to balance efficiency, robustness, and flexibility across agent designs and interaction modalities, and how to design unified systems that combine structured tool use, perceptual grounding, and adaptive memory mechanisms for real-world scientific workflows.

Acknowledgments

This research was supported by the U.S. NSF through grants IIS-2101696, OAC-2104158, IIS-2401144, and CCF-2550610, and the U.S. DOE through grants DE-SC0023145 and ECRP 51917/SCW1885. This work was performed under the auspices of the DOE by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344, reviewed and released under LLNL-CONF-2019246. The authors thank the anonymous reviewers for their insightful comments.

References

[1] S. Agashe, J. Han, S. Gan, J. Yang, A. Li, and X. E. Wang. Agent S: An open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*, 2024. doi: 10.48550/arXiv.2410.08164 1, 2

[2] J. P. Ahrens, B. Geveci, and C. C. Law. ParaView: An end-user tool for large-data visualization. In C. D. Hansen and C. R. Johnson, eds., *The Visualization Handbook*, chap. 36, pp. 717–731. Academic Press, 2004. doi: 10.1016/B978-012387582-2/50038-1 1

[3] K. Ai, P. P. Do, and C. Wang. HiLSVA: Design and evaluation of a human-in-the-loop agentic system for scientific visualization. *arXiv preprint arXiv:2606.26614*, 2026. doi: 10.48550/arXiv.2606.26614 2

[4] K. Ai, H. Miao, Z. Li, C. Wang, and S. Liu. An evaluation-centric paradigm for scientific visualization agents. In *Proc. IEEE VIS & GenAI*, 2025. doi: 10.48550/arXiv.2509.15160 1, 2

[5] K. Ai, H. Miao, K. Tang, N. Gorski, J. Sun, G. Liu et al. SciVisAgentBench: A benchmark for evaluating scientific data analysis and visualization agents. *arXiv preprint arXiv:2603.29139*, 2026. doi: 10.48550/arXiv.2603.29139 1, 2

[6] K. Ai, H. Miao, K. Tang, S. Liu, and C. Wang. SciVisAgentSkills: Design and evaluation of agent skills for scientific data analysis and visualization. *arXiv preprint arXiv:2606.05525*, 2026. doi: 10.48550/arXiv.2606.05525 2

[7] K. Ai, K. Tang, and C. Wang. NLI4VolVis: Natural language interaction for volume visualization via multi-LLM agents and editable 3D Gaussian splatting. *IEEE Trans. Vis. Comput. Graph.*, 32(1):46–56, 2026. doi: 10.1109/TVCG.2025.3633888 2

[8] Anthropic. Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku. https://www.anthropic.com/news/3-5-models-and-computer-use. 2

[9] Anthropic. Claude Code: An agentic coding tool. https://github.com/anthropics/claude-code, 2025. 1, 2

[10] Anthropic. Equipping agents for the real world with agent skills. https://claude.com/blog/equipping-agents-for-the-real-world-with-agent-skills, 2025. 4

[11] A. Biswas, T. L. Turton, N. R. Ranasinghe, S. Jones, B. Love, W. Jones et al. VizGenie: Toward self-refining, domain-aware workflows for next-generation scientific visualization. *IEEE Trans. Vis. Comput. Graph.*, 32(1):1021–1031, 2026. doi: 10.1109/TVCG.2025.3634655 2

[12] R. Bonatti, D. Zhao, F. Bonacci, D. Dupont, S. Abdali, Y. Li et al. Windows Agent Arena: Evaluating multi-modal OS agents at scale. *arXiv preprint arXiv:2409.08264*, 2024. doi: 10.48550/arXiv.2409.08264 2

[13] C. Chen, Z. Zhang, Z. Chen, E. Xu, Y. Yang, I. Khalilov et al. Comparing human oversight strategies for computer-use agents. *arXiv preprint arXiv:2604.04918*, 2026. doi: 10.48550/arXiv.2604.04918 2

[14] N. Chen, Y. Zhang, J. Xu, K. Ren, and Y. Yang. VisEval: A benchmark for data visualization in the era of large language models. *IEEE Trans. Vis. Comput. Graph.*, 31(1):1301–1311, 2025. doi: 10.1109/TVCG.2024.345632 2

[15] Z. Chen, J. Chen, S. Ö. Arik, M. Sra, T. Pfister, and J. Yoon. CoDA: Agentic systems for collaborative data visualization. *arXiv preprint arXiv:2510.03194*, 2025. doi: 10.48550/arXiv.2510.03194 2

[16] V. Dhanoa, A. Wolter, G. M. León, H.-J. Schulz, and N. Elmqvist. Agentic visualization: Extracting agent-based design patterns from visualization systems. *IEEE Comput. Graph. Appl.*, 45(6):89–90, 2025. doi: 10.1109/MCG.2025.3607741 2

[17] P. P. Do, K. Tang, K. Ai, and C. Wang. SVLAT: Scientific visualization literacy assessment test. *arXiv preprint arXiv:2603.19000*, 2026. doi: 10.48550/arXiv.2603.19000 2

[18] J. Fang, Y. Peng, X. Zhang, Y. Wang, X. Yi, G. Zhang et al. A comprehensive survey of self-evolving AI agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*, 2025. doi: 10.48550/arXiv.2508.07407 2

