
An Agent Can Move and Still Not Be Portable

Anonymous Author(s)

Affiliation

Address

email

Abstract

1 Tool calls increasingly return handles that models are expected to carry into later
2 interactions. Checkpointing can preserve such a handle without preserving the au-
3 thority that gives it meaning, so an agent may resume elsewhere while remaining
4 dependent on the system it supposedly left. *Source severance* is a paired test in
5 which identical restored state is continued with the original source either reach-
6 able or removed. A LangGraph experiment establishes the persistence premise
7 by showing that an actual tool-returned handle survives checkpoint and resume
8 even though the checkpoint representation provides no framework-defined per-
9 value place for its external authority. PermeantOS, an early open-source system
10 for live migration of agent state, provides a migration case in which an importer
11 that already enforces rebinding obligations for credentials and excluded artefacts
12 accepts a documented tool-result resource URI without attaching an equivalent
13 obligation; source severance shows that the relation remains load-bearing after
14 migration. A small prototype captures authority and object identity when the han-
15 dle enters agent state, carries that relation in the migrated package, and rejects in-
16 complete bindings before continuation. Together these results indicate that agent
17 portability requires migration obligations to follow externally governed relations
18 as they move through persistent state.

1 Portability after checkpointing

20 Agent operating systems have brought resource management back to the centre of agent architecture.
21 AIOS, for example, places access to models and tools alongside scheduling, memory and storage
22 behind an operating-system layer [Mei et al., 2025]. The difficulty considered here is older than
23 contemporary agents, although agent state gives it a new form. Classical process-migration work
24 treated residual dependence on the origin as a design choice rather than an implementation detail
25 [Milojčić et al., 2000]. Sprite retained some home-machine dependencies in exchange for simpler
26 transparent migration, whereas the V-system sought to avoid dependence on the previous host. Zap
27 later virtualised resource names so that a migrated environment did not continue to rely on source-
28 host naming, while mobile-object systems made binding and rebinding policy explicit [Douglass and
29 Ousterhout, 1991, Theimer et al., 1985, Osman et al., 2002, Tanter and Piquer, 2001].

30 Recent agent runtimes address neighbouring parts of the checkpoint problem. Crab relates agent
31 turns to OS-visible sandbox effects when deciding what can be checkpointed and restored, Zheng et
32 al. derive preservation conditions for checkpoint, fork, restore and merge from execution records,
33 and Atomix tracks external tool effects transactionally across retries and failures [Wu et al., 2026,
34 Zheng et al., 2026, Mohammadi et al., 2026]. These systems govern effects or execution state
35 already represented by the runtime. The problem examined here begins one step earlier, when a
36 provider-owned reference crosses a tool boundary whose runtime knows the governing authority
37 and then becomes ordinary agent data that need not retain that relation.

38 The use of tool-returned handles is already explicit in current protocol design. MCP’s 2026-07-28
 39 revision removes protocol-level sessions and recommends that servers needing state across calls mint
 40 explicit handles for the model to pass back later. The maintainers explicitly argue that this works
 41 better than session state hidden in the transport because the model can see the handle and thread
 42 it between tools [Soria Parra and Delimarsky, 2026]. The same revision makes resource listings
 43 and reads cacheable through `ttlMs` and `cacheScope` hints [Model Context Protocol Contributors,
 44 2026]. Model-visible handles and cached resource reads are therefore part of the contemporary
 45 agent ecosystem rather than pathologies introduced for this experiment. Once a handle has entered
 46 a message, memory or other persistent state, preserving the value does not by itself preserve the
 47 authority that will make it usable after relocation.

48 Three properties are easy to collapse in practice. *Transportability* concerns whether state can be
 49 exported and imported, *continuation fidelity* whether the target resumes from the expected execution
 50 boundary, and *source independence* whether later behaviour can still resolve historical objects and
 51 effects after the origin disappears. Here, *portability* denotes their conjunction because an agent that
 52 resumes on a target while continuing to dereference its old provider has moved computationally
 53 without becoming source-independent.

54 The missing relation appears before migration but becomes visible only when continued state needs
 55 the source again. LangGraph establishes the persistence premise by showing that a mainstream
 56 checkpoint preserves the tool-returned value without a framework-defined per-value authority
 57 record. PermeantOS then supplies the discriminating migration case because its importer already
 58 enforces rebinding obligations for some external state, making the boundary of that machinery mea-
 59 surable. The final experiment returns to the earlier boundary and asks what changes when the run-
 60 time captures the relation at ingress rather than trying to reconstruct it at migration time.

