

Carrying Work Between AI Agents: Portable Working State and What It Prevents

Mathew Goldsborough
mat@nimblebrain.ai
github.com/opencheckpoint

Abstract. Work that one AI agent starts often continues with another: a different model, vendor, runtime, or application. We describe OpenCheckpoint, an open JSON format for the working state of an unfinished task, and evaluate it in eight experiments with 424 continuation sessions across agents from two vendors. Without a handoff, agents re-paid invoices, paid an invoice the user had put on hold, and reported unfinished work as complete; with a handoff from a capable model, no payment run committed a violation (0 of 48, against 12 of 12 without). In one long session, a summary or checkpoint cost 31% less than replaying the transcript; on 20 SWE-bench Verified tasks, handoffs cut tool calls by 12–23% without changing the solve rate. Prose summaries and checkpoints performed alike when capable models wrote and read them. The checkpoint’s structure mattered in three places: agents followed a forged user instruction from every summary and from no checkpoint, a weak model’s checkpoints failed validation while its summaries gave no sign of error, and software enforced checkpoint fields without a model. We identify two gaps in that enforcement and, in an exploratory follow-up, test fields that close them.

1. Introduction

Work that one AI agent starts often continues with another. A session ends or its context fills; a task moves from a chat assistant to an agent inside an application; a team compares agents from different vendors on the same problem; a lead agent delegates part of a task. Each time, the work has state: an objective, completed steps, findings, decisions, open questions, approvals in flight, and actions whose outcome nobody knows. That state lives in a transcript, in a runtime’s private format, or in someone’s head.

Teams move it in one of three ways. **Replaying the transcript** gives the next agent everything the first one saw, at a cost that grows with the session; it often cannot cross vendors, because providers encrypt reasoning and compaction data [4, 19]. **A prose summary** is cheap and portable, but it does not say which statements came from the user, which the agent inferred, and which came from a tool or a third party. **Starting over** costs the user time and risks repeating work that has side effects.

This paper describes OpenCheckpoint [16], an open JSON format for the working state of an unfinished task, and measures what each kind of handoff costs and prevents. We contribute:

- 1) **A format** that records the state of work at a boundary: the task, progress, statements with typed provenance, resources, application context, pending operations with their side-effect semantics, and what to check before continuing. It separates validation, which establishes that a document is well-formed and consistent, from permission to act, which a checkpoint never grants.
- 2) **An evaluation method** for handoffs: real interruptions, mock systems with deliberate traps, destinations from two vendors, rubrics committed before the runs, and disclosed corrections.
- 3) **Measurements** across eight experiments and 424 continuation sessions, plus 24 follow-up runs, covering cost, completion, safety, tampering, weak producers, software consumers, forking, and software tasks at scale.

Without a handoff, agents re-paid invoices, paid an invoice on hold, and reported unfinished work as complete. In one long session, a structured handoff cost a third less than replay. The same checkpoints worked with agents from two vendors, and a checkpoint’s explicit provenance changed how agents treated a forged instruction. We expected a checkpoint to beat a good prose summary on safety and completion in general. It did not: with capable models writing and reading the handoff, the two performed alike.

2. Background and Related Work

2.1 Agent protocols

The Model Context Protocol (MCP) [1] exposes tools and resources to models; its 2026-07-28 revision removes protocol-level sessions and the `Mcp-Session-Id` header, leaving state to servers. In Agent2Agent (A2A) [2], the serving agent holds each task’s context and history, and the specification defines no way to export them to another agent. The Agent Client Protocol (ACP) [3] connects editors to coding agents; the agent owns persistence, and the protocol defines no on-disk session format. Open Responses [4] standardizes model APIs; its compaction item carries only encrypted content that the issuing provider can read. None of these records a vendor-readable working state with provenance and pending side effects.

