Externalization Barriers: An OS Abstraction for Untrusted Agent Exploration

Jinhao Hu
MPI-SWS

Bardia Mohammadi
MPI-SWS

Ashvin Goel
University of Toronto

Laurent Bindschaedler
MPI-SWS

Abstract

AI agents now test alternative plans by running untrusted code that touches external services. Copy-on-write branches roll back files and processes, but not pull requests, continuous-integration jobs, cloud reservations, or notifications. The hard case is a branch that must see a real service result before the system knows whether that branch should commit. We argue for an OS-enforced *externalization barrier*: a confinement boundary routing every externally visible action from an exploratory process tree through a trusted mediator. The mediator holds the credentials, hands each branch a handle carrying its effect's real result, and makes durable only what policy admits. Losing effects are revoked, compensated, or recorded as audited residual state. Prior effect-staging systems give cooperating tool code a commit and abort lifecycle. An OS barrier enforces that lifecycle below untrusted code, including the model-driven planner that picks the winner. We sketch *Effect Domains*, OS-managed scopes for staging, observing, and committing external effects.

1 The Externalization Gap

Agentic systems such as coding assistants and computer-use agents run code, call tools, open pull requests (PRs), start continuous-integration (CI) jobs, and reserve cloud resources while trying candidate plans [9, 21]. Copy-on-write branch contexts let each branch explore in a private workspace and cheaply revert local files and processes [18]. However, a branch context is a local abstraction, and an external effect is not local state.

Consider two branches fixing one bug. Each opens a draft PR to trigger CI and reads the real verdict before deciding its next edit. Only one branch should survive: its PR is published, while the loser's draft is withdrawn, its CI run canceled, and any notification the provider already sent is left behind as an audit record. Local rollback cannot do this. It never owned the PR, the CI job, or the notification.

An externalization mechanism owes four duties to every external action, the columns of Table 1. *Attribution* ties an action to the branch that caused it, including when an adversarial descendant issues it. *Credential custody* keeps the token out of the branch, so an unauthorized call has nothing to sign with. A *handle* names an effect staged by the mediator or the provider and returns its real result, when one exists, without making the effect durable. *Commit* releases one winner once selection happens. Existing abstractions cover some columns and miss others. Branch contexts and process sandboxes attribute an action to a branch but stop at the host-to-service boundary, so no effect is ever staged or committed there. A network-disabled sandbox avoids external effects and, with them, the PR and its CI. Access gates hold credentials and refuse unauthorized calls [17, 22], but a refused call is not a staged one, so a losing branch has nothing to read back or abort. Cooperative effect runtimes such as Atomix and Cordon [4, 13] supply exactly the two

	Attrib.	Creds	Handle	Commit
Sandboxes, branch contexts [18, 19]	✓			
Effect transactions [4, 13]			✓	✓
Access gates [17, 22]	✓	✓		
Effect Domains	✓	✓	✓	✓

Table 1: The four duties of §1: adversarial-descendant attribution (Attrib.), mediator credential custody (Creds), handles returning real results (Handle), and single-winner commit below the planner.

columns the gates lack, a full `stage`/`commit`/`abort` lifecycle, but only for tool code that opts in: a branch running `git push`, `curl`, or a child process is outside the protocol, with nothing binding credentials or branch identity beneath it.

Nor do they compose by stacking: a compensation library bolted onto a sandbox and a proxy binds the duties only by convention, which adversarial code evades. We call the resulting task the *untrusted-externalization problem*: confine an adversarial exploratory tree while letting it observe the real external effects policy permits. The lifecycle itself is old, with reversibility tiers tracing to Sagas [7] and try-confirm-cancel [8], and classic OS speculation staging effects on a single trusted path [14, 15, 20]. What is new is running it *below* untrusted code, so code that never joined the protocol still cannot externalize without the mediator.

2 Effect Domains

Three pressures shape the abstraction. First, untrusted code is everywhere: agents run model-generated shell commands, third-party plugins, and install hooks, and ingest content that can inject the planner itself [6]. Runtime refinement can harden regenerated code [10], but it cannot bound what that code externalizes, so control must bind every descendant at creation time, not at a tool API a child can walk around. Second, branches need real results, whether CI passed and which identifier a reservation got, and withholding them defeats the exploration. Finally, effects differ in kind: a draft PR can be withdrawn, a reservation canceled, an email not unsent, and services fail, partly complete, or go stale between decision and commit. Because services share no transaction protocol, someone must supply a handler per effect type, revalidation for stale results, and an audit record for what cannot be undone. A handler also decides how staging touches the live service, since raw speculative traffic trips rate limits and lets one branch's provider-visible state pollute what a sibling reads back: it can isolate the effect in a shadow namespace, throttle it, or hold it until commit.

