Attributing Subagent Work in Coding-Agent Protocol Adapters

Rajasekhar Josyula
Independent Researcher
rajasekhar.josyula@gmail.com

Lakshmi Sravya Vedantham
Independent Researcher
lakshmisravya.vedantham@gmail.com

***Abstract*—Coding-agent adapters can report a subagent's file edits against the wrong request. We find this defect in the open-source Codex and Claude adapters for the Agent Client Protocol (ACP): work dispatched for request A appears in request B's file-change report. We define an origin-attribution invariant and build TURNSTILE, a checker that runs real adapter builds against scripted providers and explores event orders in hand-written scenarios. The original adapters misattribute in 581 of 956 Codex and 39 of 111 Claude schedules; staged runs with real providers reproduce both defects. Binding children when they start still fails when a start or restart arrives during a later request. Dispatch-time binding removes the tested Codex misattributions but can omit legitimate work. For the typed Codex 0.160.1 interface, we give a source-grounded indistinguishability witness: under stated lifecycle and observation conditions, identical adapter input requires different correct reports. Truthful lineage available before reporting suffices for exact attribution. A patch using the newer `rootTurnId` field prevents misattribution across 2,423 schedules, but omits own evidence in 34 registration-race cases; a separate control exposes edit–revert omissions. A source screen of eight additional ACP implementations finds no further confirmed instance. The findings isolate a reporting failure at the adapter boundary and show why request identifiers alone do not establish the origin of delegated work.**

***Index Terms*—coding agents, Agent Client Protocol, subagents, provenance, observability, systematic concurrency testing**

I. INTRODUCTION

Editors drive coding agents through protocol adapters. The Agent Client Protocol (ACP) [1] lets an IDE talk to any compliant agent, and adapters wrap vendor runtimes (*providers*) such as Codex and Claude Code. Some adapters also derive per-request outputs, such as the files one request changed; the Codex and Claude adapters do so for JetBrains AIR [2].

Agents now delegate to subagents, and some providers let a subagent outlive the turn that started it. Both adapters we study keep per-request bookkeeping, created for each request and disposed when it returns, so a late event from an older subagent can arrive after the state for its originating request is gone (Fig. 1). Credited to the request active on arrival, it puts one request's changes in another's report, and the client cannot tell: the report has the right request id and the wrong contents.

The inspected file-report suites at the pinned adapter commits do not send a second request while a subagent of the first still runs, and a hand-picked failing order invites the objection that providers never produce it. We therefore combine source inspection, systematic exploration under explicit provider models, and staged live runs. The live runs establish reachability; schedule counts describe the explored scenarios, not failure probabilities in deployed systems. We contribute:

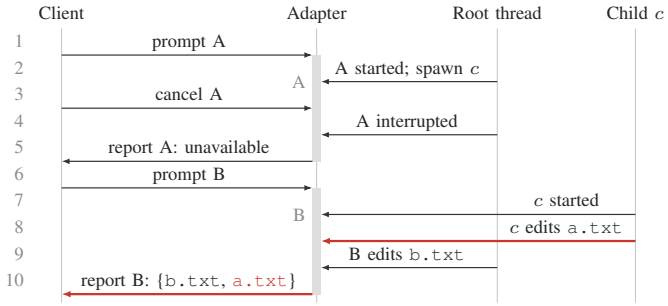


Fig. 1. Minimal failing cancel schedule (abridged; no preemption). The original adapter (step 8) and the start-time patch (step 7) credit c's edit to B; the dispatch-time patch binds c to A via the spawn item (step 2).

1) **Concrete adapter failures and repair tradeoffs.** We find one defect class in Codex ACP and Claude ACP, reproduce it with real providers, and compare start-time, dispatch-time, and explicit-lineage binding (Sec. V). A source screen of eight further implementations characterizes attribution mechanisms in the inspected sample (Sec. VII).
2) **A controlled attribution checker.** TURNSTILE applies systematic concurrency testing [3] at the provider–adapter boundary, comparing actual reports with a hidden origin ledger under explicit scheduling constraints (Sec. IV).
3) **A precise property and information boundary.** We define report-relative soundness and completeness (Sec. III), construct a Codex 0.160.1 typed-wire ambiguity under stated conditions, and state what lineage must provide to resolve it (Sec. VI).

Propagating causal context is established practice [4], [5]. Our contribution is its concrete failure at a coding-agent reporting boundary, including the distinction between information absent from an interface and information an adapter receives but loses.

II. BACKGROUND

ACP is a JSON-RPC protocol between a client and a coding agent (not the Agent Communication Protocol).

Prompt turns. A client sends `session/prompt`; the agent streams `session/updates` and answers with a stop reason. After `session/cancel` the agent may still send updates, but only before it answers [1]. ACP's subagent proposal lets "a background child that the cancelled work did not wait on" continue and scopes its output to a child session [6], so a child event during a later request is not by itself a protocol violation. A *request* is one `session/prompt` call (A, B, ...); a *turn* is provider-side work: one *root turn* per request, plus *child turns* run by subagents (*children*).