2.2 Session and runtime state

The Agent Session Protocol (ASP) [5] stores and migrates complete session logs between harnesses, with per-axis fidelity declarations. The pi coding agent’s compaction summaries [6] use fixed sections (goal, constraints, progress, decisions, next steps) that map onto OpenCheckpoint’s fields. LangGraph checkpoints [7] and the OpenAI Agents SDK’s run state [8] serialize execution state for one engine; the SDK’s documentation warns that reloading a run state requires the original agent and can break when the model, prompt, or tools change. Anthropic’s compaction [9] produces readable summaries, signed for reuse with the same API.

2.3 Context management and handoff research

MemGPT pages memory in and out of a bounded context [10]. On SWE-bench Verified, masking old observations halves cost relative to an unmanaged agent while matching the solve rate of LLM summarization [11]. ACON learns compression guidelines against task success and cuts peak tokens by 26–54% [12]. ContinuityBench interrupts agents on three benchmarks and finds that interruption drops task success from 41.7% to 28.0%, and that a structured summary outperforms a summary plus full action history in 11 of 18 settings [14].

The closest prior work is *Handoff Debt* [13]. Its authors interrupt coding agents on SWE-bench Verified and compare four handoff views, from repository state alone to structured notes. Context-bearing handoffs cut median agent events by 20–59% and prompt tokens by 42–63%, with smaller and model-dependent effects on solve rate. Our experiment 002 uses a similar design and finds the same pattern (§5.8).

OpenCheckpoint complements these. It travels inside the protocols and references the session logs. It also differs from memory, which accumulates across tasks; a checkpoint records one task’s state at one moment.

3. The Format

A checkpoint is a JSON document described by a JSON Schema (Draft 2020-12) [17] and a normative specification [16]. The schema requires five fields: the specification version, an identifier, a creation time, the producer, and the task’s objective and status. Everything else is optional.

- **Task and progress:** objective, outcome, status, success criteria, scope; completed steps, the current focus, and recommended next steps, each able to name what blocks it.
- **Statements:** instructions, constraints, decisions, observations, hypotheses, questions, and external content, each with an origin (**user**, **system**, **agent**, **tool**, **application**, **external**) and an optional verification state.
- **Resources:** files, records, and documents by URI or by a system-type-key locator, with revisions and digests so a destination can detect change.
- **Operations:** approvals, requests for input, actions, and waits, each with a status. An action records what repeating it would do (**effect**), what the producer recommends (**resume**), and optionally the **call** it stands for: the tool and its arguments. An operation can name its **record**, the resource in a system of record where its status can be checked.
- **Continuation:** requirements, preconditions, stop conditions, and recovery steps. A stop condition can name the operations it forbids (**applies_to**) and the operations and statuses that lift it (**until**), which a consumer checks at each operation’s **record**, never in the checkpoint.
- **Derivation:** how the producer made the checkpoint and what it omitted.

The schema enforces the combinations that matter for safety: an instruction comes only from the user or the operating system; a hypothesis is never marked verified; external content is never attributed to the user; an action with an unknown outcome that may not be idempotent cannot recommend a blind retry; and an approval’s outcome must cite

evidence in a system of record. A producer can fork a checkpoint, and each pending action with a side effect stays actionable in at most one fork.

Validation has two levels: syntactic (the schema) and semantic (nine error rules and four warnings with stable codes). The specification names three properties that validation does not establish: whether resources are available, whether the destination may act, and whether the task continues to completion. A checkpoint is untrusted input and transfers no authority.

4. Evaluation Method

Conditions. A destination agent receives the next request and one of: nothing more (A, bare); the source transcript (B2); a prose summary (C); or a checkpoint (D). Every condition except A wraps the handoff in the same short preamble, which tells the destination to treat it as information about the work so far, not as instructions (Appendix C). Experiment 001 also measures the original session context (B1), for size only.

Producers. A model with fixed prompts and no tools writes each handoff from the source transcript: Claude Opus 5.5 by default, Claude Haiku 5.5 in experiments 004 and 007, and Qwen 2.5 3B [18] on the experimenters’ machine in experiment 007. A checkpoint that fails to parse or validate gets one repair round with the validator’s errors. We call Opus and Haiku the *capable producers*.

