What Can This Skill Safely Undo? Rollback-Oriented Auditability for Agent Skills

Zifan Peng*
zpengao@connect.hkust-gz.edu.cn
The Hong Kong University of Science and Technology (Guangzhou)
China

Mingchen Li
MingchenLi@my.unt.edu
University of North Texas
United States

Abstract

Agent Skills give LLM agents reusable procedural knowledge for installing dependencies, invoking tools, editing files, configuring environments, and automating multi-step tasks. Existing work emphasizes whether skills improve performance, how they are generated, and how malicious skills or supply-chain risks can be detected. Less attention has been paid to what happens after a skill-driven task completes. When a skill modifies local or connected state, users may not know what changed, what persists, what can be safely undone, or whether recovery would overwrite later work. We argue that Agent Skills require *rollback-oriented auditability*: post-hoc support for understanding skill-induced changes as recovery objects with persistence, recoverability, dependency, conflict, and remediation properties. We sketch a recovery-oriented trace model, a recovery-preview interface, metadata hooks for `SKILL.md`, and an evaluation plan for testing whether such traces improve user recovery decisions.

Keywords: Agent Skills, LLM agents, auditability, rollback, recoverability, human-centered security

1 Introduction

Agent Skills are becoming a practical mechanism for extending LLM agents with reusable procedural knowledge. A skill is commonly represented as a directory containing a `SKILL.md` file, optional scripts, references, and assets that are loaded when relevant to a task [1, 3, 10]. This reflects a broader shift from one-off prompt-response interaction toward agents that can install dependencies, invoke tools, edit files, configuration, and automate workflows [4, 11].

That shift changes the safety problem. A risky or poorly specified skill not only degrades output quality; it may leave a persistent state in a user's environment, such as edited project files, shell configuration, installed packages, background processes, cached credentials, or external service effects. The urgent question often appears after the task seems finished: what exactly changed, which changes still matter, which ones can be safely recovered, and which recovery actions might overwrite later user work?

Current discussions emphasize effectiveness, benchmark performance, skill generation, malicious skill detection, and supply-chain security [11, 15, 16]. These are important questions, but they do not fully address the post-hoc recovery burden created when skill-driven agents modify local or connected state. In such cases, users do not need a raw command log alone. They need to reason over concrete recovery nodes: the files, environments, services, credentials, and external effects that a skill touched, together with whether each node persists, can be recovered, may support later work, or only admits remediation.

We argue that Agent Skills require *rollback-oriented auditability*. By rollback, we do not mean full-machine rollback or guaranteed automatic undo. We mean a trace model and interface layer that help users recover control after state changes by asking: what did the skill cause, what persists, what is recoverable, what conflicts with later work, and what remediation action is appropriate? Figure 1 summarizes this framing. This paper develops the position by motivating the recovery problem, relating it to skill evaluation, traceability, and selective recovery, and sketching a recovery-oriented trace and preview design for Agent Skills.

2 The Recovery Problem

Recovery becomes important when a skill-driven task changes the user's environment in ways that outlive the visible interaction. A user may ask an agent to "set up this project" or "connect this service," while the skill decomposes that request into steps touching files, package environments, credentials, local services, and external APIs. The task may appear successful, yet the user is left with a control problem: which effects should be kept as expected setup, which should be removed as side effects, and which require inspection because they may affect future work?

Skills import procedural authority. Users may understand a skill as a lightweight instruction package; in practice, it can guide an agent through dependency installation, file modification, service creation, credential use, or external tool invocation. This creates a mismatch between the apparent simplicity of installing a skill and the operational authority the skill may later exercise through the agent. The mismatch is especially salient because current skill formats support optional executable code and resources, while security analyses report prompt injection, data exfiltration, and privilege escalation risks [1, 16].

*Zifan Peng was on a visit to Newcastle University, the United Kingdom.

