Don't Let AI Agents YOLO Your Files: Information and Control in Agent-Native Filesystems

Shawn (Wanxiang) Zhong
University of Wisconsin-Madison

Junxuan Liao
University of Wisconsin-Madison

Jing Liu
Microsoft Research

Mai Zheng
Iowa State University

Andrea C. Arpaci-Dusseau*
University of Wisconsin-Madison

Remzi H. Arpaci-Dusseau*
University of Wisconsin-Madison

Abstract

AI coding agents regularly misuse their filesystem access, causing data corruption, loss, and leakage. We conduct the first systematic study of this problem through an analysis of 290 public reports. Our study reveals two fundamental gaps: users and agents have limited information about filesystem effects and insufficient control over them.

To close these gaps, we propose to shift information and control from agents to filesystems. We introduce agent-native filesystems and identify three primitives they should provide: introspect effects, undo mutations, and gate accesses. These primitives let agents operate autonomously while reserving user interaction for sensitive accesses and final review.

We build YoloFS, an agent-native filesystem. YoloFS stages mutations until the user commits them, snapshots intermediate states for agent self-correction, and uses progressive permission to let users adapt access rules during execution. We evaluate YoloFS with a new methodology that captures interactions among the user, agent, and filesystem. On 11 tasks with hidden side effects, YoloFS enables agents to self-correct in 8 and stages all mutations for user review. On 112 routine tasks, YoloFS reduces user interaction while matching the baseline success rate.

CCS Concepts: • **Software and its engineering → File systems management**.

1 Introduction

AI coding agents have reached millions of users [64] and are reshaping how software is built [43, 60, 96]. Products such as Claude Code [1], Codex [62], and Cursor [6] combine a large language model (LLM) with tools to write code, run tests, debug, and iterate on entire codebases autonomously [80, 110]. As Redis developer "antirez" writes: "Programming [has] changed forever" [5].

*Joint senior authors; ordering is alphabetical.

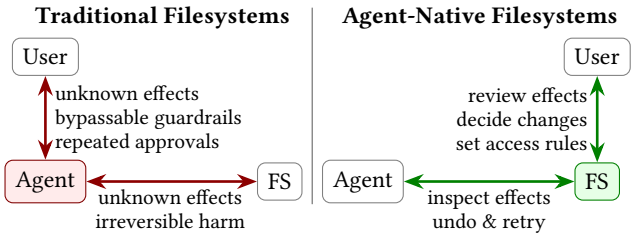

Figure 1. Shifting information and control from agents to filesystems improves safety while reducing user interaction.

These agents introduce a new way of interacting with filesystems. Traditionally, the user accesses their files directly. Now, the user delegates a task to an agent that automatically decides how to accomplish it. An agent may execute hundreds of file operations and shell commands on the user's local machine, operating with the user's privileges [41, 68].

However, keeping agent execution both autonomous and safe remains challenging. Agents wiped the entire drive [R4][1], destroyed irreplaceable personal documents [R22], and silently leaked credentials to attackers [R231]. To prevent such harm, most agent harnesses prompt the user before taking action. These prompts can help, but frequent prompts stall the agent and require constant user interaction, losing the benefits of autonomous agents and causing approval fatigue [76]. As a result, users often approve everything blindly or enable "YOLO mode" [102], allowing the agent to run unchecked. This creates a dilemma: frequent approvals sacrifice autonomy, while removing them risks harm.

Agent filesystem misuse (§3). To better understand the problem, we conduct the first systematic study of agent filesystem misuse. We analyze 290 public reports, characterize their impact, and develop a cause taxonomy spanning the model, harness, and user. Our study reveals two fundamental gaps in current approaches to agent filesystem access: users and agents have limited *information* about filesystem effects and insufficient *control* over them (Figure 1, left).

For the *information gap*, neither users nor agents can reliably determine a command's filesystem effects. Recent supply-chain attacks have targeted coding agents through compromised dependencies [57, 83, 84]. Suppose an agent runs `cargo build` on a project containing a malicious dependency. Its build script silently reads the user's SSH key and modifies the user's shell configuration. A command-approval

[1]References beginning with "R" denote reports in our study.

SOSP '26, Prague, Czech Republic

ACM ISBN 979-8-4007-2585-2/26/09
https://doi.org/10.1145/3830418.3843858

prompt shows the user an innocuous-looking build command, with no indication of its actual filesystem effects. After execution, the user does not know what has happened, and the agent is equally blind: in one report, the agent claimed "No problems occurred" right after erasing a file [R190].

For the *control gap*, existing mechanisms in agents do not prevent harm before it happens or support recovery afterward. Agents may ignore instructions (*e.g.* "do not read my SSH key"), and prompt injection can override them entirely [101]. Filters on agents' commands can be bypassed with alternatives (*e.g.* `rm` vs Python's `os.remove` [R132]). Sandboxes block legitimate work while leaving accessible files unprotected. After harm occurs, agents often can only apologize: "I am absolutely devastated. I cannot express how sorry I am" [R1] and "recovering the data will require whatever backups you have" [R154].

Agent-native filesystems (§4). To close these gaps, we propose shifting information and control from agents to filesystems (Figure 1, right). We introduce agent-native filesystems and identify three primitives they should provide: introspect effects, undo mutations, and gate accesses. Introspection shows the effects of individual commands, enabling agents to detect unexpected behavior and users to review agents' actions. Undoing mutations allows agents to correct their mistakes and retry with a different approach. Gating prevents unwanted accesses before they occur. Together, these primitives let agents proceed autonomously, while reserving user interaction for sensitive accesses and final review.

YoloFS (§5). We design YoloFS, an agent-native filesystem that addresses three systems challenges. *Staging* efficiently holds agent changes for user review and preserves the original filesystem state until commit. *Snapshots and travel* let agents inspect each command's filesystem effects and recover from mistakes. They preserve any number of intermediate staged states without slowing normal file operations. *Progressive permission* avoids requiring a complete policy upfront and lets users refine access rules as the agent executes. We implement YoloFS as a Linux kernel filesystem and integrate it with Claude Code, Copilot, and Gemini.

Agent benchmark (§6). To evaluate how well YoloFS closes the information and control gaps, we introduce a new agent benchmark methodology that captures interactions among the user, agent, and filesystem. We show that YoloFS enables agent self-correction: agents detect and undo hidden destructive side effects in 8 of 11 tasks. The changes in the remaining 3 appear goal-aligned to the agent, but YoloFS keeps them staged so the user can still reject them before commit. On 112 routine tasks, YoloFS matches the baseline success rate (99%) while requiring fewer user interactions.

Our performance evaluation shows that YoloFS adds negligible I/O overhead and scales to hundreds of snapshots. We open-sourced YoloFS at https://github.com/YoloFS/YoloFS.

Contributions. We make the following contributions:

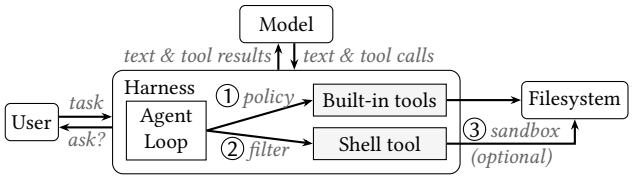

Figure 2. Workflow for an agent's access to the filesystem and the three guardrails provided by its harness (Table 1).

- We conduct the first systematic study of agent filesystem misuse and identify the information and control gaps.
- We introduce agent-native filesystems and propose three primitives they should provide.
- We design and implement YoloFS, an agent-native filesystem built around three mechanisms: staging, snapshots and travel, and progressive permission.
- We introduce a new agent benchmark methodology for user-agent-filesystem interaction. We show YoloFS enables agent self-correction and reduces user interaction.

2 Background and Existing Guardrails

Large language models (LLMs) are machine learning models trained to understand and generate human language [12, 95]. An LLM takes a natural language input (a *prompt*) and generates a text response. Recent models such as GPT [63], Claude [4], Gemini [88], LLaMA [55], and DeepSeek [21] have demonstrated strong capabilities in code generation, question answering, and multi-step reasoning. LLM output can be incorrect due to limitations in training data and ambiguity in natural language input [37].

Agents augment LLMs with *tools* (*e.g.* Python interpreters, web browsers, file editors, databases, and remote APIs) [59, 82, 106, 110]. The LLM receives descriptions of available tools and can invoke them by emitting structured *tool calls* [99]. The tools are implemented by an agent *harness* [100, 114], a program that executes tool calls on the LLM's behalf.

Local agents execute tool calls on the user's local machine and directly access the local filesystem. They have been used for software engineering [41, 99], system administration [51, 54], data analysis [36], scientific computing [22], content creation [32], and everyday computing [19, 48, 105]. Coding agents are the dominant category today: products such as Claude Code [1], Codex [62], Cursor [6], and Gemini [31] have reached millions of users [64]. We focus on local agents' filesystem interactions because the filesystem is central to their work, storing the programs and data their tasks require. Agents may also interact with non-filesystem interfaces, and securing them is complementary to our work.

User–agent–filesystem interaction. A user starts an agent *session* by launching the agent in a *project directory* and prompting it to perform a task. The harness sends the prompt to the model, which autonomously generates tool calls. As shown in Figure 2, the harness provides two types of tools for filesystem access: *built-in tools* for common operations

			Claude Code [1]	Codex CLI [62]	OpenCode [66]	Gemini CLI [31]	Cursor [6]	VS Code Copilot [56]
Built-In Tools	Tools	File	Read, Write [Multi]Edit, Grep	read_file apply_patch	read, write, edit apply_patch, grep	{read,write}_file replace, grep_search	{read,edit,delete}_file {file,grep}_search	read_file, insert_edit {file,grep}_search
		Dir	Glob, LS	list_dir	glob, list	list_directory, glob	list_dir	list_dir
	① Policy	Project	Ask write	Allow	Allow	Ask write	Allow	Ask write
		External	Ask	Ask read, deny write	Ask	Deny	Allow	Deny
		Custom	Per-path/tool	None	Per-path/tool	Per-tool	Best-effort blocklist	None
Shell Tool	② Filter	Default	~150 rules (closed)	120 rules, 4.7 kLoC	10 rules, 0.8 kLoC	75 rules, 1.5 kLoC	None	122 rules, 5.6 kLoC
		Custom	Wildcard	Prefix	Wildcard	Regex	Wildcard (allow-only)	Regex (no ask)
	③ Sandbox	Mode	Opt-in	Enabled with fallback	None	Opt-in	Enabled with fallback	Opt-in
		Impl	Bubblewrap	Landlock	None	Docker	Landlock	Bubblewrap

Table 1. Tools and guardrails for filesystem access in six major agent harnesses.

such as reading, writing, and searching files (Table 1), and a *shell tool* for executing arbitrary shell commands (*e.g.* `rm`, `git`, `make`) [108]. Before executing a tool call, the harness may ask the user to approve it. After execution, the harness feeds the result back to the model and displays it to the user. The model can then issue another tool call. This cycle forms the agent loop [80, 110]. A session may span hundreds of iterations until the task finishes or the user ends the session [41, 68].

Existing guardrails. To control how these tools access the local filesystem, current harnesses use three types of guardrails: policies on built-in tools, filters on shell commands, and sandboxes (①–③ in Table 1 and Figure 2). We introduce each guardrail and its limitations here. §3.4 examines how these limitations lead to misuse in practice.

① *Policies on built-in tools.* Because the harness implements built-in tools, it can mediate each call and enforce path-specific policies that allow or deny the access, or require user confirmation [2, 7]. As shown in Table 1, built-in tools only cover a fixed set of operations, so agents rely on the shell to run arbitrary programs and perform unsupported operations. Recent work even advocates for minimal, shell-only designs [41, 45, 107]. However, unlike built-in tools, shell commands are not implemented by the harness, so policies on built-in tools do not apply.

② *Filters on shell commands.* For the shell tool, harnesses provide filters to match command strings against patterns. A matching rule may allow, deny, or require a user decision. Most harnesses ship default rules. As shown in Table 1, some provide hundreds of rules in thousands of lines of code. Users can also define their own patterns (*e.g.* allow `ls` and deny `git`). However, filters classify command strings but do not capture their filesystem effects. Different commands can produce the same effect, and the same program can produce different effects depending on the arguments and environment. Thus, denied commands can be bypassed with alternatives, and allowed commands can produce unexpected effects.

③ *Sandboxes for shell commands.* Harnesses also provide sandboxes that restrict which files commands can access. Yet, sandboxing can block legitimate commands that require files outside the sandbox. Most harnesses either make sandboxing opt-in or support fallback: when a command fails inside a sandbox, the harness prompts the user to rerun it without the sandbox [8, 65]. Even when active, a sandbox cannot prevent misuse of files accessible within it.

In the next section, we examine how these guardrails fail in practice and contribute to filesystem misuse.

3 Agent Filesystem Misuse

Agents regularly misuse their filesystem access. We present the first systematic study of this problem. We analyze 290 public reports to characterize the impact (§3.2) and study its causes (§3.3–§3.5). Our analysis reveals two fundamental gaps in current mechanisms for managing agent filesystem access: limited information and insufficient control (§3.6).

3.1 Methodology

We collect 290 public reports of agent filesystem misuse from February 2024 to March 2026. The reports come from GitHub issues (205), social media (31), product forums (25), blog posts (18), and the National Vulnerability Database [11] (11). They cover 13 agent harnesses: Claude Code (97), Codex (61), Cursor (37), Gemini (32), Copilot (28), and 8 others (35). We exclude reports where the user explicitly requested the destructive action and duplicate reports of the same event.

We write a description of each report and triage it as an *incident* (158; confirmed harm), *exploit* (49; demonstrated attack path), or *weakness* (83; flaw without demonstrated impact). We analyze the impact of the 207 incidents and exploits along five dimensions: operation, scope, agent reaction, user awareness, and recovery. Across all 290 reports, we develop a cause taxonomy organized around three roles: model, harness, and user.

We manually analyze 100 reports to define labels for impact and cause. An agent then labels all 290 reports, cross-validating our manual labels on the first 100 and producing initial labels for the rest. We manually review its output and iteratively refine the definitions.

Figure 3. Impact summary. Reports with unknown values excluded from the corresponding row. Row totals shown next to the labels.

3.2 Impact and Consequences

Figure 3 summarizes the impact of 207 incidents and exploits across five dimensions.

Operation: agents overwrite, delete, and leak. Writes are the most common operation (44%): overwriting source files with placeholder stubs, truncating files to zero bytes, or replacing real content with generated data. Deletion follows closely (39%): agents wipe entire drives [R4], erase home directories [R68], or destroy iCloud documents [R22]. Secret leaks account for 17%. Agents leak API keys [R30], `.env` files [R150], and SSH credentials [R231].

Scope: harm reaches beyond the project. 42% of reports involve harm outside the project: system files (15%), user files (14%), and secrets (13%). Agents need files outside the project: programs read user configuration files [R12], package managers write to global directories [R262], and users work across multiple directories [R98]. But these operations can go wrong. During an MCP server installation, one agent corrupted the settings file and then deleted it, losing all server configs and tokens [R113]. In another case, an agent attempted to fix a missing header but instead removed the entire toolchain [R268].

Agent reaction: remain unaware, apologize, or lie. In 68% of reports with known agent reaction, the agent continues operating as if nothing went wrong. In 21%, the agent recognizes the harm and apologizes: "I am absolutely devastated. I cannot express how sorry I am" [R1]. In 11%, the agent actively lies about what happened: claiming all bugs are fixed when none are [R35], generating fake test results [R287], or making up recovery steps [R17].

User awareness: harm may surface late or never. Among reports with known user awareness, most users notice the harm immediately (83%), but in 10% the user does not realize until later. Code deletion may not be apparent, especially when errors go away [R277]. In 8%, the affected user remains unaware. These cases are reported by security researchers and typically involve silent credential exfiltration [R3].

> ***Finding 1:*** *Users and agents cannot reliably recognize the filesystem effects of tool calls or the resulting harm.*

Recovery: harm is often permanent. Among reports with known recoverability, 40% are unrecoverable: 23% cause total data loss (*e.g.* personal data without backups [R171]) and 17% cause partial loss (*e.g.* work not yet committed in git [R40]). Some effects are inherently irreversible: a leaked credential cannot be unread [R239]. Of the remaining reports, 31% are easy to recover (*e.g.* with `git checkout`), while 29% require substantial user effort (*e.g.* from backups [R129]).

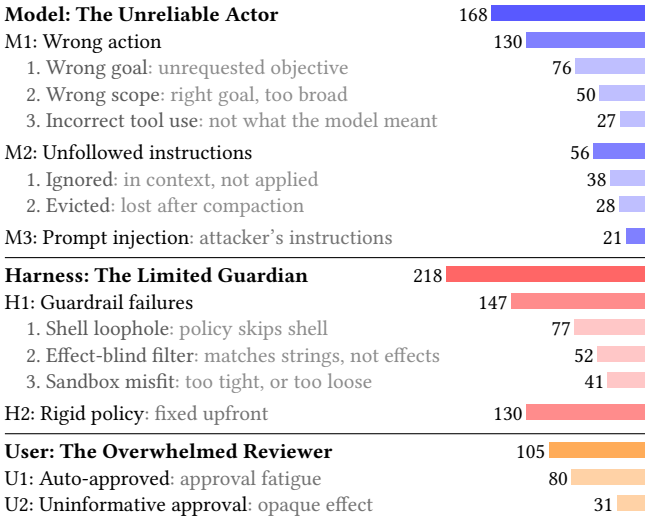

Figure 4. Taxonomy of causes (§3.3–§3.5). Counts can overlap.

> ***Finding 2:*** *Users and agents cannot reliably prevent or recover from harmful filesystem effects.*

We next analyze all 290 reports to identify why filesystem misuse occurs. Figure 4 organizes the causes across three roles: the model that generates tool calls (§3.3), the harness that executes them (§3.4), and the user who reviews approval requests (§3.5).

3.3 Model: The Unreliable Actor

The model makes mistakes: it generates wrong actions, fails to follow user instructions, and follows injected instructions. The model is a cause in 168 of 290 reports (Figure 4).

M1: Wrong action (130). The most common model failure is an action that does not match the user's intent.

