Handover: An Enforced Agent-to-Agent Handoff Document and a Trap-Based Benchmark for Measuring Context Transfer

Harshvardhan Pandey

Independent · brownlittlefish.com · github.com/geekidharsh/handover · harshvardhanpandey@hotmail.com

ABSTRACT

Coding agents lose context at every session boundary: the successor inherits a natural-language summary whose claims are neither verified against the system's current state nor checked for whether they actually prevent the successor from repeating a known failure. We present Handover, a content contract for agent-to-agent handoff artifacts built around one invariant: a cold agent given only the artifact and repository access should reach the same decisions the sender would. Handover pairs a machine-validated header — a version-control freshness anchor, a single unambiguous next action, and a verification command — with a deterministic, LLM-free scorer that separates document *shape* (a 0–100 structural rubric) from *truth* (repository-aware checks: does the anchor commit exist, how far has the codebase drifted, does a declared "done" status actually pass its own verification command). We further introduce a trap-based benchmark: repository scenarios instrumented with deliberately planted failure states — a reverted change that invites re-introduction, a path already tried and abandoned, an ambiguous next step — and a two-layer measurement of whether a handoff artifact steers a receiving agent away from them, first via a deterministic text-presence proxy and second via direct agent-in-the-loop trial. Across three scenarios, a protocol-compliant artifact scores 100/100 and covers all planted traps, while a naive session summary scores below 20/100 and covers none. In a first agent-in-the-loop pilot (one scenario, n=1 per condition), the protocol-compliant artifact was the only one of three conditions — compliant artifact, a misleading summary, and no artifact at all — that avoided reintroducing a previously-reverted client-side pricing bug; the other two planted traps showed no discrimination at this scale, for reasons we report, alongside a methodological hazard the pilot surfaced twice over in constructing trap-based fixtures. We release the protocol specification, the lint/scaffold/bench tooling (zero dependencies, no network, no model in any enforcement path), and the benchmark scenarios under Apache-2.0.

1 Introduction

An agent audited a git branch 288 commits behind the trunk it was meant to review; nearly every issue it reported had already been fixed upstream. A one-line baseline check — how far behind is this branch, really — would have caught it before the audit began. In an unrelated session, a different agent re-resolved an identical squash-merge conflict on six separate branches in a row, silently, until a human finally asked why something routine was taking so long. Neither agent was malfunctioning. What failed in both cases was the handoff: the artifact, or absence of one, that a work session leaves for whichever process picks up the workstream next.

These two incidents are anecdotes, not measurements — we have no population-level data on how often handoffs fail this way, and this paper does not claim to. What they illustrate is a structural gap, not a frequency: any agent that picks up work across a session boundary receives only what the outgoing

process chose to write down, and nothing in that summary is checked against what is actually true by the time it is read. The receiving agent cannot tell whether "the migration is applied" is still true, whether "we tried X and it failed" is recorded anywhere at all, or whether the one correct next step is the first thing it happens to attempt. Positive state — what exists — is recoverable by reading the code. Negative knowledge — what was tried and abandoned, what was deliberately left unbuilt, what was built and reverted — is not, and is exactly the part a natural-language summary is least disciplined about preserving.

Existing agent-to-agent handoff mechanisms address the transport problem: they move state, task specifications, or conversation history between processes (§2). None of the mechanisms we found verify that a transferred claim is still true against live repository state, and none measure whether a transferred artifact actually prevents a receiving agent from repeating a specific, previously-encountered failure. This paper addresses both gaps. Our contributions:

1. A content contract for lossless handoff (§3) built around one invariant, with a machine-validated header carrying the facts that must never be lost or ambiguous.
2. A deterministic, LLM-free scorer (§4) that separates a document's *shape* from its *truth*, the latter checked against live repository state rather than asserted.
3. A trap-based benchmark methodology (§5–7): repository scenarios instrumented with planted failure states, measured first by a deterministic proxy and second by direct agent-in-the-loop trial, plus three scenarios and a real (if narrow) empirical pilot — including two methodological hazards the pilot surfaced and the fixes for them, reported as findings for constructing this class of benchmark, not just patched quietly.
4. An open-source release (protocol specification, CLI tooling, benchmark scenarios) under Apache-2.0, with zero runtime dependencies and no model or network call in any enforcement path.