Effect Domains bind all four duties in one OS-managed abstraction. Each domain confines an untrusted exploratory process tree and pairs it with a trusted mediator: the sole egress path, which holds credentials and decides which effects commit. The premise is a trust inversion we state as the *externalization-authority principle*: model-driven components are prompt-injectable, so the authority to make an effect durable must sit below them, in the OS beneath

the planner. Exploration is optional: a single untrusted agent is the degenerate one-branch case.

Externalization barrier. The barrier confines a branch's whole process tree, so every externally visible action reaches the mediator by one *logical* egress path below the planner. We call this contract *complete mediation*: tool-call APIs sit too high, and raw untrusted code walks around them. Meeting it by construction requires default-deny confinement at the resource level, so a gap in the egress path costs the branch functionality instead of handing it an unmediated route. Holding that coverage as interfaces evolve is open (§3).

Externalization handle. The mediator represents each staged effect as an *externalization handle*: a typed, identity-bound record naming an effect already staged with the provider and not yet durable. Its reversibility tiers follow cooperative effect runtimes [4, 13]: *buffered* (held until commit, e.g., an outbound email), *provisional* (provider-supported drafts, e.g., a draft PR or CI run), and *compensatable* (undone by a follow-up action, e.g., releasing a cloud reservation). The handle carries the effect's real service result, but the mediator, not the branch, decides its fate: revoke or compensate on abort, release on a winning commit. An irreversible effect that must return its real result admits no safe handle: exploration works only for stageable and compensatable effects, and a truly irreversible action stays a synchronization point. Unlike a capability, which grants authority to *cause* an effect, a handle names one already caused but withheld from durable release.

Interface. The barrier exposes one lifecycle at any placement. `open(policy)` returns a confined domain whose credentials the mediator holds. `stage(d,e)` stages effect `e` and returns a handle carrying its real result. `select` picks the winning branch, `commit` releases its admissible handles, and `abort` revokes or compensates the rest, recording what cannot be undone as residual.

Two-level trust and the envelope. The enforcement substrate and the mediator are trusted. Everything else, the planner included, is untrusted and may collude to forge branch identity or externalize an unauthorized effect. The split is between *selection* and *commit*: the planner chooses which branch should win, and the mediator releases that choice only if policy admits it, no other outcome conflicts, and revalidation still holds. Policy predates execution. Before `open`, a trusted principal approves the effect-type handlers and the domain's policy, which enumerates the destinations, credentials, and effect types a domain may commit. The mediator checks both at staging and again before release. No running code can create or edit it. What this buys is one *integrity* property, the *envelope*: every effect a domain makes durable lies inside that policy, and every effect of a losing branch is buffered, revoked, compensated, or recorded as audited residual [2], whatever the branches and the planner do. This is net-effect reversal, not erasure: the barrier rolls back what becomes durable, never external history, and failed cross-service compensation leaves audited residual [3]. It buys nothing else, and the worst case is worth stating. Admissible outcomes are unranked, so an injected planner is free to select the least useful one policy allows. Confidentiality sits outside: a branch legitimately reads back real CI logs and resource identifiers, and mediation does not stop it encoding them into an admissible effect. The envelope also bounds which effects commit, not how much irreversible risk a tree may accumulate. We treat that quantity axis as a budgeting problem in companion work [12]. The property is only as good as its two premises: complete mediation and effect-type handlers that achieve the reversal they claim (§3).

Example. We now walk the race of §1 end to end. `open` confines each process tree, and the repository token stays with the mediator. Branch A stages a draft PR: the mediator signs the call, records the effect against A, and returns a provisional handle. Reading it blocks until CI reports, then yields the real verdict, green, which A uses for its next edit. Branch B gets red, then tries to `curl` its diff to an outside endpoint. That call passes no tool-call monitor, but it reaches the mediator, which has no policy entry for the destination and denies it. The planner, possibly injected, selects A. Selecting is all it can do: the mediator revalidates that A's PR still applies to the branch head, rechecks destination, credential, and effect type, and only then publishes. It then walks B's handles in reverse, withdrawing the draft PR and canceling the CI run. The review-request email the provider already sent enters the audit log as residual: nothing can unsend it.