Derived reports. Under AIR [2] the client asks for an *agent file-change report* with a fresh `requestId`, and the adapter returns the paths changed by that request before answering it; a `declaredComplete` flag can admit omissions. AIR does not say who owns paths changed by children.

III. Origin Attribution

Each turn has an *origin*: the request whose processing dispatched it, inherited by nested children. Informally, a report for r should list only paths evidenced by origin-r work. It should include all such paths observed before publication, or explicitly report possible incompleteness. A late write by A's child may be legitimate; listing it as B's violates this dispatch-origin contract. This policy follows the initiating dispatch, rather than every request that may later influence a child. AIR does not currently prescribe this choice. Violations are *turn-attribution bugs*; the usual one, *arrival-time attribution*, takes the request active *now* as the origin.

Model. Requests $r = A, B, \ldots$ arrive in order, with root turn ρ_r for request r. Each child turn w has one *dispatch* (a spawn or resume, or Codex *trigger* mail, Sec. VI), sent by turn $\delta(w)$, its recorded parent on Codex. Queue-only mail, mail taken by an already running turn, and interrupts are not dispatches. Executions without a unique dispatcher for each child turn are outside the model. Origins are recursive:

$$o(\rho_r) = r, \qquad o(w) = o(\delta(w)).$$

Thus a child thread reused by B can run a new turn of origin B even if A created the thread. Thread identity alone is not request identity.

Thread streams are FIFO, unordered across threads and unboundedly delayed; *legal* executions obey the provider. A deterministic adapter maps messages and arrival times to provider requests and, after ρ_r ends, eventually to one report (R_r, c_r). Here R_r is a set of paths and c_r asserts completeness. A report is *available* unless the client cancels r or ρ_r is interrupted without the adapter asking.

Let $W_r(x)$ be the turns from which evidence mentioning path x reached the adapter before it published R_r. Then

$$Own_r = \{x : \exists w \in W_r(x),\ o(w) = r\}.$$

The cutoff is the report's publication, not the eventual end of every child. This definition concerns observed edit evidence, rather than a final filesystem snapshot. In particular, a later revert does not erase earlier evidence from $W_r(x)$.

Definition 1: An available report is *sound* (the *origin-attribution invariant*) if each $x \in R_r$ has a $w \in W_r(x)$ with $o(w) = r$, *complete* if $Own_r \subseteq R_r$, and *safe* if sound with c_r false unless complete. An adapter is *exact* (*safe*) if, in all legal executions, its available reports are sound and complete (safe). Omission ($R_r = \emptyset$, c_r false) is safe; both adapters clear c_r (`declaredComplete`), so their safety is soundness.

Two distinctions matter in interpreting results. First, an incomplete report may safely omit work but must not assign another request's work to this one. Second, a cancelled request may have no available report at all. Exactness of available reports therefore does not imply that every child edit appears somewhere. We report these coverage gaps separately where measured.

IV. Turnstile: Controlled Provider Schedules

Provider substitution. Both adapters start their provider as a child process whose path an environment variable overrides; Turnstile points it at a relay that forwards every line to the explorer. A *provider stub* per wire protocol (Codex app-server JSON-RPC, Claude CLI stream-json) renders abstract scenario events (*root turn started*, *spawn child*, *child edit*, ...) into concrete messages; a minimal ACP client sends prompts and cancels and captures reports.

Schedule constraints. A scenario is a set of items (provider events and client actions), each on a FIFO *stream* (one per provider thread, plus the client's), with optional happens-before edges and *guards* that enable an item only after the adapter has sent a given message. For the Codex defect scenarios, the rules combine guarantees confirmed in the pinned provider's source with additional ordering restrictions. These restrictions narrow the explored orders; the provider model remains a source-derived abstraction, not a proof of the full runtime. Guarantees: events within a thread are FIFO; `turn/interrupt` reaches only the root thread, so children keep emitting after it; nothing orders events across threads; and a parent waits for children only through a wait tool. Restrictions: a child's events follow the root's spawn item, B's turn follows A's completion, a completion report follows its own completion, and a few similar causal orders. One synthetic robustness control removes an expected dispatch item; separate registration-race controls relax the spawn-first restriction, since cross-thread delivery can put a child start before its announcement. The theorem pair renders a source argument as a shared observable trace, rather than executing the provider to generate both hidden histories. For Claude we could not check provider source; the stub encodes the adapter's own assumptions (background-subagent frames share the CLI's single ordered stream, are tagged with their launching tool call, and may interleave with later root turns), each seen in live Claude runs.

A *schedule* is one total order of a scenario's 10 to 24 items respecting these constraints. The explorer runs each schedule in a fresh adapter process, delivering one item at a time and waiting for 150 ms of silence, a heuristic for quiescence; it explores message orders, not races inside the adapter. A

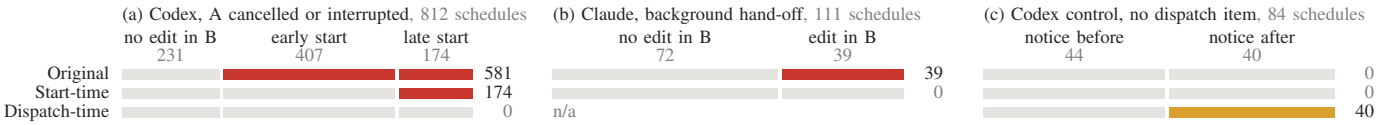


Fig. 2. Every schedule of three scenario groups, split (to scale) by when the child's evidence reaches the adapter; one bar per build (misattribution, missing output, clean), violating schedules counted at right. *In B*: after B's prompt (Codex) or its echo by the CLI (Claude), before B's report. *Early/late start*: edit in B, child started before/after B's prompt. *Notice before/after*: Codex's completion notice for the child's new turn came before/after B's report.

preemption is a switch away from a provider stream that still has a deliverable item [3]; it measures distance, not likelihood.