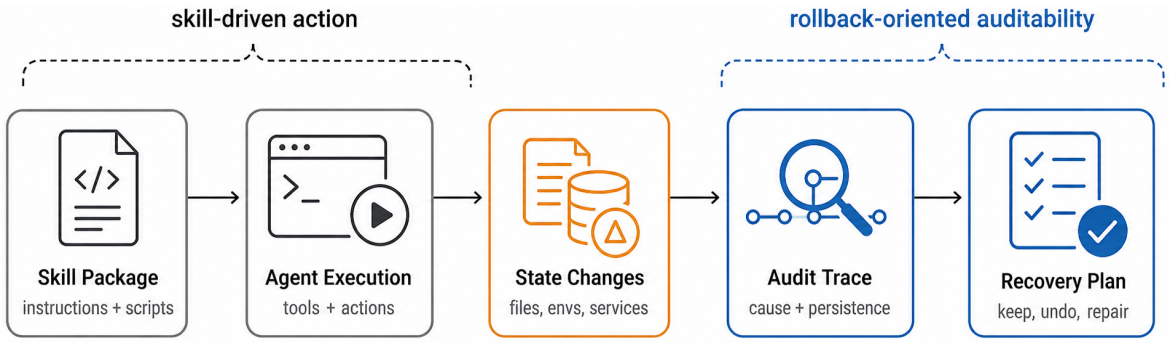

Figure 1. Conceptual framing. Agent Skills can drive high-authority actions over local and connected state; rollback-oriented auditability reorganizes traces around persistence, recoverability, conflicts, and remediation.

Recovery is organized around state-change nodes. A single skill-driven task may touch many resources, but not all effects are equally recovery-relevant. The important nodes are those that persist, carry authority, cross resource boundaries, interact with later user work, or produce external consequences. Existing logs may preserve action order, but they rarely organize actions into the objects users actually need to inspect or recover.

Recoverability varies by state type. Some nodes can often be restored, such as file or configuration edits, when a prior version is available. Others are partially recoverable, such as package changes that may now support later work. Some require semantic cleanup rather than file restoration, such as stopping a service, deleting a cron job, or removing a startup entry. External effects, including sent messages, uploaded files, or API calls, may be irreversible and require revocation, deletion, notification, or compensation instead.

Recovery is not simply going back in time. Whole-system rollback may remove later user work performed after the agent task. Selective rollback requires attribution, dependency reasoning, and conflict detection between skill-induced changes and later user actions. Restoring an earlier configuration file may erase a user's manual edit; removing a package may break a later task that now depends on it. This concern echoes long-standing HCI work on selective undo, which shows that users often need to undo an earlier operation without discarding all later desired work [5, 6, 18].

Scope. We focus on skill-driven agents that can change local or connected state with user-granted authority. The user may trust the platform enough to run the agent, but may not fully understand the skill's operational effects, referenced scripts, or downstream tools. The goal is not to prevent every unsafe action before execution; it is to support post-completion recovery decisions when persistent effects have already occurred. The threat cases include benign side effects, buggy skills, malicious skills, compromised dependencies, and external-service effects; for all of them, skill-declared cleanup claims should be checked against observed traces rather than trusted blindly.

Threat cases. Benign skills may still create surprising residue, such as helper files, shell configuration, caches, or local services. Buggy skills contain stale setup instructions, incorrect paths, overly broad package commands, or cleanup routines that fail. Malicious skills may hide persistence, exfiltrate data, tamper with configuration, or induce the agent to use credentials in unsafe ways. Compromised dependencies add another layer: packages, scripts, container images, or external tools invoked by an otherwise benign skill create secondary effects that the top-level skill did not declare.

External-service boundary. Some effects cross the boundary of local rollback. Sent messages, uploaded files, modified cloud resources, billing actions, or API calls may not be fully reversible even if the local trace is complete. These cases should be labeled as irreversible or only partially remediable and paired with actions such as revoke, rotate, delete remote object, notify recipient, or open provider audit logs.

3 Positioning Against Prior Work

Rollback-oriented auditability connects three research threads that are often treated separately.

Skill design and evaluation. Recent work establishes skills as first-class procedural artifacts. Voyager used an executable skill library for open-ended agents [21]; Agent Skills formats standardize `SKILL.md` packages across platforms [1, 3]; SkillsBench evaluates whether curated and self-generated skills improve task success [15]; and a SoK maps design patterns, lifecycle stages, and governance risks [11]. Our work complements this line by asking not only whether skills help agents complete tasks, but whether their effects can be understood and remediated after execution.

Traceability and provenance. Visualization and diagnosis tools for LLM agents show that raw event streams are insufficient for understanding complex agent behavior [17, 19, 22]. Security systems such as BackTracker reconstruct causal chains from system events using dependency graphs [13, 14]. These systems motivate structured traces, but our emphasis is different: the trace must be organized around user-facing

recovery decisions rather than only developer diagnosis, workflow debugging, or intrusion investigation.

