Graphectory Viewer: A Tool for Process-Centric Analysis of Agentic Software Trajectories

Charlie Jyu
University of Illinois Urbana-Champaign
USA
cbjyu2@illinois.edu

Shuyang Liu
University of Illinois Urbana-Champaign
USA
sl225@illinois.edu

Reyhaneh Jabbarvand
University of Illinois Urbana-Champaign
USA
reyhaneh@illinois.edu

Abstract

We present Graphectory Viewer, a web-based tool for interactive, process-centric analysis of software-agent trajectories. The tool operationalizes the Graphectory taxonomy introduced in our prior work by transforming raw trajectories into phase-aware graphs and aggregate summaries that can be explored in the browser. Graphectory Viewer supports raw trajectories from multiple agent frameworks, on-demand graph construction, node-level inspection, search and filtering over large trajectory sets, comparison of successful and failed runs, and a Sankey-style summary of phase transitions. The tool is intended for researchers and practitioners who need to inspect, compare, and reuse large trajectory corpora beyond outcome-centric evaluation. Graphectory Viewer is open source and is released together with documentation, precomputed graphs, and the trajectory corpus used in our prior large-scale study.

Code: https://github.com/Intelligent-CAT-Lab/Graphectory
Dataset: https://zenodo.org/records/17364210
Live Demo: https://graphectory-viewer-demo.vercel.app/
Screencast: https://youtu.be/Hc4hnfRkuxc

1 Introduction

Software engineering agents are increasingly used to solve real-world programming tasks, including bug localization, code editing, and test generation. Systems such as SWE-agent[21], OpenHands[15], and mini-SWE-agent[14] demonstrate that agents can autonomously resolve issues across complex codebases with limited human intervention. However, as these systems become more capable, their executions become correspondingly harder to understand.

Most existing inspection tools present trajectories as raw logs or step-by-step transcripts, which are useful for replaying individual runs but provide limited support for higher-level analysis. In practice, researchers and practitioners need to answer questions such as: Did the agent correctly localize the bug before editing? Did it repeatedly explore the same files? Did it validate its patch effectively? How do successful and failed runs differ in their strategies? These questions require moving beyond linear logs to structured, comparative views of agent behavior.

Prior work introduced Graphectory [12], a process-centric representation that maps low-level agent actions into semantic phases and graph structures, enabling analyses beyond outcome-centric evaluation. While effective for large-scale, automated studies, it is primarily used as an offline framework. Applying it often requires writing scripts, inspecting serialized graphs, and interpreting quantitative results, limiting accessibility.

We present Graphectory Viewer, an interactive, browser-based tool that makes this representation directly usable. Given a raw trajectory, the tool parses actions, assigns context-sensitive phase labels, and renders a phase-colored directed graph. Users can inspect the thought, action, and observation behind each node, navigate between repeated occurrences, and suppress low-signal commands to reduce visual clutter. A companion Sankey view summarizes phase transitions across trajectories, enabling the transition from single-run inspection to corpus-level comparison. The tool also includes precomputed graphs and trajectories from the 4,000-run corpus in [12], supporting immediate use for replication and further analysis.

In summary, this paper makes the following contributions:

- **Interactive process-centric trajectory analysis** (§3): We introduce Graphectory Viewer, a browser-based system that transforms raw agent trajectories into interactive, phase-aware graphs with coordinated views spanning node-level information, graph structure, and aggregate phase-transition summaries for debugging and pattern analysis.
- **Unified analysis across agent frameworks** (§4): The tool supports trajectories from SWE-agent, OpenHands, and mini-SWE-agent via a unified process-centric representation, enabling consistent cross-framework comparisons and lightweight extensions to additional agent scaffolds via adaptable parsing and phase-mapping rules.
- **An open and extensible analysis artifact** (§5): We release an open-source workflow with reusable graph-generation pipelines, precomputed graph artifacts, and large-scale trajectory datasets to support reproducible, extensible, and large-scale studies of software-agent behavior.

2 Related Work

Software Agents. Recent surveys document the expansion of agentic software engineering, the growing importance of tool use, and the difficulty of rigorously characterizing agent behavior beyond task-level success rates [1, 9, 10, 18, 20, 22]. Complementary evaluation frameworks such as CIFE, agentic rubrics, agent assessment frameworks, and benchmark checklists examine whether agents follow instructions, align with intended goals, or are evaluated under rigorous task and reward designs [2, 8, 11, 17, 23].