Oracle. Each scenario carries a ledger, hidden from the adapter, mapping every evidence path to its origin. Each scenario assigns a path to only one request; shared-file ownership is outside this experimental matrix. The frozen oracle reports a *misattribution* when a report lists a path of another request's origin, and *missing expected output* when it omits a required own path whose evidence arrived before it. It also flags missing, unexpected, early or late reports, false completeness claims and adapter exits; an omission in a report marked incomplete is only logged. A schedule that hangs, errors or diverges on re-run is *inconclusive*, never clean. Violations are re-run three times and count only if they fail identically. Four broken builds (no B report, dropped child paths, early report, crash) were flagged on every affected schedule.

Controls. Fixes must not win by dropping work, so controls cover late child output that is legitimately B's (a new child; new work or a resume for A's child) or must stay out of B's report (unregistered thread, capability off, late diff with no B).

V. Results

We pinned codex-acp at commit `082be156` and claude-agent-acp at `ecaa2c93`; the Codex stub follows rust-v0.160.1 (`d27764b8`), the version the pinned codex-acp locks [7]. Table I comes from one final run of every build, with no divergent or inconclusive schedule and no flaky one except as noted. In the groups of Fig. 2, an outcome depends only on when the child's evidence arrives relative to B.

A. Original Adapters

Codex ACP. The event handler assigns a child turn it has not seen to the *current* root turn. Each request builds a new handler whose disposal clears these bindings, but the session's child subscriptions survive and route into whichever request is current. We found no failure in our normal-completion scenarios, as under V2 the adapter holds A open until the child's `turn/completed`, up to a 10-minute timeout [8].

Claude ACP. The adapter records child edits into the session's mutable `activeTurn`. When the CLI ends A's turn while A's background subagent runs, the adapter holds A open [9]; a B sent meanwhile becomes active when the CLI echoes it, and a subagent edit after that echo lands in B's report. Since 2026-10-22 (`9b3a5c8`) an AIR client may decline the hold; A then settles at its result, and by source reading a later child edit is dropped before B's echo and credited to B after it.

B. Repair Tradeoffs

Start-time binding. Our first fix, for both adapters, binds each child to the root turn current when it starts. Both patches pass our reproduction tests and upstream's file-report tests (28 and 34 tests in total). The fix removes every hand-off violation, but A's child edit then appears in no report in 66 of the 111 schedules (27 under the original). These edits arrive after A's report cutoff, so Table I records no omission of prior evidence even though global coverage falls. When Claude Code restarts a stopped child on its own during B it still misattributes in 54 of 150 schedules (original 91), because the restart arrives inside B's turn. On Codex a child's `turn/started` can itself arrive after B's prompt, binding the child to B (Fig. 1): start-time binding is still an arrival-time proxy.

Dispatch-time binding (Codex). The second fix binds a child turn through the item that dispatched it: a spawn or follow-up item names the child's thread under the dispatching turn's id. As the wire does not say which dispatch started which child turn, the patch pairs them in arrival order and grades each binding *named* (Codex's completion notice names the parent turn, or the turn is a spawned thread's first), *paired* (nothing contradicts it), or *uncertain*; uncertain work is left out, and the report says so. The patch removes every misattribution in Table I, including where an A-era sibling or grandchild of A's child follows it up while B does, though it then omits B's own child edit in 226 of 996 schedules (start-time misattributes in 620). All builds pass a 254-schedule control in which B gives A's child work through a follow-up item. A control with *no* such dispatch item, although real Codex emits one, fails in 40 of 84 schedules (Fig. 2c): the only origin evidence is the completion notice naming B's turn, and when it arrives after B's report the patch omits the edit and flags the report. A fresh run of the patched adapter's suite records 1,815 passed, 36 skipped, and one known failure: the dispatch-less recall test. Codex-binary-dependent tests are excluded.

Lineage binding (Codex 0.162.0+). We extended the Codex stub to put `rootTurnId` on every turn as 0.162.0 does, by our source reading (a child turn names the root turn of the request whose turn dispatched it), and bound each child turn to that request on its `turn/started` (Sec. VI). Table II collects the lineage results. On this extended stub, the patch has no misattribution or omission from available reports in 2,389 schedules: 956 seed schedules, 401 controls, 996 sibling/grandchild schedules, and 36 cancel-race schedules, where dispatch-time binding omits B's own child edit in 226 of 996 and 40 of 84. A separate registration-race control exposes a gap: when the child's `turn/started` precedes the item

TABLE I

TURNSTILE RESULTS PER BUILD: SCHEDULES EXPLORED (EXHAUSTIVE EXCEPT NOTE C) AND SCHEDULES WHOSE REPORT LISTS ANOTHER REQUEST'S EDIT OR OMITS OWN EVIDENCE FROM AN AVAILABLE REPORT. ORIG., START, AND DISP. DENOTE ORIGINAL, START-TIME, AND DISPATCH-TIME BUILDS; EACH CELL COUNTS SCHEDULES, NOT INDEPENDENT DEFECTS.