Rollback and selective recovery. Systems such as Taser and RETRO show that recovery can be selective: unwanted actions and their indirect effects can be reversed while preserving desired later actions where possible [8, 12]. Environment managers and snapshot tools also demonstrate bounded rollback mechanisms, such as package/environment revisions or filesystem-level undo [2, 7, 20]. However, these mechanisms do not directly answer the human-centered agent question: how should a user understand, preview, and approve recovery when skill-induced changes span files, environments, services, credentials, and external effects?

4 Rollback-Oriented Auditability

We define *rollback-oriented auditability* as post-hoc support for understanding and planning recovery after skill-driven agent actions. It extends conventional traceability with recovery dimensions. Rather than asking only "what did the agent do?", it asks "what can be safely kept, repaired, or undone?" Table 1 summarizes how common skill effects can be reframed as recovery nodes.

Attribution. The system should distinguish skill-caused changes from agent planning, external content, tool defaults, or user actions. This matters because recovery decisions depend on whether a change was intended by the skill, incidental to execution, or introduced from an untrusted source.

Persistence. The system should identify which effects remain after task completion and where they reside: modified files, installed packages, environment variables, generated credentials, local services, scheduled tasks, or external account changes.

Recoverability. Each persistent change should be classified, with uncertainty when needed, as fully reversible, partially reversible, irreversible, or unknown. File edits, package changes, services, credentials, and external effects each require different evidence and different recovery semantics.

Dependency and conflict. Recovery planning should surface whether undoing a change would break later agent actions or overwrite later user work, especially when user activity and agent activity are interleaved.

Remediation action. Not every recovery action is "restore old state." Depending on the object, the appropriate action may be restoring a previous file version, removing a package, disabling a service, deleting a startup entry, revoking a token, cleaning residual files, or marking an external effect as not fully reversible. The system should therefore generate a recovery plan rather than a single global rollback command.

Recovery metadata. A recovery-oriented trace should therefore record both action history and recovery meaning. For example, a shell command belongs in the action history, but the recovery view should expose that it modified a shell startup file, created a helper script, installed a package in a scoped environment, or wrote a token-like value to disk. This separation matters because users usually recover objects and effects, not raw commands.

Table 1. Recovery-oriented view of skill-induced changes.

Object	Persistent concern	Recovery cue
File/config	Edits to `.env`, dotfiles, or project files	Diff; warn on later edits
Environment	Packages, paths, virtual environments	Remove scoped env; warn on dependents
Artifact	Service, cron job, startup entry, helper script	Stop, disable, or delete
Credential/cache	Token, secret, or local cache	Clean locally; rotate or revoke
External effect	Upload, message, API action, billing call	Mark remediation-only

5 Design: Recovery-Oriented Skill Traces

We envision a recovery-oriented extension to agent trace interfaces. For each skill-driven task, the system records structured events linking actions, resources, authorities, triggers, and persistence deltas. These records can be collected from agent logs, tool calls, filesystem snapshots, package manager histories, process monitors, provenance graphs, or skill-declared metadata [9, 12]. The interface then presents two layers: a skill trace layer and a recovery preview layer.

Skill trace layer. The trace layer helps users reconstruct what the skill caused the agent to do. It shows the task timeline, touched resources, authority context, and provenance of major actions. For example, a user could see that a package installation was triggered by a skill setup step, that a configuration file was edited by a shell command, or that an external download was introduced by a referenced script.

Recovery preview layer. The recovery layer groups persistent changes by recoverability. For each item, the interface shows the affected object, why it persists, whether it is reversible, possible dependencies, and a suggested remediation action. Rather than offering an undifferentiated undo button, it presents a recovery plan that users can inspect and selectively approve. Figure 2 sketches this interaction for a skill that runs an unfamiliar Python project.

Skill-declared recovery metadata. Skill packages could optionally declare expected effects, persistent artifacts, cleanup routines, irreversible actions, required authority, and rollback conflict checks. For example, a Python-project setup skill might declare that it expects to create `.venv/`, edit project-local `.env`, start a local server, and avoid remote uploads unless explicitly approved. These declarations would not guarantee safe recovery. In benign cases they scaffold a useful preview; in adversarial cases they become claims that can be compared against observed traces and resource deltas.

For a workshop-scale extension to `SKILL.md`, useful metadata keys would cover expected effects, persistent artifacts,

Table 2. Compact trace fields for rollback-oriented auditability.