Placement. The barrier is an abstraction, not a placement: userspace hooks, a kernel-backed mediator, and a microVM per branch are candidate realizations, at increasing mediation strength. Enforcement below the planner is already reachable in userspace, as Sandlock shows with unprivileged Landlock and seccomp [19]. What differs is what each buys. Userspace egress filtering is a denylist that new syscalls and side channels eventually evade, whereas kernel interposition shrinks the trusted core and makes coverage less dependent on convention. A confined process also forks in about a millisecond where a Firecracker microVM needs roughly 125 ms and megabytes [1, 11], though that gap bites only at fan-out we have not measured.

3 Open Problems

(i) Commit-time consistency. Services offer no transaction protocol, so the mediator must run optimistic concurrency control against them: record what each staged effect assumed, recheck it at commit, abort when the world has moved. A cheap, provider-agnostic read set is the hard part. *(ii) Compositional commit.* Releasing a conflict-free subset of several surviving branches needs a notion of conflict over external effects, on shared resources and quota, that local-state merge does not supply. We are developing composition over nested race and merge sets in the full Effect Domains paper. *(iii) Speculative isolation.* Shadow namespaces and throttling (§2) trade fidelity for isolation: which effects tolerate a recorded response is open. *(iv) Aborting cognitive state.* Abort invalidates a branch's authority, not its memory: a branch that read a red CI verdict still carries those tokens after its draft PR is withdrawn, and wiping the context invites it to retry forever. Handles give a hook, since the mediator knows which context spans derive from which aborted effect, and harness-managed agent state [5, 16] offers mechanisms to act on it. Whether to invalidate, mark counterfactual, or summarize is open. *(v) Effect-type handlers.* The practical crux: the OS cannot infer reversibility, so someone must declare it, and whether agent tools converge on reusable types decides whether the abstraction deploys. *(vi) Confinement completeness.* Closing every externalization path by construction and keeping it closed as interfaces evolve. Confidentiality (§2) compounds it: covert channels and provider-visible state sit outside.

References

[1] Alexandru Agache, Marc Brooker, Andreea Florescu, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, USA, 419–434.

[2] Laurent Bindschaedler, Quentin Botha, and Christoph Siebenbrunner. 2026. Agent Flight Recorder: Tamper-Evident Audit Trails with On-Chain Anchoring for Long-Horizon Tool-Using Agents. In *7th International Conference on Blockchain Computing and Applications (BCCA 2026)*. IEEE.

[3] Laurent Bindschaedler, Quentin Botha, and Christoph Siebenbrunner. 2026. Bonded Recourse for Smart-Contract Settlement of Compensable Agent Side Effects. In *7th International Conference on Blockchain Computing and Applications (BCCA 2026)*. IEEE.

[4] 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] doi:10.48550/arXiv.2606.17573

[5] Excel Chukwu and Laurent Bindschaedler. [n. d.]. May the Memory Be With You: Efficient and Infinitely Updatable State for Large Language Models. In *The 5th Workshop on Machine Learning and Systems (EuroMLSys '25)*. doi:10.1145/3721146.3721951

[6] Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, Vol. 37. Curran Associates, Inc., Vancouver, BC, Canada, 82895–82920. doi:10.52202/079017-2636 Datasets and Benchmarks Track.

[7] Hector Garcia-Molina and Kenneth Salem. 1987. Sagas. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data (SIGMOD '87)* (San Francisco, CA, USA). Association for Computing Machinery, New York, NY, USA, 249–259. doi:10.1145/38713.38742

[8] Pat Helland. 2007. Life beyond Distributed Transactions: an Apostate's Opinion. In *Proceedings of the Third Biennial Conference on Innovative Data Systems Research (CIDR 2007)*. CIDR, Asilomar, CA, USA, 132–141. https://cidrdb.org/cidr2007/papers/cidr07p15.pdf

[9] Coleman Hooper, Minwoo Kang, Suhong Moon, Nicholas Lee, Eric Wen, John Wawrzynek, Michael W. Mahoney, Yakun Sophia Shao, Amir Gholami, and Kurt Keutzer. 2026. Speculative Interaction Agents: Building Real-Time Agents with Asynchronous I/O and Speculative Tool Calling. arXiv:2605.13360 [cs.LG] doi:10.48550/arXiv.2605.13360