		Misattributing			Omitting own edit		
Scenario	Sched.	Orig.	Start	Disp.	Orig.	Start	Disp.
Codex							
A cancelled by client	504	341	87	0	0	0	0
A interrupted[a]	308	240	87	0	0	0	0
A completed	144	0	0	0	0	0	0
sibling or grandchild[c]	996	620	620	0	0	0	226
6 controls	401	0	0	0	0	0	40[b]
Thm. 1 pair, E	952	792	792	0	0	0	0
Thm. 1 pair, E'	952	0	0	0	0	0	792[b]
Claude							
background child hand-off	111	39	0	n/a	0	0	n/a
CLI restarts the child in B	150	91	54	n/a	0	0	n/a
2 controls	30	0	0	n/a	0	0	n/a

The lineage build is evaluated separately in the text. Codex runs use multi-agent V2; the first three rows each cover three delivery modes (first diff, changed snapshot, identical replay). CLI restart: Claude Code restarts the stopped background child on its own during B, as seen live; the edit is A's. [a]Interrupt issued by the server or another client. [b]Required paths, a violation (missing expected output, Fig. 2); other omissions are allowed, as both adapters mark every report possibly incomplete. [c]Four scenarios, every schedule without preemption; within one preemption Disp. misattributes in none of 22,074 and omits in 6,760. Every misattribution reproduced 3 of 3 (two row-E Start ones after a stalled re-run under load).

TABLE II

LINEAGE-BUILD RESULTS ON THE EXTENDED CODEX STUB. THE FIRST TWO ROWS TOTAL 2,423 SCHEDULES. THE UNRESTRICTED VARIANT OVERLAPS THE 34-CASE CONTROL AND IS NOT ADDED TO THAT TOTAL.

Cases	Sched.	Misattr.	Omit	Inconcl.
Extended matrix	2,389	0	0[a]	0
Completed registration race	34	0	34	0
Unrestricted race variant	56	0	52	4
Edit–revert audit order	1	0	1	0

Omit counts schedules missing own evidence from an available report. [a]In 36 matrix schedules A is cancelled and its child edit appears nowhere; the unavailable A report is outside this completeness check. The edit–revert row is one diagnostic order, not an exhaustive scenario.

announcing the child, codex-acp drops the start notification. With A completing normally, the patch then omits A's child edit in 34 of 34 schedules. This is safe but incomplete. In the 36 cancel-race schedules above, A's report is unavailable; the same lost work is outside report-relative completeness. The unrestricted completed-race variant explores 56 schedules per build: both builds omit early evidence in 18, the lineage build additionally omits it in the 34 announced cases, and four are inconclusive because the run hangs. These runs are retained and are not counted as clean. The 2,389-schedule matrix therefore establishes neither total retention nor correctness on all registration orders. The original adapter is unchanged by the field (47 of 84, 38 of 56).

C. Live Provider Runs

The live driver creates a fresh workspace, stages A's child to wait about 25 seconds before editing its file, and overlaps B's roughly 45-second wait with that edit. File names and prompts identify the intended owner. Each run records both sides of the adapter. These waits seek a particular interleaving; the runs are not a random sample of normal use.

In five staged cancel runs each on Codex 0.160.1 and 0.165.0 the original build put A's child edit in B's report every time (10 of 10); the start-time and dispatch-time builds (0.160.1) and the lineage build (0.165.0) did so in 0 of 5 each, and the lineage build attributed two follow-up runs correctly, each child turn's `rootTurnId` naming its dispatching request. In five queued runs with Claude Code 2.1.312 the original adapter misattributed every time and the patch never, A's child edit landing in no report. Earlier single runs (Claude Code 2.1.309) saw the CLI restart the subagent during B, where the patch also failed; TURNSTILE reproduces that order (Table I). codex-acp main (`3f6d371`, Codex 0.165.0) misattributed in 5 of 5 such runs and claude-agent-acp main (`4b4b2c4`) in 2 of 2. These runs show that the orders occur, not how often.

D. Version Checks

As of October 29, 2026, both inspected main branches retain arrival-time binding: codex-acp `3f6d371`, which pins Codex 0.165.0, and claude-agent-acp `4b4b2c4`. Repeating the seed matrix reproduces Table I's first three Codex rows and Claude hand-off row exactly: 581 of 956 and 39 of 111 misattributing schedules. Claude's newer `backgroundSubagents` capability lets an AIR client skip the held-turn wait. With that capability, the main build still misattributes in 39 of 111 schedules, while A's report loses its child edit in 30 additional schedules. Thus opting out of the wait changes retention but does not repair the observed attribution path. These claims are tied to the cited snapshots; they do not predict subsequent upstream changes.