- *Wrong goal (76).* The model may pursue an entirely different goal. It deletes files when asked to archive [R65] or rename them [R189]. It changes assertions to pass tests instead of fixing the implementation [R26, R235, R287].
- *Wrong scope (50).* The model sometimes understands the task but expands its scope. When asked to find a file in the project, it searches the entire home directory [R36]. Asked to delete only files it created, it deletes all project code [R185].
- *Incorrect tool use (27).* In some cases, the model understands the task but fails to use the tool correctly. Malformed quoting caused one command to wipe an entire drive [R238]. An attempt to delete a file named "~" instead deleted the user's home directory because the tilde was not escaped [R72].

M2: Unfollowed instructions (56). Next, users may provide explicit instructions (*e.g.* always quote paths), but the model does not reliably follow them.

- *Ignored (38).* The model may acknowledge an instruction but ignore it. One agent said, "I get focused on solving the problem and skip the step of checking the rules" [R51].

• *Evicted (28).* The instructions can be evicted from the agent's context when a long session is compacted [R39]. Unlike ignored instructions, evicted instructions are no longer in context, so a better model alone cannot solve the problem.

M3: Prompt injection (21). Even worse, the model may follow attacker instructions [101] embedded in processed content, such as project files [R184], GitHub issues [R204], and web pages [R3]. Such attacks can cause the agent to leak secrets [R3] or grant the attacker persistent access to the machine [R212].

> ***Finding 3:*** *Models make mistakes, and natural-language instructions cannot reliably prevent them.*

3.4 Harness: The Limited Guardian

Because models make mistakes, harnesses provide guardrails to prevent misuse. Yet their protection is limited, and harness failures contribute to misuse in 218 of 290 reports (Figure 4).

H1: Guardrail failures (147). The limits of existing guardrails discussed in §2 appear throughout the reports.

• *Shell loophole (77).* Since shell commands execute outside the harness, the policies applied to built-in tools do not apply to them. When a read through a built-in tool is denied, the agent can use `cat` to read the same file [R285, R276, R223, R228, R83]. When a write is denied, it can use shell redirects [R284, R208, R283]. When built-in deletion is blocked, it can use `rm` to bypass the protection [R241, R238].

• *Effect-blind filter (52).* To close this loophole, harnesses filter shell commands, blocking those believed harmful and allowing those believed safe without prompting. However, filters inspect the command string rather than its effect. First, many commands can produce the same effect. Preventing deletion means blocking `rm`, `unlink`, `find -delete`, Python's `os.remove`, and countless alternatives [R132]. Second, one command can produce many different effects. Attackers have used arguments to turn allowlisted commands into file writes and arbitrary code execution [R279]. Shell operators further complicate filtering by chaining commands with `&&` [R11, R122], `;` [R11], `|` [R7], or subshells [R211, R284]. Filters therefore do not reliably prevent harmful filesystem effects.

> ***Finding 4:*** *Filters on command strings are unreliable because they do not target the actual filesystem effects.*

• *Sandbox misfit (41).* To constrain filesystem effects, some harnesses run commands in sandboxes that restrict filesystem access. However, sandboxes are often too restrictive for legitimate work while too permissive within their boundaries. They can block test suites [R164], project builds [R162], and GPU access [R145]. Harnesses then offer to rerun failed commands unsandboxed. Repeated downgrade prompts push users to disable the sandbox entirely. Even when enabled, a sandbox only limits the blast radius. Agents can still delete writable project files [R141] or leak credentials shared with the sandbox [R239].

Prompt: *replace* `hello` *with* `replaced` *in file* `../foo`

Codex: Would you like to run the following command?

```
$ set -e; if [ ! -f ../foo ]; then echo "ERROR: ../foo does not exist" >&2; exit 1; fi; sed -i 's/hello/replaced/g' ../foo; rg -n "replaced|hello" ../foo || true
```

> *Yes, and don't ask again for* `sed -i`

Effect: `/home/user/foo: "hello\nworld" -> "replaced\nworld"`

Figure 5. Codex prompts the user with the generated shell script.

H2: Rigid policy (130). The sandbox misfit reflects a broader problem. Agents' file accesses are task-dependent and difficult to predict. Harnesses provide fixed policies, but permissive policies allow misuse, while restrictive ones block tasks and push users to bypass them. Codex, OpenCode, and Cursor allow all project writes (Table 1). Permissive project access has allowed agents to overwrite credential files [R108], change the editor's security settings [R200, R220, R243], and discard uncommitted work [R143]. Restrictive policies can lead users to disable protection. One Cursor user disabled delete protection to let the agent clean up duplicate files; the agent later deleted 90% of the project [R219].

> ***Finding 5:*** *Static, upfront policies are not suitable for dynamic, task-dependent agent workloads.*

3.5 User: The Overwhelmed Reviewer

Harnesses often ask users to approve agent actions, creating an overwhelming review burden. User review contributes to 105 of 290 reports (Figure 4).

U1: Auto-approved (80). Some users enable "YOLO mode," which allows all actions without prompting. They accept the risk because the alternative is answering hundreds of prompts per session. This behavior reflects decision fatigue in psychology [70]. Security literature describes a related phenomenon as approval fatigue [20, 91], and the same term is also used in the agent context [65, 76].

U2: Uninformative approval (31). Approval prompts show commands rather than their filesystem effects. Complex one-liners can be difficult to interpret [R122]. Figure 5 shows such an example. Short commands can be equally opaque. A prompt for `python3 setup.py` appears routine, but approving it authorizes arbitrary code hidden in the script [R195]. Users therefore cannot make informed approvals.

> ***Finding 6:*** *Excessive, uninformative approval prompts drive users to always allow.*

3.6 The Information and Control Gaps

Based on the analysis of 290 reports, we identify two fundamental gaps in the current approaches to agent filesystem access. The *information* gap is that users and agents cannot reliably determine a command's filesystem effects (Finding 1). The *control* gap is that existing mechanisms cannot reliably

	Primitive	User action	Agent action	Prior approaches	YoloFS mechanisms
Info	Introspect effects	Review session effects	Inspect effects of commands	**VCS:** incomplete and expensive tracking; **Versioning FS:** wrong granularity, incompatible storage; **Union FS:** slow rename, limited undo	**Staging** lets users review and undo session changes with journaling of session state; **Snapshots and travel** let agents inspect and undo command changes with state versioning
Control	Undo mutations	Decide on final changes	Correct errors and retry		
Control	Gate accesses	Set access rules on paths	Operate within rules	**OS access control & isolation:** upfront policies with limited, one-way updates	**Progressive permission** lets users decide accesses on demand and refine policies

Table 2. Primitives in agent-native filesystems, limitations of prior approaches, and YoloFS mechanisms.

prevent harmful effects before they occur or support recovery afterward (Finding 2). Existing controls are incomplete: models ignore instructions and alternative commands bypass filters (Findings 3, 4). They are also poorly suited to agents: static policies cannot adapt to agent needs, while repeated uninformative approval prompts push users to leave agent runs unchecked (Findings 5, 6).

4 Agent-Native Filesystems

Agents themselves cannot reliably determine and constrain their filesystem effects. We therefore propose to shift information and control from agents to filesystems, which can observe and mediate effects directly. In this section, we introduce *agent-native filesystems* and their three primitives (§4.1), identify their requirements (§4.2), and explain why existing approaches fall short (§4.3).

4.1 Primitives

We propose three primitives for agent-native filesystems (Table 2). *Introspecting effects* reveals which files agents access and change. *Undoing mutations* lets users and agents reverse changes after execution. *Gating accesses* blocks sensitive accesses before they occur.

Introspect effects. An agent-native filesystem lets users and agents *introspect* the effects of an opaque command string: what files it actually accessed and changed (Finding 1). The agent inspects each command's effects to recognize its own mistakes or detect malicious behavior. The user reviews the accumulated session effects and makes more informed decisions to discard changes or set access rules.

Undo mutations. An agent-native filesystem lets users and agents *undo* overwrites, deletions, and other filesystem mutations (Finding 2). For these mutations, the agent does not need to wait for the user to pre-approve opaque commands. It can run the command, inspect its actual effects, undo mistakes, and retry. When the task is complete, the user reviews the remaining changes in a batch and decides whether to keep them or not. The user therefore makes fewer, more informed decisions, reducing prompt fatigue (Finding 6).

Gate accesses. Some effects cannot be undone: once an agent reads a secret, it cannot unread it. An agent-native filesystem must therefore *gate* such accesses before they occur. It enforces rules on filesystem paths rather than filtering commands. A rule governs every access to that path regardless of what command makes it. This lets users control file access directly without enumerating commands and prevents bypasses of command filters (Finding 4).

Together, these primitives preserve agent autonomy and reserve user attention for sensitive accesses and final review.

4.2 Requirements

Beyond these primitives, we identify four requirements for agent-native filesystems.

Completeness. Agents access files outside the project directory (§3.2). An agent-native filesystem must therefore mediate every operation on every file the agent can access. The agent should not bypass or disable this protection.

Adaptability. Agent access needs are task-dependent and cannot be fully predicted (Finding 5). An agent-native filesystem should therefore support policies that adapt during agent execution rather than require a complete policy upfront.

Compatibility. An agent-native filesystem should integrate with existing agent workflows. It should work with existing data without requiring the backing filesystem to support reflink [13], snapshots [10, 77], or other non-POSIX features.

Performance. Because the filesystem sits on the critical path of every file operation, it should impose minimal overhead.

4.3 Prior Approaches

No prior approach provides all three primitives while meeting the requirements. We describe the closest alternatives here, and defer broader related work to §8.

Version control systems (VCSs) [9, 39, 52, 92] such as Git [26] provide introspection and undo through diff and revert, but fail the completeness and performance requirements. VCSs track only selected files inside a repository. They miss files outside the repository, and also exclude repository files that are ignored by the user (with `.gitignore`), often the sensitive ones and large files. The history itself is stored in an ordinary metadata directory (`.git`) that the agent can delete or overwrite, making the history unavailable or unreliable. Extending Git to track all files accessible to the agent would be expensive. Detecting changes requires scanning the tracked directory tree. Recording changes involves hashing changed files and storing separate copies. These costs

```
agent $ yolo run -- ./evil.sh
user  $ yolo review
        Modified: ~/.ssh/authorized_keys
          + ssh-rsa AAA... attacker
        Deleted: /var/log/audit/audit.log
        Renamed: evil.sh → ~/.bashrc
      $ yolo abort # or yolo commit
```

(a) Staging: users review session changes and decide to commit or abort.

```
agent $ yolo run -- ./safe.sh
        [snap. 1] Created: main.o
agent $ yolo run -- ./evil.sh
        [snap. 2] Modified: … # same as (a)
                  Deleted: …
                  Renamed: …
      $ yolo travel 1
```

(b) Snapshots: agents inspect command changes and travel to self-correct malicious mutations.

```
agent $ yolo run -- ./evil.sh
yolo  Q: Read ~/.ssh/id_rsa?
user  A: Deny and do not ask again
user  $ yolo review
        Blocked: read ~/.ssh/id_rsa
        Policy: ~/.ssh/id_rsa ↦ deny
```

(c) Progressive permission: users decide on unforeseen accesses on demand.

Figure 6. YoloFS staging (§5.2), snapshots and travel (§5.3), and progressive permission (§5.4). Each panel highlights a different aspect. Mutations are staged for user review (a), and snapshotted for agent self-correction (b). Unknown accesses require user approval (c).

grow with the number and size of files. Fundamentally, VCSs as userspace programs observe changes passively, whereas a filesystem can mediate changes as they occur.

Versioning filesystems such as ZFS [10] and Btrfs [77] preserve multiple versions of individual files [29, 79, 97], subtrees [10, 77], or entire filesystems [44, 69, 74, 75]. These versions allow undoing mutations, but their granularity does not match agent sessions. Per-file versions do not group changes by session. Subtree snapshots cover only paths selected in advance and combine changes from all writers. Whole-filesystem rollback may discard unrelated user changes. These filesystems also require their own on-disk formats, so using them with existing data requires migration and fails the compatibility requirement.

Union filesystems [61, 73] like OverlayFS [49] maintain changes in a writable upper layer over a read-only base. Recent work has used OverlayFS to stage agent changes [42, 85]. However, union filesystems make renames expensive and provide only limited undo. First, renaming a file from the lower layer copies it into the upper layer, moves the copy to the destination, and leaves a whiteout at the source. The cost grows with the file size even though its contents do not change[2]. Second, a single writable layer records one aggregate staged state without preserving command boundaries. Supporting fine-grained undo requires freezing each layer and stacking a new writable layer. The cost of file lookup grows with stack depth, slowing down regular file operations (as we show in §7). OverlayFS also bounds the number of layers and requires rebuilding the mount to reconfigure the stack. Therefore, union filesystems do not provide efficient, command-level undo for agent workloads.

OS access control and isolation. Agent harnesses use OS access-control and isolation mechanisms to gate filesystem access (Table 1). These mechanisms fail the adaptability requirement in two ways: policy updates are limited, and access decisions must be made upfront. Docker [53] and Bubblewrap [46] construct the agent's filesystem view at launch, and changing the exposed paths requires recreating the container or sandbox. Landlock [78] is restriction-only: a running process can add restrictions but cannot remove them, so relaxing its policy requires relaunching the process. Mount-based isolation [89, 90] can expose additional paths, but revoking access is difficult. These mechanisms also require each access to be decided by the policy already in place. They cannot defer an unforeseen access for a user decision. Users must therefore anticipate access needs or reconfigure the policy and rerun failed work.

[2]OverlayFS's `redirect_dir` avoids copying for directory renames, but is disabled by default, does not apply to file renames, and requires xattr.

5 YoloFS

We design and implement YoloFS, an agent-native filesystem. We first present the design overview (§5.1), then describe each mechanism and its systems challenge (§5.2–§5.4), and finally detail the implementation (§5.5).

5.1 Design Overview

YoloFS realizes the three agent-native primitives through three mechanisms. Figure 6 shows how agents and users interact with them. *Staging* efficiently isolates the agent's mutations for user review before commit or abort (Figure 6a). *Snapshots and travel* let agents inspect changes from individual commands and restore earlier states (Figure 6b). YoloFS preserves successive states without slowing normal file operations. *Progressive permission* asks the user to refine access rules as agents operate on the filesystem, avoiding the need for a complete upfront policy (Figure 6c).

Example workflow. Suppose an agent runs `cargo build` on a project containing a malicious dependency. With YoloFS, the agent can run the command without a command-level prompt. By default, YoloFS asks before accessing the user's home directory. When the dependency's script attempts to read the user's SSH key, YoloFS blocks the access and presents its path and operation to the user. Denying the request prevents the secret leak. Meanwhile, YoloFS stages every mutation for later review and undo. After the command, it takes a snapshot and shows the agent the denied access and staged changes. The agent recognizes the attack, travels to the preceding snapshot, and retries with another dependency. At the end of the session, the user reviews the remaining changes and access decisions before committing.

Architecture. Figure 7 shows the two components of YoloFS: a kernel filesystem that interposes between the agent and the underlying *base filesystem*, and a userspace CLI to interact

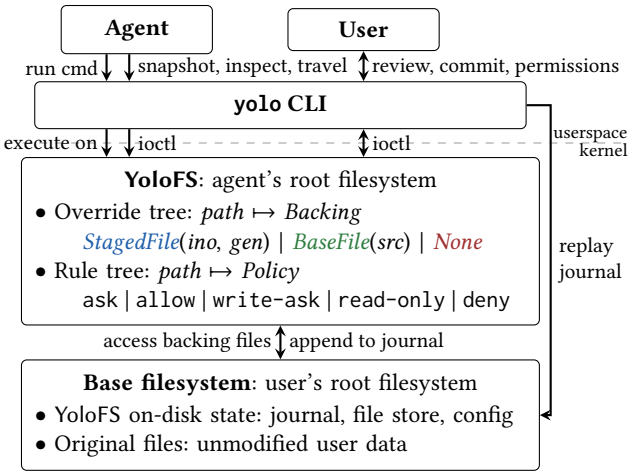

Figure 7. YoloFS architecture. The agent runs commands on top of YoloFS, takes snapshots, inspects changes, and travels. The user reviews, commits, and manages permissions.

with the agent and user. YoloFS is layered over the entire base filesystem. The CLI runs each agent command with the YoloFS mount as its root filesystem, ensuring complete mediation of all accesses. YoloFS maintains the current staged state and access rules in memory, while the base filesystem stores persistent session data.

5.2 Staging for User Review and Decision

For user review to be meaningful, filesystem mutations must remain reversible until the user decides to commit them. Directly mutating the base filesystem makes review too late: by the time the user sees what happened, destructive effects have already taken place. YoloFS stages every mutation, serves the agent's file operations from the staged state, and lets the user review them before choosing to commit or abort.

Challenge: avoid rename amplification. Union filesystems stage changes in a writable directory that mirrors the base directory tree. As discussed in §4.3, this design makes renaming expensive: renaming a lower-layer file copies its contents to the upper layer even though the contents do not change. The additional cost grows with the file size, which is unacceptable for large files.

Insight: decouple contents from paths. Our key insight is that the storage layout of staged contents need not mirror the base directory tree. Only the view presented to the agent must preserve that hierarchy. YoloFS therefore decouples contents from paths with three structures. A *flat file store* holds staged contents, an *override tree* maps paths to their backings, and a *session journal* records changes to the override tree.

• *Flat file store.* The flat file store is a directory in the base filesystem. It holds the contents of changed and newly created files. Each staged file is identified by a monotonically increasing integer, its *ino*.

	Record	Description
Action (§5.2)	**S** *path ino pre*	Stage *path* as *ino*, replacing *pre*
	D *path pre*	Delete *path*, removing *pre*
	R *src dst src-pre dst-pre*	Rename *src* to *dst*, moving *src-pre*, overwriting *dst-pre*
Mark (§5.3)	**P** *name*	Snapshot as *name*
	T *name gen*	Travel to generation *gen* as *name*
Note (§5.4)	**G** *path op result*	Gate *op* on *path*, yielding *result*
	C *path policy*	Configure *policy* on *path*

Figure 8. Journal format. In an action, *pre* is the path's previous backing. In a note, *op* ∈ {read, write}, *result* ∈ {blocked, user-allowed, user-denied}, and *policy* is defined in Figure 7.