61 2 Source severance

62 Source severance pairs two post-migration conditions and begins, before export, by enumerating
 63 the historical external relations that the chosen continuation will exercise. For each relation the
 64 experiment records the source authority, source handle and target equivalent when one exists. A
 65 historical object is therefore not any arbitrary piece of remote state, but an externally governed
 66 object created or resolved before export and intentionally revisited only after import. The package
 67 is exported once and cloned into matched target environments.

68 One environment leaves the original namespace reachable, while the other removes it after import
 69 without changing legitimate target credentials or services. Resolution caches are cold or bypassed,
 70 and no tested object is prefetched after import. This matters in an ecosystem where resource reads
 71 may legitimately be cached, because a warm cache can make a source-dependent continuation ap-
 72 pear independent of its source [Model Context Protocol Contributors, 2026]. Instrumentation at the
 73 source and target resolver boundary attributes every post-cutover lookup to one namespace or the
 74 other.

75 The intervention is related to controlled replay and fault injection but asks a different question.
 76 AgentCheck, for example, replays recorded MCP tool responses under injected failures [Mazumder
 77 and Lia, 2026]. Source severance instead keeps the migrated state byte-identical and changes only
 78 the post-import reachability of an authority whose historical relation is fixed in advance. A failure
 79 after the cut therefore says that the relation remains load-bearing; it does not merely show that a
 80 selected tool endpoint can fail.

81 For migrated state S , validator V and a pre-enumerated continuation E , let $D_E(S)$ contain the
 82 historical external relations that E requires after import. Let $O_V(S)$ contain the relations for which
 83 V enforces a post-migration obligation before continuation, such as a target-side binding or an
 84 unresolved-rebind condition that must be handled before source fallback. The validator covers the
 85 relations exercised by E when

$$D_E(S) \subseteq O_V(S).$$

86 If the source-connected continuation succeeds by consulting the old namespace while the matched
 87 source-severed continuation cannot resolve relation r , severance establishes that r remains load-
 88 bearing. Inspection of the validator then distinguishes a missing obligation, $r \notin O_V(S)$, from an
 89 obligation whose later execution is faulty. Because $D_E(S)$ is defined by the continuation under

90 test, successful severance is only a lower bound on coverage rather than a completeness claim about
91 arbitrary future state.

92 3 What checkpointing carries

93 LangGraph supplies the persistence premise for the migration experiment. Its checkpoint path makes
94 no cross-runtime rebinding promise, but it allows a narrower question to be tested directly. A tool-
95 returned provider handle can enter ordinary framework state and remain behaviourally important
96 after checkpoint and resume, while the checkpoint representation itself provides no framework-
97 defined per-value field for the authority or post-migration resolution relation that gives the han-
98 dle meaning. In LangGraph’s pinned checkpoint interface, arbitrary channel state is persisted in
99 `channel_values` without a corresponding framework-defined per-value authority slot [LangGraph
100 Project, 2026a]. The relation is therefore not lost during migration because, at this layer, it was never
101 carried in the first place.

102 The probe uses LangGraph 1.2.11, `langgraph-checkpoint` 4.2.0 and the SQLite saver 3.1.1 from
103 the same monorepo commit 11ee1859 [LangGraph Project, 2026a,b]. A LangChain tool invoked
104 inside a StateGraph returns a source-owned receipt URI, after which execution is interrupted im-
105 mediately before the resolver and persisted through the SQLite checkpointer. The restored graph
106 resumes from that boundary without executing the tool again, and the handle is unchanged in both
107 environmental arms.

108 With the source reachable, resumed execution makes one source lookup and completes correctly,
109 whereas removing the source leaves the byte-identical restored state attempting the same lookup and
110 becoming blocked. That behaviour is consistent with LangGraph’s checkpoint contract rather than a
111 defect in it, but it establishes the premise needed here. Faithful framework persistence can preserve
112 the handle while leaving the governing relation outside the checkpoint representation, locating the
113 problem at state entry and explaining why a migration-time rule alone cannot recover semantics that
114 were never carried.