Visualization Tools. Existing tools for inspecting agentic systems span several adjacent goals, but differ in representation and intended use. Many tools provide practical ways to replay or inspect individual runs, but they remain largely tied to raw traces and framework-local abstractions rather than a shared process-centric representation [6, 14, 19]. AgentDiagnose provides a solid base for generalist agent systems, but lacks specialized phase assignments

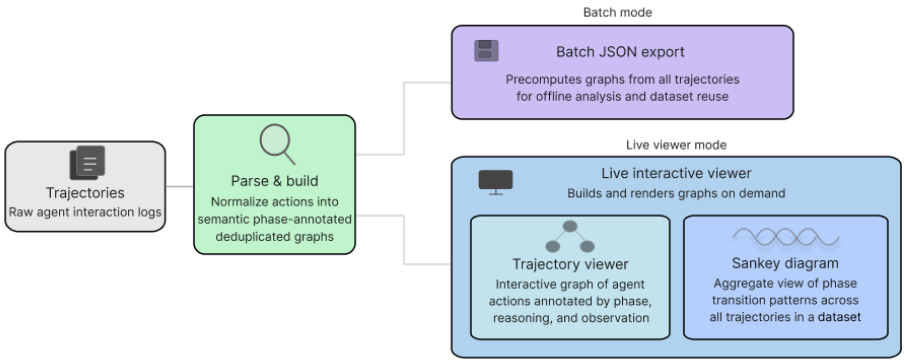

Figure 1: High-level architecture of Graphectory Viewer, showing the flow from raw trajectories through parsing, phase mapping, graph construction, and interactive browser-based visualization.

to actions and relies on difficult to understand embedding visualizations [16]. AgentLens and AGDebugger visualize LLM-based agentic system behaviors but focus on real time debugging and intervention, not post-hoc analysis [7, 13].

3 Tool Overview

The main browser interface is organized around a navigation sidebar and a main visualization pane. As shown in Figure 3, users begin with the *Data source* panel by supplying the trajectories and report. The sidebar then populates a searchable instance list with status badges and trajectory counts.

For trajectory-level analysis, selecting an instance from the sidebar loads a Graphectory into the main canvas. Figure 2 shows a graphectory of SWE-agent using Deepseek to solve issue astropy-13033. The agent begins (step 0) by creating a script to replicate the issue from the problem description and running it with python. Steps 0-2 across the first four purple nodes represent the localization steps the agent takes to find the source of the issue. The second node (representing steps 0,1,4) is colored both as purple for localization and blue for validation because the command it represents occurs across multiple trajectory steps that belong to different phases. Step 3 shows the agent patching the code after localizing the issue then validating its correctness in step 4 by running the issue replication script. After thinking in step 5 about step 4's validation results, the model submits, briefly cleaning up the code in steps 7 and 8 before finally submitting. Solid edges show the main execution flow, while dashed blue edges represent intra-step links between multiple commands issued within the same trajectory step. Arrowhead size of an edge shows the length of the thought in the node it points to.

Clicking a node opens the inspection sidebar, which shows the raw step's thought-action-observation tuple. When a node is referenced by multiple steps, the interface provides tabs for each step. The graphectory view includes several display options. Switching *Verbose node labels* leaves only the action name, reducing clutter. *Exclude quotes in thought length* changes how thought length is measured so that copied or quoted text in the agent's think step does not dominate the visual encoding. *Filter cd (show hat)* compresses repetitive leading directory-change commands. This can be especially useful for agent frameworks that repeatedly return from the repository root into the same testbench directory before running a script. *Show observation indicators* highlights where environment feedback may have influenced the next action, but this signal is optional because observations can range from a few tokens to several thousand and can otherwise overwhelm the view. *Unique think nodes* controls whether explicit reasoning steps are merged or separated; this is particularly helpful for agent frameworks where many dedicated thought steps exist but often concern different local issues.

The Phase Sankey Diagram, shown in Figure 3, provides the second major interface mode by summarizing each run as a sequence of meaningful phase transitions. Users can filter the displayed trajectories by status, adjust the maximum number of displayed transitions, suppress low-frequency flows with a minimum-flow slider, and enable or disable entire phase categories.