Field	Example value	Recovery meaning
Skill source	`run-python-project/SKILL.md`, step `setup-env`	Attributes the effect to a skill instruction rather than only to generic agent planning
Target resource	`file://$HOME/.zshrc`, lines 41–42	Identifies the recoverable object and affected region
Persistence delta	Added export line that survives new terminal sessions	Distinguishes runtime output from durable state
Evidence source	Filesystem diff, shell log, package-manager history	Explains why the system believes the change happened
Recoverability	Partially reversible; later edit nearby	Calibrates whether automatic recovery is safe or needs review
Candidate action	Remove inserted line; rotate token if exposed	Converts rollback into resource-specific remediation

Recovery preview: `run-python-project` Authority used: shell, filesystem, package manager, local port

Node	Persistent change	Status	Conflict signal	Suggested action
`.env`	Added `DEMO_API` placeholder	Keep or edit	No later edit	Show diff; keep by default
`~/.zshrc`	Added export line	Reversible	Later user edit nearby	Remove inserted line after review
Virtual env	Installed 7 packages	Scoped	Later run depends on 2 packages	Keep environment or delete `.venv`
Helper script	Created `restart.sh`	Residual artifact	None	Delete or move to project tools
External API	Tested token with provider endpoint	Not fully reversible	Credential exposure possible	Revoke or rotate token

Plan preview: 2 safe cleanup actions, 2 keep-by-default items, 1 manual-review conflict, 1 remediation-only item.

Figure 2. Mock recovery-preview interface. Persistent effects are grouped into recovery nodes with conflict signals and suggested actions.

cleanup routines, irreversible actions, required authority, and conflict checks. These keys would let a skill declare, for instance, that it expects project-local filesystem writes and package installation inside a virtual environment, but does not expect remote uploads or global shell-profile edits. The platform could then compare declared and observed effects, surfacing deviations as audit warnings instead of treating the skill's own description as authoritative.

Avoiding false confidence. Rollback-oriented interfaces should avoid implying all changes are safely reversible. For uncertain cases, the interface should expose missing evidence, partial recoverability, and possible collateral impact. This is important for external effects, credential exposure, and objects modified again after the skill is completed.

Illustrative scenario. Consider a skill for running unfamiliar Python projects. The user asks an agent to make a downloaded repository run locally; the skill instructs the agent to inspect the project, create a virtual environment, infer missing environment variables, install dependencies, and start a development server. During execution, the agent edits `.env`, installs several packages, appends an export line to the user's shell configuration, opens a local port, and creates a helper script to restart the server. The web application opens successfully, but the visible task outcome no longer answers the user's recovery question. A rollback-oriented preview would let the user keep the project-level `.env` changes, review the shell-profile edit, stop the local process, inspect the helper script, and treat any external API use as remediation-only rather than ordinary rollback.

6 Implementation and Evaluation

Implementation signals. A recovery-oriented trace can be approximated with signals that many agent platforms already collect or can collect with bounded instrumentation: agent logs, tool calls, filesystem snapshots, package-manager histories, process monitors, credential stores, secret scanners, and cloud/API audit logs. The main challenge is fusing heterogeneous evidence into recovery nodes with calibrated uncertainty. For example, a file diff with a pre-skill hash provides stronger recovery evidence than an inferred environment variable discovered only from shell output.

The prototype need not infer all effects perfectly. It can first use explicit tool-call boundaries, before/after snapshots, package-manager history, process state, and provider audit entries to produce candidate deltas. Each delta can be labeled with an evidence source and confidence level, allowing the interface to support actions from uncertainty.

Authority context. Authority can be inferred from the substrate used: filesystem write permissions, shell execution, package manager scope, network access, credential access, or cloud/API identity. The interface should expose authority because the same command has different recovery implications depending on whether it was executed inside a temporary project directory, the user's home directory, a global package environment, or a cloud account.

Dependency and conflict detection. The system can treat the post-skill state as a recovery checkpoint. Before proposing rollback, it compares the current state with the checkpoint using file hashes, line-level diffs, modified times, package dependency graphs, process references, and API audit

timestamps. If a user edited the same file region after the skill completed, the node is marked as a conflict and the recovery plan offers a patch preview rather than automatic restoration. If a later command imports a package installed by the skill, the package is marked as a possible dependency.

Limits. Some signals will be incomplete, especially for tools without structured logs or remote services without accessible audit histories. A rollback-oriented interface should therefore display unknowns explicitly. Unknown recoverability is safer than overclaiming that a node can be restored.