115 4 Validator coverage in PermeantOS

116 PermeantOS v0.1.0 is an open-source project for live migration of agent state, and its pinned security
117 policy describes the project as a research preview [PermeantOS Project, 2026f]. Its importer already
118 contains explicit rebinding machinery. That existing machinery makes the system a discriminating
119 case, because an uncovered relation can be compared directly with recognised external-state paths
120 that the same importer already knows how to govern.

121 The evaluation uses commit c76c489d and its complex Agent Memory Graph path [PermeantOS
122 Project, 2026b]. The unmodified export contains 27 nodes and 25 edges, including four artefacts,
123 two durable memories, four tool calls and two pending tool calls. The importer verifies graph in-
124 tegrity, security attestation, artefact packaging, retrieval and vector state, prompt and simulated-KV
125 hashes, replay policy and deterministic continuation. The same graph construction is used by the
126 project’s published AWS complex-agent migration path, although that run does not exercise the
127 receipt relation introduced below [PermeantOS Project, 2026a,c].

128 The import rules expose an asymmetry between explicitly governed external state and the more
129 general tool-result reference field. Credential references carry an explicit import contract in which
130 every `credential_ref` node must set `redaction_state` to `external_only`, set `required` to
131 `True`, and include a non-empty binding; malformed declarations are rejected. Excluded artefacts
132 have a parallel contract in which `restore_policy` is `external_rebind`, the blob is omitted, and
133 `rebind_required` is `True`. Tool-result `resource_refs`, by contrast, are documented simply as
134 optional resource URIs returned by the tool, and the schema accepts string values without restricting
135 them to packaged artefacts or attaching an analogous rebinding requirement [PermeantOS Project,
136 2026d,e,b]. The stock graph happens to use this field for `artifact:reports/result.json`,
137 whose referent is governed separately by the artefact manifest. The controlled receipt URI there-
138 fore follows the documented purpose of `resource_refs` while exercising the case in which the
139 referenced object remains external and is not represented by a packaged artefact.

Two deterministic namespaces contain the same receipt under source identifier R7K381AX9 and target identifier M2P904QZ1. The source relation is instantiated in four forms across three state classes: assistant-message state; durable memory; the authority-opaque URI receipt:R7K381AX9 in tool-result resource_refs; and the visibly source-owned URI receipt://source.example/R7K381AX9 in the same field. The message and memory placements carry no structural resource marker. All affected hashes, retrieval evidence, vector state, attestation and deterministic-continuation evidence are regenerated through PermeantOS’s own functions before import, while post-import resolution occurs only through the instrumented fixture gateway. Matched environmental conditions therefore have identical graph hashes and differ only in source reachability or, for the mapping control, the presence of a target translation.

The controls exercise the same importer’s recognised validation paths so that the severance result can be interpreted against mechanisms known to work. A migrated target map lets the evaluator route through a supplied target translation, although PermeantOS itself does not execute the map. Excluding reports/result.json activates the native resource path and causes import to report a rebinding obligation. Four malformed credential or artefact declarations test the enforced branches, while a dependency-free continuation checks that severance alone does not create a source lookup.

Table 1: Source severance in PermeantOS: four handle forms across three state classes, 18 continuation trials and four import-only malformed controls. S/T count attempted source/target lookups; * mapping is evaluator-executed.

State / handle form	Source up	Source down	Down + map
Message; memory; tool result (opaque or source URI)	correct; S=1	blocked; S=1	correct; T=1
Evaluator map + valid credential declaration	correct; T=1*	correct; T=1*	n/a
Native external artefact + strict evaluator	rebind; S=0,T=0	rebind; S=0,T=0	n/a
No planted relation	correct; S=0,T=0	correct; S=0,T=0	n/a

Every package carrying the planted receipt passes import_session, yet across all four handle forms the subsequent behaviour remains source-dependent. Source-connected continuation succeeds by looking up the receipt at the source, while source-severed continuation attempts the same source lookup and is blocked. Supplying the target translation redirects continuation to the target with no source callback. The opaque and source-visible tool-result forms test the documented resource-URI field; message and memory show the same relation after structural typing has been lost. The four malformed declarations on recognised paths are rejected, and the dependency-free continuation makes no source lookup in either environment.

