AgentGUI: An Interface for Observing and Steering Long-Running AI Agents

Xuan Zhao **Jiwoong Sohn** **Qinyue Zheng** **Michael Moor**

ETH Zürich

arXiv:2607.26300v1 [cs.CL] 28 Jul 2026

Abstract

AI agents are increasingly adept at tackling complex, long-running tasks. With the rapid surge of autonomous capabilities, human oversight is systematically lagging behind due to limited human-centered interfacing. Aiming to address this, we introduce AgentGUI, a user-friendly, locally hosted GUI for seamlessly observing and steering AI agents amid multiple concurrent, long-running sessions. AgentGUI features 1) rich agent trajectory visualizations, 2) effective manual and automated steering, and 3) integration with and coordination between open-source and frontier agent frameworks. A controlled user study demonstrates statistically significant reduction in the time it takes to identify key elements from agent traces (38% faster, $p = 0.023$). In a preliminary experiment, AgentGUI's automated drift prevention feature raises the task completion rate of small local agents by as high as 34 pp across a 0.8B–9B model ladder (N=50 runs per model). AgentGUI is publicly available through its project website[1] and open-source repository[2], along with a demo video[3].

1 Introduction

Tool-using LLM agents have greatly advanced in recent years, often capable of running for hours or days on end. These agents can autonomously tackle tasks that used to require full human attention, such as completing software engineering tasks end-to-end (Wang et al., 2025; Huang et al., 2024; Chan et al., 2025), generating research hypotheses and running experiments (Lu et al., 2026; Gottweis et al., 2026; Kon et al., 2025; Jiang et al., 2025), and computer use (Anthropic, 2024; OpenAI, 2025).

Greater capabilities bring about greater complexity. A long-running agent leaves behind a messy transcript of interleaved reasoning steps, tool calls, and file accesses. For a human, supervising this agent's action can be formidable, and spending time studying agent traces partly defeats the time-savings promised by delegation. Furthermore, not spending the effort to understand agent action hinders the opportunity to improve and customize agent behavior by human intervention.

Therefore, we introduce AgentGUI, an open-source, local GUI for observing and managing fleets of long-running AI agents. It combines real-time trajectory visualization, manual and automated steering, and multi-agent coordination across agent harnesses in a single interface. We demonstrate how AgentGUI helps users understand agent trajectories both faster and more accurately, and proof-of-concept result for the automated audit feature. Together, AgentGUI provides a human-centered approach to keep agents easily manageable in personal workflows.

2 Related Work

2.1 LLM Agents and Harness

LLM agents are systems that use language models to reason, select tools, and take actions over multiple turns, often through interleaved reasoning traces (Yao et al., 2023) or executable code (Wang et al., 2024). The software environment enabling the agent to manage its context and act over long horizons is increasingly referred to as the harness. Notable frameworks include SWE-agent (Yang et al., 2024), OpenHands (Wang et al., 2025), OpenClaw (Steinberger and the OpenClaw community, 2026), and Hermes (Nous Research, 2026; Teknium et al., 2024).

2.2 Observing Agent Trajectories

Humans are not the only ones to struggle to read machine-centered agent transcripts. Frontier LLMs, tasked with debugging agent behavior from raw