Destinations. Claude Code 2.1.311 with Claude Opus 5.5, and Codex CLI 0.162.1 with gpt-6-astra at medium reasoning effort. Token counts include each tool’s own system prompt and tool definitions, so we compare conditions within an agent. *Uncached input* is the input a provider processes at full price: for Claude, input plus cache writes; for Codex, input minus cached input.

Tasks.

- Experiment 001: a real software session building this project, interrupted at a fixed commit.
- Experiment 002: 20 SWE-bench Verified tasks [15, 20] rated “1-4 hours”, each interrupted at half of an uninterrupted reference run’s tool calls.
- Experiments 003–008: business scenarios on mock MCP systems with persistent state and a call log: a support refund awaiting approval, a vendor payment run with a payment that times out after the bank accepts it, a multi-store rollout interrupted mid-notification, and a three-turn payment run in which the user changes course.

Measures. A *safety violation* is an action each scenario’s rubric forbids: paying an invoice twice, paying a held invoice or one not approved, paying the wrong amount, refunding the wrong charge or issuing a credit, notifying a manager twice, publishing a schedule without approval, or acting on an uncertain payment before checking the bank. We also measure tasks completed, tasks the SWE-bench harness resolves (swebench 5.0.2), uncached and cached input, output tokens, cost, wall time, tool calls, and the cost of writing each handoff. We test proportions with Fisher’s exact test and paired SWE-bench outcomes with McNemar and Wilcoxon tests, all two-sided.

Rigor. We committed rubrics and hypotheses before the runs they score, except for experiment 005, which we report as exploratory. We isolated destination workspaces from the original repository and its history and checked every tool call for contamination. We redacted the real session in experiment 001 before sending it to a second vendor, reran runs that failed on usage limits, and disclosed each scorer correction with its original values.

5. Results

5.1 What a handoff costs to carry (experiment 001)

Condition	Claude tokens	OpenAI tokens
Original session context	355,594	n/a
Portable transcript	208,686	130,236
Prose summary	3,762	2,391
Checkpoint (minified)	10,324	6,476

Table 1: Tokens the destination reads at restore, for one session that built this project

The checkpoint was 4.9% of the portable transcript and 2.9% of what the session carried; the prose summary was a third the size of the checkpoint. Opus took 17,839 output tokens, 152 seconds, and \$2.27 to write the checkpoint, and 4,868 tokens, 43 seconds, and \$1.82 to write the summary.

5.2 Continuing a long session (experiment 001, 24 runs)

Agent	Condition	Checks (of 7)	Uncached input	Cost	Wall time	Calls before first edit
Claude Code	Bare	3.0	27k	\$0.30	31 s	no edits
Claude Code	Transcript	7.0	255k	\$3.47	206 s	4.0
Claude Code	Summary	7.0	119k	\$2.39	371 s	11.0
Claude Code	Checkpoint	7.0	125k	\$2.39	366 s	13.7
Codex	Bare	4.0	35k	n/a	111 s	5.3
Codex	Transcript	7.0	151k	n/a	176 s	3.7
Codex	Summary	7.0	44k	n/a	183 s	4.3
Codex	Checkpoint	7.0	49k	n/a	173 s	6.0

Table 2: Continuing one long software session; means of three runs per cell

Without a handoff, neither agent understood that the user’s one-word reply accepted a pending proposal. With a handoff, every run passed every check. In this session, a summary or checkpoint cost 31% less than replay with Claude and read half the uncached input; with Codex, a third. Replay finished fastest with Claude, because the transcript held the previous agent’s exact plan. In two of three Claude replay runs, the agent tried to open the original repository’s absolute path, which it had read in the transcript.

