Verification Allocation in Five Stage LLM Pipelines

Louis Wren Hernandez David

University of Alberta

With Apart Research

Abstract

Limited oversight creates a practical allocation question: should a verifier check early agents with greater downstream influence, or sample positions across a workflow? An existing probability simulation favored random allocation, but contained no LLM participants. We retain that baseline and add an instrumented experiment with genuine model responses. A native Codex proof of concept executes two matched scenarios through five fresh worker agents under four verification policies. A separate programmatic Codex runner then evaluates 20 matched scenarios using fresh ChatGPT-authenticated LLM sessions. Policies allocate zero, two random, two upstream, or five verifier calls; injections and tasks are matched across conditions. Verifiers independently calculate corrections rather than receive answer keys. Main final accuracy is 0.0%, 80.0%, 40.0%, 100.0% respectively; the paired random-minus-upstream difference is +40.0 percentage points, with a 95% bootstrap interval of [20.0, 60.0] percentage points. Crucially, a prediction frozen before the main run shows that the global-prefix verification protocol structurally favors random allocation when calculations are correct. The observed final outcomes agree with that prediction in 80/80 condition trials. Our contribution is inspectable evidence of real handoffs, injected-error propagation and model-generated correction, alongside an explicit account of this design constraint. The study does not establish random verification's superiority on natural tasks, calibrate the simulation's contagion parameters, or demonstrate swarm alignment. Context and tool-use audits support observed separation but do not establish a hard filesystem sandbox.

Research conducted at the Swarm Alignment Hackathon, November 2026.

1. Introduction

Multi-agent workflows rely on intermediate artifacts that can transmit mistakes. A monitor with a limited budget must choose which artifacts to inspect. An upstream-priority heuristic checks positions with the greatest number of descendants; a random policy distributes coverage. Their relative value depends on the error prior, the task, the information available to verifiers, and when a correction is applied.

The preexisting project simulated five agents by drawing from fixed error probabilities. It did not observe agents communicating or reasoning. This takeover adds actual LLM participants, retains the simulation unchanged, and corrects claims that exceeded its evidence. Contributions are a native five-agent control demonstration, a reproducible fresh-session LLM study, an independent evidence audit, and a structural explanation that limits what the observed policy comparison can establish. This is a controlled reliability study, not a test of collusion, deceptive intent or independent agent preferences.

2. Related Work

Greenblatt et al. [1] evaluate oversight protocols when an untrusted model may deliberately subvert its monitor. Our faults are researcher-induced arithmetic mutations, without an adversarial model objective. Hammond et al. [2] identify risks from interaction among advanced agents; our serial relay addresses a narrow information-transmission failure, rather than their broader incentive-driven risks. Codex documentation [3] describes delegated agent contexts and inherited model settings; documentation alone does not establish isolation, so we inspect actual execution records. Apart's Swarm Alignment guidelines [4] motivate reproducible experiments and inspectable evidence. The report uses the official template linked from those guidelines [5]. Unverified references in the historical paper are not reused as support for behavioral probabilities.

3. Methods

3.1 Preserved simulation baseline

Version 2 contains 200 matched scenarios under four policies, with 121 injected-error scenarios shared across conditions. It assumes contagion probability 0.60, spontaneous error probability 0.05, verifier detection probability 0.90, and false-positive probability 0.02. These are modeling assumptions, not estimates from our LLM study. Its factual subquestions are independent; dependence is imposed by the probability function. Corrections use the answer key and can never introduce a wrong answer, including after false alarms. We retain all original code, datasets, result files and PDFs with a pre-edit SHA256 inventory.

3.2 Actual LLM tasks and intervention

Twenty distinct numerical scenarios use five integer operations: inventory receipt, batch multiplication, dispatch subtraction, packing by floor division, and a final stock adjustment. The schedule, random verification positions and condition order are fixed using seed 20261107 before participant outcomes. Injection positions are balanced, with four main scenarios at each position 1 through 5. Every main scenario contains one injection, so accuracy estimates are conditional on an injected fault, not deployment error prevalence. Two separate pilot scenarios are excluded from main inference.

Each stage is a genuinely new model response. Stage 1 sees the initial inventory and its operation. Later workers see all earlier released reports and their current operation, and must use the latest value. They receive no separate authoritative specification of the full source task, although prior explanations may reveal source values or operations. This restriction creates a genuine dependency but is not a perfect information barrier. At the specified position, the researcher adds 24 to the generated value and replaces its explanation with a neutral report. The original response and altered artifact are both retained; the alteration is never called an LLM response. Exact integer arithmetic supplies objective researcher-side evaluation after generation.

3.3 Verification and correction

The policies are no verification (k=0), random verification of two distinct positions (k=2), upstream verification of positions 1 and 2 (k=2), and verification of all five positions (k=5). Verification is online, before an artifact reaches the next worker. A new verifier receives the original inventory and operation prefix, earlier released reports and the current artifact. It recomputes the prefix without an answer key and returns a flag and its own integer result. A flagged verifier's result becomes the downstream handoff. The

evaluator never supplies an oracle repair. Corrections therefore can be wrong. Random and upstream policies have equal call budgets, but later prefix checks may consume more tokens. This differs from the simulation's post-hoc oracle repair and downstream rerun, preventing direct validation of its effect sizes.