E. Validation and Sensitivity

The original enumerations used a ledger-based oracle that detects foreign ownership but treats unknown paths as informational and does not separately reject an own path reported before its evidence. An independent strict audit checks both cases. Re-evaluating 64,078 archived non-native schedule records, including historical and supplemental runs, reproduces every saved verdict; there are no unknown reported paths and no own paths listed before evidence. These audit records are not a new experimental denominator. The revised checker treats both cases as soundness failures, with focused regression cases; the original oracle and its experiment hashes remain preserved.

For settling sensitivity, we replayed 11 build/order pairs at 50, 150, and 500 ms, covering five failure classes: late child start, missing dispatch, Claude hand-off, Claude automatic restart, and the lineage registration race. All 33 executions completed and each pair retained its verdict at all three delays. No archived schedule reached the settling cap. This is a selected robustness check, not exhaustive exploration at three delays.

A further edit–revert control exposes a semantic limit of the existing collector: both the original Codex build and the lineage patch omit a path whose earlier nonempty diff is replaced by an empty diff. This is safe but incomplete under our observed-evidence definition; it may be correct for a net-change report. It is not another cross-request attribution defect.

VI. What the Typed Interface Can Establish

The following result concerns exact attribution under the dispatch-origin contract of Sec. III. It is a source-grounded application of the standard two-execution indistinguishability argument [5], [10]. It does not rule out safe omission, changing the ownership contract, or adapters that control child execution differently.

Provider premises. For Codex 0.160.1 multi-agent V2 with no hooks, our source reading establishes P1–P5 on the witness path [7]; P6 is a modeling assumption.

- P1. `followup_task` and `send_message` emit the same typed `subAgentActivity` item under their caller's turn. Only the former sends trigger mail, carrying the caller's turn as parent and its start settings.
- P2. Mail can remain queued while a child executes a long tool call; it need not be consumed before the child's next model call. The witness uses neither `wait` nor `sleep`, which react to mailbox activity.
- P3. An interrupt aborts only its target's turn. After `interrupt_agent`, waiting trigger mail starts a new turn with a fresh random id. It takes all queued mail, the triggers' common parent, and the last trigger's settings.
- P4. Apart from the interaction items in P1 and the completion activity in P5, the following typed sources do not expose the mail-tool choice, trigger flags, or parent/root *turn* ids: notifications; replies to turn, goal, and thread-opening requests; `thread/read`, `thread/list`, `thread/turns/list`, and `thread/items/list`; and background-terminal lists. Raw events and rollout files can expose the missing information and are excluded.
- P5. For the witness's child turns, interrupted without a preceding terminal error, no completion mail or parent-naming completion activity is sent. Normal completion could reveal the recorded parent.
- P6. Model API replies, including tool calls and token usage, random ids, and scheduling delays are adversarial choices, subject to provider ordering guarantees. Thus different model inputs may produce equal observable outputs; this is not an assumption about their probability.

	Same in E and E'	Hidden in E	Hidden in E'
1	ρ_A spawns c; c's turn v runs		
2	ρ_A mails c	trigger	plain
3	client cancels A; ρ_A ends		
4	B's prompt is forwarded; ρ_B starts		
5	ρ_B mails c	plain	trigger
6	ρ_B interrupts c; v ends		
7	new turn u takes both mails, edits x	parent ρ_A	parent ρ_B
8	ρ_B interrupts u; ρ_B ends; report due		
	origin of u, so owner of x	A	B

Fig. 3. The witness pair of Theorem 1: identical typed observations, but a different trigger mail and therefore a different dispatch parent for u and origin for x.

Adapter class. The deterministic adapter must eventually emit the reports required by Sec. III. For a client that only prompts and cancels, require:

- Q1. After a client cancel during ρ_r, it answers r and forwards the next prompt without waiting for r's children.
- Q2. It runs the requests on one live, hook-free thread, permits the V2 agent tools and the witness's child edits to run without user approval, and acts on children only through reads and replies to their server requests.
- Q3. It receives all typed notifications and obtains thread data only through P4's listed sources.

These conditions exclude, for example, fresh threads per prompt, disabling delegation, stopping children on cancel, and consulting raw logs. The pinned codex-acp follows this path with V2 enabled, native subagent reporting negotiated, no hooks or configuration disabling the tools, and a working sandbox permitting the edit. Its cancel must interrupt the root before normal completion; otherwise it waits for children. Notification receipt here means receipt at the transport, including messages the adapter later discards.

Witness. In E, ρ_A spawns child c. Its first turn v enters a long tool call, and ρ_A sends it a `followup_task`. The client cancels A during ρ_A and sends B after A's answer. ρ_B sends c a `send_message`, then interrupts v. The new turn u takes both mails but has parent ρ_A, the sole trigger's sender, so $o(u) = A$. Only u edits x; its diff reaches the adapter before ρ_B interrupts u and ends without edits. In E', swap the two mail tools: the sole trigger is B's and $o(u) = B$. An interrupt is not a dispatch under our contract; this is why the request interrupting v does not determine u's origin.