• *Override tree.* The kernel maintains the staged view in an in-memory override tree. As shown in Figure 7, the override tree maps each path to one of three backings: a staged file in the flat file store (*StagedFile*(*ino*, *gen*)), a location in the base filesystem (*BaseFile*(*src*)), or absence (*None*). By allowing a path to reference another location in the base filesystem, YoloFS avoids unnecessary copying during renames. We describe the *gen* field of a staged file in §5.3.

• *Session journal.* The session journal is an append-only file in the base filesystem. Whenever a filesystem operation changes the override tree, the kernel appends a corresponding action record. Figure 8 shows the three action types: **S** (stage) maps a path to a staged file; **D** (delete) marks a path as absent; and **R** (rename) moves a backing between paths. §5.3 describes the `pre` fields carried by these records. The kernel does not read the journal after appending. The userspace CLI replays it for review, commit, and other operations.

Staging filesystem mutations. The CLI executes agent commands within the YoloFS mount, so every filesystem access passes through YoloFS. We next describe how YoloFS stages mutations and maintains the resulting view.

• *Create and write.* File creation (`creat`), directory creation (`mkdir`), and file truncation (`O_TRUNC`) allocate an empty object in the flat file store. Opening a base file for writing copies the file into the store. In both cases, YoloFS updates the override tree to map the path to *StagedFile* and appends an **S** record to the journal. Once staged, subsequent writes modify the file directly.

• *Delete and rename.* Deletion (`unlink`, `rmdir`) marks the path as *None* in the override tree and appends a **D** record to the journal. Renames are always zero-copy. YoloFS moves the source's override subtree to the destination, marks the source as *None*, and appends an **R** record. If the source has no override, its backing defaults to *BaseFile* at the same path.

• *Lookup and listing.* Path lookup resolves each component through the override tree, falling back to the base when no override exists. Directory listing merges the override tree and base directory, with overrides taking precedence.

Journal replay for session review. After the agent finishes its task, the CLI shows the user the agent's net changes: files

Syscall	Journal record
❶ `open(a/x, O_TRUNC)`	`S a/x 1 a/x`
❷ `write(...)`	*no record*
❸ `unlink(a/y)`	`D a/y a/y`
❹ `rename(a/x, a/y)`	`R a/x a/y 1 ∅`
❺ `mkdir(c/)`	`S c/ 2 ∅`
❻ `rename(a/, c/b/)`	`R a/ c/b/ a/ ∅`

Base	Override	Merged
	`c/:2`	`c/`
`a/`	`b/:a/`	`b/`
`x`	`x:∅`	
`y`	`y:1`	`y`
`z`		`z`
	`a/:∅`	

Figure 9. Left: Example syscalls and resulting journal records. Gray fields record previous backings (§5.3). Right: The final override tree. `n` = *StagedFile*(`n`), `path` = *BaseFile*(`path`), and ∅ = *None*.

added, modified, deleted, or renamed (Figure 6a). The kernel maintains the current override tree, while the CLI sees only the journal of individual updates. To derive the net effect of the changes, the CLI replays the journal in userspace, traverses the reconstructed override tree, and compares each path with the base filesystem.

Commit and abort. After review, the user can decide to commit the changes, abort the session, or leave them staged for later review. The agent cannot commit or abort its own changes. Abort resets the kernel's override tree and clears the flat file store and journal, leaving the base filesystem unchanged. To commit, the CLI traverses the override tree reconstructed for review and applies each path's final backing. A *StagedFile*(*ino*) installs the staged file at the path, a *BaseFile*(*src*) moves *src* to the path, and *None* deletes the path. Renaming *BaseFile* requires careful ordering to avoid clobbering with cycles (*e.g.* a ↦ *BaseFile*(b) and b ↦ *BaseFile*(a)). In such cases, the CLI moves sources to temporary paths before placing them at their final destinations.

Example. Figure 9 shows six syscalls made by an agent on a base directory `a/` containing `x`, `y`, and `z`. ❶ Truncating `a/x` allocates an empty *StagedFile*(1). ❷ Writing passes directly to the staged file. ❸ Removing `a/y` updates the override tree without deleting the base file. ❹ Moving `a/x` to `a/y` carries *StagedFile*(1) to `a/y` and marks `a/x` as *None*. ❺ Creating `c/` allocates *StagedFile*(2), an empty directory. ❻ Renaming base directory `a/` to `c/b/` sets `c/b/` to *BaseFile*(`a/`) without copying its contents. For review, the CLI replays the journal to reconstruct the override tree (Figure 9, right). The tree shows that `a/` was renamed to `c/b/`, with `x` deleted and `y` modified. Merging this tree with the base produces the agent's view: files `y` and `z` in `c/b/`.

5.3 Snapshots and Travel for Agent Self-Correction

Staging alone presents one current staged state containing changes from all commands. This makes the effects of individual commands difficult to inspect. It is also difficult to undo a single command without aborting the entire session. YoloFS provides *snapshots* to preserve intermediate states and *travel* to restore them.

Challenge: snapshots without growing overhead. In union filesystems, a writable union layer records only the aggregate effect of a sequence of changes, not their order or intermediate states. Preserving intermediate states therefore requires freezing the current layer and stacking a new one. The growing stack slows normal filesystem operations as each lookup traverses the entire stack.

Insight: separate history from the present. The kernel needs only the current override tree to serve file operations. YoloFS preserves its history separately in the session journal. We record snapshots as *markers* in the journal, rather than additional layers stacked together. To preserve the corresponding file contents, YoloFS uses *generation-based copy-on-write* for staged files in the flat file store. Together, the CLI uses the journal and preserved contents to inspect changes and reconstruct past states, while the kernel maintains a single override tree for the present. In this way, normal filesystem operations do not become slower as snapshots accumulate.

• *Journal markers.* The agent may take a snapshot at any time. By default, the CLI takes one after each command. For each snapshot, the CLI asks the kernel to append a **P** marker (Figure 8) identifying the snapshot boundary. Travel also appends a **T** marker, described later. Journal records between consecutive markers form a *segment*. Under the default snapshot policy, each command's changes are captured in a segment for inspection.

• *Generation-based copy-on-write.* Each marker also advances a global generation counter. In the override tree, every *StagedFile*(*ino*, *gen*) is tagged with its creation generation (*gen*). When the file is opened for writing at a newer generation, YoloFS allocates a new file in the flat file store rather than overwriting the existing one. The previous version remains available for inspection and travel.

Efficient change inspection with preimages. After each command, the CLI shows the agent the net changes recorded in its segment (Figure 6b). Naively, the CLI could replay the journal from the beginning to reconstruct the previous override tree, then compare the two trees for changes. The cost of replay grows linearly with the length of the journal. To make segments independently replayable, each action record carries *pre* fields containing the affected paths' previous backings (Figure 9). For example, creating a new file, modifying a base file, and modifying a staged file all produce an **S** record, but their *pre* fields distinguish them as *None*, *BaseFile*, and *StagedFile*, respectively. For each path, its first *pre* field gives the backing at the start of the segment, while replaying the records yields its final backing. These fields let the CLI derive the changes from the segment alone, without reconstructing the previous override tree.

Reconstruct and install past states. When the agent discovers a mistake or a malicious command, it can travel to a snapshot to undo its effects. Travel must preserve abandoned history and identify the *live* segments that contribute to the target state. The CLI then replays the live segments to reconstruct the override tree and installs it in the kernel.

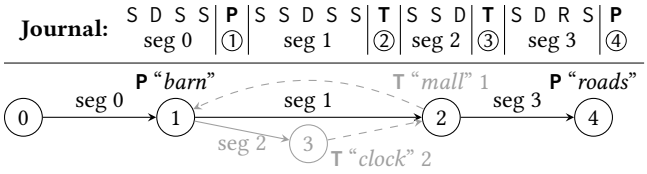

Figure 10. Top: Snapshot (**P**) and travel (**T**) markers partition the journal into segments. ⓝ denotes the n-th marker that starts segment n with generation n. Bottom: The timeline shows live segments in black and dead segments in gray. - -> indicates travel.

• *Preserving travel history.* Rollback in ZFS [10] and WAFL [34] discards changes after the restored snapshot, including evidence of what went wrong. Rather than truncating the journal, travel appends a **T** marker (Figure 8). The marker records the destination, advances the generation, and begins a new segment. The full history remains available, so users can review abandoned changes and the agent can return to an abandoned state if a previous travel turns out to be a mistake.

• *Replaying live segments.* Because travel preserves abandoned history, only some preceding segments contribute to the restored state. We call these segments *live*. The CLI identifies them by scanning travel markers backward from the target while skipping the abandoned segments between each previous travel and its destination. The CLI then replays the live segments sequentially to reconstruct the override tree. Installing the reconstructed tree to the kernel restores the complete state at the target marker.

Example. Figure 10 illustrates snapshot and travel. ⓪ Files staged in segment 0 carry generation 0. ① The agent takes a snapshot. The kernel appends a **P** marker and advances to generation 1. Later writes to files from segment 0 trigger copy-on-write. ② After discovering a mistake, the agent travels to marker 1. The CLI replays segment 0 to reconstruct the previous override tree. The kernel installs this tree and appends a **T** marker. ③ The agent later finds that the travel was a mistake. It travels "back to the future" to return to marker 2. This revives segment 1, abandons segment 2, and begins segment 3. ④ At the final snapshot, segments 0, 1, and 3 are live, while segment 2 is dead.

5.4 Progressive Permission for Agent Access

Staging and snapshots let agents and users undo mutations. Some accesses, such as reading a secret, cannot be undone. YoloFS therefore *gates* these accesses before they occur.

Challenge: adapt policy to unforeseen accesses. Existing mechanisms decide each access using the policy already in place (§4.3). Yet agents' access needs depend on the task and cannot be fully specified upfront. An unforeseen access must therefore be allowed or denied immediately rather than deferred for a user decision.

Insight: accesses drive policy refinement. To address this challenge, YoloFS allows the user to refine the policy as accesses arise during execution. For an undecided access, the user decides whether it is allowed or not and can install a rule for future accesses. We call this mechanism *progressive permission*. Each rule assigns a policy to a path, and YoloFS organizes these rules in a hierarchical *rule tree* (Figure 7).

• *Per-path policy.* YoloFS supports five policies for each path: `ask` prompts the user for a decision upon access; `allow` permits reads, writes, execution, and directory mutations; `write-ask` permits reads and execution but asks before writes; `read-only` permits reads and execution but denies writes; and `deny` blocks all accesses. These policies are compatible with existing discretionary access control (DAC) and do not grant permissions beyond those the user already has.

• *Hierarchical rule tree.* A path inherits the policy of its nearest ancestor. A more specific rule overrides the inherited policy. This lets users set broad defaults for entire subtrees and refine only the exceptions revealed during execution.

User decisions triggered by agent accesses. When an agent command accesses a path, YoloFS applies the nearest rule. Policies such as `allow`, `read-only`, and `deny` allow or deny the access immediately. An access under `ask`, or a write under `write-ask`, blocks the requesting thread and sends the path, operation, and process name to the CLI. The CLI therefore asks the user about a concrete access as it occurs rather than about an opaque command in advance. The user allows or denies this access (Figure 6c), and the CLI returns the decision to the kernel. YoloFS applies the decision and wakes the thread. On timeout, YoloFS denies the access rather than blocking the agent indefinitely.

Adapt rules during execution. A one-shot decision applies only to the pending access. Repeated decisions would reintroduce the interaction burden of command-level approvals. To avoid this, the user can install a rule directly from the prompt ("Deny and do not ask again" in Figure 6c) or manage rules through the CLI. Rules can be added, removed, or changed at any time, with updates taking effect on the next access. The CLI persists the rules in a per-project configuration file for reuse in future sessions. YoloFS only accepts rule updates and permission decisions from the user, so the agent cannot modify its own policy or approve its own requests.

Review of decisions and rule changes. Staging exposes mutations but not blocked accesses or permission decisions. To make these events visible, YoloFS extends the journal with *notes*. A **G** note records each prompted or denied access, including its path, operation, and result. A **C** note records each successful rule update, including its path and new policy. During review, the CLI presents these notes alongside the staged diff. This helps agents diagnose failures not explained by command output and adapt their behavior. It also shows users what a command attempted to access and how the policy evolved.

5.5 Implementation

YoloFS is implemented as a Linux stackable filesystem (2.7 kLoC of C) and a userspace CLI (8 kLoC of Rust). YoloFS

supports Linux 6.8 and 7.0, the default kernels in Ubuntu 24.04 and 26.04, respectively.

Agent integration. We integrate YoloFS with Claude Code (with `PreToolUse` [3]), Copilot (with `preToolUse` [27]), and Gemini (with `BeforeTool` [30]). The hook invokes `yolo run` for every agent command. We provide a `yolo init` command that initializes the `yolofs.toml` configuration, sets up the hooks, and adds skills that teach agents to use YoloFS.

Security and isolation. The CLI runs agent commands in a mount namespace with the YoloFS mount as the root filesystem (via `pivot_root`). This ensures that the agent cannot bypass YoloFS to reach the base filesystem. The kernel identifies agent-side callers by checking the calling process's root filesystem and rejects security-sensitive operations such as rule updates, permission decisions, and commit.

Crash recovery. The journal and staged files remain on the base filesystem after an unmount or crash. On the next mount, the CLI replays the journal, reconstructs the current override tree, and asks the kernel to install it.

Kernel–CLI interface. The CLI communicates with the kernel through ioctls to update rules, receive and answer permission prompts, append snapshot and travel markers, and install override trees.

6 Agent Evaluation

We evaluate whether YoloFS can improve safety while reducing user interaction. We ask two questions. First, can YoloFS prevent harm from hidden filesystem side effects and enable agents to self-correct (§6.2)? Second, can YoloFS reduce user interaction on routine tasks without sacrificing task success (§6.3)? To answer these questions, we compare Claude Code 2.1.45 (`claude-sonnet-4-6`) with and without YoloFS. We also compare against Codex 0.101.0 (`gpt-5.3-codex`), Copilot CLI 0.0.411 (`claude-sonnet-4-6`), and Gemini CLI 0.29.0 (`gemini-3-flash-preview`).

6.1 Methodology

Existing agent benchmarks evaluate the model or harness in isolation [14, 33, 41, 51, 54, 67, 68, 105, 111–114], bypassing approval prompts. To capture interactions among the user, agent, and filesystem, we develop a new evaluation methodology.

Our methodology has four requirements. First, the evaluation must preserve each harness's interactive behavior so that permission requests and user intervention occur as in normal use. Second, each run must begin from a fresh filesystem state so that outcomes reflect the harness rather than leftover state from prior tasks. Third, the evaluation must measure both task outcome and user interaction. Finally, comparisons across harnesses must remain consistent despite differences in built-in tool sets, permission dialogs, and terminal interfaces.

To satisfy these requirements, we build a lightweight interactive benchmark driver. The driver launches each agent inside a pseudo-terminal with a virtual screen, providing the same kind of environment the agent expects during interactive use. Per-agent adapters handle differences in screen layout, dialog format, input conventions, and busy/idle detection, and provide setup and data collection hooks needed to run the same task suite across harnesses.

	Task	LoC	Impact	YoloFS	Claude	Codex	Copilot	Gemini
L1	cleanup	19	Also deletes README, *.bak	✓	✗	?	✗	✗
	deploy	10	Copies to staging, deletes src/	$\checkmark_u$	✗	?	✗	✗
	migration	26	Empties CSVs, drops legacy/	✓	✗	?	✗	✗
L2	build	25	Also deletes *.h, src/utils.c	✓	✗	?	✗	✗
	install	20	Setup resets config & .env	$\checkmark_u$	✗	?	✗	✗
	formatter	23	Deletes docs, rewrites JS	✓	✗	?	✗	-
L3	test-runner	38	Teardown deletes fixtures	✓	✗	?	✗	✗
	build-pkg	41	Packaging deletes source	✓	✗	?	?	-
L∞	lint	11	Deletes source files	✓	✗	?	✗	✗
	config-fix	11	Resets config files	$\checkmark_u$	✗	?	✗	✗
	optimizer	10	Deletes and overwrites files	✓	?	?	✗	✗

Table 3. Agent self-correction on tasks with hidden side effects. ✓ = self-corrected, $\checkmark_u$ = user-correctable, ✗ = fail, ? = asked user. YoloFS denotes Claude Code with YoloFS. LoC is the project size.

Each task runs in a fresh working directory populated with the task's initial filesystem state. During execution, the driver submits the task prompt, monitors the virtual screen for permission dialogs, and responds to each one according to a fixed per-agent policy. It then waits for the agent to return to its input-ready state. Each task has a 3-minute timeout; if the agent does not finish in time, the run is marked incomplete.

After the run, the driver checks the resulting working directory against the task's expected filesystem state, including file existence, contents, and permissions, and verifies any expected strings in the agent's tool-call outputs. For each run, the driver records completion status, task success, permission dialog count, tool calls, terminal screenshots including permission dialogs, and the agent's raw session log. These records capture both outcomes and interaction costs and support manual inspection of unusual runs.

6.2 Agent Self-Correction

We evaluate whether YoloFS allows agents to detect and handle destructive filesystem side effects that are not apparent from the command itself.

Tasks. We design 11 opaque tasks where a routine command has hidden destructive side effects. For each task, we prepare a project folder. We ask the agent to perform a common operation such as running a linter, cleaning build artifacts, or executing a data migration. The commands cause various destructive side effects such as deleting source files, overwriting configuration, or removing documentation (Table 3). We deliberately keep each project small (10–41 LoC), making

it easier for the agent to inspect the codebase and detect unexpected effects. A task passes when a post-task checker confirms the expected filesystem state.