[19] J. Y. Koh, R. Lo, L. Jang, V. Duvvur, M. C. Lim, P.-Y. Huang et al. Evaluating multimodal agents on realistic visual web tasks. In *Proc. ACL*, pp. 881–905, 2024. doi: 10.18653/v1/2024.acl-long.50 2

[20] Letta AI and contributors. Letta: The platform for building stateful ai agents. https://github.com/letta-ai/letta, 2026. 1, 2

[21] S. Liu, H. Miao, and P.-T. Bremer. ParaView-MCP: An autonomous visualization agent with direct tool use. In *Proc. IEEE VIS (Short Papers)*, pp. 61–65, 2025. doi: 10.48550/arXiv.2505.07064 1, 2

[22] S. Liu, H. Miao, Z. Li, M. Olson, V. Pascucci, and P.-T. Bremer. AVA: towards autonomous visualization agents through visual perception-driven decision-making. *Comput. Graph. Forum*, 43(3):e15093, 2024. doi: 10.1111/cgf.15093 2

[23] H. Miao, Z. Li, K. Ai, K. Tang, C. Wang, P.-T. Bremer et al. Toward AI VIS co-scientists: A general and end-to-end agent harness for solving complex data visualization tasks. *arXiv preprint arXiv:2605.21825*, 2026. doi: 10.48550/arXiv.2605.21825 1

[24] D. Nguyen, J. Chen, Y. Wang, G. Wu, N. Park, Z. Hu et al. GUI agents: A survey. In *Proc. ACL Findings*, pp. 22522–22538, 2025. doi: 10.18653/v1/2025.findings-acl.1158 1

[25] Open Interpreter. Open Interpreter: A natural language interface for computers. https://github.com/openinterpreter/open-interpreter, 2023. 1, 2

[26] OpenAI. OpenAI Codex: Lightweight coding agent that runs in your terminal. https://github.com/openai/codex, 2025. 1, 2

[27] T. Peterka, T. Mallick, O. Yildiz, D. Lenz, C. Quammen, and B. Geveci. ChatVis: Large language model agent for generating scientific visualizations. In *Proc. IEEE LDAV*, pp. 22–32, 2025. doi: 10.1109/LDAV68558.2025.00007 1, 2

[28] Z. Sun, Z. Liu, Y. Zang, Y. Cao, X. Dong, T. Wu et al. SEAgent: Self-evolving computer use agent with autonomous learning from experience. *arXiv preprint arXiv:2508.04700*, 2025. doi: 10.48550/arXiv.2508.04700 2

[29] J. Z. Tam, P. Grosset, D. Banesh, N. Ramachandra, T. L. Turton, and J. Ahrens. InferA: A smart assistant for cosmological ensemble data. In *Proc. ACM/IEEE SC Workshops*, pp. 20–28, 2025. doi: 10.1145/3731599.3767342 2

[30] K. Tang, K. Ai, J. Han, and C. Wang. TexGS-VolVis: Expressive scene editing for volume visualization via textured Gaussian splatting. *IEEE Trans. Vis. Comput. Graph.*, 32(1):933–943, 2026. doi: 10.1109/TVCG.2025.3634643 2

[31] Z. Wu, C. Han, Z. Ding, Z. Weng, Z. Liu, S. Yao et al. OS-Copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024. doi: 10.48550/arXiv.2402.07456 2

[32] Y. Yang and S. Oney. Vizcode: A practical real-time tool for in-class computer programming tutoring. In *Proc. ACM Learning@Scale*, pp. 544–546, 2024. doi: 10.1145/3657604.3664716 2

[33] Y. Yang, A. G. Zhang, S. Oney, and A. Y. Wang. Spark: Real-time monitoring of multi-faceted programming exercises. In *Proc. IEEE VL/HCC*, pp. 81–92, 2025. doi: 10.1109/VL-HCC65237.2025.00018 2

[34] Z. Yang, Z. Zhou, S. Wang, X. Cong, X. Han, Y. Yan et al. MatPlotAgent: Method and evaluation for LLM-based agentic scientific data visualization. In *Proc. ACL Findings*, pp. 11789–11804, 2024. doi: 10.18653/v1/2024.findings-acl.701 2

[35] C. Zhang, S. He, J. Qian, B. Li, L. Li, S. Qin et al. Large language model-brained GUI agents: A survey. *arXiv preprint arXiv:2411.18279*, 2024. doi: 10.48550/arXiv.2411.18279 1

[36] C. Zhang, H. Huang, C. Ni, J. Mu, S. Qin, S. He et al. UFO^2: The desktop agentOS. *arXiv preprint arXiv:2504.14603*, 2025. doi: 10.48550/arXiv.2504.14603 1, 2

[37] C. Zhang, L. Li, S. He, X. Zhang, B. Qiao, S. Qin et al. UFO: A UI-focused agent for windows OS interaction. In *Proc. ACL: Human Language Technologies*, pp. 597–622, 2025. doi: 10.18653/v1/2025.naacl-long.26 1, 2

[38] C. Zhang, L. Li, H. Huang, C. Ni, B. Qiao, S. Qin et al. UFO^3: Weaving the digital agent galaxy. *arXiv preprint arXiv:2511.11332*, 2025. doi: 10.48550/arXiv.2511.11332 1, 2