2 Related Work

Agent handoff as a runtime primitive. The OpenAI Agents SDK's `handoffs` construct, Google's Agent2Agent (A2A) protocol, LangGraph's `Command`-based routing, AutoGen [Wu et al., arXiv:2308.08155], and CrewAI all provide in-flight task or conversation transfer between agent processes. AutoGen in particular formalizes multi-agent conversation as the coordination substrate itself, agents exchanging messages to jointly solve a task. These are delegation envelopes in the sense of §1: a bounded, ephemeral handoff between a parent and a child process. None produce a durable, file-based artifact with a freshness anchor that can be checked against repository state after the fact, and none are scored.

Handoff bundle files. Community projects such as `llm-handoff` and the `session-handoff` skill in `agent-toolkit` use a durable markdown artifact, closer in spirit to this work. Neither validates a freshness anchor against live repository state or runs a verification command; both are unscored.

Deterministic scoring of agent context files. Schliff scores `AGENTS.md` / `CLAUDE.md`-style instruction files 0–100 deterministically, with anti-gaming measures — the closest prior art to our STRUCTURE layer. It scores static instruction quality; it does not check a document against live repository state, and nothing in it executes.

Context and memory persistence. MemGPT [Packer et al., arXiv:2310.08560] and its production successor Letta manage long-running memory within a single continuing agent via virtual context paging between fast and slow memory tiers. This addresses within-agent memory capacity; it is not a discrete, auditable transfer between independent agent processes, and carries no freshness anchor a receiver can check against a repository that has since moved.

Benchmarks. SWE-bench [Jimenez et al., arXiv:2310.06770] established the now-standard genre this work's benchmark belongs to *in spirit* — a repository-grounded task with a checkable outcome. Its methodology is not ours: SWE-bench mines real GitHub issues and their merged pull requests at scale as ground truth, where our benchmark is three hand-authored synthetic scenarios built specifically to contain a planted trap; the two are not comparable in scale or provenance, only in genre. Its direct descendant, SWE Context Bench [Zhu et al., arXiv:2602.08316], is the closest sibling to our benchmark specifically: it measures whether coding agents improve on a task by reusing summarized experience from related prior tasks, across 1,100 base tasks and 376 related tasks spanning 51 repositories. Both measure task resolution or context *reuse* in aggregate; neither plants a specific adversarial trap and measures whether a single transferred artifact steers an agent away from it. The closer methodological analog is adversarial agent-safety benchmarks such as AgentDojo [Debenedetti et al., arXiv:2406.13352], which plants a malicious instruction in an agent's environment and measures whether the agent acts on it. The plant-and-measure structure is shared with our traps; the threat model is not — AgentDojo measures resistance to an adversarial actor's intent, while our traps encode a benign but incomplete or misleading handoff (deception vs. omission). To our knowledge, no existing benchmark applies a planted-trap structure to context-transfer artifacts specifically. Separately, work on negative knowledge as a failure-aware shared memory for autonomous research agents [Wang, arXiv:2606.21024] argues, in a different domain (scientific research agents rather than coding agents), that failed attempts should be curated into typed, explicitly adopted-or-rejected records rather than discarded — a conceptual match for our §2b priority on negative knowledge, applied here to repository handoffs and made machine-checkable rather than curator-mediated.

3 The Handover Protocol

The protocol serves one invariant: *a handoff is lossless if and only if a cold agent — given only the artifact and repository access, and nothing from prior conversation — reaches the same decisions the sender would; it is seamless if and only if the receiver's first action is unambiguous.* Every rule in the protocol exists to protect that sentence.

Two ideas are load-bearing. First, every assertion is a claim, not a fact: a handoff records belief at a moment, and belief drifts from reality, so every state claim must carry either a verify command or an explicit `[belief, unverified]` tag. Second, negative knowledge is the irreducible core: positive state is re-derivable by reading code, but what was tried and abandoned, deliberately left unbuilt, or built and reverted is not — so when a handoff must be cut for length, positive state is cut first.

A Handover document is a strict, machine-validated YAML header followed by a human-authored prose body linted loosely (fenced code and blockquotes masked, so quoted or illustrative text cannot trip the checks meant for live assertions). The header carries seven required fields:

Field	Purpose
`true_at_sha`	the freshness anchor — the commit the claims below were true at
`first_action`	exactly one unambiguous next step
`verify_cmd`	the command that proves the whole state still holds
`status`	`in_progress` or `done` — `done` is refused without a non-empty `verify_cmd`
`supersedes`	explicit pointer when a newer doc replaces an older one
`handoff` , `shape`	workstream identity and interaction shape (handoff / delegation / broadcast)

Nine hard rules (H1–H9) make the protocol's MUSTs testable: self-containment (no reliance on chat history), claims-carry-checks, absolute dates, addressable references (URLs and SHAs, never bare

names), one first action, explicit supersession, a freshness stamp, no silent “done,” and a deterministic, discoverable file location. Each is either gate-checkable or lint-scorable, which is what makes the protocol enforceable rather than aspirational.