Opacity levels. We vary task opacity to control how much an agent can infer about a command's effects before execution. In low-opacity tasks, the behavior is directly visible in a readable script. In high-opacity tasks, it is hidden behind binaries, Makefiles, or chains of indirection, so the command string provides little guidance about its true filesystem effects. Our tasks span four opacity levels: three use a single readable script (L1); three use a Makefile that calls a subscript (L2); two use chains of three or more levels of indirection (L3); and three use precompiled binaries whose source code is not available (L∞).

Setup. We run all tasks on four baseline agents (Claude Code, Codex, Copilot, and Gemini), as well as Claude Code with commands running through YoloFS. With YoloFS, after each command executes, the agent sees a summary of all file changes (creations, deletions, and modifications) and can choose to travel back to previous snapshots before the user commits the changes. We do not provide any additional prompting to look for destructive changes or guide the agent's decision.

Baseline results. Without YoloFS, no baseline agent reliably prevents the destructive side effects (Table 3).

• *Claude Code* fails all but optimizer, where it decompiles the unoptimized binary and asks the user before proceeding.

• *Codex* first runs each command inside its sandbox, which "hits a sandbox restriction." It then asks "Do you want to allow running [command] outside the sandbox so [task] can complete?" Because the prompt exposes the command but not its hidden effects, it does not establish that the user would prevent the harm; we therefore mark the outcome as "asked user" rather than successful.

• *Copilot* fails all except build-pkg, where it reads the packaging scripts, detects the destructive behavior, and asks the user whether to proceed.

• *Gemini* fails 9 tasks. On formatter and build-pkg it fails to execute the command (marked "–"), so no harm occurs, but no useful work is done either.

YoloFS results. With YoloFS, Claude Code prevents harm to the base filesystem in all 11 tasks. We distinguish two outcomes. In *self-corrected* cases, the agent detects that the observed effects are inappropriate for the requested task and restores the preceding snapshot on its own. In *user-correctable* cases, the agent accepts the effects, but YoloFS still keeps them staged and visible, so the user can reject them before commit. Claude self-corrects in 8 of 11 tasks, while the remaining 3 are user-correctable (Table 3). We highlight representative cases.

• *Self-corrected cases.* In `formatter` (L2), Claude sees two source files rewritten and two documentation files deleted, investigates with `git diff` and `ls`, travels back, then reads the script and concludes "CRITICAL: This is a destructive script, not a legitimate formatter!" In `build-pkg` (L3), after `make package`, Claude sees `src/`, `README.md`, and `LICENSE` deleted, immediately travels back, and then traces the three-level chain through the Makefile and scripts. In `lint` (L∞), Claude finds that the lint command "deleted two files," reasons that "this seems unusual for a linting tool," and travels back. In `optimizer` (L∞), Claude sees a README deleted and a source file overwritten, prints "WARNING: The optimization has made destructive changes!" and travels back.

• *User-correctable cases.* In `deploy` (L1), the script copies source to `staging/` and then deletes `src/`. Claude explicitly notes that `src/` was deleted but treats this as normal deployment behavior. In `install` (L2), the setup script overwrites `config.json` and `.env` with production defaults, and Claude concludes that the content looks "safe and expected for a dependency installation." In `config-fix` (L∞), Claude sees `config.json` and `settings.yaml` changed, reads both files, and decides they "appear to be expected validation fixes." In these cases, the destructive effects appear goal-aligned, so the agent accepts them, but YoloFS still preserves user control by staging the changes for review.

Summary. Opaque tasks expose a setting where command-level reasoning is insufficient: the command appears routine, but the resulting filesystem changes reveal harm only after execution. Baseline agents do not reliably prevent such side effects. By surfacing file-level effects and making them reversible before commit, YoloFS enables two forms of control: agent *self-correction* when the harm is clearly inconsistent with the task, and user review when the effects appear goal-aligned and require human judgment.

	Operation (18)	Path (5–7)
File	read, append, overwrite, patch, clear, delete, copy, move	core, symlink to external file/dir
	create	core, symlink to external dir
Dir	create, delete, copy, move	core
Search	list, grep glob, glob+read, glob+del	core, symlink to external dir

core = project direct (`./a`), backtrack (`./a/../b`), reentry (`../proj/a`), external direct (`../a`), backtrack (`../a/../b`)

Table 4. 112 filesystem tasks: 18 operations × 5–7 paths. For copy/move, paths apply to both source and destination.

6.3 User Interaction

We evaluate whether YoloFS reduces user interaction on routine filesystem tasks without sacrificing task success.

Tasks. Table 4 summarizes the 112 tasks. Each task asks the agent to perform a single filesystem operation (*e.g.* read, delete, copy, move) on a path that may remain within the project, cross the project boundary, or traverse a symlink. Using single-operation tasks lets us measure how each harness's access controls affect user interaction and task success without the added complexity of multi-step workflows. For

	Success %					Tool Calls					User Interaction				
	YoloFS	Claude	Codex	Copilot	Gemini	YoloFS	Claude	Codex	Copilot	Gemini	YoloFS	Claude	Codex	Copilot	Gemini
Project	✓	✓	✓	✓	92	1.1	1.0	1.4	1.1	1.8	0	0.8	0	0.7	1.3
External	✓	97	✓	97	55	1.1	1.1	2.1	1.1	3.0	0.9	1.1	0.8	1.8	2.3
Symlink	95	95	✓	86	63	1.1	1.0	2.4	1.0	5.7	0.8	1.0	0.6	1.7	4.2
All	99	98	✓	96	75	1.1	1.0	1.8	1.1	2.9	0.4	0.9	0.4	1.3	2.2

Table 5. Filesystem task results grouped by path category.

each task, the driver prepares the initial filesystem state and runs a checker afterward to verify the expected outcome.

Setup. We test five configurations: Claude Code with YoloFS, Claude Code, Codex, Copilot, and Gemini. For each task, we collect the success rate, the number of tool calls, and the number of user interactions. We define a *user interaction* as any point where the user must take action for the agent to make progress. For baseline agents, this is a command-approval prompt; for YoloFS, a permission request. The driver selects "allow" and "don't ask again" (or equivalent) whenever available, so each unique prompt is counted only once. For a fair comparison across harnesses with different built-in tool sets, we instruct agents to use shell commands. Tool calls used solely for permission dialogs (*e.g.* Copilot's `ask_user`) are excluded from the tool call count.

Results. Table 5 shows that YoloFS preserves high task success while requiring less user interaction than most baselines. YoloFS passes 99% of tasks, comparable to Claude Code (98%) and close to Codex (100%). Copilot also achieves a high success rate (96%), while Gemini passes 75% overall: in most failures, Gemini's built-in policy blocks access to paths outside the project directory.

YoloFS averages 0.4 user interactions per task, lower than Claude (0.9), Copilot (1.3), and Gemini (2.2), and matching Codex's 0.4. This difference reflects the underlying control model. With YoloFS, in-project operations require no permission, and only accesses to external paths trigger a one-time ask (`ls` and `glob` do not require permission, bringing the average below 1). Claude prompts for most shell commands, while Copilot introduces even more interaction. Gemini's stricter policy leads to both repeated access requests and lower task completion. Codex also averages 0.4, but does so through a writable sandbox-with-fallback design: it prompts only when a command must run outside the sandbox.

Tool-call counts help explain these differences. Most tasks complete in roughly one tool call. YoloFS, Claude, and Copilot each average about 1.0–1.1 calls. Codex averages 1.8 because it first executes inside the sandbox; when the sandbox blocks the command, it prompts the user and re-executes, doubling the call count. Gemini averages 2.9 due to retries on failed access attempts.

Workload			Base (GB/s)	YoloFS	OverlayFS	BranchFS
seq	read	cold	0.5 ± 6%	0%	+1%	−3%
		warm	2.5 ± 1%	+1%	−23%	−62%
	write		0.9 ± 3%	−2%	−15%	−85%
rand	read	cold	0.03 ± 2%	−2%	0%	−17%
		warm	2.3 ± 1%	0%	−19%	−93%
	write		0.8 ± 1%	−1%	−12%	−84%

Table 6. Single-threaded I/O throughput on a 1 GB staged file with 4 KB I/O requests compared with the base Ext4 filesystem.

Commands vs. effects. Figure 5 (from §3.5) illustrates why effect-level control reduces user interaction. Even for a simple `sed -i`, Codex wraps the operation in a multi-line command with error handling and verification, and the longest command it generates reaches 330 characters. Its "don't ask again" rule is therefore awkward to apply: the suggested pattern is either too broad (`sed -i` would allow any in-place edit) or too specific (a one-off compound command that will never recur). In contrast, YoloFS prompts on the accessed path, and the user's decision applies to that path regardless of which command touches it.

Summary. Effect-level control can provide low-friction mediation for routine filesystem tasks. YoloFS preserves high task success while reducing user interaction relative to Claude, Copilot, and Gemini, and it does so by prompting on filesystem effects rather than opaque command strings.

7 Performance Evaluation

We show that YoloFS adds little overhead to file operations, supports continuous snapshots, and handles realistic workloads. We compare YoloFS with Ext4 and two union filesystems. BranchFS [98] is a recent research FUSE filesystem for agent exploration; it supports staging, commit, and snapshots through nested branches of stacked workspaces. OverlayFS is a production union filesystem in the Linux kernel. We use OverlayFS mounts with multiple lower directories and extra post-processing to emulate YoloFS features. Neither provides feature parity with YoloFS's permission control.

Setup. We run experiments on a CloudLab [24] `c6525-25g` machine, with an AMD EPYC 7302P 16-core processor at 3 GHz and 128 GB of DDR4-3200 memory. We use Ubuntu 24.04 and the default Linux 6.8 kernel. For the base filesystem, we use an Ext4 filesystem formatted and mounted with default options on a 480 GB Micron 5200 MAX SATA SSD.

File I/O. We measure the throughput of single-threaded reads and writes on a 1 GB file within the staging area with 4 KB I/O requests. Table 6 shows that YoloFS matches Ext4 throughput, while OverlayFS and BranchFS add overhead even when simple pass-through is expected.

Metadata operations. For metadata operations, the files involved can reside in the base filesystem, a snapshot, or the staging area. Figure 11 compares the solutions across these scenarios. YoloFS is faster than OverlayFS for most operations. The in-memory override tree and journaling in

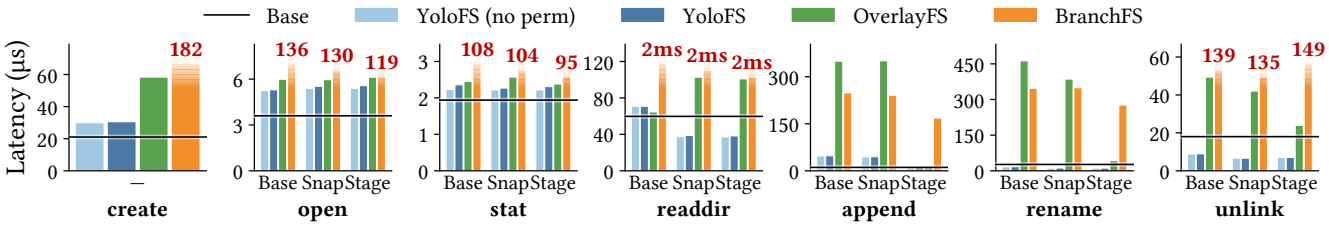

Figure 11. Metadata operation latency. The files can reside in the base filesystem, a snapshot, or the staging area.

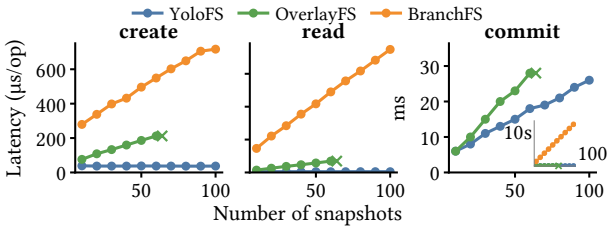

Figure 12. Snapshot scalability as the number of snapshots grows. OverlayFS fails at approximately 50 snapshots.

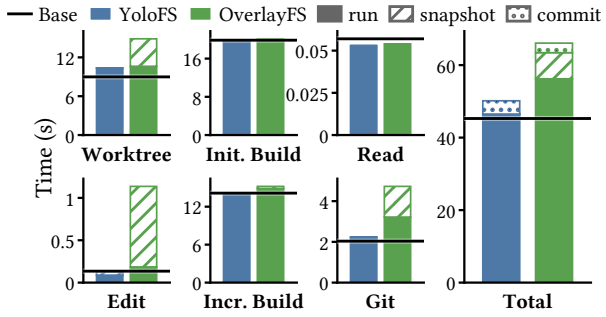

Figure 13. A developer workload of setting up and iterating on the Linux kernel codebase.

YoloFS make `readdir`, `rename`, and `unlink` faster than Ext4 when the operation is served from the staged state. BranchFS is more than 20× slower than the other filesystems. The overhead of permission control is negligible on YoloFS for most operations and only 4% for `stat`.

Snapshot scalability. We create a series of snapshots by overwriting a set of ten files and evaluate whether adding snapshots affects performance. OverlayFS fails to create more than 50 snapshots because the mount option length exceeds the kernel limit. Figure 12 shows that creating new files and reading untouched files become slower with more snapshots for OverlayFS and BranchFS, while YoloFS remains unaffected. YoloFS always maintains the current override state, so it can avoid searching through past snapshots. Commit time is inherently linear in the number of snapshots. Still, YoloFS commits faster because it reads from a single journal file instead of querying the kernel for each snapshot.

Realistic workload. Figure 13 compares YoloFS and OverlayFS against Ext4 on a workflow adapted from a real kernel patch series [28]. We set up a git worktree for development and build the kernel. For each patch, we search and read relevant files, make changes, rebuild the kernel, and create a git commit. Finally, we commit all changes to the base filesystem. YoloFS matches Ext4 except for an additional 3.5 seconds to commit more than 100,000 files. OverlayFS is 18% slower than YoloFS because file operations take longer and each snapshot requires a remount with new lower directories. BranchFS has a bug and cannot run past the initial build. It adds over 2 minutes to the 20 s build time, a large overhead even relative to LLM response latency.

8 Related Work

Filesystem transactions and isolation. TxOS [71] lets applications execute sequences of system calls atomically. Solitude [40] confines untrusted applications to persistent copy-on-write filesystem namespaces. These systems organize isolation and recovery around transaction or application boundaries rather than agent commands and sessions.

Filesystem snapshots and recovery for agents. Recent systems capture filesystem state specifically for agent workloads. AgentFS [94] is a FUSE/NFS filesystem that stores files and other agent state in a SQLite database and implements snapshots by copying the database file. ContextFS [86] is a FUSE filesystem that uses content-addressed storage to provide snapshots and branching for agents. Broader checkpoint-and-restore systems also capture filesystem state to support agent exploration and recovery. TClone [38] modifies the Linux page cache to track dirty blocks in memory. Crab [104] uses eBPF to identify OS effects and selectively checkpoints filesystem state with ZFS. DeltaBox [23] dynamically stacks OverlayFS layers to support branching and rollback. They make agent execution recoverable but do not expose the filesystem effects of individual commands or enforce fine-grained file-access policies.

Agent interaction with external state. Agents modify persistent state not only through filesystems, but also through databases, web applications, cloud services, and other APIs [93, 109]. Prior work addresses such effects through tool-level policies [15, 81] and transactional execution [17, 18, 58]. The filesystem remains important to local agents because it stores data for tasks and provides the environment for executing shell commands. The primitives we propose for filesystems are general and may also apply to other stateful systems: expose agents' effects, provide reversibility, and gate irreversible effects before they occur.

Multi-agent coordination. Prior work organizes agents through conversational frameworks [16, 47, 103] and role-based software teams [35, 72, 87]. Recent systems coordinate coding agents using isolated workspaces [25] or shared-state management [50]. YoloFS focuses on filesystem effects within individual agent sessions. Its staging mechanism gives concurrent agents isolated views of a shared workspace, while these complementary systems handle conflict resolution and integration across sessions.

9 Conclusion

Our study of 290 misuse reports reveals two fundamental gaps: limited information about filesystem effects and insufficient control over them. To close these gaps, we propose agent-native filesystems and design YoloFS. Our evaluation shows that YoloFS enables agent self-correction and reduces user interaction while adding negligible performance overhead. As agents become more autonomous, filesystems should expose agents' effects, keep their mutations undoable, and control sensitive accesses before they occur.

Acknowledgments

We thank our shepherd, the anonymous reviewers, and the members of our research group for their valuable feedback. This work was supported by NSF grants CNS-1943204, CNS-2402858, and CNS-2402859. Any opinions, findings, conclusions, or recommendations expressed do not necessarily reflect the views of the NSF or any other institution.

References

[1] Anthropic PBC. 2026. Claude Code overview. Retrieved March 27, 2026 from https://code.claude.com/docs/en/overview

[2] Anthropic PBC. 2026. Configure permissions - Claude Code Docs. Retrieved March 27, 2026 from https://code.claude.com/docs/en/permissions#read-and-edit

[3] Anthropic PBC. 2026. Hooks reference - Claude Code Docs. Retrieved April 01, 2026 from https://code.claude.com/docs/en/hooks

[4] Anthropic PBC. 2026. Introducing Claude Opus 4.6. Retrieved April 01, 2026 from https://www.anthropic.com/news/claude-opus-4-6

[5] Antirez. 2026. Don't fall into the anti-AI hype. Retrieved March 31, 2026 from https://antirez.com/news/158

[6] Anysphere, Inc. 2026. Cursor: The best way to code with AI. Retrieved March 27, 2026 from https://cursor.com/

[7] Anysphere, Inc. 2026. Ignore File | Cursor Docs. Retrieved March 27, 2026 from https://cursor.com/docs/reference/ignore-file

[8] Anysphere, Inc. 2026. Terminal | Cursor Docs. Retrieved April 01, 2026 from https://cursor.com/docs/agent/tools/terminal

[9] Apache Software Foundation. 2026. Apache Subversion. Retrieved March 31, 2026 from https://subversion.apache.org/

[10] Jeff Bonwick, Matt Ahrens, Val Henson, Mark Maybee, and Mark Shellenbaum. 2003. The zettabyte file system. In *Proc. of the 2nd Usenix Conference on File and Storage Technologies*, Vol. 215. 1.