Lemma 1: Under P1–P6, for an adapter satisfying Q1–Q3 on the witness path, E and E' can be scheduled legally with identical adapter inputs.

Proof: P1–P3 and Q2 permit the spawn, queued mail, interrupts, and edit; Q1 permits B while v still runs. Delay any adapter requests that could end v or ρ_B until after u is interrupted (P6); child-directed actions are restricted by Q2. This preserves request ordering: B's `turn/start` is processed before later root requests. Inductively, equal inputs cause equal adapter requests. The two executions differ only in the mail-tool names, trigger flags and headers, and u's parent, root, and inherited settings. P4 and Q3 hide these differences; u invokes no MCP tool, and its model replies can agree despite different settings (P6). Choose equal ids, timestamps, texts, and token counts. Neither v nor u produces the parent-revealing completion activity (P5). After their interruption, waiting or making the permitted reads does not reveal the difference. ∎

Theorem 1 (No exact attribution on the typed Codex 0.160.1 wire): Under P1–P6, every adapter satisfying Q1–Q3 on the witness path is unsound in E or incomplete in E' under dispatch origins. A safe adapter must set c_B false in E, even though Own_B is empty.

Proof: B is not cancelled and no external actor interrupts ρ_B, so the model requires an available report. Lemma 1 makes this the same (R_B, c_B) in both executions. If $x \in R_B$, it is unsound in E: only origin-A turn u supplied evidence for x. Otherwise it omits $x \in Own_B$ in E'. Safety therefore forces c_B false in E' and, by indistinguishability, in E. ∎

The schedule may depend on the adapter; the theorem does not assert that every schedule fails. TURNSTILE renders the pair as the same messages with two ledgers, testing how each build resolves the ambiguity rather than independently proving its provider feasibility. In 792 of 952 schedules, u's diff precedes B's report: the original and start-time builds list x, failing in E, while the dispatch-time build omits it, missing B's edit in E'. Other evaluated scenarios do not start a child turn on two requests' mail, so this theorem explains none of their failures.

Information sufficient for exact attribution. Let M_r contain all edit-evidence messages received before R_r. Using only messages received by that cutoff, a lineage rule λ maps each $m \in M_r$ to a request or $\perp$ (unresolved). It is *truthful* if every resolved value is the sender turn's origin, and *timely* if every message resolves by the cutoff, in all executions under consideration. The ideal collector T_λ retains this evidence and reports x for r exactly when some message for x resolves to r. It sets c_r true exactly when no message remains unresolved.

Proposition 1 (Lineage suffices): If λ is truthful, T_λ is safe. If λ is also timely, T_λ is exact.

Proof: Truthfulness supplies an origin-r witness for every reported path. If c_r is true, every message witnessing $x \in Own_r$ resolves, necessarily to r, and thus x is reported. Timeliness resolves every message, giving completeness for every available report. ∎

Codex 0.162.0 exposes `rootTurnId` on turns and their lifecycle notifications [11]. It separates E from E': u names A's root in one execution and B's in the other. More generally, our source reading gives a truthful, timely rule under these conditions: V2 turn ids are distinct; each request starts a turn that is its own root; every report follows the `turn/start` reply for every request forwarded before that report and precedes the next prompt's forwarding; and every non-root turn in an evidence sender's dispatch chain was started by spawn or trigger mail from one turn. Map each evidence message through its turn's `turn/started rootTurnId` to the corresponding request. Spawn and trigger mail inherit the dispatcher's root, so induction on the dispatch chain proves truthfulness. FIFO delivery puts each sender's start before its evidence; the report ordering ensures its request's root is already known, proving timeliness. This statement assumes the collector retains those messages.

The experimental lineage patch is not the ideal collector: it preserves the adapter's child-registration filter and latest-diff storage. It can therefore omit received evidence, including an edit later reverted to an empty diff. Its finite evaluation supports the ownership rule, not unconditional patch exactness. The Claude patch also binds by observation time, at launch or resume registration, and is not a truthful lineage rule when earlier work registers a child during a later request.

VII. Source Screen of ACP Implementations

Writing a provider stub is the main per-adapter cost, so we read the event-to-report path of eight more ACP implementations at pinned main-branch commits and classified how each attributes per-turn output. Candidates came from the ACP agents list and registry; closed-source implementations could not be inspected, and a Gemini fork was excluded to avoid counting it as an independent implementation. The screen concerns the inspected output path, not every behavior of each system (Table III). Two of the eight, fast-agent and OpenCode, we also ran under TURNSTILE through a scripted model API (192 schedules within one preemption, three replays each): no cross-request attribution of usage.

Child evidence crossing a request's attribution window creates the risk; background execution alone does not establish a failure. OpenCode lets children survive a cancelled turn but hands them to a muted consumer that cannot write B's report. Guarded implementations carry origin explicitly (Muse), keep state unreachable across requests (Gemini CLI), or prevent arrival during B by rejecting or serializing prompts or draining the previous turn.

VIII. Limitations and Threats to Validity

Scope and frequency. Exploration is exhaustive only within hand-written scenarios shaped by defects source review had found, with a preemption bound for four Codex scenarios and the native-agent screen. We did not measure how often providers produce these orders (the live workloads deliberately stage overlap) and found no open-source client that consumes the report, so we claim neither a frequency nor harm to users.