4 Workflow and Architecture

The workflow centers on transforming raw software-agent trajectories into a unified process-centric graph representation. The backend accepts either trajectory directories or OpenHands-style `output.jsonl` files and converts them into Graphectories through three core stages: parsing, phase assignment, and graph construction.

`commandParser.py` normalizes raw trajectory actions into structured records containing tool names, shell commands, subcommands, arguments, and flags. Unlike framework-specific replay tools that operate directly on raw logs, the parser handles heterogeneous software-agent behaviors such as chained shell commands, framework-specific tool invocations, and editor-style operations. It uses Bash-aware parsing to decompose compound shell actions into semantically meaningful sub-actions before graph construction [5]. This normalization enables trajectories from systems such as SWE-agent and OpenHands to be mapped into a shared intermediate representation despite differences in logging structure and tool APIs.

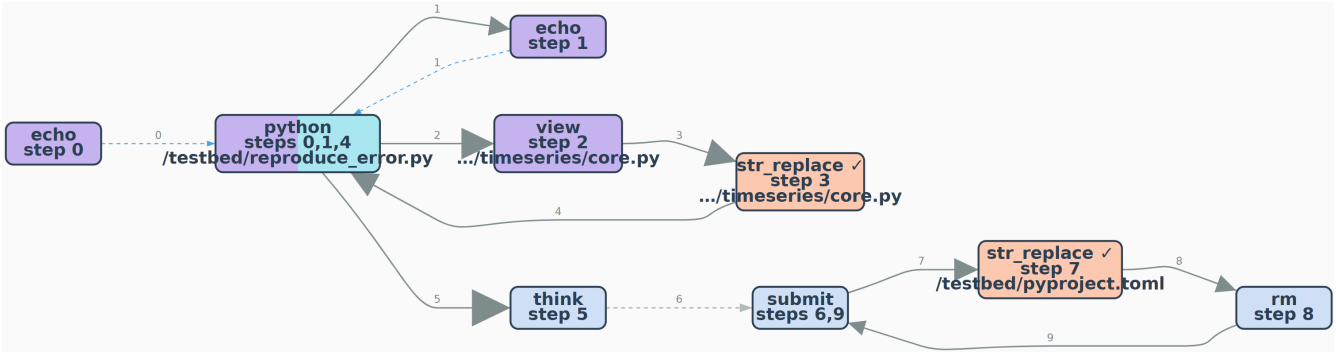

Figure 2: The Graphectory for a SWE-agent trajectory repairing `astropy-13033`. Phase-colored nodes summarize actions, solid edges show execution flow, and dashed blue edges indicate intra-step command links.

`mapPhase.py` then assigns localization, patch, validation, or general labels using context-sensitive heuristics derived from the original Graphectory taxonomy [12]. Phase assignment is not purely syntactic. For example, a command such as `pytest` may indicate localization before any source modification has occurred, but validation after a patch has been introduced.

Finally, `buildGraph.py` constructs the Graphectory itself by merging repeated actions into shared nodes while preserving per-occurrence metadata such as step indices, observations, and thought lengths. This allows repeated behaviors, revisits, and loops to become explicit structural patterns rather than remaining buried in long transcripts. Nodes may accumulate multiple phase labels across different occurrences, enabling the representation of semantically reused actions. The resulting graph contains many types of edges to represent temporal flow and hierarchical edges representing navigation through files and code regions. The same construction pipeline is reused to export serialized graph artifacts for offline analysis.

To analyze aggregate phase transitions, a Sankey-style visualizer is provided. Consecutive repetitions are collapsed and trajectories are converted into compact semantic paths, enabling comparison of higher-level behavioral motifs such as localization loops or premature validation.

5 Utility, Quality, and Availability

Evidence of utility. To evaluate the tool more systematically, we measured it over the released trajectory corpus used in our broader Graphectory workflow. This corpus contains 3,973 trajectories spanning eight collections: four SWE-agent runs and four OpenHands runs over SWE-bench Verified tasks. Across the full corpus, raw trajectories average 35,664 lines and 3,047,249 characters, but the corresponding Graphectories average only 34.53 nodes and 49.60 edges. Even at this scale, the viewer preserves the underlying evidence because each node remains linked to its full thought, action, and observation tuples.