[1] https://agent-gui-project.github.io/

[2] https://github.com/eth-medical-ai-lab/agent-gui

[3] https://youtube.com/watch?v=GSDyxN1gTF0

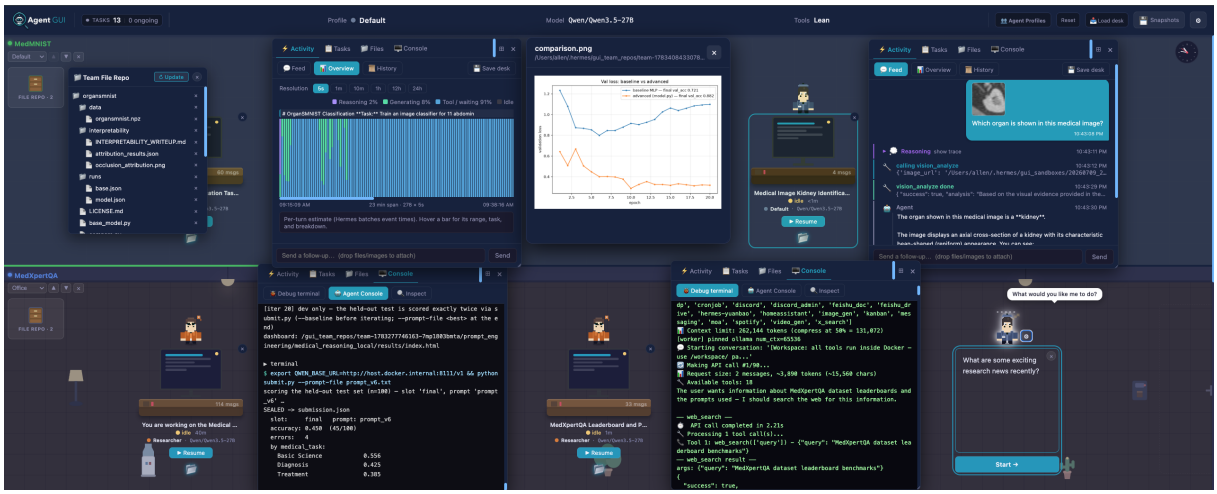


Figure 1: The AgentGUI dashboard. It presents all agent sessions and can simultaneously show different levels of agent details. The upper panel shows a team of agents working on image analysis tasks, featuring shared file storage, execution wall-time, file preview, and activity feed. The lower panel shows a team of agents working on medical question-answering tasks, featuring terminal actions and API-call-level messages.

traces, localize only a small fraction of errors on the TRAIL benchmark (Deshpande et al., 2025). A first line of work improves the visualization and diagnosis of trajectories. Examples include Agentflow, which renders a live coding-agent session as a branching graph of tool calls and subagent activity (Patole, 2026), and IBM's Agent Trajectory Explorer (Desmond et al., 2025). For debugging, AgentDiagnose extracts and visualizes trajectory statistics (Ou et al., 2025), and AgentLens scales such analytics to multi-agent simulation histories (Lu et al., 2025). These systems make agent behavior more legible, but provide no mechanism for intervening in or redirecting an ongoing run.

2.3 Steering Agents at Runtime

A second line of work contributes to steering agent behavior. Operating on the AutoGen framework (Wu et al., 2024), AutoGen Studio provides a no-code builder and debugger for multi-agent workflows (Dibia et al., 2024), and AGDebugger adds interactive message editing and resets for steering multi-agent teams (Epperson et al., 2025). Magentic-UI supports co-planning and co-tasking between human and agent (Mozannar et al., 2025). ResearStudio streams a deep-research agent's plan and actions to a live interface where the user can intervene (Yang and Weng, 2025).

Despite prior works, observability, steering, and collaboration between open-harness agents rarely co-occur. Trajectory visualizers and debuggers often target explainability but leave out steering, whereas steering interfaces are often bound to a specific harness.

3 AgentGUI

AgentGUI targets researchers and developers who run fleets of agents across open-source harnesses and need to monitor their behavior, detect and correct agent drift, and coordinate collaboration among them. We describe AgentGUI through a user's journey, from managing agents on the dashboard to observing and steering individual agents, and then highlight several notable engineering features.

3.1 Agent Configuration and Collaboration

The main dashboard (Fig. 1) presents running agents as workers in a pixel-art office, with each agent assigned its own desk. Starting a task is as simple as selecting an empty desk and entering a prompt, and optionally a single drag-and-drop action to provide context files. Primary support targets Hermes agents, and experimental support covers the Claude Agent SDK (Anthropic, 2025a), which exposes the agent loop behind Claude Code (Anthropic, 2025b). Users can customize agent profiles, including their memory, system prompts, model configurations, and tool settings, directly from the GUI. Agents working on the same task can be grouped into teams and collaborate through artifact sharing, enabling use cases e.g. a powerful agent reviewing and refactoring the work of a long-running local agent.

Figure 2: Per-desk views of a single run. **a.** Activity feed: a live event timeline with visual separation between different types of actions. **b.** Overview: a wall-clock timeline of the trajectory. **c.** Debug terminal: per-call API and token telemetry. **d.** Agent console: the terminal/code execution stream.

3.2 Agent Observation

An agent work desk consists of 4 major tabs, visualizing agent activity, task definition, workspace files, and debug messages. Agent trajectory is displayed in 4 minor tabs with varying levels of detail. The activity feed (Fig. 2a) decomposes agent traces into reasoning and generation content, and tool requests and responses, with distinct visual cues for skimming. The overview feed (Fig. 2b) renders the time agent spends on each task. The console tab includes a comprehensive turn-level debug log with token telemetry (Fig. 2c), as well as a lightweight, code-centered terminal console (Fig. 2d). The latter demonstrates only agent actions in code execution, which, coming from experience of heavy coding agent users, provide valuable insights and quick comprehension of agent actions. Sub-agents spawned by Hermes agent's delegate tool would be visualized as mini expandable avatars next to the main agent's desk, with traces available. Additionally, the files tab offers one-click previews of agent work directories and artifacts.