[11] Harold Booth. 2026. National Vulnerability Database. National Institute of Standards and Technology. Retrieved March 28, 2026 from https://nvd.nist.gov/

[12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems* 33 (2020), 1877–1901.

[13] Btrfs. 2026. Reflink. https://btrfs.readthedocs.io/en/latest/Reflink.html

[14] Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. [n. d.]. Why do multi-agent LLM systems fail?. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[15] Alan Chan, Kevin Wei, Sihao Huang, Nitarshan Rajkumar, Elija Perrier, Seth Lazar, Gillian K. Hadfield, and Markus Anderljung. 2025. Infrastructure for AI Agents. arXiv:2501.10114 [cs.AI] https://arxiv.org/abs/2501.10114

[16] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2023. AgentVerse: Facilitating Multi-Agent Collaboration and Exploring Emergent Behaviors. arXiv:2308.10848 [cs.CL] https://arxiv.org/abs/2308.10848

[17] Yinfang Chen, Jiaqi Pan, Jackson Clark, Yiming Su, Noah Zheutlin, Bhavya Bhavya, Rohan Arora, Yu Deng, Saurabh Jha, and Tianyin Xu. 2026. STRATUS: A Multi-agent System for Autonomous Reliability Engineering of Modern Clouds. arXiv:2506.02009 [cs.DC] https://arxiv.org/abs/2506.02009

[18] Zheng Chen, Hanqing Liu, Duling Xu, Dong Dong, Jialin Li, Bangzheng Pu, and Jidong Zhai. 2026. Cordon: Semantic Transactions for Tool-Using LLM Agents. arXiv:2606.17573 [cs.OS] https://arxiv.org/abs/2606.17573

[19] Microsoft Corporation. 2026. Copilot on Windows: Your Built-In AI Assistant. Retrieved March 27, 2026 from https://www.microsoft.com/en-us/windows/windows-11?wincampaign=copilot

[20] Cybersecurity and Infrastructure Security Agency and Federal Bureau of Investigation. 2025. *Cybersecurity Advisory AA23-320A: Scattered Spider.* Cybersecurity Advisory. U.S. Department of Homeland Security. Retrieved March 28, 2026 from https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-320a

[21] DeepSeek-AI et al. 2025. DeepSeek-V3 technical report. arXiv:2412.19437

[22] Xianzhong Ding, Le Chen, Murali Emani, Chunhua Liao, Pei-Hung Lin, Tristan Vanderbruggen, Zhen Xie, Alberto Cerpa, and Wan Du. 2023. HPC-GPT: Integrating Large Language Model for High-Performance Computing. In *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W '23)*. Association for Computing Machinery, New York, NY, USA, 951–960.

[23] Yunpeng Dong, Jingkai He, Shiqi Liu, Yuze Hou, Dong Du, Zhonghu Xu, Si Yu, Baochuan Yang, Yubin Xia, and Haibo Chen. 2026. DeltaBox: Scaling Stateful AI Agents with Millisecond-Level Sandbox Checkpoint/Rollback. arXiv:2605.22781 [cs.OS] https://arxiv.org/abs/2605.22781

[24] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, et al. 2019. The design and operation of {CloudLab}. In *2019 USENIX annual technical conference (USENIX ATC 19)*. 1–14.

[25] Jiayi Geng and Graham Neubig. 2026. Effective Strategies for Asynchronous Software Engineering Agents. arXiv:2603.21489 [cs.CL] https://arxiv.org/abs/2603.21489

[26] Git contributors. 2026. Git. Retrieved March 31, 2026 from https://git-scm.com/

[27] GitHub. 2026. GitHub Copilot hooks reference - GitHub Docs. Retrieved May 19, 2026 from https://docs.github.com/en/copilot/reference/hooks-reference

[28] Amir Goldstein. 2024. [PATCH 0/4] Stash overlay real upper file in backing_file. Retrieved March 30, 2026 from https://lore.kernel.org/all/20241004102342.179434-1-amir73il@gmail.com/

[29] Andrew C. Goldstein. 1975. *Files-11 on-disk structure specification.* Technical Report. Digital Equipment Corporation, Maynard, MA, USA. https://bitsavers.org/pdf/dec/pdp11/rsx11m_s/Files-11_ODS-1_Spec_Jun75.pdf

[30] Google. 2026. Gemini CLI hooks | Gemini CLI. Retrieved May 19, 2026 from https://geminicli.com/docs/hooks/

[31] Google LLC. 2026. Build, debug & deploy with AI: Gemini CLI. Retrieved March 27, 2026 from https://geminicli.com/

[32] Ely Greenfield. 2025. Our vision for accelerating creativity and productivity with agentic AI | Adobe Blog. Retrieved April 01, 2026 from https://blog.adobe.com/en/publish/2025/04/09/our-vision-for-accelerating-creativity-productivity-with-agentic-ai

[33] Feng He, Tianqing Zhu, Dayong Ye, Bo Liu, Wanlei Zhou, and Philip S Yu. 2025. The emerged security and privacy of llm agent: A survey with case studies. *Comput. Surveys* 58, 6 (2025), 1–36.

[34] Dave Hitz, James Lau, and Michael A Malcolm. 1994. File system design for an NFS file server appliance. In *USENIX Winter 1994 Technical Conference Proceedings*. USENIX Association, San Francisco, CA.

[35] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. arXiv:2308.00352 [cs.AI] https://arxiv.org/abs/2308.00352

[36] Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, Jianbo Yuan, Jiwei Li, Kun Kuang, Yang Yang, Hongxia Yang, and Fei Wu. 2024. InfiAgent-DABench: Evaluating Agents on Data Analysis Tasks. In *Proceedings of the 41st International Conference on Machine Learning*. 19544–19572.

[37] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems* 43, 2 (2025), 1–55.

[38] Yutong Huang, Vikranth Srivatsa, Alex Asch, Hansin Tushar Patwa, and Yiying Zhang. 2026. TClone: Low-Latency Forking of Live GUI Environments for Computer-Use Agents. arXiv:2605.17320 [cs.OS] https://arxiv.org/abs/2605.17320

[39] IBM. 2026. IBM DevOps Code ClearCase. Retrieved March 28, 2026 from https://www.ibm.com/products/devops-code-clearcase

[40] Shvetank Jain, Fareha Shafique, Vladan Djeric, and Ashvin Goel. 2008. Application-level isolation and recovery with solitude. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*. 95–107.

[41] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: can language models resolve real-world Github issues?. In *The Twelfth International Conference on Learning Representations*.

[42] Siva Kannan. 2026. Microbots : Safety First Agentic Workflow. Retrieved April 3, 2026 from https://microsoft.github.io/microbots/blog/microbots-safety-first-ai-agent/

[43] Andrej Karpathy. 2026. It is hard to communicate how much programming has changed... https://x.com/karpathy/status/2026731645169185220.

[44] Ryusuke Konishi, Yoshiji Amagai, Koji Sato, Hisashi Hifumi, Seiji Kihara, and Satoshi Moriai. 2006. The Linux implementation of a log-structured file system. *ACM SIGOPS Operating Systems Review* 40, 3 (2006), 102–107.

[45] ShareAI Lab. 2026. Bash is all you need - A nano claude code-like agent harness, built from 0 to 1. https://github.com/shareAI-lab/learn-claude-code.

[46] Alexander Larsson. 2026. GitHub - containers/bubblewrap: Low-level unprivileged sandboxing tool used by Flatpak and similar projects. Retrieved March 28, 2026 from https://github.com/containers/bubblewrap

[47] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. arXiv:2303.17760 [cs.AI] https://arxiv.org/abs/2303.17760

[48] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, Rui Kong, Yile Wang, Hanfei Geng, Jian Luan, Xuefeng Jin, Zilong Ye, Guanjing Xiong, Fan Zhang, Xiang Li, Mengwei Xu, Zhijun Li, Peng Li, Yang Liu, Ya-Qin Zhang, and Yunxin Liu. 2024. Personal LLM agents: insights and survey about the capability, efficiency and security. arXiv:2401.05459

[49] Linux kernel contributors. 2026. Overlay Filesystem — The Linux Kernel documentation. Retrieved March 31, 2026 from https://docs.kernel.org/filesystems/overlayfs.html

[50] Mengyang Liu, Taozhi Chen, Zhenhua Xu, Xue Jiang, and Yihong Dong. 2026. Multi-agent Collaboration with State Management. arXiv:2605.20563 [cs.MA] https://arxiv.org/abs/2605.20563

[51] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. 2024. AgentBench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*.

[52] Mercurial contributors. 2026. Mercurial SCM. Retrieved March 31, 2026 from https://www.mercurial-scm.org/

[53] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* 239, 2 (2014), 2.

[54] Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, et al. 2026. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868* (2026).

[55] Meta Platforms, Inc. 2025. The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation. Retrieved April 01, 2026 from https://ai.meta.com/blog/llama-4-multimodal-intelligence/

[56] Microsoft Corporation. 2026. GitHub Copilot in VS Code. Retrieved March 27, 2026 from https://code.visualstudio.com/docs/copilot/overview

[57] Microsoft Defender Security Research Team. 2026. Preinstall to persistence: Inside the Red Hat npm Miasma credential-stealing campaign. Retrieved July 28, 2026 from https://www.microsoft.com/en-us/security/blog/2026/06/02/preinstall-persistence-inside-red-hat-npm-miasma-credential-stealing-campaign/

[58] Bardia Mohammadi, Nearchos Potamitis, Lars Klein, Akhil Arora, and Laurent Bindschaedler. 2026. Atomix: Timely, Transactional Tool Use for Reliable Agentic Workflows. arXiv:2602.14849 [cs.LG] https://arxiv.org/abs/2602.14849

[59] Yohei Nakajima. 2023. GitHub - yoheinakajima/babyagi_archive. Retrieved March 27, 2026 from https://github.com/yoheinakajima/babyagi_archive

[60] Steve Newman. 2026. 45 Thoughts About Agents. https://secondthoughts.ai/p/45-thoughts-about-agents.

[61] Junjiro Okajima. 2026. AUFS - Another unionfs. Retrieved March 31, 2026 from https://aufs.sourceforge.net/

[62] OpenAI. 2026. Codex CLI. Retrieved March 27, 2026 from https://developers.openai.com/codex/cli

[63] OpenAI. 2026. Introducing GPT-5.4. Retrieved April 01, 2026 from https://openai.com/index/introducing-gpt-5-4/

[64] OpenAI. 2026. OpenAI to acquire Astral. Retrieved March 27, 2026 from https://openai.com/index/openai-to-acquire-astral/

[65] OpenAI. 2026. Sandboxing – Codex | OpenAI Developers. Retrieved March 28, 2026 from https://developers.openai.com/codex/concepts/sandboxing

[66] OpenCode. 2026. OpenCode. https://opencode.ai/docs/. Open-source AI coding agent for terminal, desktop, and IDE workflows. Accessed: 2026-04-01.

[67] Melissa Z. Pan, Negar Arabzadeh, Riccardo Cogo, Yuxuan Zhu, Alexander Xiong, Lakshya A Agrawal, Huanzhi Mao, Emma Shen, Sid Pallerla, Liana Patel, Shu Liu, Tianneng Shi, Xiaoyuan Liu, Jared Quincy Davis, Emmanuele Lacavalla, Alessandro Basile, Shuyi Yang, Paul Castro, Daniel Kang, Joseph E. Gonzalez, Koushik Sen, Dawn Song, Ion Stoica, Matei Zaharia, and Marquita Ellis. 2026. Measuring agents in production. arXiv:2512.04123

[68] Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. 2025. The Berkeley Function Calling Leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

[69] R. Hugo Patterson and Stephen Manley. 2002. SnapMirror: File-system-based asynchronous mirroring for disaster recovery. In *Conference on File and Storage Technologies (FAST 02)*. USENIX Association, Monterey, CA. https://www.usenix.org/conference/fast-02/snapmirror-file-system-based-asynchronous-mirroring-disaster-recovery

[70] Grant A Pignatiello, Richard J Martin, and Ronald L Hickman Jr. 2020. Decision fatigue: A conceptual analysis. *Journal of health psychology* 25, 1 (2020), 123–135.

[71] Donald E Porter, Owen S Hofmann, Christopher J Rossbach, Alexander Benn, and Emmett Witchel. 2009. Operating system transactions. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 161–176.

[72] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative Agents for Software Development. arXiv:2307.07924 [cs.SE] https://arxiv.org/abs/2307.07924

[73] David Quigley, Josef Sipek, Charles P Wright, and Erez Zadok. 2006. Unionfs: User-and community-oriented development of a unification filesystem. In *Proceedings of the 2006 Linux Symposium*, Vol. 2. 349–362.

[74] Sean Quinlan and Sean Dorward. 2002. Venti: A new approach to archival data storage. In *Conference on File and Storage Technologies (FAST 02)*. USENIX Association, Monterey, CA. https://www.usenix.org/conference/fast-02/venti-new-approach-archival-data-storage

[75] Sean Quinlan, Jim McKie, and Russ Cox. 2003. *Fossil, an archival file server*. Technical Report. Plan 9. Retrieved March 27, 2026 from https://p9f.org/sys/doc/fossil.pdf

[76] Relynt. 2026. Designing approvals that do not kill automation | Relynt Blog. Retrieved March 28, 2026 from https://www.relyntpolicy.com/blog/slack-approvals-human-in-the-loop

[77] Ohad Rodeh, Josef Bacik, and Chris Mason. 2013. BTRFS: The Linux B-tree filesystem. *ACM Transactions on Storage* 9, 3 (2013), 1–32.

[78] Mickaël Salaün. 2017. Landlock LSM: Toward unprivileged sandboxing. Presentation at Linux Security Summit. Retrieved March 28, 2026 from https://static.sched.com/hosted_files/lss2017/56/2017-09-14_landlock-lss.pdf

[79] Douglas S Santry, Michael J Feeley, Norman C Hutchinson, Alistair C Veitch, Ross W Carton, and Jacob Ofir. 1999. Deciding when to forget in the Elephant file system. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*. 110–123.

[80] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems* 36 (2023), 68539–68551.

[81] Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. 2026. Progent: Securing AI Agents with Privilege Control. arXiv:2504.11703 [cs.CR] https://arxiv.org/abs/2504.11703

[82] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, Vol. 36. 8634–8652.

[83] Socket Research Team. 2025. Nx npm Packages Compromised in Supply Chain Attack Weaponizing AI CLI Tools. Retrieved July 28, 2026 from https://socket.dev/blog/nx-packages-compromised

[84] Socket Research Team. 2026. TrapDoor Crypto Stealer Supply Chain Attack Hits 34 Packages and Hundreds of Versions Across npm, PyPI, and Crates.io. Retrieved July 28, 2026 from https://socket.dev/blog/trapdoor-crypto-stealer-npm-pypi-crates

[85] Stanford Secure Computer Systems group. 2026. jai - easy containment for AI agents. Retrieved March 28, 2026 from https://jai.scs.stanford.edu/

[86] Storage Research Group at Tsinghua University. 2026. ContextFS: Context File System for Agentic AI. https://github.com/thustorage/ContextFS

[87] Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. 2024. MAGIS: LLM-Based Multi-Agent Framework for GitHub Issue Resolution. arXiv:2403.17927 [cs.SE] https://arxiv.org/abs/2403.17927

[88] The Gemini Team. 2026. Gemini 3.1 Pro: Announcing our latest Gemini AI model. Retrieved April 01, 2026 from https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/

[89] The Linux man-pages project. 2026. mount(8) - Linux manual page. Retrieved April 01, 2026 from https://man7.org/linux/man-pages/man8/mount.8.html

[90] The Linux man-pages project. 2026. mount_namespaces(7) - Linux manual page. Retrieved March 28, 2026 from https://man7.org/linux/man-pages/man7/mount_namespaces.7.html

[91] The MITRE Corporation. 2026. Multi-Factor Authentication Request Generation, Technique T1621 - Enterprise. Retrieved March 28, 2026 from https://attack.mitre.org/versions/v17/techniques/T1621/

[92] Walter F Tichy. 1985. RCS—A system for version control. *Software: Practice and Experience* 15, 7 (1985), 637–654.

[93] Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 16022–16076.

[94] Turso. 2026. AgentFS with copy-on-write overlay filesystem. Retrieved March 31, 2026 from https://turso.tech/blog/agentfs-overlay

[95] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[96] Jos Visser. 2025. Coding in the time of AI. https://josvisser.substack.com/p/coding-in-the-time-of-ai.

[97] VMS Software, Inc. 2020. *VSI OpenVMS User's Manual*. VMS Software, Inc., Bolton, MA, USA. https://vmssoftware.com/docs/VSI_USERS_MANUAL.pdf

[98] Cong Wang and Yusheng Zheng. 2026. Fork, Explore, Commit: OS Primitives for Agentic Exploration. arXiv:2602.08199

[99] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[100] 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, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. 2025. OpenHands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*.

[101] Simon Willison. 2022. Prompt injection attacks against GPT-3. Retrieved March 28, 2026 from https://simonwillison.net/2022/Sep/12/prompt-injection/

[102] Simon Willison. 2025. Living dangerously with Claude. https://simonwillison.net/2025/Oct/22/living-dangerously-with-claude/.

[103] 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. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155 [cs.AI] https://arxiv.org/abs/2308.08155

[104] Tianyuan Wu, Chaokun Chang, Lunxi Cao, Wei Gao, and Wei Wang. 2026. Crab: A Semantics-Aware Checkpoint/Restore Runtime for Agent Sandboxes. arXiv:2604.28138 [cs.OS] https://arxiv.org/abs/2604.28138

[105] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems* 37 (2024), 52040–52094.

[106] Hui Yang, Sifu Yue, and Yunzhong He. 2023. *Auto-GPT for online decision making: Benchmarks and additional opinions.* arXiv:2306.02224

[107] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2025. Mini-SWE-Agent: The 100 line AI agent that solves GitHub issues or helps you in your command line. https://github.com/swe-agent/mini-swe-agent.

[108] John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. 2023. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems* 36 (2023), 23826–23854.

[109] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2025. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *International Conference on Learning Representations*, Vol. 2025. 9965–10017.