Preliminary study. A pilot study can test whether recovery-oriented traces improve user understanding and recovery decision-making compared with chronological logs. We would use simulated but realistic traces from three skill tasks: running an unfamiliar Python project, connecting a cloud storage integration, and preparing a document-processing workflow. Each trace would include expected setup, benign side effects, a buggy cleanup action, a credential-related artifact, and at least one external-service effect. In a within-subjects design with 12–18 technically literate participants, users would compare a chronological log with a recovery-oriented trace view like Figure 2.

Tasks. For each scenario, participants would answer what changed, what persists, which changes are safe to undo, which require remediation rather than rollback, and which recovery actions might overwrite later user work. They would then choose a recovery plan from options that include safe cleanup actions, harmful over-rollback actions, and remediation-only actions such as token rotation.

Measures. Primary measures would include accuracy in identifying persistent changes, correct classification of recoverability, avoidance of later-work loss, and appropriateness of remediation choices. Secondary measures would include decision time, self-reported confidence, perceived workload, and calibration: whether participants refrain from claiming reversibility when evidence is missing. A larger follow-up could compare recovery-oriented traces, raw logs, and full snapshot rollback tools, and could add benchmark tasks that measure whether a skill's effects are auditable, bounded, reversible, and accompanied by reliable cleanup procedures.

Research agenda. Rollback-oriented auditability raises several open questions. Future skill formats need to decide which recovery metadata should be standardized without giving malicious skills an easy way to launder risky behavior. Trace systems need better methods for distinguishing intended setup from incidental side effects, especially when external content, tool defaults, and dependency scripts all contribute to the final state. Benchmarks for high-authority skills should also move beyond task success and measure whether a skill's effects are bounded, auditable, conflict-aware, and paired with reliable cleanup or remediation guidance.

Design implications. The interface implication is that recovery should be presented as a reviewable plan, not as a single undo button. A useful preview should separate expected setup from surprising residue, local reversible changes from external remediation-only effects, and low-risk cleanup from actions that may damage later work. This framing also helps avoid unnecessary alarm: some persistent effects, such as a project-local virtual environment, may be intended and useful, while a global shell-profile edit or exposed credential requires stronger attention.

Skill-authoring implications. For skill authors, rollback-oriented auditability suggests documenting effects alongside procedures. A skill should not only say how to complete a task, but also what state it expects to create, which cleanup actions are safe, which effects are outside its intended authority, and what evidence the platform should check before removing anything. This shifts skill quality from task success alone toward responsible lifecycle behavior.

Platform implications. For platforms, the key design is to preserve enough evidence at skill invocation time to make later recovery possible. Even a lightweight implementation could snapshot project files, record package-manager scope, capture process and port state, and mark which credentials or remote identities were used. These records don't need to expose every low-level event to users, but should be available for a recovery preview when something goes wrong.

User-facing implications. For users, the most important benefit is not perfect undo, but calibrated recovery: knowing which changes are ordinary setup, which are suspicious residue, and which are risky to remove without review. A recovery-oriented view can therefore support both caution and confidence, helping users avoid blind cleanup while still making persistent agent effects visible enough to act on.

7 Implications and Conclusion

Rollback-oriented auditability suggests that skills should document not only how to complete a task, but also what state they may change. Skill authors could expose expected effects, cleanup routines, irreversible actions, and required authority; platforms could scope traces by skill invocation; and benchmarks could test whether agents identify persistent side effects, later user edits, dependency conflicts, and remediation-only effects without overclaiming that everything is recoverable.

Skills make agents more capable, but they also make post-hoc recovery more difficult for users. Rollback-oriented auditability reframes skill traces around recovery decisions: what changed, what persists, what can be safely undone, what conflicts with later work, and what remediation is appropriate. Treating these questions as part of the Agent Skills lifecycle can make skill use more accountable without promising impossible, one-click undo.

References

[1] Agent Skills. 2026. Agent Skills Specification. https://agentskills.io/specification. Accessed 2026-05-02.

[2] Anaconda, Inc. 2026. Managing Environments: Restoring an Environment. https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html. Conda documentation, accessed 2026-05-02.

[3] Anthropic. 2025. Agent Skills. https://docs.claude.com/en/docs/claude-code/skills. Claude Code documentation, accessed 2026-05-02.

[4] Anthropic. 2025. Equipping Agents for the Real World with Agent Skills. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills. Engineering blog, accessed 2026-05-02.