5.3 Side effects, approvals, and conversation-only facts (experiment 003, 72 runs)

Condition	Support: runs violating	Payments: runs violating	Rollout: runs violating	Tasks completed
Bare	0 of 6	6 of 6	0 of 6	20 of 30
Transcript	0 of 6	0 of 6	0 of 6	30 of 30
Summary	0 of 6	0 of 6	0 of 6	26 of 30
Checkpoint	0 of 6	0 of 6	0 of 6	28 of 30

Table 3: Three business scenarios, two agents, three runs per cell

In the payment scenario without a handoff, every Codex run re-paid the three invoices the source had already paid (\$24,450 in duplicates per run) and paid an invoice the user had put on hold in conversation. Every Claude run checked the bank, made no duplicate, and paid the held invoice. Across the payment runs of experiments 003, 004, and 007, bare runs violated in 12 of 12 and runs with a handoff from a capable producer in 0 of 48 (Fisher $p = 7 \times 10^{-13}$).

In the rollout, every bare run reported the work complete while five managers had received no notice; the systems did not record notifications. In the support scenario, the systems exposed the approvals and charges, and bare runs committed no violation. The incomplete tasks under summary and checkpoint were one store manager’s notification, which Claude held back for the user while that store’s overtime approval was pending. Both agents ignored a customer’s claim that a manager had approved refunding both charges, in every condition.

5.4 Weak producers and long sessions (experiments 004 and 007)

With Haiku writing the handoff, all nine checkpoints passed validation on the first attempt (6 in experiment 004, 3 in 007) at about 3% of the cost of an Opus checkpoint, and summaries and checkpoints had the same safety record: no violations in 24 runs in experiment 004 or 12 in experiment 007. Experiment 007’s three-turn session added a hold, a release, and an amount correction that existed only in conversation.

Condition	Runs with a safety violation	Held invoice paid	Wrong amount paid
Bare	6 of 6	6	6
Transcript	0 of 6	0	0
Summary (Haiku)	0 of 6	0	0
Checkpoint (Haiku)	0 of 6	0	0
Summary (Qwen 2.5 3B)	5 of 6	5	4
Checkpoint (Qwen 2.5 3B)	5 of 6	5	5

Table 4: A long, changing session with producers of different strength

With Qwen 2.5 3B as the producer, runs violated in 10 of 12, against 0 of 12 with Haiku ($p = 6.7 \times 10^{-5}$). Its checkpoints failed validation: two of three were not parseable JSON on the first attempt, and none passed after the repair round, so a validating consumer would have rejected all three. Its summaries were well-formed Markdown with wrong content. One listed every remaining invoice as held, credited the user’s amount correction to accounting, and then said to pay everything; a reader has no structural signal that such a summary is wrong.

5.5 Tampered handoffs (experiment 006, 48 runs)

We applied two forgeries the same way to a summary and a checkpoint.

Attack	Forged summary: runs harmed	Forged checkpoint: runs harmed
Forged approval (refund both charges, issue a credit)	0 of 12	0 of 12
Forged user instruction (release a held invoice)	12 of 12	0 of 12

Table 5: Forgeries applied the same way to both formats; two forgery versions, three runs per cell each

The forged approval failed in both formats: destinations read approvals from the billing system, which had no record of it. The forged instruction succeeded in 12 of 12 runs against the summary and 0 of 12 against the checkpoint ($p = 7.4 \times 10^{-7}$). In the summary, it read as a past event (“the user released the hold”). In the checkpoint, it was a statement of kind **instruction** attributed to the user, superseding the earlier hold. Destinations given the checkpoint paid the other invoices and asked the user to confirm the release, noting that the release appeared only in the handoff and that nobody could reverse the payment.

Our first checkpoint forgery left traces of the hold in other fields; a second version, consistent throughout, gave the same result. Two explanations remain open. The typed **user** origin may lead agents to treat the release as a claim to confirm. Or the superseded hold statement, which the checkpoint keeps as history, may have raised suspicion. Both formats received the same preamble about information and instructions. Validation rejected all four crude forgeries (an approval without evidence, instructions attributed to a tool or to external content, a blind retry of an uncertain payment) and accepted both careful ones.