4 Deterministic Verification

A score has to say what it is scoring. We split the lint output into two layers with different trust values, because only one of them can be judged from the text alone.

4.1 STRUCTURE — the deterministic 0–100 rubric

Pure text analysis; same input, same score; proves a document is *built to be checkable*, never that a claim is *true*.

Component	Points	What it checks
Header validity	45	all seven required fields present and well-formed; placeholders (`FILL IN` , `TBD`) fail
Negative knowledge	20	5 per §2b category, counted only with real content on its own line — keyword-stuffing scores 0
Section completeness	15	canonical sources, a marked next action, a verification section — detected only on structural lines
Self-containment	12	−6 per distinct H1/injection violation, floored at 0
Prose cleanliness	8	−2 per H3/H4 nit (relative dates, bare issue refs), floored at 0

4.2 VERIFICATION — repository-aware truth checks (opt-in)

Structure cannot know whether `true_at_sha` is real or whether `status: done` is earned. The verification layer can, but only inside the document's own repository, so it is opt-in: `--repo` confirms the anchor commit exists (a fabricated or foreign SHA fails), computes drift via `git rev-list` , and flags a future date; `--verify` additionally runs `verify_cmd` — the only check that converts “done” from belief to fact; `--claims` goes further, running *each* claim row's own verify command and localizing which specific claims no longer hold rather than invalidating the document as a whole. All repository probes run without a shell (`execFileSync` , never string interpolation), so a hostile document's header can never itself execute — only the document's own declared commands run, and only when a human opts in.

5 The Trap-Based Benchmark

Lint measures whether a document is well-formed. It cannot measure whether the document actually stops a cold agent from making a mistake the sender already made. For that we plant deliberate traps.

A trap associates a detectable repository state with (a) a `guard_pattern` — a regular expression checked against a candidate artifact's text, the deterministic proxy layer — and (b) a `detect_fell_in` probe, an executable check run against the repository after an agent has acted, the empirical layer. The deterministic layer answers a necessary-condition question (does the artifact even mention the guard?) cheaply and repeatably; the empirical layer answers the sufficient-condition question (did a real agent actually avoid it?) at the cost of running a real trial.

Three scenarios ship with this release, nine traps total:

Scenario	Traps (by kind)

`reverted-feature`	reverted-do-not-re-add (client-side pricing); tried-and-failed (reused member data model); ambiguous-next-action (migration-vs-route order)
`failed-dependency-upgrade`	tried-and-failed (a dependency major-version bump); reverted-do-not-re-add (a shim that looks dead but isn't); ambiguous-next-action (a misleading first test failure)
`descoped-admin-export`	deliberately-out-of-scope (a compliance-blocked export feature); decision-relitigation (a settled library choice); reverted-do-not-re-add (a client-side cache)

Scenario authoring follows one rule for the deterministic proxy specifically: traps are written from an observed failure mode *before* the compliant artifact is drafted. Deriving `guard_pattern` from an artifact already written reverses the direction of evidence and turns the proxy layer into a self-fulfilling regex. §7 reports two related but distinct measurement-integrity failures in the empirical layer — negative knowledge leaking into fixture source code, and a detection probe keyed to identifiers no trial used — neither of which is a `guard_pattern`-circularity problem; we separate them here rather than treat "author traps first" as a fix for every way a trap-based benchmark can fail.

6 Deterministic Results

Across all three scenarios, a protocol-compliant artifact and a naive "session-notes"-style summary score as follows: STRUCTURE score, header validity, deterministic trap coverage, and a combined *quality* metric equal to the mean of the STRUCTURE score and the trap-coverage percentage (validity is not a separate weighted term; it correlates with STRUCTURE score, since an invalid header already caps that score low):

Scenario	Artifact	Score	Valid	Traps covered	Quality
reverted-feature	good	100/100	yes	3/3	100/100
	bad	16/100	no	0/3	8/100
failed-dependency-upgrade	good	100/100	yes	3/3	100/100
	bad	8/100	no	0/3	4/100
descoped-admin-export	good	100/100	yes	3/3	100/100
	bad	12/100	no	0/3	6/100

This layer is honest about its own limit: it checks that the negative-knowledge warning is *present in the text*, not that a receiving agent obeys it. It is the necessary-condition half of the measurement; §7 is the sufficient-condition half.