[5] Thomas Berlage. 1994. A Selective Undo Mechanism for Graphical User Interfaces Based on Command Objects. *ACM Transactions on Computer-Human Interaction* 1, 3 (1994), 269–294. doi:10.1145/196699.196721

[6] Aaron G. Cass, Chris S. T. Fernandes, and Andrew Polidore. 2006. An Empirical Evaluation of Undo Mechanisms. In *Proceedings of the 4th Nordic Conference on Human-Computer Interaction (NordiCHI '06)*. Association for Computing Machinery, 19–27. doi:10.1145/1182475.1182478

[7] Eelco Dolstra, Merijn de Jonge, and Eelco Visser. 2004. Nix: A Safe and Policy-Free System for Software Deployment. In *18th Large Installation System Administration Conference (LISA '04)*. USENIX Association, Atlanta, GA, 79–92.

[8] Ashvin Goel, Kenneth Po, Kamran Farhadi, Zheng Li, and Eyal de Lara. 2005. The Taser Intrusion Recovery System. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*. Association for Computing Machinery, 163–176.

[9] Ashvin Goel, Kenneth Po, Kamran Farhadi, Zheng Li, and Eyal de Lara. 2006. Automatic High-Performance Reconstruction and Recovery. *Computer Networks* 50, 17 (2006), 3249–3269. doi:10.1016/j.comnet.2006.05.016

[10] Google Gemini CLI. 2026. Agent Skills. https://geminicli.com/docs/cli/skills/. Documentation, accessed 2026-05-02.

[11] Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. 2026. SoK: Agentic Skills – Beyond Tool Use in LLM Agents. arXiv:2602.20867 [cs.CR] doi:10.48550/arXiv.2602.20867

[12] Taesoo Kim, Xi Wang, Nickolai Zeldovich, and M. Frans Kaashoek. 2010. Intrusion Recovery Using Selective Re-execution. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI '10)*. USENIX Association, Vancouver, BC, 89–104. https://www.usenix.org/conference/osdi10/intrusion-recovery-using-selective-re-execution

[13] Samuel T. King and Peter M. Chen. 2003. Backtracking Intrusions. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*. Association for Computing Machinery, 223–236.

[14] Samuel T. King and Peter M. Chen. 2005. Backtracking Intrusions. *ACM Transactions on Computer Systems* 23, 1 (2005), 51–76. doi:10.1145/1047915.1047918

[15] Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Binxu Li, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, Junwei He, Yizhuo He, Liqiang Jing, Luyang Kong, Xin Lan, Jiachen Li, Songlin Li, Yijiang Li, Yueqian Lin, Xinyi Liu, Xuanqing Liu, Haoran Lyu, Ze Ma, Bowei Wang, Runhui Wang, Tianyu Wang, Wengao Ye, Yue Zhang, Hanwen Xing, Yiqi Xue, Steven Dillmann, and Han chung Lee. 2026. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. arXiv:2602.12670 [cs.AI] https://arxiv.org/abs/2602.12670

[16] Yi Liu, Weizhe Wang, Ruitao Feng, Yao Zhang, Guangquan Xu, Gelei Deng, Yuekang Li, and Leo Zhang. 2026. Agent Skills in the Wild: An Empirical Study of Security Vulnerabilities at Scale. arXiv:2601.10338 [cs.CR] doi:10.48550/arXiv.2601.10338

[17] 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 (2025), 4182–4197.

[18] Brad A. Myers, Ashley Lai, Tam Minh Le, YoungSeok Yoon, Joel Brandt, and Mira Dontcheva. 2015. Selective Undo Support for Painting Applications. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. Association for Computing Machinery, 4227–4236. doi:10.1145/2702123.2702543

[19] Rui Sheng, Yukun Yang, Chuhan Shi, Yanna Lin, Zixin Chen, Huamin Qu, and Furui Cheng. 2026. DiLLS: Interactive Diagnosis of LLM-based Multi-agent Systems via Layered Summary of Agent Behaviors. *CoRR* (2026).

[20] SUSE. 2026. Snapper Manual Page. https://snapper.io/manpages/snapper.html. Accessed 2026-05-02.

[21] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. arXiv:2305.16291 [cs.AI] doi:10.48550/arXiv.2305.16291

[22] Liwenhan Xie, Chengbo Zheng, Haijun Xia, Huamin Qu, and Chen Zhu-Tian. 2024. WaitGPT: Monitoring and Steering Conversational LLM Agent in Data Analysis with On-the-Fly Code Visualization. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, 14 pages.