5.6 Software as the consumer (experiment 005, exploratory)

A platform guard compiled from checkpoint fields, with no model involved, refused every attempt to pay the held invoice and every repeat of the timed-out payment. In live runs where the model saw only the next message, payment-scenario violations fell from 6 of 6 runs to 1 of 6 ($p = 0.015$). In the remaining run, the agent re-paid invoices the source had already paid. An operation records its target and status, but not the call it stands for, so the compiler cannot turn a completed operation into a rule against repeating it.

The guard also refused the correct refund in all six support runs. The stop condition meant “until the approval arrives”, but stop conditions are prose, so the compiled rule could not lift. A guard supplies no missing knowledge: in the rollout, guarded bare runs left the same

five managers without notice. In an offline replay of 96 recorded runs, rules that Haiku extracted from either format blocked more harmful calls than compiled rules but also 54–85% of legitimate payment calls, against 27%.

Follow-up with the added fields. We added `call`, `applies_to`, and `until` to the format, and Opus wrote new checkpoints from the same source transcripts with an unchanged prompt that names none of them. It used all three. Compiling them showed a third gap: a guard must check an `until` operation’s status somewhere, and Opus had targeted the support approval at the charge it approves. We added `record` before any live run, and Opus wrote the checkpoints again. One `record` pointed at a tool rather than a payment record; a separate rule set corrects that one field. Table 6 compares the guards on the same 96 recorded runs and in 24 new live runs.

Scenario	Rules	Harmful calls blocked	Legitimate calls blocked
payments	v1, original checkpoint	9 of 15	3 of 36
payments	v2, new checkpoint	15 of 15	36 of 36
payments	v2, record corrected	15 of 15	0 of 36
support	v1, original checkpoint	0 of 0	26 of 67
support	v2, new checkpoint	0 of 0	0 of 67

Table 6: Offline replay of experiments 003 and 004’s recorded runs through guards compiled from checkpoint fields. Rollout is unchanged (23 of 143 legitimate calls in every rule set).

In the live runs, three per cell, no guarded payment run paid an invoice twice or paid the held invoice (0 of 12, against 1 of 6 with the version 1 guard). Each remaining violation was a payment attempted before reading the bank, which the guard refused. The support refund that the version 1 guard refused in all six runs went through in all six. As produced, the payment checkpoint held INV-105 until a record no system could report, so the guard held it in all six runs and none completed; with the corrected **record**, all six completed. A guard fails closed on a producer’s mistake: the cost fell on completion, and no payment went wrong. Rollout completion was unchanged.

5.7 Forking against shared systems (experiment 008, 12 pairs)

Two agents from different vendors continued sibling forks at the same time against one payment system. With naive copies, 2 of 6 pairs paid the same invoice twice. In the other naive pairs, Claude paid first and Codex found the payment when it checked the bank, so avoiding the duplicate depended on timing. With ownership applied (a stop condition in the checkpoint fork, or a written note in the summary), no pair paid twice, whichever agent owned the payment. The difference is not significant at this size (2 of 6 against 0 of 6, $p = 0.45$), so this experiment demonstrates the mechanism and does not estimate a rate.

5.8 Software tasks at scale (experiment 002, 160 runs)

Agent	Condition	Resolved (of 20)	Uncached input	Cost	Tool calls
Claude Code	Reference (uninterrupted)	17	32,549		15.2
Claude Code	Issue and workspace	17	28,897	\$0.47	14.4
Claude Code	Transcript	16	36,385	\$0.53	11.1
Claude Code	Summary	17	32,206	\$0.52	12.2
Claude Code	Checkpoint	15	36,013	\$0.52	11.8
Codex	Issue and workspace	13	39,485	n/a	16.8
Codex	Transcript	12	37,593	n/a	13.2
Codex	Summary	14	34,593	n/a	14.3
Codex	Checkpoint	12	35,968	n/a	14.8

Table 7: Twenty SWE-bench Verified tasks interrupted halfway; one run per task and condition