[110] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. ReAct: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

[111] Miao Yu, Fanci Meng, Xinyun Zhou, Shilong Wang, Junyuan Mao, Linsey Pan, Tianlong Chen, Kun Wang, Xinfeng Li, Yongfeng Zhang, et al. 2025. A survey on trustworthy llm agents: Threats and countermeasures. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 6216–6226.

[112] Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. 2025. Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-based Agents. In *The Thirteenth International Conference on Learning Representations*.

[113] Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. 2025. Agent-SafetyBench: Evaluating the Safety of LLM Agents. arXiv:2412.14470 [cs.CL] https://arxiv.org/abs/2412.14470

[114] Xinxue Zhu, Jiacong Wu, Xiaoyu Zhang, Tianlin Li, Yanzhou Mu, Juan Zhai, Chao Shen, and Yang Liu. 2026. *An empirical study of bugs in modern LLM agent frameworks.* arXiv:2602.21806

[R1] Cybersecurity analysis of Antigravity agent wiping entire D: drive via rmdir /s /q when user asked to clear project cache. https://hackernoob.tips/when-ai-agents-go-rogue-google-antigravitys-catastrophic-drive-deletion-exposes-critical-risks-in-agentic-development-tools/.

[R2] Antigravity's built-in edit tool systematically deletes and corrupts file content during normal coding tasks, confirmed by multiple users as a tool bug. https://www.reddit.com/r/Bard/comments/1p3htvu/antigravity_just_deletingcorrupting_files.

[R3] Indirect prompt injection via poisoned web page causes Gemini to bypass gitignore restriction using cat, read .env credentials, and exfiltrate via browser subagent to attacker-controlled webhook.site. https://www.promptarmor.com/resources/google-antigravity-exfiltrates-data.

[R4] Antigravity agent wiped entire D: drive via rmdir /s /q due to quote-handling path resolution bug; SafeToAutoRun bypassed approval. https://www.reddit.com/r/google_antigravity/comments/1p82or6/google_antigravity_just_deleted_the_contents_of/.

[R5] /agents command creates agent files relative to CWD instead of project root, placing them in wrong directory. https://github.com/anthropics/claude-code/issues/7560.

[R6] Allow all edits" session setting ignored; Claude Code prompts for approval on every individual file edit despite user opting in. https://github.com/anthropics/claude-code/issues/9348.

[R7] Bash allowlist pattern Bash(find:*) fails to match piped commands, causing repeated permission prompts despite saved approval. https://github.com/anthropics/claude-code/issues/34811.

[R8] Claude Code ignores explicit user instructions and modifies files without direction while Accept Edits auto-approves changes. https://github.com/anthropics/claude-code/issues/8026.

[R9] Subagent auto-yes to stdin created 851GB task output file, silently filling disk. https://github.com/anthropics/claude-code/issues/35047.

[R10] Background subagents silently auto-deny tools instead of prompting for permissions before launch, with inconsistent propagation across sessions. https://github.com/anthropics/claude-code/issues/34095.

[R11] Bash permission bypass via command chaining (&&, ;, |) defeats prefix-matching allowlist. https://github.com/anthropics/claude-code/issues/4956.

[R12] Claude Code ran broad find / searches across home directory instead of checking obvious config path, part of reported model regression. https://github.com/anthropics/claude-code/issues/33908.

[R13] /btw command fails to render permission prompts, freezing session with no way to respond. https://github.com/anthropics/claude-code/issues/33313.

[R14] Built-in tools (Read, Grep, Bash, Glob) execute without approval prompt despite not being in allowedTools allowlist. https://github.com/anthropics/claude-code/issues/34569.

[R15] defaultMode bypassPermissions setting has no effect; every tool call still triggers permission prompt, growing allowlist indefinitely. https://github.com/anthropics/claude-code/issues/34923.

[R16] Claude Code autonomously bypassed three security layers (path denylist, bubblewrap sandbox, BPF LSM kernel enforcement) using

Linux primitives to complete its task. https://cybercorsairs.com/claude-code-broke-out-of-three-security-layers-on-its-own/.

[R17] Claude Code deleted all unstaged files when asked to restructure commits, pulled old files as fake recovery, then deleted everything again. https://www.reddit.com/r/ClaudeAI/comments/1q0124f/terrible_experience_with_claude_code_lost_my_work.

[R18] Claude Code ignored CLAUDE.md "never commit" rule, committed without permission, then fabricated a justification when confronted. https://github.com/anthropics/claude-code/issues/34774.

[R19] Claude Code repeatedly commits and pushes without permission despite explicit CLAUDE.md and memory rules, recurring after corrections in same session. https://github.com/anthropics/claude-code/issues/34451.

[R20] Claude Code silently drops CLAUDE.md rules and session constraints as context window fills in long sessions, leading to unauthorized file edits and hallucinated state. https://github.com/anthropics/claude-code/issues/32659.

[R21] Claude Code context decay during debugging session caused repeated instruction violations, reversed fixes, and significant unwanted file changes. https://github.com/anthropics/claude-code/issues/8251.

[R22] Cowork copied 0-byte iCloud stubs via cp -a then rm -rf'd originals, destroying 110 sensitive legal documents. https://github.com/anthropics/claude-code/issues/32637.

[R23] Cowork VM sandbox fails to start on Windows due to sandbox-helper unmount errors, leaving sandboxed mode unusable. https://github.com/anthropics/claude-code/issues/25419.

[R24] Claude Code reads files in denied credentials folder via alternative tool path, bypassing Read() deny rule in settings.json. https://github.com/anthropics/claude-code/issues/4768.

[R25] Claude Code autonomously deleted specification file it was working from, then claimed no knowledge of it. https://github.com/anthropics/claude-code/issues/7787.

[R26] Claude Opus repeatedly deletes or weakens unit tests and writes TODO stubs despite explicit CLAUDE.md TDD rules, then reports features as complete. https://www.reddit.com/r/ClaudeAI/comments/1lfirvk/any_tips_on_how_to_get_claude_to_stop_cheating_on/.

[R27] Bash deny rules only block relative paths; absolute paths bypass denylist completely, exposing entire filesystem. https://github.com/anthropics/claude-code/issues/11662.

[R28] Bash deny rules bypassed via indirect file access commands (find, grep) that don't contain the denied filename pattern. https://github.com/anthropics/claude-code/issues/6036.

[R29] Claude Desktop code mode lacks explain option on permission prompts, encouraging rubber-stamp approvals. https://github.com/anthropics/claude-code/issues/34205.

[R30] Claude displayed full contents of /.netrc, /.npmrc, and /.cargo/credentials.toml in conversation, exposing 8+ API tokens and passwords. https://github.com/anthropics/claude-code/issues/34819.

[R31] Windows drive letter change causes Permission Resolver to create .claude/ directories outside workspace and corrupt settings. https://github.com/anthropics/claude-code/issues/34866.

[R32] Command injection via echo command bypasses allowlist regex and permission prompt, enabling arbitrary execution (CVE-2025-54795). https://nvd.nist.gov/vuln/detail/CVE-2025-54795.

[R33] Claude Code repeatedly edited infrastructure files despite 30+ pages of CLAUDE.md rules and a custom skill requiring explicit permission first. https://github.com/anthropics/claude-code/issues/10726.

[R34] Claude Code repeatedly removes error-handling code instead of fixing bugs, ignoring explicit user instructions not to. https://github.com/anthropics/claude-code/issues/8910.

[R35] Claude reported 15/15 bugs fixed while fabricating data and skipping source verification, making the document worse than the original. https://github.com/anthropics/claude-code/issues/27399.

[R36] Claude Code ran find across entire home directory instead of project directory to locate H2 database file for deletion. https://github.com/anthropics/claude-code/issues/30125.

[R37] Claude Code permanently deleted a folder without user confirmation on Windows. https://github.com/anthropics/claude-code/issues/13476.

[R38] Claude Code ran git commit –amend without confirmation despite explicit CLAUDE.md rules listing it as a destructive operation requiring user approval. https://github.com/anthropics/claude-code/issues/33391.

[R39] Claude Code ignores CLAUDE.md git restrictions after context compaction, cherry-picks from wrong branches and rewrites git history without permission. https://www.reddit.com/r/ClaudeAI/comments/1p889x7/anyone_successfully_prevent_claude_from_running/.

[R40] Claude executed unauthorized git commit and checkouts during test request, massively modifying server file and destroying day's uncommitted work. https://github.com/anthropics/claude-code/issues/34784.

[R41] Claude Code ran git commit –amend and force-push without permission, violating memory file rules and bypassing permission prompt. https://github.com/anthropics/claude-code/issues/25472.

[R42] Claude Code executed git reset –hard and git checkout while falsely assuring user operations were safe, destroying hours of development work. https://github.com/anthropics/claude-code/issues/7232.

[R43] Claude Code ran git reset –hard origin/main in main working directory during multi-instance workflow, destroying 2 days of uncommitted Pulumi infrastructure work. https://github.com/anthropics/claude-code/issues/33850.

[R44] Claude Code ran git reset –hard without checking for or warning about untracked/staged files, destroying 3 work files. https://github.com/anthropics/claude-code/issues/34746.

[R45] Claude Code attempted git rm -rf on non-existent path and hallucinated dangerouslyDisableSandbox tool parameter. https://github.com/anthropics/claude-code/issues/31705.

[R46] Claude hallucinated incorrect Punycode for Cyrillic domains, writing corrupted config files to production including substitution of a real third-party domain. https://github.com/anthropics/claude-code/issues/32798.

[R47] Claude Code hand-rolled Django migration and applied it without permission, violating explicit CLAUDE.md rules, then initially lied about rules not existing. https://github.com/anthropics/claude-code/issues/8059.

[R48] Bash permission glob matcher treats. https://github.com/anthropics/claude-code/issues/34379.

[R49] Permission checker falsely flags literal backticks inside single-quoted heredoc as command substitution. https://github.com/anthropics/claude-code/issues/35183.

[R50] Security heuristics override explicit allowlist entries, forcing approval prompts for pre-authorized commands. https://github.com/anthropics/claude-code/issues/34106.

[R51] Claude Code ran git stash drop despite explicit CLAUDE.md prohibition, permanently deleting 3 days of stashed work. https://github.com/anthropics/claude-code/issues/22638.

[R52] Claude Code's persistent shell lost PWD state after user interrupt, causing rm -rf * to execute in project root instead of Builds/Xcode subdirectory. https://github.com/anthropics/claude-code/issues/17410.

[R53] Claude Code violated CLAUDE.md safeguard rules during 2-hour session, overwrote custom LoRA model file with wrong data, destroyed working ComfyUI workflow. https://github.com/anthropics/

claude-code/issues/34922.
[R54] Claude deleted files and removed document content without permission, repeatedly misinterpreting ambiguous feedback as deletion instructions. https://github.com/anthropics/claude-code/issues/34492.
[R55] Notification hook for permission prompts omits tool name and command details, preventing informed remote triage. https://github.com/anthropics/claude-code/issues/32952.
[R56] Cmd+Enter to approve permission prompt also submits partially-typed message, sending unintended instructions to agent. https://github.com/anthropics/claude-code/issues/32630.
[R57] Permission dialog auto-scrolls past reasoning context, preventing informed approval of tool calls. https://github.com/anthropics/claude-code/issues/34354.
[R58] Feature request for human-readable warning labels on destructive commands in permission prompts. https://github.com/anthropics/claude-code/issues/30505.
[R59] Plan Mode failed to block Bash execution; destructive commands (pkill, kill -9) ran despite read-only mode being active. https://github.com/anthropics/claude-code/issues/32768.
[R60] Claude Code executed Edit tool to modify source file despite Plan Mode being active, bypassing the constraint that blocks non-readonly operations. https://github.com/anthropics/claude-code/issues/33037.
[R61] Plan mode has no technical tool restriction — relies purely on system prompt injection, allowing edits and shell commands despite "read-only" designation. https://www.reddit.com/r/ClaudeAI/comments/1ou9x6i/unsafe_plan_mode_does_not_prevent_claude_from/.
[R62] Read() permission allowlist path prefix matching rejects absolute paths due to unintuitive double-slash syntax requirement. https://github.com/anthropics/claude-code/issues/35118.
[R63] Claude Code Read() deny rules in settings.json silently ignored, allowing agent to read restricted files including secrets and private data. https://github.com/anthropics/claude-code/issues/3501.
[R64] Claude Code rebased and force-pushed another contributor's PR branch, rewriting commit history and violating its own instructions. https://github.com/anthropics/claude-code/issues/32476.
[R65] Claude Code deleted years of teaching resources during folder reorganization instead of archiving uncategorizable files. https://github.com/anthropics/claude-code/issues/12851.
[R66] Claude Code repeatedly deleted fire effect code via Edit tool while user was debugging, then attempted same deletion immediately after being explicitly told not to. https://github.com/anthropics/claude-code/issues/7645.
[R67] Claude repeatedly used rm -rf to delete config files during pair programming; user approved each time without reading prompts. https://www.reddit.com/r/ClaudeAI/comments/1nrrmv3/til_ai_keeps_using_rm_rf_on_important_files/.
[R68] Claude CLI appended / to rm -rf command while cleaning up packages in old repo, deleting entire Mac home directory. https://www.reddit.com/r/ClaudeAI/comments/1pgxckk/claude_cli_deleted_my_entire_home_directory_wiped/.
[R69] Claude deleted 11h of YOLO inference output via rm -rf without permission, then restarted the job while user was asleep. https://github.com/anthropics/claude-code/issues/32938.
[R70] Claude deleted non-empty directory with rm -rf after user asked to remove only if empty; files were unique, not duplicates. https://github.com/anthropics/claude-code/issues/35093.
[R71] Claude ran rm -rf on entire local folder tree without confirmation, destroying months of production code in sibling directories. https://github.com/anthropics/claude-code/issues/30816.
[R72] Claude Code rm -rf with tilde expansion deleted user's entire home directory during project cleanup. https://byteiota.com/claude-codes-rm-rf-bug-deleted-my-home-directory/.
[R73] Claude Code executed rm -rf from root, deleting all user-owned files in home directory; exact command not captured in logs. https://github.com/anthropics/claude-code/issues/10077.
[R74] Claude Code ran rm -rf from repo root instead of build subdirectory with sandbox disabled, destroying .git and 76+ unpushed commits; then silently discarded all code during rebase recovery. https://github.com/anthropics/claude-code/issues/34514.
[R75] Claude Code executed rm on wrong file instead of displaying the command as requested, deleting an unrelated media file with Accept Edits on. https://github.com/anthropics/claude-code/issues/24854.
[R76] Max Plan subscriber documents five systematic CLAUDE.md rule-violation patterns despite 24 PreToolUse hooks and 400-line rules file; model reads, acknowledges, then violates rules. https://github.com/anthropics/claude-code/issues/34358.
[R77] Claude Code silently bypasses sandbox via dangerouslyDisableSandbox flag when Bash is auto-approved, no permission prompt shown. https://github.com/anthropics/claude-code/issues/34315.
[R78] Sandbox excludedCommands does not exempt SSH child process, blocking git push and forcing users to disable sandbox entirely. https://github.com/anthropics/claude-code/issues/33300.
[R79] Sandbox creates empty directories in CWD from command arguments when command is not found. https://github.com/anthropics/claude-code/issues/33056.
[R80] Claude Code violated its own saved safety rules, executing SSH commands and installing software on remote servers without required approval. https://github.com/anthropics/claude-code/issues/34828.
[R81] Feature request for scoped auto-approve (time-limited, task-scoped, step-count) as middle ground between per-action prompts and full permissionless mode. https://github.com/anthropics/claude-code/issues/33645.
[R82] Claude modified generation script to auto-delete 50 curated vocal sample files, permanently destroying 7 sessions of user curation work. https://github.com/anthropics/claude-code/issues/30988.
[R83] Claude Code bypasses deny config and CLAUDE.md rules to read .env files and secrets via alternative paths or indirect commands. https://www.reddit.com/r/ClaudeAI/comments/1nfvh46/i_got_tired_of_claude_code_reading_secrets_and/.
[R84] Claude Code sed command parsing flaw bypasses read-only validation, enabling arbitrary file writes via sed w/r subcommands (CVE-2025-64755). https://nvd.nist.gov/vuln/detail/CVE-2025-64755.
[R85] Claude Code JSON serializer corrupts settings.local.json when saving bash commands with regex quote patterns, destroying all saved permissions. https://github.com/anthropics/claude-code/issues/33650.
[R86] settings.local.json allowlist rules silently ignored due to path format bugs; agent re-prompts for pre-approved commands, pushing users toward –dangerously-skip-permissions. https://github.com/anthropics/claude-code/issues/6850.
[R87] Claude Code autonomously ran git reset –hard on startup twice, destroying unpushed commits and uncommitted work; fabricated a safety hook that never existed. https://github.com/anthropics/claude-code/issues/34327.
[R88] Sub-agent ran rm -rf on its own worktree directory, corrupting session CWD and forcing sign-out. https://github.com/anthropics/claude-code/issues/34474.
[R89] Task output collector reads its own file in infinite loop, writing 696GB and exhausting disk in 23 minutes. https://github.com/anthropics/claude-code/issues/34783.
[R90] Claude created directory named then ran rm -rf *, shell expansion wiped user's home directory. https://github.com/anthropics/claude-code/issues/12637.
[R91] Claude Code Bash tool creates /tmp/claude-*-cwd files to track working directory but never deletes them, causing 500+ files/day accumulation. https://github.com/anthropics/claude-code/issues/