The native artefact behaves differently because PermeantOS reports an unresolved rebinding obligation whether or not the source is available, after which the strict evaluator refuses continuation before any source fallback. PermeantOS supplies the signal while the evaluator supplies the fail-closed policy, so the control establishes declaration enforcement rather than autonomous rebinding. In the planted-receipt packages the exercised receipt relation lies in $D_E(S)$ but not in $O_V(S)$, whereas the native external artefact enters $O_V(S)$ through its enforced rebinding declaration. This need not be a PermeantOS defect: its enforced paths make measurable the broader boundary of migration validation, which can guarantee only relations made obligatory by the validator’s representation.

5 Carrying the relation through migration

Because the same relation can move between representations, a repair cannot stop at the field in which the experiment first exposes it. A rule attached only to resource_refs would cover the measured tool-result placement but would lose the same relation after a model or runtime copied it into a message or memory. The difficult boundary is therefore not one field versus another, but the moment when an externally governed value crosses from a context in which its authority is known into model-facing state that may preserve only the value itself.

A comparison of interchange formats has to distinguish useful metadata from an enforceable migration contract. Here a post-migration obligation counts only when the published schema or normative text requires target-side resolution metadata, or when an importer rejects state that omits equivalent semantics; optional provenance, naming or content-addressing fields do not count. Under that rule, AAIF carries state provenance but does not require target translation for each external reference in

its value-bearing fields [van Bussel, 2026]; Portable Agent Memory provides memory provenance and content addressing without a required external-object translation contract [Ravindran, 2026]; and MCP assigns URIs to resources while leaving migration of application-level handles outside protocol scope [Model Context Protocol Contributors, 2026, Soria Parra and Delimarsky, 2026]. These formats preserve useful information, but persistence and provenance alone do not determine how a historical external relation should resolve after the source is gone.

The corresponding design boundary is tested over the same PermeantOS package path. At tool ingress, while the runtime still knows what produced the value, a small prototype receives the provider handle together with explicit source authority and object identity. It records that relation, emits a stable slot such as `extref:receipt-1` into model-facing state, and carries the relation in the migrated package together with its target binding. On restore, the wrapper validates every encountered slot against the package-carried registry before continuation. It never recovers authority by parsing arbitrary model-facing strings, and the raw source identifier is absent from the graph state seen by the agent.

The more informative result appears when the registry is incomplete. A registry with no entry for the slot and a registry whose entry lacks the target binding both pass the underlying PermeantOS importer unchanged, yet the wrapper rejects them before continuation. By contrast, the six complete registries resolve through the target as expected because the binding is captured at ingress and consumed at restore. Those passing trials show that the same package-carried relation works without a source callback after the slot has been placed in message, memory or tool-result state. Table 2 makes both sides of the comparison explicit.

Table 2: Package-carried reference registry. The six valid cells are deterministic continuation trials; the malformed row summarises two import-valid registry mutations.

Placement / condition	Source up	Source down
Message	correct; S=0,T=1	correct; S=0,T=1
Memory	correct; S=0,T=1	correct; S=0,T=1
Tool result	correct; S=0,T=1	correct; S=0,T=1
Malformed registry	PermeantOS import passes; wrapper rejects	

Explicitly storing relation metadata somewhere in the package is therefore not sufficient; the relation has to participate in an enforced migration contract. Figure 1 shows where that contract is attached in the prototype.

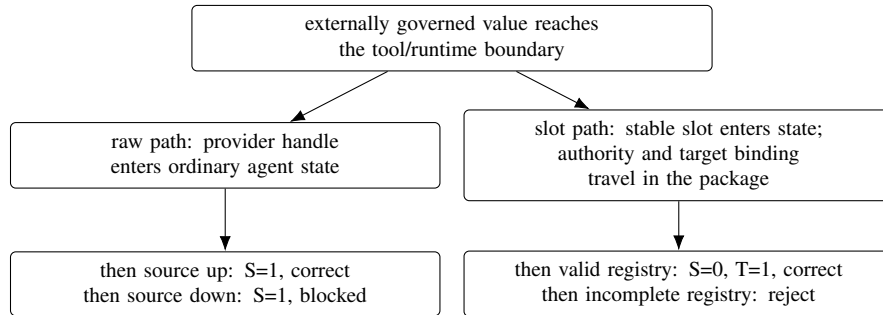


Figure 1: The measured raw-handle path leaves the source load-bearing, whereas the prototype couples a stable model-facing slot to package-carried resolution semantics.