3.3 Agent Steering

A human can intervene and redirect the agent's current turn by direct input (Fig. 3a). One can also modify the agent's task definition from the task tab, which will be reviewed by the agent after the current turn completes. One implicit steering method includes switching the agent profile during the same task's execution, so a more powerful model could take over a stalled task, or a local model could take over a monitoring task.

An LLM-powered automated manager audits agent trajectory and artifacts to steer when necessary (Fig. 3b), intervening upon a user-initiated audit, or on a configurable interval when a desk is idle and not yet marked solved. The manager first decomposes the agent task into verifiable criteria, then gathers evidence from agent transcript and workspace files, and finally judges each criterion against the evidence and leaves an audit report. A session is either marked solved, or prompted to resume by reading the manager's audit output.

3.4 Engineering Highlights

We took extensive engineering measures to ensure accessibility for users with varying setups, and security. AgentGUI provides quick-start instructions for running agents from entirely on the user's hardware using Ollama (Ollama, 2023), to using remote GPU servers and hosted inference options. Inherited from Hermes implementation, each desk owns a persistent Docker sandbox that isolates agents from the host, and from one another. A locally hosted FastAPI server executes each agent turn in an isolated worker process and streams events to a React frontend over WebSockets. Claude Code agents can instead use the user's existing Claude

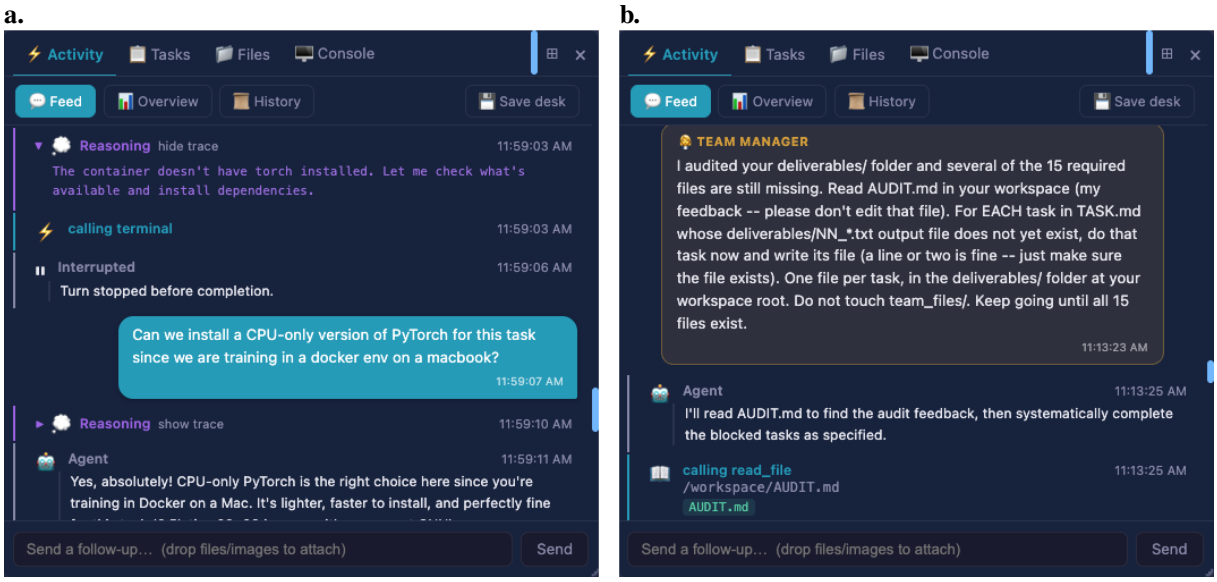


Figure 3: Manual and automated steering channels in AgentGUI. **a.** User intervention: user message interrupts and redirects the agent's current turn. **b.** Manager audit: a manager audit detects drift and auto-resumes the agent with corrective feedback.

subscription without requiring a separate API key and billing. Desks can be saved and fully restored, including both trajectories and workspaces snapshots for portability and sharing.