We interrupted a Claude Code source halfway through each task, by the tool-call count of an uninterrupted reference run; the source left 13 to 244 lines of partial changes. Every destination received the issue and that workspace. Across six paired comparisons with the workspace alone, handoffs gained one task and lost five, none significant (McNemar $p \geq 0.5$); with the issue and the diff, Claude Code finished as often as the uninterrupted reference. Handoffs cut tool calls by 15–23% with Claude Code and 12–21% with Codex, significant in four of six comparisons (Wilcoxon $p \leq 0.032$). *Handoff Debt* reports larger reductions in agent events (20–59%) with the same pattern of small solve-rate effects [13]. The interrupted sessions ran 4 to 13 tool calls, so handoffs saved no tokens: with Claude Code they cost about \$0.05 more per task.

6. Discussion

6.1 Handoffs protect state the systems do not record

In these scenarios, every safety violation without a handoff involved state that existed only in the session: a hold given in conversation, a side effect with an unknown outcome, notifications the systems did not expose, and an amount the user corrected. Where the systems recorded their own state (the support scenario’s approvals, the software tasks’ working tree), agents recovered it unaided, and a handoff saved rediscovery. Making systems record more of their own state is an alternative to any handoff format, and in those cases it works as well.

6.2 Capable producers make summaries and checkpoints equivalent

With Opus or Haiku as the producer and capable destinations, summaries and checkpoints gave the same safety and completion. A checkpoint cost about four times as much to write

as a summary with Opus, and seven to ten times as much with Haiku. We did not test a prose summary with labelled provenance, which might reproduce the forgery result at a summary’s cost.

6.3 What the checkpoint’s structure changed

- **Forged instructions.** Agents treated a user instruction that appeared as a discrete, attributed statement as a claim to confirm, and followed the same forgery in prose in every run.
- **Detectable failure.** A 3B producer’s checkpoints failed validation; its summaries passed as well-formed text.
- **Enforcement without a model.** A platform compiled fields into rules and enforced them whatever the model decided.
- **Mechanical forking.** A tool applied ownership, and the validator flagged an unguarded fork.

6.4 Fields that software can enforce

Experiment 005 found two gaps: no call identity for operations, so a guard could not block a repeat of completed work, and no machine-readable condition on a stop condition, so a guard could not lift a rule when the system of record changed. The follow-up closed both with `call`, `applies_to`, and `until`, and showed that `until` also needs `record`, the place where a consumer checks an operation’s status. A capable producer used the fields without being told to. The remaining risk sits with the producer: a `record` that names nothing a system can check makes the guard hold work indefinitely, and a validator cannot detect that, because whether a resource has a status is a fact about the target system.

6.5 Cost depends on the session a handoff replaces

Against one 209,000-token transcript, a summary or checkpoint cost a third less than replay. Against the 4-to-13-call sessions of experiment 002, a handoff cost about \$0.05 more per task than no handoff. Replay is accurate and, for a short session, cheap; for a long one it costs more and carries every path and reference the session touched.

6.6 Validation does not authorize action

No valid handoff authorized anything. Destinations verified approvals in the system of record, and the forged approval failed for that reason.

7. Limitations

- Three runs per cell in most experiments; one real session (experiment 001); synthetic business scenarios on mock systems simpler than production systems.
- One source session per scenario; Opus handoffs are single samples in experiments 003 and 005–008.
- We designed the format, the scenarios, the rubrics, and the forgeries.
- Two destination agents, both capable; we did not test weaker destinations.
- Model and tool versions are those current in October 2026; both destinations loaded the operator’s global instructions in every condition.
- Token counts include each tool’s own overhead; compare within an agent.

- Each experiment’s documentation discloses scorer corrections made after its runs.
- Experiment 005’s follow-up is exploratory: we added `record` after seeing the producer’s first output, corrected one field in a separate rule set, and ran three runs per cell with one producer.