Table 1 shows that this compression is consistent across agents and models, while also revealing meaningful behavioral differences. For example, SWE-agent with DeepSeek-V3 yields especially compact graphs, averaging only 14.99 nodes and a shortest execution path of 7.50 despite raw trajectories averaging 15,895 lines. At the other extreme, SWE-agent with Claude Sonnet 4 produces much larger logs, averaging 144,587 lines and over 10 million characters, yet the corresponding graphs still average only 46.49 nodes. Deduplication is substantial across all collections, reducing effective graph size by 9.60% to 28.22% on average depending on the source. This indicates that the viewer is not merely shrinking trajectories visually, but surfacing repeated actions, revisits, and loops that would otherwise remain buried in long transcripts.

Table 1: Trajectory and graph average statistics across released SWE-agent and OpenHands collections.

Source	N	lines	nodes	dedup.	path
All collections	3973	35,664	34.53	14.71%	16.70
OH Claude-4	500	2,004	59.90	9.60%	30.49
OH DeepSeek-V3	500	5,219	25.98	11.87%	13.75
OH DeepSeek-R1	474	3,681	19.46	11.07%	12.57
OH Devstral	500	12,446	72.53	10.37%	35.66
SA DeepSeek-V3	499	15,895	14.99	20.71%	7.50
SA Claude-4	500	144,587	46.49	11.94%	16.35
SA DeepSeek-R1	500	7,466	12.57	13.71%	6.40
SA Devstral	500	92,314	23.50	28.22%	10.66

Beyond compression, the viewer enabled several recurring behavioral patterns to become apparent across large trajectory collections. We frequently observed agents entering repeated localization or validation loops on difficult tasks, repeatedly revisiting the same actions without transitioning into new phases or making meaningful progress. The process-centric representation additionally revealed that thought length alone is an unreliable proxy for reasoning complexity: agents sometimes execute complex multi-stage commands following only shallow thoughts, while lengthy reasoning traces may precede relatively trivial actions. These patterns were difficult to identify consistently from raw transcripts alone, but become substantially more visible through aggregate graph and phase-transition analysis.

Availability and reuse. The artifact is publicly available under the University of Illinois/NCSA Open Source License. The repository provides installation instructions, usage examples, sample

Figure 3: The Phase Sankey Diagram interface with status filtering, phase chips, and sliders for transition depth and minimum displayed flow.

inputs, graph-export scripts, trajectory-metric scripts, a screencast, and links to the Zenodo archive. A live demo is available at https://graphectory-viewer-demo.vercel.app/. The codebase supports extension to new SWE-agent tool configurations and new trajectory schemas, making it reusable as agent frameworks, model backbones, and orchestration strategies evolve [3, 4].

6 Conclusion

This paper presents Graphectory Viewer, a web-based tool for process-centric analysis of software-agent trajectories that combines graph-based visualization, node-level inspection, and Sankey-style phase summaries.

References

[1] Mohamad Abou Ali, Fadi Dornaika, and Jinan Charafeddine. 2025. Agentic AI: A Comprehensive Survey of Architectures, Applications, and Future Directions. *Artificial Intelligence Review* 59, 1 (2025), 11.

[2] Sreemaee Akshathala, Bassam Adnan, Mahisha Ramesh, Karthik Vaidhyanathan, Basil Muhammed, and Kannan Parthasarathy. 2025. Beyond Task Completion: An Assessment Framework for Evaluating Agentic AI Systems. arXiv:2512.12791 [cs.MA] https://arxiv.org/abs/2512.12791

[3] Anthropic. 2025. How We Built Our Multi-Agent Research System. https://www.anthropic.com/engineering/multiagent-research-system. Article.

[4] Ajay Bandi, Bhavani Kongari, Roshini Naguru, Sahitya Pasnoor, and Sri Vidya Vilipala. 2025. The rise of agentic ai: A review of definitions, frameworks, architectures, applications, evaluation metrics, and challenges. *Future Internet* 17, 9 (2025), 404.

[5] bashlex developers. 2026. bashlex. https://github.com/idank/bashlex. Python parser for Bash. Accessed: 2026-04-28.