4 System evaluation

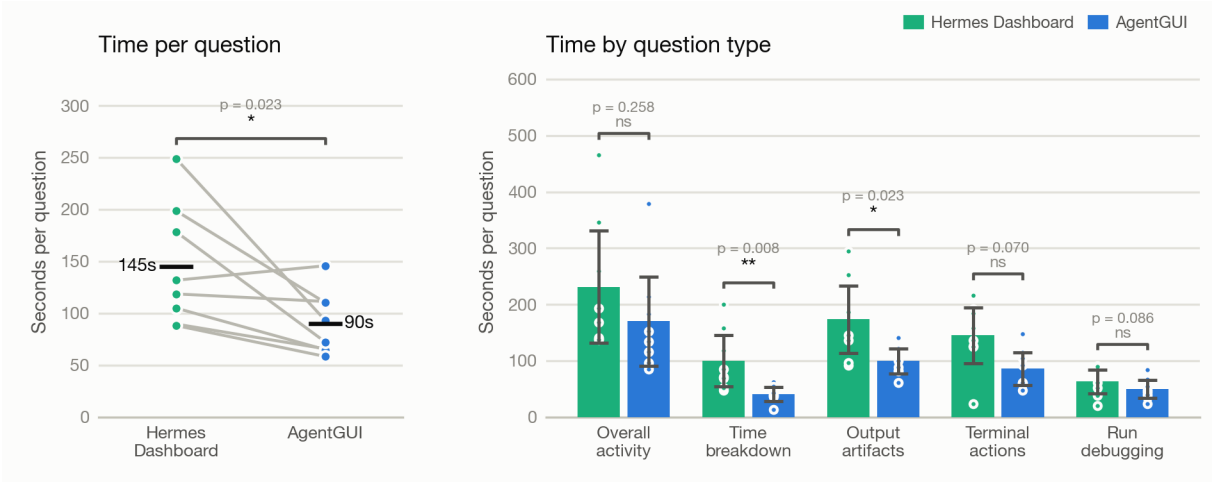
4.1 User Study: Trajectory Comprehension

Does AgentGUI help users better understand agent trajectories? We measured the time and accuracy with which $N{=}8$ MSc/PhD students (non co-authors of the manuscript) in quantitative fields identified key information from agent trajectories. The baseline compared against is Hermes Dashboard (v0.16.0), a native visualization tool for Hermes Agent trace.

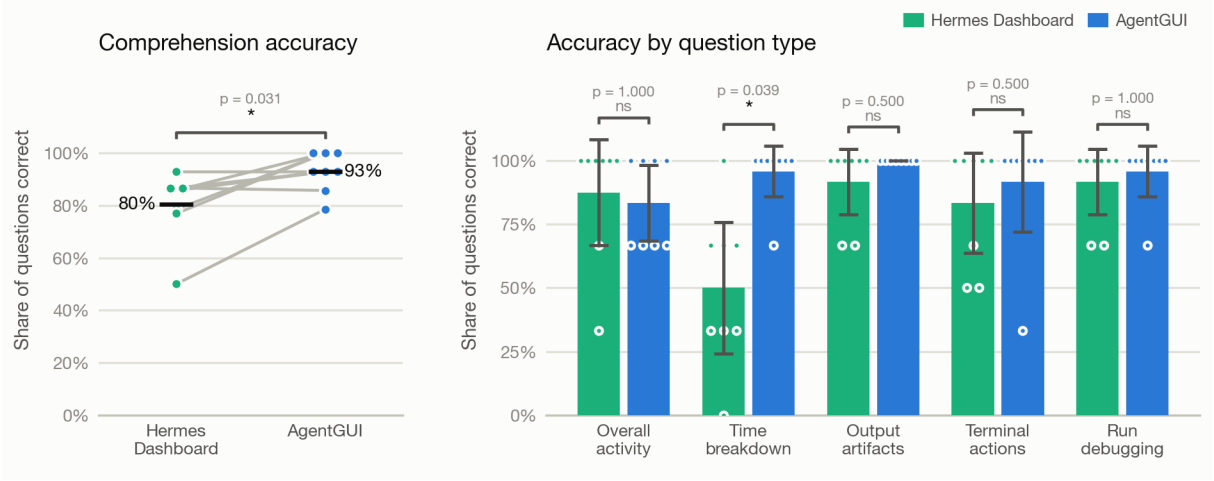
Design We defined two research tasks: training a CNN on OrganSMNIST (Yang et al., 2023) against a frozen scorer, and iterating a system prompt for answering MedXpertQA (Zuo et al., 2025) questions. For each task, we generated two rollouts with a Hermes agent on a Qwen3.5-27B backbone (Qwen Team, 2026) (Appendix Table 2), so that no comparison hinges on a single idiosyncratic trace. For each rollout, we authored 14–15 questions, around 2–3 each on the agent's overall activity, time breakdown, output artifacts, terminal actions, and run debugging (example questions in Fig. 4c). Each participant answered questions on two trajectories, one from each research task, one viewed in AgentGUI and one in the dashboard, so that memorization could not carry over between interfaces. Interface order and rollout assignment were counterbalanced: four participants saw AgentGUI first, and each rollout was seen by exactly two participants per interface (assignment in Appendix Fig. 7). To reduce noise from unfamiliarity with either visualizer, each participant was given five minutes of UI exploration, and the quiz included guidance on the location of relevant information. Both interfaces exposed the same information categories (Appendix Table 1). The study thus measures the interfaces' support for trajectory comprehension and information lookup, rather than familiarity with a particular UI.