TABLE III
SOURCE SCREEN AT THE INSPECTED OCTOBER 22, 2026 SNAPSHOTS. THE EIGHT ADDITIONAL IMPLEMENTATIONS ARE A PURPOSIVE SAMPLE. A GUARD RECORDS A SOURCE-LEVEL EXPLANATION FOR FINDING NO REACHABLE FAILURE; IT IS NOT A PROOF OF CORRECTNESS.

Implementation	Derived output	Attribution path or observed guard	Commit
Codex ACP	AIR file paths	Arrival-time binding; child subscriptions survive request state.	`082be156`
Claude ACP	AIR file paths	Child edits enter the mutable active turn after a hand-off.	`ecaa2c93`
Muse ACP	AIR file paths	Explicit source turn id must match an in-flight record.	`d3be20c6`
Pi ACP	Usage and files	Arrival window, protected by prompt settlement and stream draining.	`ffea89d2`
OpenCode	Usage	Message-id anchor, one prompt at a time, muted background consumer.	`d93568f3`
Gemini CLI	Usage	Accumulators local to each prompt invocation.	`451a02ff`
Goose	Usage	One run per session; cancellation drains the previous stream.	`b89ab563`
fast-agent	Usage	Prompt lock; child usage merged before parent tool returns.	`5c4453b8`
Mistral Vibe	Session total	No per-request usage window to test under this property.	`63d0e8f1`
Cline	None in scope	No per-turn file report; usage events are not reported.	`590785ef`

Model limits. TURNSTILE does not model timers (including codex-acp's 10-minute wait), Codex's V1 protocol, or Claude cancellation. Newer Codex versions are covered only by the `rootTurnId` stub extension and live examples, not by complete version-specific provider models. In the cancel and interrupt scenarios the child's turn ends 'interrupted', though a root interrupt does not stop children. The proofs rest on our reading of Codex and codex-acp source, not a mechanized model, and the Claude model is based on adapter assumptions checked against staged live runs. Its stream cadence and handling of completion notifications are abstractions, not a complete model of the closed-source CLI.

Contract. Both adapters and their tests aggregate child-turn paths, which the AIR documentation leaves unassigned. The invariant is our reading and has not yet been raised with the maintainers.

Oracle and path semantics. The scenario ledger assigns each path one origin, while Definition 1 permits evidence from multiple origins. The matrix therefore does not evaluate two requests editing the same file. Diff reversion is covered only by the separate audit control, and oversized diffs are untested. The lineage stub constructs its fields from the declared scenario origin; it tests whether the adapter uses truthful lineage, not whether the real provider always supplies it. That latter claim rests on source analysis under stated conditions and the live examples.

Fixes. All patches are experimental. Besides the gaps of Sec. VI, the Codex start-time and `rootTurnId` patches drop child turns started before their spawn item. The dispatch-time patch can still misattribute a child turn without an observed dispatch, including a previously unobserved sleeping child woken by mail; that wake-up behavior is outside the modeled scenarios.

Independence. Violations within an adapter share one mechanism, so counts measure the failing region of selected scenarios, not independent bugs. Staged provider runs and purposively selected source implementations do not establish ecosystem prevalence.

IX. RELATED WORK

Causal context and attribution. X-Trace propagates a task identifier through causally related requests [4]. W3C Trace Context standardizes trace and parent identifiers across component boundaries [12]. Mishra and Sharad [5] show that delegation membership is not identifiable from telemetry lacking that membership, and bind it at execution time. The general information-loss argument and the principle of carrying origin are therefore not new. Our witness concerns a particular typed Codex interface, its dispatch semantics, and adapters required to remain responsive after cancellation. Our experiments show the resulting error in actual per-request file reports and measure repairs that trade misattribution for omission. An external tracing layer can resolve the ambiguity only if it observes or propagates information excluded by the theorem's interface conditions.

Testing agent infrastructure. AgentThread model-checks protocol security and composition properties and replays counterexamples against SDKs and reference servers [13]. TURNSTILE instead varies provider-event order around an origin invariant on derived reports. Both approaches combine abstraction with implementation-level evidence; our claim is not that testing implementations is new. CHESS [3] supplies the established systematic concurrency-testing approach; our scheduler controls protocol messages and uses timed settling, rather than controlling the adapter's internal thread scheduler.

Effects and lifecycle failures. Stop Means Stop [14], Lingering Authority [15], and Li [16] study enforcement, revocation, or duplicate effects. We ask which request a permitted effect is reported against. Upstream work already addresses background-subagent lifecycle and held-turn settlement [9]; the defect here is specifically a foreign path in an AIR report with the correct outer request id. Bug studies of agent frameworks [17] and coding agents [18] mine issue trackers. MAST [19] classifies multi-agent conversation failures. These broader studies motivate infrastructure checks, but do not substitute for an origin oracle at this reporting boundary.