[6] Timothy Bula, Saurabh Pujar, Luca Buratti, Mihaela Bornea, and Avirup Sil. 2025. SeaView: Software Engineering Agent Visual Interface for Enhanced Workflow. arXiv:2504.08696 [cs.SE] https://arxiv.org/abs/2504.08696

[7] Will Epperson, Gagan Bansal, Victor Dibia, Adam Fourney, and Saleema Amershi. 2025. Interactive Debugging and Steering of Multi-Agent AI Systems. arXiv:2503.02068 [cs.HC] https://arxiv.org/abs/2503.02068

[8] Sravani Gunnu, Shanmukha Guttula, and Hima Patel. 2025. CIFE: Code Instruction-Following Evaluation. *arXiv preprint arXiv:2512.17387* (2025).

[9] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Transactions on Software Engineering and Methodology* 33, 8 (2024), 1–79.

[10] Zhonghao Jiang, David Lo, and Zhongxin Liu. 2025. Agentic Software Issue Resolution with Large Language Models: A Survey. *arXiv preprint arXiv:2512.22256* (2025).

[11] Christopher Koch and Joshua A. Wellbrock. 2026. Beyond Task Success: An Evidence-Synthesis Framework for Evaluating, Governing, and Orchestrating Agentic AI. arXiv:2604.19818 [cs.SE] https://arxiv.org/abs/2604.19818

[12] Shuyang Liu, Yang Chen, Rahul Krishna, Saurabh Sinha, Jatin Ganhotra, and Reyhaneh Jabbarvand. 2026. Process-Centric Analysis of Agentic Software Systems. *Proc. ACM Program. Lang.* 10, OOPSLA1, Article 163 (April 2026), 28 pages. doi:10.1145/3798271

[13] Jiaying Lu, Bo Pan, Jieyi Chen, Yingchaojie Feng, Jingyuan Hu, Yuchen Peng, and Wei Chen. 2024. AgentLens: Visual Analysis for Agent Behaviors in LLM-Based Autonomous Systems. *IEEE Transactions on Visualization and Computer Graphics* (2024). doi:10.1109/TVCG.2024.3399756

[14] mini-SWE-agent documentation. 2026. Overview. https://mini-swe-agent.com/latest/. Accessed: 2026-04-28.

[15] OpenHands. 2024. OpenHands: Code Less, Make More. https://github.com/All-Hands-AI/OpenHands

[16] 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*. Association for Computational Linguistics, Suzhou, China, 207–215. doi:10.18653/v1/2025.emnlp-demos.15

[17] Mohit Raghavendra, Anisha Gunjal, Bing Liu, and Yunzhong He. 2026. Agentic Rubrics as Contextual Verifiers for SWE Agents. *arXiv preprint arXiv:2601.04171* (2026).

[18] Leon Staufer, Kevin Feng, Kevin Wei, Luke Bailey, Yawen Duan, Mick Yang, A. Pinar Ozisik, Stephen Casper, and Noam Kolt. 2026. The 2025 AI Agent Index: Documenting Technical and Safety Features of Deployed Agentic AI Systems. arXiv:2602.17753 [cs.CY] https://arxiv.org/abs/2602.17753

[19] SWE-agent documentation. 2026. Trajectory Inspector. https://swe-agent.com/latest/usage/inspector/. Accessed: 2026-04-28.

[20] Huanting Wang, Jingzhi Gong, Huawei Zhang, Jie Xu, and Zheng Wang. 2025. AI Agentic Programming: A Survey of Techniques, Challenges, and Opportunities. *arXiv preprint arXiv:2508.11126* (2025).

[21] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Proceedings of the 41st International Conference on Machine Learning*.

[22] Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2025. Survey on Evaluation of LLM-based Agents. arXiv:2503.16416 [cs.AI] https://arxiv.org/abs/2503.16416

[23] Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy K. Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, Fazl Barez, Rahul Gupta, Jwala Dhamala, Jacob Merizian, Mario Giulianelli, Harry Coppock, Cozmin Ududec, Antony Kellermann, Jasjeet S. Sekhon, and Jacob Steinhardt. 2025. Establishing Best Practices in Building Rigorous Agentic Benchmarks. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*. https://openreview.net/forum?id=H2Za8cmNB1