Results Participants completed questions 38% faster with AgentGUI than with the baseline interface, taking on average 90 s rather than 145 s per question ($p = 0.023$, Appendix B, Fig. 4). Completion time is observed to reduce across all five question types (Fig. 4a), with statistical significance in time breakdown questions (59 s faster; $p = 0.008$) and output artifacts questions (74 s faster; $p = 0.023$). Accuracy improved to 93% from 80% ($p = 0.031$, Fig. 4), although this absolute increase could be impacted by one participant scoring a low (50%) score on the baseline interface. Importantly, accuracy with AgentGUI was not significantly worse in any question type (Fig. 4b).

a.



b.



c.

Overall activity	Time breakdown	Output artifacts	Terminal actions	Run debugging
The task prompt prescribed a 7-step workflow. Did the agent adhere to it?	How much time approximately did the agent spend on tool calls?	What are the default learning rates defined in base_model.py and model.py?	How many epochs did each successful model.py run request, and how many actually ran?	The agent's very first model.py run failed immediately. What was the error?

Figure 4: User-study results (N=8, within-participant). **a.** Time: mean seconds per question, per participant and interface (left; grey lines connect a participant's two sessions, black ticks are interface means) and by question type (right). **b.** Accuracy: share of questions answered correctly, same layout. Brackets are exact paired sign-flip permutation tests on within-participant deltas; bars are means across sessions, error bars 95% t-CIs across participants, dots individual sessions. **c.** An example question for each type of question.

We found no evidence of a speed–accuracy trade-off (left panels of Fig. 4a and Fig. 4b). Users report statistically significant (Fig. 5) less mental demand, frustration, and effort (adapted from the NASA-TLX (Hart and Staveland, 1988)) when using AgentGUI.

4.2 Automated Steering against Drift

Design We ran a proof-of-concept experiment to test whether the automated manager can improve task completion rate. The task simulates an agent navigating through a synthetic patient chart consisting of 98 files, backed by a local open-source

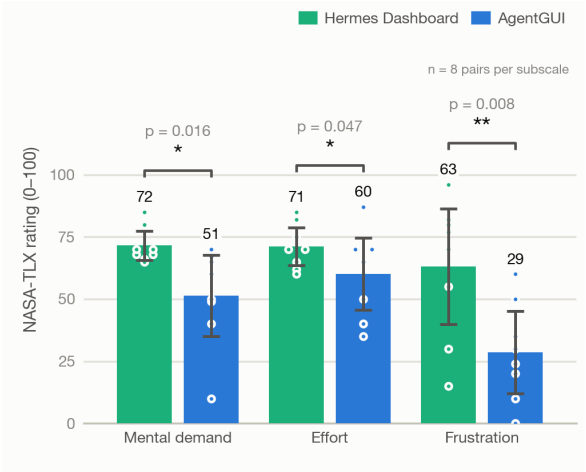


Figure 5: Self-reported workload under AgentGUI and the Hermes Dashboard (N=8). Lower scores indicate lower workload; dots show participants, bars show means, and error bars show 95% confidence intervals. Brackets report exact paired permutation-test p-values.

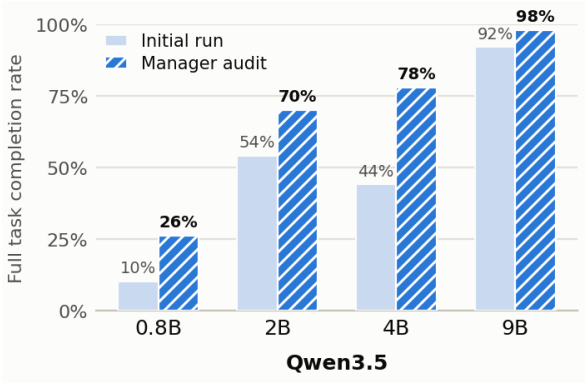


Figure 6: Effect of one manager audit on task completion across Qwen3.5 worker sizes (N=50 runs per model). Solid bars show completion before the audit, while hatched bars show completion after steering.

model to preserve data privacy. The agent is asked to create 15 deliverables for 15 data aggregation tasks. A programmatic scorer checks the presence of the deliverables, and a manager (Qwen3.5-27B) audits the workspace if the agent did not complete the task. The workspace is scored again when the agent addresses the manager's comments.

