Safe Agentic Workflows for Isabelle

Maximilian Schäffeler[1], Lukas Stevens[2], Kevin Kappelmann[3], Mohammad Abdulaziz[1], Andrei Popescu[3], and Dmitriy Traytel[2]

[1] Department of Informatics, King's College London, United Kingdom
[2] Department of Computer Science, University of Copenhagen, Denmark
[3] Department of Computer Science, University of Sheffield, United Kingdom

Abstract. Large language models (LLMs) and proof assistants are in the early days of a fruitful marriage. LLM-based agents draft and mechanize, while proof assistants provide infrastructure to construct proofs and check the results. We consider two aspects of safety when integrating agents with the Isabelle proof assistant. First, we survey technological methods for isolating agents and restricting what they can do through Isabelle. Second, we discuss different agentic workflows and the human intervention needed to assure the resulting artifacts' logical consistency.

1 Introduction

We consider the following general architecture: an LLM is paired with a coding agent that interacts with a proof assistant via the Model Context Protocol (MCP). In our own setup [9], the coding agent is OpenCode, the MCP server is Isabelle/Q [1], and the proof assistant is Isabelle2025-2. Bryant et al. [3] describe an alternative setup that avoids MCP in favor of the standard `isabelle build`, but uses specific Isabelle/ML tools to observe the context, find theorems, or use proof automation at a given point in the Isabelle theory.

Either architecture raises two distinct safety concerns. The first is *operational*: there are by now well-documented reports of agents causing serious harm to the systems they run on, including bugs [16], security vulnerabilities [4,7], and outright data loss—a widely circulated example being a Claude instance that wiped a user's database while operated via Cursor [13]. The second concern is *logical*: it is surprisingly easy to generate large amounts of formal material that is accepted by Isabelle yet makes no mathematical sense—definitions or locales with contradictory assumptions, for instance. This is not a new problem; definitions and lemma statements have always needed human inspection. What changes with agents is the scale at which such material can be produced, and consequently the scale at which it needs to be reviewed.

2 Operational Safety

Giving an LLM agent access to a proof assistant inevitably means giving it some foothold on a computer. This foothold has real consequences for operational safety. Isolating the agent from the rest of the system is our main mitigation. We experimented with five approaches, differing along three axes: usability, portability, and the degree of isolation they actually provide.

YOLO. The simplest approach is to install the agent and the proof assistant side by side on the host machine and take no further measures. This maximizes usability but provides no isolation whatsoever. This is where most users start.

Note that in this setting, the user is exposed to significant risks even if the agentic coding harness itself does not provide shell access, as the agent can still execute arbitrary code on the host via the MCP interface: The agent can, for instance, insert a malicious ML code block into a theory, which is then executed by Isabelle.

Bubblewrap [5]. The `bubblewrap` tool creates lightweight sandboxes for Linux applications based on namespaces and bind mounts. Careful configuration of the sandbox is required to ensure that the agent has access to the necessary Isabelle files and tools while being restricted from accessing sensitive parts of the host system. To display Isabelle/jEdit, the sandbox shares the host's X server environment, which opens up a potential attack surface. This surface could be reduced with a nested X server such as `Xpra` [15] or by restricting the agent's X client via the X11 SECURITY extension [14], so that it cannot spy on or inject events into other windows on the host.

Bubblewrap works well on Linux but is not available on other operating systems. Alternatives like `sandbox-exec` for macOS exist, but there is no established cross-platform solution.

Docker [11]/Podman [6] containers. Containers for both Isabelle and OpenCode exist and provide a reasonable degree of portability. A complication is that Isabelle/Q requires a running instance of Isabelle/jEdit to function, so the container needs a graphical environment. Our solution is to expose Isabelle/jEdit over VNC (a protocol for sharing graphical desktops over a network), allowing the user to watch and interact with the formalization from the host. We initially experimented with X-Forwarding, but discarded it due to graphical issues on macOS and the large attack surface of an X server running on the host.

A somewhat unexpected benefit of the container approach is that placing OpenCode and Isabelle in *separate* containers limits the ways the agent can interact with the proof assistant. Without this restriction, agents tend to get confused about how to access theory files and start editing them directly rather than through MCP, even when explicitly instructed not to. Concretely in our setup, the Isabelle container holds the Isabelle release, Isabelle/Q, the Archive of Formal Proofs (AFP), and a writable project directory. It exposes the Isabelle and AFP sources as well as the project directory to the OpenCode container as *read-only* mounts. As a result, the MCP is genuinely the only channel through which the agent can modify any file in the project. Docker Compose conveniently orchestrates the two-container setup. One practical challenge we encountered is correctly handling file ownership and user mappings when mounting volumes, which varies across operating systems.

Virtual machine (VM). A full virtual desktop inside a VM, e.g., executed via the QEMU emulator [2], provides the strongest isolation short of hardware separation. We use a declarative Nix-based build [8] to reproduce the environment.