Although the prototype closes the exercised failure, it does not solve dependency discovery and assumes both that an external value is recognised at ingress and that the stable slot survives later transformations of agent state. A model that paraphrases the slot away, or an external relation created outside a mediated tool/runtime boundary, falls outside the mechanism. Source severance is likewise continuation-relative, so a failure supplies a witness to residual dependence while a successful finite test can only establish a lower bound on what the migration contract covers. Deciding which values may become externally load-bearing before their future use is known remains the harder systems problem.

6 Limits and reproducibility

The experiments are deterministic and mechanism-oriented rather than an estimate of failure prevalence. PermeantOS is an early research system, the receipt relations are controlled fixtures, and the published AWS migration is not itself source-severed. The LangGraph experiment establishes that a tool-originated external relation can remain load-bearing across an actual framework checkpoint and resume, not that LangGraph contains the same validator boundary. The study also does not exercise committed external effects, adversarial source/target collusion or long-running natural traces in which dependencies arise without being planted.

The anonymised supplementary artifact contains the benchmark runners, pinned PermeantOS source and hash, migrated packages, raw condition records, validator controls, the LangGraph checkpoint-resume probes, the package-carried registry prototype and reproduction instructions. The vendored PermeantOS source is byte-identical to the pinned upstream file, while the experimental instrumentation remains in separate runners; the bundle retains the upstream Apache-2.0 licence. LangGraph code is installed from its MIT-licensed packages rather than redistributed. The artifact contains no author repository URL, credentials, production APIs, cloud resources or personal data.

7 Conclusion

An agent can preserve an externally governed value after losing, or never carrying, the relation that makes the value resolvable. The LangGraph probe locates that problem at state entry, while the PermeantOS experiment shows how it can remain outside a migration validator even when neighbouring external-state paths already have explicit rebinding obligations. Source severance makes the residual dependence visible, and the prototype shows why the practical boundary for an Agentic OS lies before migration itself. External-reference semantics have to be captured while the governing authority is known and kept coupled to the value through persistence and migration; otherwise the checkpoint may travel while a load-bearing part of the agent remains behind.

References

- Fred Douglass and John K. Ousterhout. Transparent process migration: Design alternatives and the Sprite implementation. *Software: Practice and Experience*, 21(8):757–785, 1991. doi: 10.1002/spe.4380210802.
- LangGraph Project. LangGraph checkpoint base interface, August 2026a. URL https://github.com/langchain-ai/langgraph/blob/11ee185999b86bfea2d8c0e69cef9a5e37acf686/libs/checkpoint/langgraph/checkpoint/base/__init__.py. langgraph-checkpoint 4.2.0; implementation pinned at monorepo commit 11ee185999b86bfea2d8c0e69cef9a5e37acf686.
- LangGraph Project. LangGraph sqlite checkpoint saver, August 2026b. URL https://github.com/langchain-ai/langgraph/blob/11ee185999b86bfea2d8c0e69cef9a5e37acf686/libs/checkpoint-sqlite/langgraph/checkpoint/sqlite/__init__.py. langgraph-checkpoint-sqlite 3.1.1; implementation pinned at monorepo commit 11ee185999b86bfea2d8c0e69cef9a5e37acf686.
- Aritra Mazumder and Nusrat Jahan Lia. AgentCheck: A reproduce–intervene–mitigate workbench for LLM agents over MCP. *arXiv preprint arXiv:2607.11098*, 2026. URL <https://arxiv.org/abs/2607.11098>.
- Kai Mei, Xi Zhu, Wujiang Xu, Mingyu Jin, Wenyue Hua, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. AIOS: LLM agent operating system. In *Proceedings of the 2nd Conference on Language Modeling (COLM 2025)*, 2025. URL <https://openreview.net/forum?id=L4HHkCDz2x>.
- Dejan S. Milošević, Fred Douglass, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. Process migration. *ACM Computing Surveys*, 32(3):241–299, 2000. doi: 10.1145/367701.367728.

Model Context Protocol Contributors. Model context protocol specification: Resources, July 2026. URL <https://modelcontextprotocol.io/specification/2026-07-28/server/resources>. Specification version 2026-07-28.