[10] Jinhao Hu, Ashvin Goel, and Laurent Bindschaedler. 2026. Don't Trust the Code, Check Its Effects: Runtime Refinement for Regenerated Systems Code Under an Adversarial Generator. In *4th Workshop on Practical Adoption Challenges of Machine Learning for Systems (PACMI '26), co-located with SOSP*. Prague, Czechia. https://sites.google.com/view/pacmi

[11] Zijun Li, Jiagan Cheng, Quan Chen, Eryu Guan, Zizheng Bian, Yi Tao, Bin Zha, Qiang Wang, Weidong Han, and Minyi Guo. 2022. RunD: A Lightweight Secure Container Runtime for High-density Deployment and High-concurrency Startup in Serverless Computing. In *Proceedings of the 2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, USA, 53–68.

[12] Bardia Mohammadi and Laurent Bindschaedler. 2026. The Irreversibility Budget: Fleet-Level Risk Accounting and Admission Control for Agent Operating Systems. In *2nd Workshop on Operating Systems Design for AI Agents (AgenticOS @ SOSP 2026)*. Prague, Czechia. https://os-for-agent.github.io/

[13] Bardia Mohammadi, Nearchos Potamitis, Lars Henning Klein, Akhil Arora, and Laurent Bindschaedler. 2026. Atomix: Timely, Transactional Tool Use for Reliable Agentic Workflows. In *Proceedings of the ICLR 2026 Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD)*. OpenReview.net, Rio de Janeiro, Brazil. https://openreview.net/forum?id=UeRbEpSVUz

[14] Edmund B. Nightingale, Peter M. Chen, and Jason Flinn. 2005. Speculative execution in a distributed file system. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles (SOSP '05)* (Brighton, United Kingdom). Association for Computing Machinery, New York, NY, USA, 191–205. doi:10.1145/1095810.1095829

[15] Donald E. Porter, Owen S. Hofmann, Christopher J. Rossbach, Alexander Benn, and Emmett Witchel. 2009. Operating System Transactions. In *Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles (SOSP '09)* (Big Sky, MT, USA). Association for Computing Machinery, New York, NY, USA, 161–176. doi:10.1145/1629575.1629591

[16] Mofasshara Rafique and Laurent Bindschaedler. 2026. ClawVM: Harness-Managed Virtual Memory for Stateful Tool-Using LLM Agents. In *Proceedings of the Sixth European Workshop on Machine Learning and Systems (EuroMLSys '26)*. ACM, New York, NY, USA.

[17] Igor Santos-Grueiro. 2026. Lingering Authority: Revocable Resource-and-Effect Capabilities for Coding Agents. arXiv:2606.22504 [cs.CR] doi:10.48550/arXiv.2606.22504

[18] Cong Wang and Yusheng Zheng. 2026. Fork, Explore, Commit: OS Primitives for Agentic Exploration. In *1st Workshop on Operating Systems Design for AI Agents (AgenticOS 2026), co-located with ASPLOS 2026*. AgenticOS Workshop, Pittsburgh, PA, USA, 8 pages. https://os-for-agent.github.io/papers/AgenticOS_2026_paper_8.pdf

[19] Cong Wang and Yusheng Zheng. 2026. Sandlock: Confining AI Agent Code with Unprivileged Linux Primitives. arXiv:2605.26298 [cs.CR] doi:10.48550/arXiv.2605.26298

[20] Benjamin Wester, Peter M. Chen, and Jason Flinn. 2011. Operating system support for application-specific speculation. In *Proceedings of the Sixth European Conference on Computer Systems (EuroSys '11)* (Salzburg, Austria). Association for Computing Machinery, New York, NY, USA, 229–242. doi:10.1145/1966445.1966467

[21] Naimeng Ye, Arnav Ahuja, Georgios Liargkovas, Yunan Lu, Kostis Kaffes, and Tianyi Peng. 2026. Speculative Actions: A Lossless Framework for Faster AI Agents. In *Proceedings of the Fourteenth International Conference on Learning Representations (ICLR 2026)*. OpenReview.net, Rio de Janeiro, Brazil. https://openreview.net/forum?id=P0GOk5wslg

[22] Yusheng Zheng, Tianyuan Wu, Quanzhi Fu, Tong Yu, Wenan Mao, Tao Ma, Dan Williams, Wei Wang, and Andi Quinn. 2026. ActPlane: Programmable OS-Level Policy Enforcement for Agent Harnesses. arXiv:2606.25189 [cs.OS] doi:10.48550/arXiv.2606.25189