Operating system (OS) and hardware isolation. An OS-native approach is to run the agent and the proof assistant as a separate OS user. The usability of this approach depends on how easily the host system allows users to switch accounts. At the most conservative extreme, using a dedicated physical machine completely isolates the agent from the user's primary environment.

Even in the YOLO mode, our experience has been that with default settings, coding agents typically *ask* before accessing files outside the expected working directory—a mild form of self-imposed caution. However, there is nothing structural that forces them to ask, as illustrated by an incident in which a prompt that mentioned the Isabelle mailing list led the agent to attempt to read the user's emails. Isolation approaches that restrict what the agent *can* do, rather than relying on what it *chooses* to do, are preferable. We provide ready-to-use configurations for the Bubblewrap, Docker, and VM approaches as part of our artifact [12].

Our analysis so far focused on the agent's ability to harm the host system. An orthogonal risk is *exfiltration*: a misaligned agent may transmit sensitive material from the user's Isabelle project to third parties. Filesystem isolation does not address this, since the agent needs network access to reach the model. Mitigations require explicit *egress control*, e.g. restricting outbound traffic to the model endpoint only. Exposure to the LLM provider itself is avoided only by running a local model.

3 Logical Safety

Isabelle's kernel guarantees that any accepted proof, whether human-created or not, is logically valid with respect to the definitions in scope. What the kernel cannot check, however, is whether the definitions and theorem statements faithfully capture what the user intended to represent. This is the crux of the logical safety question, also known as *semantic consistency* [10]. The degree to which semantic consistency is achieved depends heavily on how the LLM is integrated into the proving workflow. The key distinction for logical safety is whether the human fixes the definitions and theorem statements in advance. If so, the kernel's guarantee covers proof correctness, and the user only needs to verify that the agent did not silently modify definitions or statements. If not, the user bears full responsibility for checking that the agent-generated definitions and statements faithfully capture the intended mathematics—a task Isabelle cannot assist with. We next consider the workflows with which we had successful experiments. The first two workflows assume all definitions and statements are fixed; the remaining three do not. We nonetheless distinguish five workflows, as they differ in what the human specifies upfront and in their typical use cases.

Proof cleanup. The human writes all definitions, theorem statements, and proofs. The agent rewrites proofs with the goal of improving readability, maintainability, or conformance to AFP style. A typical use case is converting exploratory apply-style proofs to clean Isar proofs. When the cleanup is limited to rewriting proofs, the kernel ensures that the rewritten proofs are still correct and the human only needs to judge whether the new proof style is acceptable. Any potential changes to definitions or lemma statements must be inspected, of course.

Proof search. The human provides definitions and theorem statements. The agent is used as a variant of Sledgehammer to find proofs (which internally may use Sledgehammer and other proof automation Isabelle offers). The agent is not expected to introduce new definitions or lemmas. Its role is to fill in proofs for statements that the human already fixed. The user should verify during or after the session that no definitions or theorem statements were modified and the proofs use no unsound oracles or axioms.

Safeguarded autoformalization. The human provides a desired specification or top-level theorem statement. The agent is expected to draft the supporting mathematical infrastructure, including auxiliary definitions and theorems. Unlike proof search, introducing new definitions and lemmas is the point of this workflow. The human retains control over the high-level mathematical claim.

Full autoformalization. Given some source, e.g., an informal description, an existing implementation, or a pen-and-paper proof of varying detail, the agent drafts formal definitions, statements, and proofs without a human-provided specification. The kernel still certifies every proof, but now the definitions and top-level statements are agent-generated.

Human review of the definitions and the main statements is essential before trusting the development's mathematical content. The human reviewer may also want to request from the agent test cases, counterexamples, or locale instantiations to check that the definitions and statements are not vacuous or contradictory. Prior work has also explored the use of LLMs as critics for checking the semantic consistency of agent-generated definitions and statements [10], e.g., by re-informalizing the Isabelle developments and comparing them against the original informal source.

Cross-assistant translation. The agent translates a formal development between proof assistants with different logical foundations—for example, from Lean to Isabelle or vice versa. This workflow inherits all the logical risks of full autoformalization, and adds the further risk that differences in foundations or libraries may cause the translation to silently alter the mathematical meaning of definitions or theorems. Careful human review is required.

4 Discussion

The operational and logical safety dimensions both deserve attention from the formal methods community, though they call for different responses. Opera-

tional safety (Section 2) is ultimately a systems problem: the right containment technology can prevent an agent from doing harm on the host machine regardless of what the agent does. Several solutions are available today. The question is whether users are willing to accept the respective setup and usability cost. Logical safety (Section 3) currently requires human oversight, though improvements in LLM reasoning capabilities may reduce this burden over time. Isabelle's kernel is a reliable gatekeeper for proof correctness, but it cannot guard against a formally valid development that captures the wrong mathematics.