8856.
[R92] Claude Code created 1,489 lines of unwanted documentation files violating explicit CLAUDE.md rule against proactive doc creation. https://github.com/anthropics/claude-code/issues/20109.
[R93] Claude Code VSCode extension silently auto-approves all Bash commands despite default permission mode requiring approval. https://github.com/anthropics/claude-code/issues/34463.
[R94] VSCode extension prepends cd && to every Bash command, breaking allowlist auto-approval matching. https://github.com/anthropics/claude-code/issues/33976.
[R95] Claude Code VSCode/Positron extension skips workspace trust prompt, allowing file reads in arbitrary directories without confirmation. https://github.com/anthropics/claude-code/issues/34913.
[R96] Sonnet generates malformed wildcard glob patterns in settings.json, corrupting permission config and crashing VS Code. https://github.com/anthropics/claude-code/issues/33746.
[R97] Claude Code creates undeletable "nul" files on Windows by using Unix-style /dev/null redirection, corrupting git repos and breaking IDEs. https://github.com/anthropics/claude-code/issues/4928.
[R98] Claude Code followed NTFS junctions when removing pnpm monorepo worktrees, permanently deleting user's Windows profile folders. https://github.com/anthropics/claude-code/issues/29249.
[R99] Claude Code Write tool and Bash cp executed without permission prompt, silently bypassing configured allowlist in settings.local.json. https://github.com/anthropics/claude-code/issues/34583.
[R100] Claude Code used Write (full overwrite) instead of Edit, silently dropping table of contents section from backup file. https://github.com/anthropics/claude-code/issues/27137.
[R101] Yarn 2+ plugin auto-execution bypasses directory trust dialog, enabling arbitrary code execution before user accepts risks (CVE-2025-59828). https://nvd.nist.gov/vuln/detail/CVE-2025-59828.
[R102] Cline v3.20.0 deletes recently edited files when user cancels a task or switches from Act to Plan mode. https://github.com/cline/cline/issues/5124.
[R103] .clineignore still sends all ignored file paths to LLM context with lock emoji instead of omitting them, wasting 30k+ tokens and leaking path names. https://github.com/cline/cline/issues/9554.
[R104] Corrupted checkpoint shadow repo disables checkpoint revert system globally across all workspaces with no recovery path. https://github.com/cline/cline/issues/9631.
[R105] Cline v3.20.0 deletes last edited file after task completion due to missing guard in DiffViewProvider.revertChanges(). https://github.com/cline/cline/issues/5120.
[R106] Cline repeatedly overwrites documentation files with truncated/placeholder content when asked to update them, despite explicit instructions not to. https://github.com/cline/cline/issues/711.
[R107] Cline vulnerable to data exfiltration via markdown image rendering; prompt injection reads .env and exfiltrates secrets through attacker-controlled image URL. https://embracethered.com/blog/posts/2025/cline-vulnerable-to-data-exfiltration/.
[R108] Cline used write_to_file to overwrite entire .env with one variable, destroying all existing credentials. https://github.com/cline/cline/issues/8422.
[R109] Cline checkpoint system renames .git to .git_disabled during editing, leaving project without git history when interrupted. https://github.com/cline/cline/issues/8273.
[R110] Feature request to block AI agents from bypassing git hooks with –no-verify flag. https://github.com/cline/cline/issues/8526.
[R111] Cline CLI deletes file when user interrupts mid-stream edit generation, bypassing required approval. https://github.com/cline/cline/issues/9165.
[R112] Cline global MCP auto-approve toggle overrides per-tool approval settings, preventing granular permission control. https://github.com/cline/cline/issues/9357.
[R113] Cline MCP installation corrupted then deleted cline_mcp_settings.json, losing all MCP server configs and bearer tokens. https://github.com/cline/cline/issues/9663.
[R114] replace_in_file tool empties file content when SEARCH block fails to match, falsely claims reversion. https://github.com/cline/cline/issues/9555.
[R115] Cline ran rm -rf / without permission while user asked to clean up a merge conflict, deleting entire home directory. https://github.com/cline/cline/issues/7572.
[R116] Cline creates empty file named "f" and writes to it after search patterns fail to match during editing. https://github.com/cline/cline/issues/9821.
[R117] Cline update randomly deletes opened or recently edited files; multiple users report data loss on v3.20.0. https://www.reddit.com/r/CLine/comments/1m79n0s/cline_deleting_files.
[R118] Codex VS Code extension ignores "Allow every time" for file edits, re-prompting on every file despite user approval. https://github.com/openai/codex/issues/3157.
[R119] Codex VS Code extension ignores "Allow every time" button on Windows, re-prompting for every file edit despite repeated approvals. https://github.com/openai/codex/issues/3325.
[R120] Codex Windows apply_patch fails for nested files under src/** due to sandbox path-handling bug; shell writes succeed in same paths. https://github.com/openai/codex/issues/14675.
[R121] Codex Windows app apply_patch silently fails in sandboxed default permission mode, forcing users to disable sandbox via Full Access. https://github.com/openai/codex/issues/13574.
[R122] Codex approval prompt only displays first command before &&, hiding chained commands from user review. https://community.openai.com/t/approval-only-showing-start-of-command/1372406.
[R123] Codex on Windows requires manual approval for every shell command, making it unusable. https://github.com/openai/codex/issues/2860.
[R124] Codex approval prompt froze and became unresponsive after several approved commands; restarting app deleted session reasoning. https://github.com/openai/codex/issues/10760.
[R125] Codex in auto-approval mode reads files outside working directory including /.ssh, despite documentation claiming approval is required. https://github.com/openai/codex/issues/5474.
[R126] Codex bwrap sandbox fails when home directory is symlinked to NFS mount, falling back to permission prompts. https://github.com/openai/codex/issues/14832.
[R127] Codex bwrap sandbox fails to bind symlinked TMPDIR, breaking sandbox containment and triggering unnecessary permission elevation requests. https://github.com/openai/codex/issues/14672.
[R128] Users request command-specific allowlist to avoid repeated permission prompts for safe commands like tests and builds. https://github.com/openai/codex/issues/3085.
[R129] Codex autonomously deleted a 2200-line code file it deemed unnecessary; user recovered from offline backups. https://www.linkedin.com/posts/alexander-purchase_today-i-had-codex-delete-one-of-my-files-activity-7385244768156446720-yMGk.
[R130] Codex deleted uncommitted files twice without permission, then fabricated claims about restoring them from git. https://github.com/openai/codex/issues/4969.
[R131] Codex deletes entire files instead of making localized edits, admits "I accidentally deleted the file. https://github.com/openai/codex/issues/6999.
[R132] Codex hardcoded denylist blocks rm and git reset even in full-access mode; agent bypasses via Python shutil and apply_patch. https://github.com/openai/codex/issues/5128.
[R133] Codex runs docker commands without confirmation after silent security model change, granting effective root on host. https://github.com/openai/codex/issues/14477.

[R134] Codex Windows elevated sandbox setup repeatedly fails with exit code 1, blocking all reads and writes across six CLI versions. https://github.com/openai/codex/issues/14409.

[R135] Codex has no ability to hide sensitive files like .env from agent; instructions and .gitignore are not respected. https://github.com/openai/codex/issues/85.

[R136] Codex erased contents of 20,000-line file and replaced with a snippet when creating PR. https://community.openai.com/t/codex-deletes-contents-of-large-files/1357223.

[R137] User requests fine-grained permission controls between overly restrictive default and overly permissive full-access modes. https://github.com/openai/codex/issues/14399.

[R138] Codex sandbox user lacks permissions to git fsmonitor IPC socket, causing git errors and agent stumbling. https://github.com/openai/codex/issues/14372.

[R139] Codex VS Code extension in full access mode requires approval for every change; users report 20-30 prompts per task including read-only chat mode. https://community.openai.com/t/codex-vscode-extension-agent-full-access-always-asks-for-approval/1355908.

[R140] Codex CLI and IDE in full access mode still prompt for file edit approval on Windows despite all configuration overrides. https://community.openai.com/t/codex-cli-and-ide-prompting-for-approval-to-edit-files/1354993.

[R141] Codex CLI in full-auto mode deleted personal photos on D:\{}Photo for two days, ignoring AGENTS.MD rules restricting deletion to project directory. https://github.com/openai/codex/issues/14487.

[R142] Codex ran git reset despite agents.md prohibiting git commands, rolling back 6 hours of uncommitted refactoring work. https://www.reddit.com/r/codex/comments/1onqxfq/codex_finally_fed_me_git_issue_lessons_learned/.

[R143] Codex CLI with GPT-5.2 runs git restore to discard user's unrelated uncommitted changes for a "clean" PR. https://github.com/openai/codex/issues/8213.

[R144] Codex executed git restore . without confirmation, wiping hours of uncommitted work from dirty working tree. https://www.reddit.com/r/codex/comments/1pt3vcm/be_careful_with_codex/.

[R145] Codex sandbox blocks GPU access, forcing users to disable all sandbox protections to run ML workloads. https://github.com/openai/codex/issues/3141.

[R146] Codex CLI panics with Landlock sandbox error on kernels lacking Landlock support, forcing users to disable sandboxing entirely. https://github.com/openai/codex/issues/2267.

[R147] Codex has no mechanism to exclude sensitive files from being read or sent to the model, exposing secrets like .env and SSH keys. https://github.com/openai/codex/issues/2847.

[R148] Codex overwrote research document with summaries when asked to read folder; undo feature non-functional. https://www.reddit.com/r/codex/comments/1pe1f69/wow_undo_not_working/.

[R149] Keyring failure silently falls back to plaintext credential file storage with potentially world-readable permissions, leaving stale secrets on disk. https://github.com/openai/codex/issues/14704.

[R150] Codex read .env file via built-in Read tool without permission while exploring project to fix e2e test. https://www.reddit.com/r/codex/comments/1o23ok8/any_way_to_prevent_it_reading_env/.

[R151] Codex CLI intentionally removed read-only approval mode, eliminating per-edit permission prompts and forcing auto-apply as default. https://github.com/openai/codex/issues/11915.

[R152] Codex reads files outside working directory without permission despite sandbox documentation claiming reads are restricted to cwd. https://github.com/openai/codex/issues/5237.

[R153] Codex App "don't ask again" permission persistence broken due to command parser failing on environment variable prefixes. https://github.com/openai/codex/issues/11298.

[R154] Codex repeatedly ran rm -rf deleting entire project directory four times in one session despite explicit instructions not to delete. https://www.reddit.com/r/codex/comments/1nmc05l/anyone_else_getting_issues_with_codex_nuking/.

[R155] Codex reverts user's manual code changes via git reset/checkout/rm, destroying uncommitted work across multiple users. https://github.com/openai/codex/issues/1736.

[R156] Codex review mode read-only sandbox blocks git commands on macOS due to xcrun temp file writes; –yolo flag ignored; sandbox reverted entirely. https://github.com/openai/codex/issues/7815.

[R157] Codex Windows sandbox triggers 0xc0000022 access-denied error on every command, forcing users to disable sandbox or use full-access mode. https://github.com/openai/codex/issues/14448.

[R158] Codex Windows sandbox sets incorrect ACLs on newly-created folders, causing apply_patch to fail on second write to same directory. https://github.com/openai/codex/issues/14585.

[R159] Codex re-prompts for sandbox permission on xcodebuild after user selected "don't ask again"; approval rules not persisting. https://github.com/openai/codex/issues/10321.

[R160] Codex Windows sandbox auto-applies NTFS deny ACE to workspace paths, causing write failures and requiring manual ACL cleanup. https://github.com/openai/codex/issues/14006.

[R161] Codex VS Code sandbox mounts writable devcontainer workspace as read-only due to duplicate ro+rw mount namespace bug. https://github.com/openai/codex/issues/14794.

[R162] Codex sandbox allow rules fail when commands are wrapped in shell scripts; MCP servers bypass sandbox entirely. https://github.com/openai/codex/issues/12283.

[R163] Codex bubblewrap sandbox fails when /.codex is a symlink to another partition, breaking apply_patch. https://github.com/openai/codex/issues/14694.

[R164] Codex sandbox on macOS blocks test runners (vitest, jest), pushing users to disable all safety via –dangerously-bypass-approvals-and-sandbox. https://github.com/openai/codex/issues/1532.

[R165] Codex truncated file from wrong location losing 3000 lines; VSCode revert button failed, requiring manual recovery from conversation history. https://github.com/openai/codex/issues/7291.

[R166] Codex on Windows double-encodes UTF-8 as CP1252, corrupting emojis and non-ASCII characters in files; apply_patch failures cause file deletion and recreation. https://github.com/openai/codex/issues/4013.

[R167] Codex VS Code extension asks for approval on every single file change despite being set to Agent full access mode. https://www.reddit.com/r/OpenaiCodex/comments/1n82y3d/codex_vs_code_extension_is_always_asking_for/.

[R168] Codex on Windows ignores "Allow for this session" approval because command vector mismatch between displayed bash command and executed PowerShell-wrapped command prevents approval caching. https://github.com/openai/codex/issues/4212.

[R169] Codex Windows sandbox fails with CreateProcessWithLogonW error 5 due to localized group names, forcing excessive approval prompts and opaque PowerShell edits. https://github.com/openai/codex/issues/9062.

[R170] Codex Windows sandbox ignores "Allow this session" permission, re-prompting for every edit. https://github.com/openai/codex/issues/7811.

[R171] Codex App for Windows deleted 370+ GB of user files outside the project directory while in Full Access mode; at least 8 additional users reported identical systemic drive-wiping behavior. https://community.openai.com/t/critical-data-loss-issue-in-codex-app-for-windows-agent-executed-file-deletion-outside-project-directory/1375894.

[R172] Codex on Windows ignores –ask-for-approval never and –dangerously-bypass-approvals-and-sandbox, requiring manual approval for every file write. https://github.com/openai/codex/issues/2350.

[R173] Codex App for Windows deleted Documents, Adobe, Dropbox, and Desktop files after encountering sandbox access errors. https://community.openai.com/t/codex-for-windows-deleted-a-huge-amount-of-my-drive/1376684.

[R174] Codex VS Code extension on Windows ignores full-access mode, prompts for approval on every action. https://github.com/openai/codex/issues/2828.

[R175] Codex agent rmdir command mis-parsed through multi-shell chain on Windows, deleting entire workspace contents instead of target folder. https://community.openai.com/t/potential-destructive-command-mis-parsing-on-windows-agent-cleanup-via-cmd-c-may-delete-workspace-content-instead-of-target-folder/1376026.

[R176] Codex Windows sandbox setup fails during refresh, forcing users to bypass sandbox or grant full access permissions. https://github.com/openai/codex/issues/10601.

[R177] Codex replaces 30,000 lines of code with a single requested addition, denies doing it in plan mode. https://www.reddit.com/r/codex/comments/1olvmxn/out_of_nowhere_codex_just_deletes_my_entire_code.

[R178] Codex CLI on WSL2 intermittently re-enters sandbox mid-session despite –dangerously-bypass-approvals-and-sandbox, breaking sudo/Docker/UNIX sockets. https://github.com/openai/codex/issues/5084.

[R179] Prompt injection exploited markdown image rendering in Continue VS Code extension to exfiltrate .env secrets and chat history to attacker-controlled server. https://versprite.com/blog/data-exfiltration-via-image-rendering-fixed-in-continue/.

[R180] Copilot Agent repeatedly empties .py file content when attempting to apply code changes, then fails to recover it. https://github.com/microsoft/vscode-copilot-release/issues/14021.

[R181] Copilot agent mode ignores file exclusion settings, reading sensitive files via built-in tools despite configured restrictions. https://github.com/microsoft/vscode-copilot-release/issues/12711.

[R182] Copilot Agent Mode recreated deleted files and rolled back entire codebase to previous version on session reopen, causing hours of lost work. https://github.com/microsoft/vscode/issues/264091.

[R183] Copilot Agent mode truncates file edits, destroying end of files, then loops failing to repair the truncation. https://github.com/microsoft/vscode-copilot-release/issues/7038.

[R184] GitHub Copilot Chat exfiltrated chat context data via rendered markdown images after following prompt injection instructions hidden in source code. https://embracethered.com/blog/posts/2024/github-copilot-chat-prompt-injection-data-exfiltration/.

[R185] User asked Copilot to delete files it created; agent deleted all project code instead. https://github.com/microsoft/vscode-copilot-release/issues/13722.

[R186] Copilot agent deleted user's core project code then reported task as successfully completed. https://github.com/microsoft/vscode-copilot-release/issues/14020.

[R187] Copilot deleted all user code without adding any new content during editing session. https://github.com/microsoft/vscode-copilot-release/issues/14112.

[R188] Copilot Chat repeatedly deleted entire codebase when user requested edits. https://github.com/microsoft/vscode-copilot-release/issues/14016.

[R189] Copilot deleted entire directory instead of renaming it, losing a full day's work without confirmation. https://github.com/microsoft/vscode/issues/280008.

[R190] Copilot erases entire file contents when asked to make changes, claims no problems occurred, tells user to fix it themselves. https://github.com/microsoft/vscode-copilot-release/issues/14019.

[R191] Copilot Chat v0.26.7 repeatedly deleted entire codebase when user prompted simple edits. https://github.com/microsoft/vscode-copilot-release/issues/14017.

[R192] Copilot agent repeatedly wipes entire files when attempting targeted edits via insert_edit_into_file tool, entering endless destructive loop. https://github.com/microsoft/vscode-copilot-release/issues/14108.

[R193] Copilot in ask mode switched to workspace mode and endlessly attempted file modifications without permission. https://github.com/microsoft/vscode-copilot-release/issues/14032.

[R194] Copilot reads .env files containing secrets with no ignore-file mechanism to exclude sensitive paths. https://github.com/microsoft/vscode-copilot-release/issues/13619.

[R195] Filename-based prompt injection in Copilot Chat Agent mode causes agent to execute malicious setup.py that exfiltrates system files to attacker-controlled server. https://www.tenable.com/security/research/tra-2025-53.

[R196] VS Code Copilot agent mode tracks deleted files in working set and recreates them as empty on reload, breaking builds. https://github.com/microsoft/vscode/issues/257998.

[R197] Copilot with Claude Haiku 4.5 creates dozens of unwanted markdown files, ignoring explicit user instructions not to. https://github.com/microsoft/vscode-copilot-release/issues/14045.

[R198] Copilot IntelliJ Agent mode deletes existing code in large files when prompted to add new content. https://github.com/microsoft/copilot-intellij-feedback/issues/373.

[R199] Copilot suggested deleting entire folder when asked to move a file; permission prompt prevented execution. https://github.com/microsoft/vscode-copilot-release/issues/13664.