X. DISCUSSION AND CONCLUSION

Arrival time does not establish which request dispatched delegated work. In the two adapters studied, request-local

collection state and surviving child streams allow A's work to enter B's report. Binding at child start moves the arrival-time decision to another event; dispatch-time binding removes the misattributions we tested but sometimes leaves legitimate work unassigned. The typed Codex 0.160.1 witness isolates a separate information problem: within its stated conditions, no adapter can recover an origin the observable interface does not distinguish.

Truthful lineage available before reporting resolves that information problem (Proposition 1), but retaining and applying it is an implementation obligation. The upstream `rootTurnId` field enables a patch with no misattribution across the 2,389-schedule matrix and 34 additional registration-race schedules. Those 34 cases omit own evidence; latest-snapshot storage also limits retention. These results support explicit origin and source-turn identity on delegated events, with an unassigned-work channel for events whose origin cannot be recovered. A separate session-level record can preserve work arriving after its originating request has already been answered. We do not claim that a late event can make an already published report retroactively complete.

Data Availability

A reproducibility package accompanies this submission as supplementary material. It contains the tool, scenarios, patches, source maps, redacted live logs, selected result summaries and raw traces, and scripts to regenerate Table I. A separate historical archive supports the full 64,078-record audit. The package distinguishes recorded experiment outputs from fresh audit checks; these supplementary materials are not publicly released.

Acknowledgment

AI assistants (Anthropic Claude and OpenAI Codex) contributed to all sections, including source analysis, experiment code and execution, figures, text, proofs, and internal reviews. The authors take full responsibility for the work.

References

[1] Agent Client Protocol contributors, "Agent Client Protocol: Prompt turn and cancellation," https://github.com/agentclientprotocol/agent-client-protocol, commit 3c229ec5, Oct. 2026.

[2] Agent Client Protocol contributors, "codex-acp: AIR extensions, agent file change report," https://github.com/agentclientprotocol/codex-acp/blob/082be156/docs/air-extensions.md, docs/air-extensions.md, commit 082be156, Oct. 22, 2026.

[3] M. Musuvathi, S. Qadeer, T. Ball, G. Basler, P. A. Nainar, and I. Neamtiu, "Finding and reproducing heisenbugs in concurrent programs," in *Proc. 8th USENIX OSDI*, 2008, pp. 267–280.

[4] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica, "X-Trace: A pervasive network tracing framework," in *Proc. 4th USENIX NSDI*, 2007, pp. 271–284.

[5] A. Mishra and K. Sharad, "Observability for delegated execution in agentic AI systems," arXiv:2606.09692, 2026.

[6] V. Briliantov and B. Brandt, "Subagent sessions (RFD)," https://agentclientprotocol.com/rfds/subagents, commit bfeaad0c, Oct. 16, 2026; RFD revised Oct. 27, 2026.

[7] OpenAI, "Codex: app-server and core source, release rust-v0.160.1," https://github.com/openai/codex/tree/d27764b8, commit d27764b8, 2026.

[8] Agent Client Protocol contributors, "codex-acp source: CodexSubagentEventRouter," https://github.com/agentclientprotocol/codex-acp/blob/082be156/src/subagents/CodexSubagentEventRouter.ts, commit 082be156, Oct. 22, 2026.

[9] Agent Client Protocol contributors, "claude-agent-acp pull request #870: fix: hold a turn open while its background subagents are still live," https://github.com/agentclientprotocol/claude-agent-acp/pull/870, Jul. 2026.

[10] N. A. Lynch, "A hundred impossibility proofs for distributed computing," in *Proc. 8th ACM PODC*, 1989, pp. 1–28.

[11] OpenAI, "Codex pull request #51415: Expose and persist turn lineage across the app server," https://github.com/openai/codex/pull/51415, 2026.

[12] World Wide Web Consortium, "Trace context, level 1," https://www.w3.org/TR/2021/REC-trace-context-1-20211123/, w3C Recommendation, Nov. 23, 2021.

[13] S. Zheng, Q. Zhang, Z. Zhang, H. Li, and C. Hauser, "Formal security analysis of agent protocol composition," arXiv:2606.28690, 2026.

[14] S. Khan, "Stop means stop: Measuring and repairing the enforcement gap in agent-framework control primitives," arXiv:2607.14166, 2026.

[15] I. Santos-Grueiro, "Lingering authority: Revocable resource-and-effect capabilities for coding agents," arXiv:2606.22504, 2026.

[16] J. Li, "Where does exactly-once live? Model, harness, and tool-contract effects on duplicate side effects in LLM agents," arXiv:2609.29095, 2026.

[17] X. Zhu, Y. Mu, X. Zhang *et al.*, "Where agent frameworks fall short: Examining functional challenges and usability concerns," arXiv:2602.21806, 2026.

[18] R. Zhang, W. Dai, H. V. Pham, G. Uddin, J. Yang, and S. Wang, "Engineering pitfalls in AI coding tools: An empirical study of bugs in Claude Code, Codex, and Gemini CLI," in *Proc. 34th ACM FSE*, 2026, pp. 393–403.

[19] M. Cemri, M. Z. Pan, S. Yang *et al.*, "Why do multi-agent LLM systems fail?" in *Advances in Neural Information Processing Systems*, vol. 38, 2025.