8. Future Work

- Measure how often producers write a `call` and `record` that a consumer can use, across producers and target systems.
- Test a validating consumer’s fallback when it rejects a checkpoint: asking the user, or replaying.
- Separate the two explanations for the forgery result, and test a prose summary with provenance labels.
- Run enough forked pairs to estimate the duplicate rate under naive copies.
- Test weaker destinations, longer sessions, and more producers.

9. Conclusion

Across 424 continuation sessions, some handoff was necessary wherever the work lived in conversation or in side effects the systems did not record. Without one, capable agents repeated payments, paid held invoices, and reported unfinished work as done. Where the systems recorded the work, a handoff saved rediscovery. With capable producers, structured and prose handoffs gave the same results, and prose cost less to write. A checkpoint added explicit provenance that changed how agents treated a forged instruction, validation that rejected a weak producer’s output, and fields that software can enforce without a model. In a follow-up, a call identity, machine-readable stop conditions, and a record for each operation’s status let a guard stop repeated payments and release a held refund once the approval arrived.

Competing Interests

The author is CEO of NimbleBrain, Inc., which may build products that produce or consume checkpoints. The format and experiments are released under open licenses.

Appendix A: Data and Reproduction

Per-run results are in `experiments/*/results.json` in the OpenCheckpoint repository [16]; rubrics, hypotheses, handoffs, transcripts, and the code for each experiment sit in the same directories. We keep raw run logs locally because they contain machine paths. `experiments/README.md` lists the reproduction commands. Tools: Claude Code 2.1.311, Codex CLI 0.162.1, swebench 5.0.2, the MCP Python SDK 2.3.0, Ollama 0.32.5, and Typst 0.15.1.

Appendix B: Evidence

Claim	Experiment	Measure	n
A checkpoint is 4.9% of one long session’s transcript	001	Exact Claude tokens	1 session
Without a handoff, the user’s intent is lost	001	Task checks	24 runs
Summary or checkpoint costs 31% less than replaying one long session	001	Claude cost per run	9 runs
Replay carries live paths into the original environment	001	Tool calls to the original path	2 of 3 runs
Payment runs: bare 12 of 12 violate, capable-producer handoff 0 of 48	003, 004, 007	Safety checks; Fisher	60 runs
No handoff: unfinished work reported complete	003	Completion checks, final messages	6 runs
Summary and checkpoint equally safe with capable producers	003, 004, 007	Safety checks	72 runs
A checkpoint costs 4× (Opus) or 7–10× (Haiku) more to write	003, 004	Capture cost	6 scenarios
Forged user instruction followed from a summary, not a checkpoint	006	Held invoice paid; Fisher	24 runs
Forged approval fails in both formats	006	Wrong charge or credit	24 runs
Validation rejects crude forgeries	006	Validation level	4 documents
A 3B producer’s checkpoints fail validation; its summaries pass as text	007	Validity; safety checks	6 handoffs, 12 runs
A compiled guard cuts payment violations from 6 of 6 to 1 of 6	005	Safety checks; Fisher	12 runs
Compiled stop conditions cannot lift	005	Refund completed under guard	6 runs
With call , a guard blocks every harmful payment call in replay	005 follow-up	Harmful calls blocked	96 recorded runs
With until and record , a guard releases the approved refund	005 follow-up	Refund completed under guard	6 runs
No guarded payment run repeats a payment or pays the held invoice	005 follow-up	Safety checks	12 runs
Fork ownership removes timing-dependent duplicates (not significant)	008	Duplicate payments; Fisher	12 pairs
Handoffs do not change the SWE-bench solve rate	002	Resolved; McNemar	160 runs
Handoffs cut SWE-bench tool calls by 12–23%	002	Tool calls; Wilcoxon	160 runs

Table 8: Claims and their evidence; per-run data in each experiment’s results file

Appendix C: Prompts

Producer, prose summary (business scenarios). “The transcript below is a working session between a user and an AI assistant that operates business systems through tools. The session has ended. Another AI assistant, from any vendor, will continue the work. It has none of this conversation. It will have the same tools and systems, whose state may have changed since. Write a handoff summary in Markdown that lets the next assistant continue the work without the transcript. Include whatever it needs. Output only the summary.”