Bardia Mohammadi, Nearchos Potamitis, Lars Klein, Akhil Arora, and Laurent Bindshaedler. Atomix: Timely, transactional tool use for reliable agentic workflows. *arXiv preprint arXiv:2602.14849*, 2026. URL <https://arxiv.org/abs/2602.14849>.

Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh. The design and implementation of Zap: A system for migrating computing environments. In *5th Symposium on Operating Systems Design and Implementation (OSDI 02)*. USENIX Association, 2002. URL <https://www.usenix.org/conference/osdi-02/design-and-implementation-zap-system-migrating-computing-environments>.

PermeantOS Project. PermeantOS white paper: Live migration for AI agent state, June 2026a. URL <https://www.permeantos.org/whitepaper/>.

PermeantOS Project. Agent memory graph local agent example, 2026b. URL https://github.com/kabudu/permeant-os/blob/c76c489d39d257c3313299033217741d94205427/examples/agent-memory-graph/local_agent.py. PermeantOS v0.1.0, commit c76c489d39d257c3313299033217741d94205427.

PermeantOS Project. AWS real-runtime complex agent E2E — 2026-06-20, June 2026c. URL <https://github.com/kabudu/permeant-os/blob/c76c489d39d257c3313299033217741d94205427/docs/aws-real-runtime-complex-agent-2026-06-20.md>. PermeantOS v0.1.0, commit c76c489d39d257c3313299033217741d94205427.

PermeantOS Project. Agent memory graph v0 documentation, June 2026d. URL <https://github.com/kabudu/permeant-os/blob/c76c489d39d257c3313299033217741d94205427/docs/agent-memory-graph.md>. Pinned at commit c76c489d39d257c3313299033217741d94205427.

PermeantOS Project. Agent memory graph v0 json schema, June 2026e. URL <https://github.com/kabudu/permeant-os/blob/c76c489d39d257c3313299033217741d94205427/docs/schemas/agent-memory-graph-v0.schema.json>. Pinned at commit c76c489d39d257c3313299033217741d94205427.

PermeantOS Project. PermeantOS security policy, June 2026f. URL <https://github.com/kabudu/permeant-os/blob/c76c489d39d257c3313299033217741d94205427/SECURITY.md>. Pinned at commit c76c489d39d257c3313299033217741d94205427; describes the project as a research preview.

Santhosh Kumar Ravindran. Portable agent memory: A protocol for cryptographically-verified memory transfer across heterogeneous AI agents. *arXiv preprint arXiv:2605.11032*, 2026. URL <https://arxiv.org/abs/2605.11032>. v1, 10 May 2026.

David Soria Parra and Den Delimarsky. The 2026-07-28 model context protocol specification, July 2026. URL <https://blog.modelcontextprotocol.io/posts/2026-07-28/>.

Éric Tanter and José Miguel Piquer. Managing references upon object migration: Applying separation of concerns. In *XXI International Conference of the Chilean Computer Science Society*, pages 264–272. IEEE, 2001. doi: 10.1109/SCCC.2001.972656.

Marvin M. Theimer, Keith A. Lantz, and David R. Cheriton. Preemptable remote execution facilities for the V-System. In *Proceedings of the 10th ACM Symposium on Operating Systems Principles (SOSP)*, pages 2–12. ACM, 1985.

Bob van Bussel. Autonomous agent interchange format (AAIF) — SC-006 v3.4.0 agent state checkpoint schema, June 2026. URL <https://github.com/Observalytics-SL/aaif/blob/3d4c7238c5df93554b7d175c73b1ebcb6215f31c/schema/agent-state.schema.json>. Public Draft; JSON Schema 2020-12; schema pinned at commit 3d4c7238c5df93554b7d175c73b1ebcb6215f31c.

- 313 Tianyuan Wu, Chaokun Chang, Lunxi Cao, Wei Gao, and Wei Wang. Crab: A semantics-aware
314 checkpoint/restore runtime for agent sandboxes. *arXiv preprint arXiv:2604.28138*, 2026. URL
315 <https://arxiv.org/abs/2604.28138>.
- 316 Yusheng Zheng, Xiaoyu Song, Yanpeng Hu, Lebin Cheng, Yuxi Huang, and Wei Zhang. When can
317 agents safely checkpoint, fork, restore, and merge? exact checking for execution edits. *arXiv*
318 *preprint arXiv:2608.22928*, 2026. URL <https://arxiv.org/abs/2608.22928>.