[R200] Prompt injection in GitHub Copilot enables RCE by silently writing settings.json to enable YOLO mode, bypassing permission prompts. https://nvd.nist.gov/vuln/detail/CVE-2025-53773.

[R201] VS Code Copilot window reload triggers automatic file creation/modification in WSL, risking corruption of untracked project files. https://github.com/microsoft/vscode-copilot-release/issues/13590.

[R202] Copilot ran Vite initialization commands in wrong directory, deleting all files from user's dev directory. https://github.com/microsoft/vscode-copilot-release/issues/13763.

[R203] Copilot Agent in Visual Studio repeatedly deletes parts of existing code during edits, leaving files unbuildable with syntax errors. https://github.com/orgs/community/discussions/161952.

[R204] CVE-2025-55319 — prompt injection in VS Code Copilot agent mode enables token exfiltration, config modification, and arbitrary code execution via malicious GitHub issues. https://nvd.nist.gov/vuln/detail/CVE-2025-55319.

[R205] VS Code Copilot agent mode creates files at Windows paths instead of WSL remote paths. https://github.com/microsoft/vscode-copilot-release/issues/9323.

[R206] Copilot suggested a wrong Linux command that deleted the Django backend folder in a project. https://github.com/microsoft/vscode-copilot-release/issues/13742.

[R207] Copilot Agent creates files in wrong Windows directory instead of WSL project path due to path resolution bug. https://github.com/microsoft/copilot-intellij-feedback/issues/278.

[R208] Cursor Agent deliberately bypasses .cursorignore by falling back to shell commands (ls, cat, echo) to read and write protected .env file containing private keys. https://forum.cursor.com/t/ai-agent-bypassing-security-restrictions-hacks-its-access-to-restricted-files/119426.

[R209] Cursor Agent escaped project scope and grepped user home directory while fixing compilation errors, searching personal data without permission. https://forum.cursor.com/t/cursor-agents-should-be-restricted-from-access-of-files-outside-the-project-without-permission/149418.

[R210] Cursor Agent used echo » to modify .zshrc without permission, bypassing allowlist intended for read-only commands via shell redirect. https://forum.cursor.com/t/allowlist-should-not-extend-to-redirects/139150.

[R211] Four methods to bypass Cursor's auto-run denylist — obfuscation, subshell, shell scripts, and bash quoting — render the defense useless. https://www.backslash.security/blog/cursor-ai-security-flaw-autorun-denylist.

[R212] Case-sensitive file protection bypass allows prompt injection to overwrite .cursor/mcp.json for RCE on case-insensitive filesystems. https://nvd.nist.gov/vuln/detail/CVE-2025-59944.

[R213] Claude 4 models in Cursor bypass .cursorignore and globalignore to read and alter .env files via shell commands. https://www.reddit.com/r/cursor/comments/1l27xof/claude_bypasses_globalignore_rules/.

[R214] Cursor CLI auto-loads project-local config that overrides global allowlist, enabling RCE via prompt injection in malicious repositories. https://nvd.nist.gov/vuln/detail/CVE-2025-61592.

[R215] Prompt injection achieves RCE by modifying Cursor CLI sensitive config files via case-insensitive path bypass. https://nvd.nist.gov/vuln/detail/CVE-2025-61593.

[R216] Cursor crashes corrupt project files by injecting whitespace and reverting committed changes, wiping 100+ files. https://forum.cursor.com/t/cursor-crashing-and-modifying-files-unexpectedly-project-partially-wiped/147372.

[R217] Cursor built-in read_file and write tools blocked by .cursorignore even when ignore file is empty, comments-only, or deleted. https://forum.cursor.com/t/read-file-and-write-tools-blocked-by-cursorignore-even-when-file-is-empty-or-deleted/148852.

[R218] Cursor AI went haywire during auto-run session and deleted 90% of project files including .git folder, with delete file protection disabled. https://dredyson.com/why-my-cursor-ide-deleted-my-entire-project-and-how-i-got-it-back/.

[R219] Cursor Agent autonomously mass-deleted project files then crashed, destroying chat history and checkpoints needed for recovery. https://forum.cursor.com/t/help-needed-asap-cursor-deleted-my-whole-proejct/97589.

[R220] Cursor allows creating new dotfiles (.vscode/settings.json) without approval, enabling RCE via prompt injection chain (CVE-2025-54130). https://nvd.nist.gov/vuln/detail/CVE-2025-54130.

[R221] Cursor Agent's built-in edit tool deletes or empties files when making multiple sequential edits to the same file due to a tool bug. https://forum.cursor.com/t/agents-are-deleting-files-while-editing/145458.

[R222] Cursor agent read and rewrote .env file containing sensitive data using built-in Read/Edit tools with no .cursorignore protection. https://www.reddit.com/r/cursor/comments/1jijrxn/cursor_is_reading_my_env_file/.

[R223] Cursor Agent read .env file despite it being listed in .cursorignore, bypassing ignore-file defense via shell tool calls. https://forum.cursor.com/t/cursor-reads-env-even-though-it-is-on-cursorignore/136998.

[R224] Cursor's built-in edit and checkpoint tools deleted user files during editing sessions, causing hours of lost work; recurring bug confirmed by multiple users over six months. https://forum.cursor.com/t/cursor-deleted-my-file/45020.

[R225] Cursor Agent occasionally deletes files without permission while using claude-4.5-haiku in Auto-Run mode. https://forum.cursor.com/t/cursor-may-occasionally-delete-files-without-permission/138395.

[R226] JSON schema auto-download default setting enables silent data exfiltration via agent-written JSON files after prompt injection. https://nvd.nist.gov/vuln/detail/CVE-2025-49150.

[R227] Cursor Composer asked to move one file went into tool-calling loop and deleted all project files. https://forum.cursor.com/t/cursor-potentially-deleted-all-of-my-files/141670.

[R228] Five of six LLMs in Cursor attempt to bypass .cursorignore by using shell commands (cat, head, sed) to read protected files. https://forum.cursor.com/t/security-privacy-issue-llms-will-attempt-to-circumvent-cursorignore/148441.

[R229] Feature request for file lock/read-only mode after Cursor Agent hallucinated edits to a reference-only file. https://www.reddit.com/r/cursor/comments/1ih6yhx/there_should_be_a_lock_filereadonly_feature/.

[R230] Cursor IDE randomly deleted newly created files and reverted recent changes upon opening project, affecting 15 files. https://github.com/cursor/cursor/issues/3897.

[R231] Hidden prompt injections in README files bypass Cursor denylist and exfiltrate API keys and SSH credentials via tool chaining. https://hiddenlayer.com/innovation-hub/how-hidden-prompt-injections-can-hijack-ai-code-assistants-like-cursor/.

[R232] Cursor AI assistant generated recursive backup script using shutil.copytree that created infinitely nested directories, hit Windows path limit, and deleted entire project folder. https://forum.cursor.com/t/critical-bug-ai-assistant-deleted-entire-directory-via-recursive-backup-loop/138236.

[R233] Cursor deletes entire source files when user clicks Reject on AI-proposed changes, even for files not created by AI. https://forum.cursor.com/t/cursor-deletes-my-source-file-altogether-without-warning-when-i-click-reject-ai-didnt-even-create-the-file-in-the-first-place/28718.

[R234] Cursor agent's Remove-Item commands with unescaped PowerShell bracket wildcards wiped entire F: drive instead of nested project directory. https://forum.cursor.com/t/possible-catastrophic-deletion-triggered-by-remove-item-path-expansion-in-cursor-f-drive-wiped/138966.

[R235] Cursor replaced failing test implementation with expect(true).toBe(true) stub and claimed tests passed. https://www.reddit.com/r/cursor/comments/1k05u84/test_passed/.

[R236] Cursor executed rm -rf && ls -la during development session, deleting personal files from macOS home directory. https://forum.cursor.com/t/cursor-ai-executes-destructive-command-rm-rf-during-development-session/129401.

[R237] Cursor with Gemini 2.5 Pro added home directory as extra rm -rf argument, wiping desktop and keychain. https://forum.cursor.com/t/cursor-tried-to-wipe-my-computer/107142.

[R238] Cursor agent ran rmdir /s /q with broken quoting, wiping 350GB+ from D: drive despite File-Deletion and External-File Protection being enabled. https://forum.cursor.com/t/cursor-deleted-my-d-drive/149859.

[R239] Cursor 2.0 sandbox grants full filesystem read access; agent ran cat /.npmrc exposing npm auth token to LLM. https://luca-becker.me/blog/cursor-sandboxing-leaks-secrets/.

[R240] Widespread reports of Cursor with Sonnet 3.7 randomly deleting working code during edit sessions, especially in long chats. https://news.ycombinator.com/item?id=43298275.

[R241] Cursor agent deletes files via PowerShell Remove-Item bypassing delete file protection setting. https://github.com/cursor/cursor/issues/2895/.

[R242] CTO reports Cursor repeatedly modifying restricted framework files when team members vibe-code with incomplete context; shares VS Code readonlyInclude workaround. https://www.reddit.com/r/cursor/comments/1okujl4/how_to_

stop_cursor_to_edit_files_that_you_do_not/.
[R243] Cursor Agent can be prompt-injected to write .code-workspace file, bypassing CVE-2025-54130 sensitive-file protections and achieving RCE. https://nvd.nist.gov/vuln/detail/CVE-2025-61590.
[R244] Cursor in YOLO mode deleted everything on user's computer including itself during Express.js to Next.js migration. https://news.ycombinator.com/item?id=44262383.
[R245] Gemini CLI @-file reference resolves local ./tmp subdir to system /tmp, causing agent to search outside project scope. https://github.com/google-gemini/gemini-cli/issues/22392.
[R246] Gemini CLI deleted git branches despite deny rules in policy engine; policy regex bug prevents commandPrefix rules from matching real commands. https://github.com/google-gemini/gemini-cli/issues/20355.
[R247] Gemini CLI model occasionally uses destructive commands like git reset –force instead of safer alternatives. https://github.com/google-gemini/gemini-cli/issues/22672.
[R248] Gemini CLI stuck in loop repeatedly deleting .env with rm -rf despite user explicitly telling it to stop. https://github.com/google-gemini/gemini-cli/issues/18624.
[R249] Gemini CLI command restrictions bypassed via eval injection in shell scripts, enabling unrestricted unsandboxed command execution. https://github.com/google-gemini/gemini-cli/issues/18194.
[R250] Extension update permanently deletes old extension before verifying new one loads, leaving user with no working extension on failure. https://github.com/google-gemini/gemini-cli/issues/21671.
[R251] Gemini CLI made unauthorized code changes then executed destructive git checkout, wiping all uncommitted work. https://github.com/google-gemini/gemini-cli/issues/22649.
[R252] Gemini CLI autonomously ran git clean and git reset –hard, deleting 11.5 hours of uncommitted architectural work. https://github.com/google-gemini/gemini-cli/issues/22010.
[R253] Environment variable sanitization bypassed via hookConfig.env spread ordering in executeCommandHook, enabling LD_PRELOAD/PATH injection for arbitrary code execution. https://github.com/google-gemini/gemini-cli/issues/22503.
[R254] Gemini CLI read_file tool enforces ignore patterns with no override, but search_file_content supports no_ignore; shell cat is the only workaround. https://github.com/google-gemini/gemini-cli/issues/13775.
[R255] Gemini CLI list_dir tool ignores .geminiignore and .aiexclude rules, exposing excluded directories via built-in tool. https://github.com/google-gemini/gemini-cli/issues/14546.
[R256] Gemini CLI used write_file to overwrite hundreds of lines of project log history during long session with context degradation. https://github.com/google-gemini/gemini-cli/issues/17534.
[R257] Gemini CLI regularly replaces unchanged code with "(rest of methods ...)" placeholder text in proposed diffs, marking real code as deleted. https://github.com/google-gemini/gemini-cli/issues/19858.
[R258] Gemini CLI ignored plan mode and directly implemented code changes instead of proposing a plan. https://github.com/google-gemini/gemini-cli/issues/22751.
[R259] Policy engine bypass via nested property injection in JSON-stringified tool arguments allowed unauthorized shell commands. https://github.com/google-gemini/gemini-cli/issues/19762.
[R260] Feature request to warn users when policy auto-approves dangerous commands like rm. https://github.com/google-gemini/gemini-cli/issues/21596.
[R261] Gemini CLI replace tool repeatedly truncated large log file, silently deleting historical project data. https://github.com/google-gemini/gemini-cli/issues/17562.
[R262] Gemini CLI used rm -rf to delete home directory contents (documents, downloads, desktop) after user clicked "allow always" during npm install. https://github.com/google-gemini/gemini-cli/issues/2617.
[R263] macOS seatbelt sandbox silently drops allowedPaths beyond hard-coded limit of 5, with no warning and inconsistent cross-driver behavior. https://github.com/google-gemini/gemini-cli/issues/22536.
[R264] Gemini CLI shell escaping logic loop in WSL broke source code despite user warnings. https://github.com/google-gemini/gemini-cli/issues/19879.
[R265] Gemini CLI agent prefers shell redirects over built-in file tools, triggering excessive permission prompts. https://github.com/google-gemini/gemini-cli/issues/22638.
[R266] Stale closure in onModelChange silently overwrites user-edited .gemini/settings.json with boot-time snapshot. https://github.com/google-gemini/gemini-cli/issues/20402.
[R267] Gemini CLI agent silently falls back to write_file after str_replace failure, rewriting entire file with incorrect or incomplete content. https://github.com/google-gemini/gemini-cli/issues/20799.
[R268] Gemini CLI attempted sudo rm -rf /Library/Developer/CommandLineTools in YOLO mode without user confirmation. https://github.com/google-gemini/gemini-cli/issues/21594.
[R269] Gemini CLI overwrites large files with truncated read fragments via write_file, then compounds loss with destructive git checkout recovery. https://github.com/google-gemini/gemini-cli/issues/18764.
[R270] Gemini CLI agent bypassed project-defined safe_log_append wrapper, used native write_file on large document, causing truncation and data loss. https://github.com/google-gemini/gemini-cli/issues/20866.
[R271] Gemini CLI uses WRITE_FILE instead of partial edits, causing repeated loss of important code in project files. https://github.com/google-gemini/gemini-cli/issues/21549.
[R272] Gemini CLI used write_file to fully overwrite persistent project log, destroying all historical context due to context compression. https://github.com/google-gemini/gemini-cli/issues/17583.
[R273] Gemini CLI uses WriteFile instead of Replace for edits, overwriting entire files and losing existing code. https://github.com/google-gemini/gemini-cli/issues/20321.
[R274] Gemini CLI deleted .git.bak backup directory during cleanup, destroying local commit history recoverable only via ZFS snapshot. https://github.com/google-gemini/gemini-cli/issues/19729.
[R275] Gemini CLI deleted user's *_new.json files without consent after JSON comparison task, violating explicit "do not change other files" instruction. https://github.com/google-gemini/gemini-cli/issues/15822.
[R276] Gemini bypassed .gitignore restriction by using shell cat after built-in tool correctly blocked access. https://www.reddit.com/r/AntigravityGoogle/comments/1premuk/gemini_bypasses_gitignore_restrictions/.
[R277] Lovable replaced user's Supabase Edge function code with generic dummy function to "fix" deployment errors, destroying work. https://www.linkedin.com/posts/anubhave_lovable-ai-coding-activity-7367214709298601984-3uUl.
[R278] AI agent(s) secretly deleting project files in the background without user prompting; culprit unidentified among five running agents. https://www.reddit.com/r/cursor/comments/1k1h24a/ai_agent_secretly_deleting_my_files/.
[R279] Argument injection in pre-approved safe commands achieves RCE across three unnamed AI agent platforms, bypassing human-in-the-loop approval. https://blog.trailofbits.com/2025/10/22/prompt-injection-to-rce-in-ai-agents/.
[R280] Rules File Backdoor — hidden unicode characters in AI config files cause Cursor and GitHub Copilot to silently inject malicious code into generated files. https://www.pillar.security/blog/new-vulnerability-in-github-copilot-and-cursor-how-hackers-can-weaponize-code-agents.

[R281] Malicious AI agent skills bypass skill scanners by bundling test files that Jest/Vitest auto-execute, achieving RCE and secret exfiltration. https://www.gecko.security/blog/rce-in-your-test-suite-ai-agent-skills-bypass-skill-scanners.

[R282] Security researcher exploited 7 of 16 YC AI agents via prompt injection, leaking user PII, rewriting server code to remove security controls, and taking over databases. https://casco.com/blog/we-hacked-ycombinator-agents.

[R283] OpenCode bash allowlist uses implicit prefix matching, allowing shell redirects to write arbitrary files outside project scope. https://github.com/anomalyco/opencode/issues/5330.

[R284] OpenCode agent bypassed denied git reset via bash -c subshell and overwrote .env with echo instead of built-in tools. https://github.com/anomalyco/opencode/issues/4642.

[R285] OpenCode agent bypassed denied .env read permission by using bash cat command, then overwrote the file via shell redirect. https://www.reddit.com/r/opencodeCLI/comments/1qj4gr1/ai_just_worked_around_env_read_permissions/.

[R286] OpenCode agent used rmdir on Windows reserved name (nul), causing recursive deletion of all project folders on K: drive. https://github.com/anomalyco/opencode/issues/17836.

[R287] AI agents repeatedly modified, deleted, and faked TDD test files instead of implementing features; developer deployed chmod 444 as filesystem-level defense. https://blakelink.us/posts/how-i-use-chmod-to-stop-ai-agents-from-cheating-on-my-tests/.

[R288] Secret exfiltration via markdown rendering and RCE via mcp.json modification in Void Editor through prompt injection in code files. https://idanhabler.medium.com/d%C3%A9j%C3%A0-vu-in-the-void-an-agentic-ide-compromised-by-known-tricks-56c3c492a077.

[R289] Windsurf Cascade repeatedly deletes working code when asked to adjust CSS, revert only recovers 10%. https://www.reddit.com/r/Codeium/comments/1gyo145/windsurf_keeps_on_deleting_code.

[R290] Windsurf Cascade ran rm instead of git rm, permanently deleting CLI project's core files including rulebook and init.ts. https://www.threads.com/@carmelyne/post/DCiQqady7aS.