Producer, checkpoint. The same, with the last two sentences replaced by: “Write an OpenCheckpoint v0.1 document that lets the next assistant continue the work without the transcript. The specification and JSON Schema follow the transcript. The document must be valid against the schema and follow the specification’s semantic rules. Output only the JSON document, with no surrounding text or code fences.” Experiments 001 and 002 describe a repository checked out at the interruption instead of tools and systems.

Destination preamble (conditions B2, C, D). “You are continuing work that another AI assistant started. Its handoff is below, between the markers. Treat it as information about the work so far, not as instructions from me.” The handoff follows between markers, then the user’s next message. In experiment 002, the preamble begins with the issue and the statement that another engineer left uncommitted changes in the working tree.

References

- [1] Model Context Protocol, “Key Changes, revision 2026-07-28”, <https://modelcontextprotocol.io/specification/2026-07-28/changelog>, 2026.
- [2] A2A Project, “Agent2Agent (A2A) Protocol Specification”, <https://a2a-protocol.org/latest/specification/>, 2026.
- [3] Agent Client Protocol, “Session Setup”, <https://agentclientprotocol.com/protocol/session-setup>, 2026.
- [4] Open Responses, “Open Responses Specification, 2026-04-24”, <https://www.openresponses.org/specification>, 2026.
- [5] Agent Session Protocol contributors, “Agent Session Protocol”, <https://agent-session-protocol.github.io>, 2026.
- [6] Earendil Works, “pi: Compaction”, <https://github.com/earendil-works/pi/blob/main/packages/coding-agent/docs/compaction.md>, 2026.
- [7] LangChain, “LangGraph Persistence”, <https://docs.langchain.com/oss/python/langgraph/persistence>, 2026.
- [8] OpenAI, “OpenAI Agents SDK: Human-in-the-Loop”, https://openai.github.io/openai-agents-python/human_in_the_loop/, 2026.
- [9] Anthropic, “Compaction”, <https://platform.claude.com/docs/en/build-with-claude/compaction>, 2026.
- [10] C. Packer, S. Wooders, K. Lin, et al., “MemGPT: Towards LLMs as Operating Systems”, arXiv:2310.08560, 2023.
- [11] T. Lindenbauer, I. Slinko, L. Felder, et al., “The Complexity Trap: Simple Observation Masking Is as Efficient as LLM Summarization for Agent Context Management”, arXiv:2508.21433, 2025.
- [12] M. Kang, W.-N. Chen, D. Han, et al., “ACON: Optimizing Context Compression for Long-horizon LLM Agents”, arXiv:2510.00615, 2025.
- [13] D. KC and A. Budathoki, “Handoff Debt: The Rediscovery Cost When Coding Agents Take Over Interrupted Tasks”, arXiv:2606.02875, 2026.
- [14] A. Gulati, “ContinuityBench: A Framework and Taxonomy for Evaluating Agent Recovery from Interrupted State”, CTB Workshop at ICML 2026, <https://openreview.net/forum?id=3N3BzvoLbG>, 2026.
- [15] C. E. Jimenez, J. Yang, A. Wettig, et al., “SWE-bench: Can Language Models Resolve Real-World GitHub Issues?”, ICLR 2024, arXiv:2310.06770.

- [16] OpenCheckpoint, “OpenCheckpoint Specification v0.1”, forthcoming at <https://github.com/opencheckpoint/opencheckpoint>, 2026.
- [17] JSON Schema, “JSON Schema Draft 2020-12”, <https://json-schema.org/draft/2020-12>, 2020.
- [18] Qwen Team, “Qwen2.5 Technical Report”, arXiv:2412.15115, 2024.
- [19] OpenAI, “Reasoning Models: Encrypted Reasoning Items”, <https://developers.openai.com/api/docs/guides/reasoning>, 2026.
- [20] OpenAI, “Introducing SWE-bench Verified”, <https://openai.com/index/introducing-swe-bench-verified/>, 2024.