Results Across N=50 runs per model size (Fig. 6), initial task completion rates vary by model capability, but increases with model size. The 4B's slightly lower unaided completion likely reflects its tendency to address outputs by absolute path, which the sandbox write-guard rejects, so some of its deliverables fail to land until the manager audit flags the gap and prompts a corrected rewrite. After a single audit, completion recovers a clean monotonic ordering in model size and improves at every scale—evidence that the benefit is general, not tied to any one worker. The lift is largest where the worker leaves partial work, while saturating near weakest and strongest models: 10%→26% (0.8B), 54%→70% (2B), 44%→78% (4B), and 92%→98% (9B).

Overall, manager tokens are much cheaper than agent execution, making up no more than 1% of the total token usage per model (Fig. 8). We observe additionally that the smallest local models spend more tokens than the larger, more capable ones on this task. Larger models, e.g. the 9B model, has higher first-shot completion rate, invoking the manager not so frequently and uses fewer tokens.

5 Conclusion

We presented AgentGUI, an open-source (MIT License), locally hosted GUI for observing, steering, and coordinating fleets of long-running AI agents. User study and proof-of-concept experiment attests to the effectiveness of observability, and to some extent drift-prevention. Observability is a precondition for trusting delegated work: an interface that makes agent activity legible and correctable in place turns opaque transcripts into outcomes a human can verify, steer, and rely on. We release AgentGUI publicly and envision it serving as a substrate for future work on live supervision, broader harness support, and automated audits that target open-ended quality.

This study has several limitations. First, the small and rather homogeneous user study limits the statistical test power and may not be fully generalizable. Second, the automated manager steering experiment focuses only on quantitative completion given small local model constraint. Finally, we have only presented experiments to benchmark potential observability and automated steering improvement AgentGUI offers. Experiments that require live human steering, preferably on a multi-agent scale and emphasize qualitative evaluation, would be a both interesting and important future direction to explore.

Acknowledgments

We thank the participants of our user study for their time and engagement.

Ethics Statement

Participation in the user study was voluntary. All participants gave informed consent through an in-app consent screen before beginning, and were free to withdraw at any time. The study posed minimal risk, involved no vulnerable populations, and collected no personally identifiable information beyond anonymized task responses and timing measurements. Participants received no compensation.

References

Anthropic. 2024. 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. Accessed 2026-07-10.

Anthropic. 2025a. Building agents with the Claude Agent SDK. https://www.anthropic.com/engineering/building-agents-with-the-claude-agent-sdk. Accessed 2026-07-10.

Anthropic. 2025b. Claude Code. Software.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. 2025. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *The Thirteenth International Conference on Learning Representations*.

Darshan Deshpande, Varun Gangal, Hersh Mehta, Jitin Krishnan, Anand Kannappan, and Rebecca Qian. 2025. Trail: Trace reasoning and agentic issue localization. *Preprint*, arXiv:2505.08638.

Michael Desmond, Ja Young Lee, Ibrahim Ibrahim, James M. Johnson, Avirup Sil, Justin MacNair, and Ruchir Puri. 2025. Agent trajectory explorer: Visualizing and providing feedback on agent trajectories. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(28):29634–29636.

Victor Dibia, Jingya Chen, Gagan Bansal, Suff Syed, Adam Fourney, Erkang Zhu, Chi Wang, and Saleema Amershi. 2024. AUTOGEN STUDIO: A no-code developer tool for building and debugging multi-agent systems. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 72–79, Miami, Florida, USA. Association for Computational Linguistics.

Will Epperson, Gagan Bansal, Victor C Dibia, Adam Fourney, Jack Gerrits, Erkang (Eric) Zhu, and Saleema Amershi. 2025. Interactive debugging and steering of multi-agent ai systems. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25, New York, NY, USA. Association for Computing Machinery.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, Felix Weissenberger, Alessio Orlandi, Dan Popovici, Anil Palepu, Keran Rong, Ryutaro Tanno, Khaled Saab, Fan Zhang, Jacob Blum, Andrew Carroll, Kavita Kulkarni, Nenad Tomašev, and 32 others. 2026. Accelerating scientific discovery with co-scientist. *Nature*, 655(8122):487–496.