Both operational and logical safety risks can be further categorized into unintended errors and malicious behavior. For logical safety, the former is already a well-known issue in formal verification. The latter, malicious behavior, is a new concern that arises with the increased autonomy of LLM agents. A misaligned agent (whether due to a prompt injection attack, a bug in the agent scaffolding, or simply a confused model) may deliberately produce definitions and statements that are formally valid but semantically misleading, with the goal of manipulating the user.

We raise these questions here more than we answer them, hoping to prompt a discussion the community needs to have. What review processes are adequate for agent-generated definitions and lemma statements as well as agent-generated proofs, for example in the AFP? How do we avoid a situation where the scale of LLM-generated formal material overwhelms human reviewers?

Acknowledgements We acknowledge support from the AI for Math Fund, an initiative by Renaissance Philanthropy with funding support from XTX Markets.

References

1. Amazon Web Services: AutoCorrode software verification framework for Isabelle/HOL. https://github.com/awslabs/AutoCorrode (2025), accessed: 2026
2. Bellard, F.: Qemu, a fast and portable dynamic translator. In: USENIX ATC 2005. pp. 41–46. USENIX (2005), http://www.usenix.org/events/usenix05/tech/freenix/bellard.html
3. Bryant, D., Huerta y Munive, J.J., Kaliszyk, C., Urban, J.: Munkres' general topology autoformalized in Isabelle/HOL (2026), https://arxiv.org/abs/2604.07455
4. Chen, A., Wu, Y., Zhang, J., Xiao, J., Yang, S., Huang, J., Wang, K., Wang, W., Wang, S.: A survey on the safety and security threats of computer-using agents: JARVIS or Ultron? CoRR **abs/2505.10924** (2025). https://doi.org/10.48550/ARXIV.2505.10924
5. Containers Project: Bubblewrap: Unprivileged sandboxing tool. https://github.com/containers/bubblewrap (2016), accessed: 2026
6. Containers Project: Podman: A tool for managing OCI containers and pods. https://github.com/containers/podman (2017), accessed: 2026
7. Datta, S., Nahin, S.K., Chhabra, A., Mohapatra, P.: Agentic AI security: Threats, defenses, evaluation, and open challenges. CoRR **abs/2510.23883** (2025). https://doi.org/10.48550/ARXIV.2510.23883

8. Dolstra, E., de Jonge, M., Visser, E.: Nix: A safe and policy-free system for software deployment. In: Damon, L. (ed.) LISA 2004,. pp. 79–92. USENIX (2004), http://www.usenix.org/publications/library/proceedings/lisa04/tech/dolstra.html
9. Kappelmann, K., Schäffeler, M., Stevens, L., Abdulaziz, M., Popescu, A., Traytel, D.: Just type it in Isabelle! AI agents drafting, mechanizing, and generalizing from human hints (2026), https://arxiv.org/abs/2604.15713
10. Li, Z., Wu, Y., Li, Z., Wei, X., Zhang, X., Yang, F., Ma, X.: Autoformalize mathematical statements by symbolic equivalence and semantic consistency. In: Globersons, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J.M., Zhang, C. (eds.) NeurIPS 2024 (2024), http://papers.nips.cc/paper_files/paper/2024/hash/6034a661584af6c28fd97a6f23e56c0a-Abstract-Conference.html
11. Merkel, D.: Docker: Lightweight Linux containers for consistent development and deployment. Linux Journal **2014**(239) (2014)
12. Schäffeler, M., Stevens, L., Kappelmann, K., Abdulaziz, M., Popescu, A., Traytel, D.: Safe agentic workflows for Isabelle: Setups (2026). https://doi.org/10.5281/zenodo.20108905
13. Udinmwen, E.: 'It took 9 seconds': tech founder outlines how rogue Claude-powered AI tool wiped entire company database and backups (May 2026), https://web.archive.org/web/20260509/https://www.techradar.com/pro/it-took-9-seconds-tech-founder-outlines-how-rogue-claude-powered-ai-tool-wiped-entire-company-database-and-backups-but-says-theres-no-such-thing-as-bad-publicity, archived at the Internet Archive Wayback Machine
14. Wiggins, D.P.: Security extension specification, version 7.1. X Consortium Standard, https://www.x.org/releases/X11R7.7/doc/xextproto/security.html (1996), accessed: 2026
15. Xpra Project: Xpra: Multi-platform screen and application forwarding. https://xpra.org (2025), accessed: 2026
16. Zhang, R., Dai, W., Pham, H.V., Uddin, G., Yang, J., Wang, S.: Engineering pitfalls in AI coding tools: An empirical study of bugs in Claude Code, Codex, and Gemini CLI. CoRR **abs/2603.20847** (2026). https://doi.org/10.48550/ARXIV.2603.20847