7 Empirical Pilot

We report the full trajectory of a first agent-in-the-loop pilot on the `reverted-feature` scenario, including a methodological failure and its fix, rather than only the final result. A benchmark paper that reports a clean result without describing how the measurement was made trustworthy is asking to be taken on faith; we would rather not ask that.

7.1 Setup

We materialized `reverted-feature`'s premise as a real, small TypeScript fixture repository (a stubbed API route, a pending SQL migration, a member data model, a pricing module, a frontend summary component, a test suite) — previously the scenario existed only as artifact text and an abstract premise.

Three fresh copies were prepared, each with its own git history seeded from one identical initial commit, differing only in what a cold agent receives at the repository root: the protocol-compliant artifact (*good*), a realistic misleading "session notes" summary that actively points toward two of the three traps (*bad*), or nothing (*none*, the control). A single model (claude-sonnet-5, via the Claude Agent SDK) was given an identical task description — finish wiring guest checkout end to end — and worked to completion in each condition, one trial per condition (n=1). Each trial ran as a general-purpose coding agent with full read/write/edit/shell tool access to the fixture repository — the same surface a human developer would have via a terminal and editor — and no explicit turn or step cap; a trial ended when the agent itself reported the task complete. We did not record or standardize tool-call counts across conditions, so we cannot report whether one condition took materially more exploration to reach a self-reported completion than another; completion was judged only by the mechanical probes in §7.4, never by the agent's own report. We report this scope precisely because it does not license a general claim; it is a pilot, not a study.

7.2 A confounded first run, and why

The first materialized fixture leaked the answer to every trap directly into its own source comments: the pricing module read, in part, *"Do not add a client-side total — see the guest-checkout handover doc's negative-knowledge section."* All three conditions — including the no-artifact control — avoided every trap, because any agent reading the code saw the negative knowledge regardless of what artifact, if any, it had been given. A second and third flaw compounded it: the task prompt stated the correct migration-then-route order directly, removing the ambiguity the third trap was meant to test; and the "applied" state for the migration was never wired to anything a probe could mechanically check, so that trap's detector could not fire under any outcome.

7.3 The fix

Every "do not X" and "this was reverted because" comment was removed from the fixture's source files; negative knowledge was left to live only in the compliant artifact, never in code an agent can read independent of condition. The migration-apply step was made mechanically real rather than documentary: a small script records applied migrations to a JSON marker, and the data-access module genuinely throws if used before that marker exists — a naive agent that skips the migration now hits a real failure, not a missing comment. The task prompt was rewritten to state the goal without specifying step order. The second scenario's `detect_fell_in` probe, which had matched the substring "nullable" even inside a negated sentence ("do not make user_id nullable"), was corrected to check the target field's actual type signature.

7.4 Result

With the fixed fixture, three fresh trials were run and verified against the actual probes — not the agents' self-reports:

Condition	Trap 1 — client-side pricing	Trap 2 — reused data model	Trap 3 — migration order
good (compliant artifact)	avoided	avoided	avoided
bad (misleading summary)	fell in	avoided	avoided
none (control)	fell in	avoided	avoided

Trap 1 discriminates: the compliant artifact was the only one of three conditions that did not reintroduce a previously-reverted bug — both the misleading summary and no artifact at all led the agent to import

the server's pricing function directly into frontend code, architecturally reintroducing the exact class of bug (client and server able to disagree on tax/rounding) the original revert was about, even though neither trial used the specific function name our first `detect_fell_in` probe checked for. That probe (originally matching the literal identifiers `computeTotalClientSide` / `clientTotal`) was corrected, during verification, to check the real architectural signal — any frontend import of the server pricing module — and re-run against the same completed trials without needing new agent runs.

Traps 2 and 3 show no discrimination at this scale. For trap 2, the fixed fixture's own pre-built data-access module (added to make trap 3 mechanically real) gave every condition an obviously easier correct path than touching the shared data model, independent of which artifact it received — a real methodological tradeoff the fix introduced, reported rather than hidden. For trap 3, this model sequenced the migration before the route in all three conditions regardless of artifact; whether that reflects a capable model not needing the hint for this particular trap, or a probe still not sensitive enough, is genuinely unresolved at n=1 and left as an open question for §9.