Sandra G. Hart and Lowell E. Staveland. 1988. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In Peter A. Hancock and Najmedin Meshkati, editors, *Human Mental Workload*, volume 52 of *Advances in Psychology*, pages 139–183. North-Holland.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024. Mlagentbench: evaluating language agents on machine learning experimentation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. 2025. Aide: Ai-driven exploration in the space of code. *Preprint*, arXiv:2502.13138.

Patrick Tser Jern Kon, Jiachen Liu, Qiuyi Ding, Yiming Qiu, Zhenning Yang, Yibo Huang, Jayanth Srinivasa, Myungjin Lee, Mosharaf Chowdhury, and Ang Chen. 2025. Curie: Toward rigorous and automated scientific experimentation with ai agents. *Preprint*, arXiv:2502.16069.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. 2026. Towards end-to-end automation of ai research. *Nature*, 651(8107):914–919.

Jiaying Lu, Bo Pan, Jieyi Chen, Yingchaojie Feng, Jingyuan Hu, Yuchen Peng, and Wei Chen. 2025. Agentlens: Visual analysis for agent behaviors in llm-based autonomous systems. *IEEE Transactions on Visualization and Computer Graphics*, 31(8):4182–4197.

Hussein Mozannar, Gagan Bansal, Cheng Tan, Adam Fourney, Victor Dibia, Jingya Chen, Jack Gerrits, Tyler Payne, Matheus Kunzler Maldaner, Madeleine Grunde-McLaughlin, Eric Zhu, Griffin Bassman, Jacob Alber, Peter Chang, Ricky Loynd, Friederike Niedtner, Ece Kamar, Maya Murad, Rafah Hosn, and Saleema Amershi. 2025. Magentic-ui: Towards human-in-the-loop agentic systems. *Preprint*, arXiv:2507.22358.

Nous Research. 2026. Hermes Agent. https://github.com/NousResearch/hermes-agent. Software, accessed 2026-07-28.

Ollama. 2023. Ollama. https://github.com/ollama/ollama. Accessed: 2026-07-28.

OpenAI. 2025. Introducing Operator. https://openai.com/index/introducing-operator/. Accessed 2026-07-10.

Tianyue Ou, Wanyao Guo, Apurva Gandhi, Graham Neubig, and Xiang Yue. 2025. AgentDiagnose: An open toolkit for diagnosing LLM agent trajectories. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 207–215, Suzhou, China. Association for Computational Linguistics.

Simon Patole. 2026. Agent Flow. Software.

Qwen Team. 2026. Qwen3.5: Accelerating productivity with native multimodal agents.

Peter Steinberger and the OpenClaw community. 2026. OpenClaw: Your own personal AI assistant. `https://github.com/openclaw/openclaw`. Accessed 2026-07-10.

Ryan Teknium, Jeffrey Quesnelle, and Chen Guang. 2024. Hermes 3 technical report. *Preprint*, arXiv:2408.11857.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better LLM agents. In *Forty-first International Conference on Machine Learning*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 5 others. 2025. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*.

Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, and Bingbing Ni. 2023. Medmnist v2 - a large-scale lightweight benchmark for 2d and 3d biomedical image classification. *Scientific Data*, 10(1):41.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Linyi Yang and Yixuan Weng. 2025. ResearStudio: A human-intervenable framework for building controllable deep research agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 896–905, Suzhou, China. Association for Computational Linguistics.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Yuxin Zuo, Shang Qu, Yifei Li, Zhang-Ren Chen, Xuekai Zhu, Ermo Hua, Kaiyan Zhang, Ning Ding, and Bowen Zhou. 2025. MedxpertQA: Benchmarking expert-level medical reasoning and understanding. In *Forty-second International Conference on Machine Learning*.

A User Study Setup

Each participant completed two quiz blocks. Interface order and rollout assignment were counterbalanced: four of the eight participants started with AgentGUI, and each rollout was read by exactly two participants per interface.

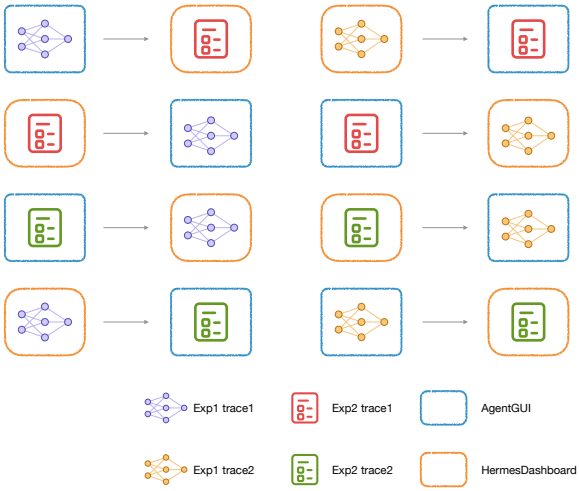


Figure 7: Participant–block assignment. Each pair is one participant; the two cards are their two quiz blocks, a model-training rollout (network icon) and a prompt-engineering rollout (clipboard icon), placed in the column of the interface they were read in, with the arrow pointing from the participant's first block to their second.

B Statistical Testing and Analysis

Per rollout, we compute accuracy (the share of the rollout's questions answered correctly) and the seconds it took a participant to answer each question. Each participant contributes one value per interface, and the unit of analysis is the within-participant difference $\Delta_i = \text{AgentGUI}_i - \text{Dashboard}_i$, which cancels between-person variance in skill and reading speed.

Significance is assessed with the exact two-sided sign-flip permutation test on the mean delta: under the null hypothesis the interface labels are exchangeable within a participant, so all $2^8{=}256$ sign assignments of the observed deltas are equally likely, and

$$p \;=\; \#\{\mathbf{s} \in \{\pm 1\}^8 : \left|\tfrac{1}{8}\textstyle\sum_i s_i \Delta_i\right| \geq |\bar{\Delta}|\} \,/\, 2^8,$$

i.e., the share of sign assignments whose mean is at least as extreme as the one observed. Each contrast is reported with a 95% paired-t confidence interval on $\bar{\Delta}$. One interrupted question was dropped (an 833 s timer, due to the participant being interrupted by unforeseen circumstances); one question was excluded for all participants after a post-hoc review found it has no correct answer; two questions' accepted-answer sets were widened to two defensible readings; and one participant's first three questions are flagged for a hardware issue (kept in the primary analysis, excluded in a sensitivity variant).

	AgentGUI	**Hermes Dashboard**
Message transcript	Activity feed	Sessions tab
Tool calls + results	Typed cards	Inline blocks
Time breakdown	Overview chart	Message timestamps
Files written	Files tab	In-app file directory
Terminal output	Console tab	Inline blocks

Table 1: Trajectory information available in the two interfaces used in the user study. Both interfaces exposed the same five information categories but differed in how they organized and presented them.

Task	**Trace**	**Len**	**Behavioural Signature**
Training	`rollout 1`	23 m	PASS (test 0.74); `pip` disk-full recovery; scored the sealed test twice
Training	`rollout 2`	20 m	PASS (0.72); uses a todo-list planner; 3 failed training runs
Prompt eng.	`rollout 1`	19 m	*overfit*: dev 1.00 → test 0.42; final prompt hard-codes dev answers, violating the prompt
Prompt eng.	`rollout 2`	40 m	*honest*: catches itself hard-coding mid-run and rewrites; dev 0.40, test 0.35 → 0.45

Table 2: Description of four agent rollouts used to generate study questions.

C Cost Analysis For Manager Steering

Fig. 8 reports token costs of the experiment in Section 4.2. Agent tokens are the usage the serving endpoint reported for each completion, accumulated per desk. Manager calls are tokenized using the tokenize endpoint on vllm.

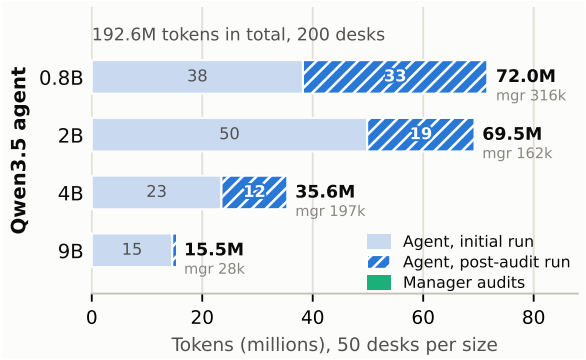


Figure 8: Token budget of the steering experiment, by agent size (N=50 desks each). Bars are stacked: the agent's initial unaided run, the agent's run after the Manager's nudge, and the Manager's audit calls. The Manager segment is 0.19–0.56% of each bar and is therefore barely visible; its value is printed at the bar end.