8 Discussion and Limitations

- **Small n throughout.** Three scenarios; one pilot scenario; one trial per condition; one model. None of this licenses a general claim about "does Handover work" — it licenses the narrower, real claim that a protocol-compliant artifact produced one measurable, discriminating effect in one trial, and that building a fair trap-based fixture is harder than it first appears.
- **The proxy and the empirical layer can disagree, and did not obviously agree here either.** The deterministic layer only checks text presence in the artifact, not repository outcome; a full accounting of where the two layers agree and disagree, across more trials, is future work rather than something this pilot settled.
- **The §6 deterministic comparison is same-author, same-scorer.** Both the compliant and naive artifacts in every scenario, and the scorer that grades them, were built by this paper's author. The 100-vs-below-20 separation is real and reproducible, but it is closer to a unit test of the scorer against artifacts it was designed to distinguish than to independent evidence about naive summaries a different author would actually write. This is distinct from the next limitation (STRUCTURE being gameable by a dishonest author) — that concerns adversarial gaming; this concerns ordinary authorship bias in constructing the evaluation itself.
- **The trap taxonomy is not independent of the protocol it evaluates.** The four trap kinds (reverted-do-not-re-add, tried-and-failed, ambiguous-next-action, decision-relitigation) are drawn directly from the same four negative-knowledge categories STRUCTURE itself scores (§2b, §4.1). This is a stronger claim than same-author bias above: even a different author's naive summary is being measured against a benchmark whose pass condition is definitionally close to "did the artifact contain the kind of statement the protocol asks for." The §6 separation is therefore partly true by construction, not solely evidence that any effective handoff — regardless of shape — would score well here. An evaluation built from a trap taxonomy independent of the protocol's own categories would be stronger evidence and is not yet done.
- **STRUCTURE is gameable by construction.** A fluent, dishonest author can write four plausible negative-knowledge lines that are false; only the VERIFICATION layer, run by a receiver who chooses to run it, and a human reading the prose, judge truth. This is documented as a design limitation, not discovered as a flaw.
- **Ecosystem and language bias.** All three scenarios are Node/TypeScript-flavored and in English; whether the protocol and benchmark transfer to other ecosystems is untested.
- **Single-model empirical layer.** No comparison across model capability tiers exists yet; whether weaker models show discrimination on traps 2 and 3 where this model did not is an open,

answerable question, not yet answered.

- **Can an agent author a passing Handover document?** Every artifact scored in this paper, compliant or naive, was written by a human. The protocol's premise is agent-to-agent handoff, so the more load-bearing question — whether an agent, not a human, can reliably produce a document that earns a high STRUCTURE score and (harder) actually carries the negative knowledge a receiving agent needs — is untested here.

9 Future Work

A multi-model, multi-repeat empirical study across all three scenarios is the direct next step, sized to actually answer whether traps 2 and 3's null result at n=1 is a fixture property or a capability property. Additional scenarios, in other ecosystems, would test whether the protocol's benefit generalizes past a Node/TypeScript fixture. The per-claim localization mechanism (`--claims`, §4.2) is itself a measurable claim — whether localized drift reporting changes what a receiving agent does differently from whole-document staleness — not yet studied here.

10 Conclusion

Handover is released as a protocol specification, a zero-dependency deterministic CLI (lint, scaffold, and bench), and three benchmark scenarios, all under Apache-2.0. The deterministic layer shows a large, reproducible separation between compliant and naive artifacts across all three scenarios — though, as §8 details, that separation is same-author and partly definitional, closer to a unit test of the scorer than to independent evidence about handoffs a different author would write. The empirical layer, run once, honestly, end to end, shows one real discriminating signal and two null results with documented reasons — and, arguably as valuable, two concrete methodological lessons for anyone building this class of trap-based benchmark: negative knowledge must live only in the artifact under test, and a detection probe keyed to specific literal identifiers will miss the same violation expressed a different way.

References

Debenedetti, E., Zhang, J., Balunović, M., Beurer-Kellner, L., Fischer, M., & Tramèr, F. (2024). AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. arXiv:2406.13352.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2023). SWE-bench: Can Language Models Resolve Real-World GitHub Issues? arXiv:2310.06770.

Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., & Gonzalez, J. E. (2023). MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560.

Wang, H. (2026). Negative Knowledge as Failure-aware Shared Memory for AutoResearch. arXiv:2606.21024.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., et al. (2023). AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155.

Zhu, J., Wu, J., Hu, M., Zhu, S., Pan, J., Shen, W., Yang, Y., Liu, F., Hao, J., Jin, Y., Ho, Q., & Xu, M. (2026). SWE Context Bench: A Benchmark for Context Learning in Coding. arXiv:2602.08316.