3.4 Native control and programmatic scale

Native participants use collaboration.spawn_agent with fork_turns=none, model gpt-6.1-sol and reasoning_effort=low. Each worker and verifier has a unique context and is never reused. Development assistants and the research orchestrator are explicitly excluded from the participant dataset. An initial context control used inherited xhigh reasoning; it is excluded from all study outcomes. The native pilot includes 8 complete condition trials and 58 actual worker/verifier responses. The median prepared-request-to-logged-response interval was 18.53 seconds (mean 20.39), including orchestration and logging. Extrapolating that mean over 638 pilot-plus-main calls gives approximately 3.6 hours of sequential execution. This is a measurement of the present sequential orchestration workflow, not a general benchmark of native model speed.

The main transport therefore uses new ephemeral Codex CLI processes with the same requested model and low reasoning effort, authenticated through the existing ChatGPT login. These are actual LLM sessions, clearly distinguished from native collaboration subagents. Each process starts in a new empty temporary directory. User configuration and rules are ignored, project-document loading is set to zero, API-key environment variables are removed, and the OpenAI provider and ChatGPT login are forced. At most three independent scenarios run concurrently; stages and conditions remain sequential within a scenario. No external paid API credentials are used. A separate two-scenario CLI pilot precedes the main run.

3.5 Isolation and evidence controls

Participant prompts omit policy labels, injection positions, other conditions' results, hypotheses and answer keys. Every actual response, handoff, injection, verifier decision and correction is retained. Native fork-none controls showed environment and standing instructions but no substantive parent research conversation. Participants are instructed to use no tools, files, memory, web or other agents. A no-model CLI probe nevertheless found exposed tool definitions despite disable flags: tool abstention is an instruction, not a proven capability boundary. Native and CLI event audits check actual tool activity and fresh thread identities. Any CLI tool use or incomplete generation fails the call and stops further dispatch.

Native task-message plaintext is encrypted in local rollout files. Prompts are logged before dispatch and are explicitly orchestrator-attested; the independent audit can corroborate native final responses, identities, configuration and tool activity, but cannot decrypt prompt equivalence. The programmatic runner saves exact stdin and arguments before launch, raw stdout, stderr, process status, final output, thread ID and usage. Independent deterministic replay checks handoffs and evaluation. Hidden server-side context, model checkpoint and sampling parameters remain unavailable. Model generation is not bitwise repeatable.

3.6 Metrics and analysis

The primary metric is exact final released accuracy. We separately score original worker outputs, pre-gate artifacts and released artifacts at every stage. A wrong-stage count is the number of incorrect released stages, not a geometric cascade depth. Direct injection detection requires a verifier flag at the injected position on an actually wrong artifact. Downstream incorrect reports are descriptive; without an

uncorrupted counterfactual, they are not uniquely attributable to injection rather than independent model mistakes.

The primary contrast is paired random-minus-upstream final accuracy. We report exact two-sided McNemar p-values over discordant matched scenarios, marginal Wilson intervals and a seeded 20,000-replicate paired percentile bootstrap interval. The ordinary paired bootstrap resamples entire scenarios across injection positions and does not preserve the fixed four-per-position balance; its interval is a descriptive conditional summary, not design-exact coverage for this stratified schedule. Other contrasts are exploratory. Repeated transformations, a shared model and small sample size further constrain inferential generality. No claim that repeated questions make tests conservative is justified. Planned, failed, incomplete and matched-complete counts are retained; no partial trial is silently counted as complete.

4. Results

4.1 Native proof of concept

Both native scenarios completed all four policies: eight condition trials, 40 worker responses and 18 verifier responses. In pilot 1, the stage 1 worker generated 379; the researcher changed the artifact to 403. Without verification, subsequent agents produced 2821, 2800, 560 and 568, yielding an incorrect final value. A fresh verifier independently flagged 403 and proposed 379; the next fresh worker used the corrected value and generated 2653. Full verification finished at the correct 534. Pilot 2 injected at stage 4. None finished incorrectly; Random k=2 finished incorrectly; Upstream k=2 finished incorrectly; Full k=5 finished correctly. These pilots establish observed orchestration and correction control; they are excluded from main inference.

Native pilot 1 controlled handoffs

	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5
No verification	403	2821	2800	560	568
Full verification	379	2653	2632	526	534

Worker 379 > researcher artifact 403 > verifier correction 379

Figure 1. Actual native pilot 1 released values. The stage 1 worker generated 379; the researcher altered it to 403. Full verification independently restored 379 before stage 2. These are recorded model handoffs, not additional trials.

4.2 Main observed LLM outcomes

Policy	Correct / n	Final accuracy (95% CI)	Wrong-stage mean	Verifier calls
None	0/20	0.0% (0.0%-16.1%)	3.000	0
Random k=2	16/20	80.0% (58.4%-91.9%)	1.150	40
Upstream k=2	8/20	40.0% (21.9%-61.3%)	1.200	40
Full k=5	20/20	100.0% (83.9%-100.0%)	0.000	100

Table 1. Actual programmatic LLM final accuracy on 20 matched injected-fault scenarios. Parenthesized intervals are 95% Wilson intervals. Verifier budgets count actual calls. Pilots and transport controls are excluded.

The primary paired difference is +40.0 percentage points; 95% paired percentile bootstrap interval [20.0, 60.0] percentage points. There are 8 random-only correct pairs and 0 upstream-only correct pairs; exact two-sided McNemar p=0.0078125. This is a conditional benchmark comparison, not evidence for a general deployment advantage. Independent replay found 400/400 locally correct worker calculations, 180/180 correct verifier recalculations and 180/180 correct flag decisions. Local worker correctness is evaluated against the received input, so a worker can calculate correctly while propagating a globally wrong artifact. All 80 planned main condition trials completed; no failed or partial main trials were omitted.

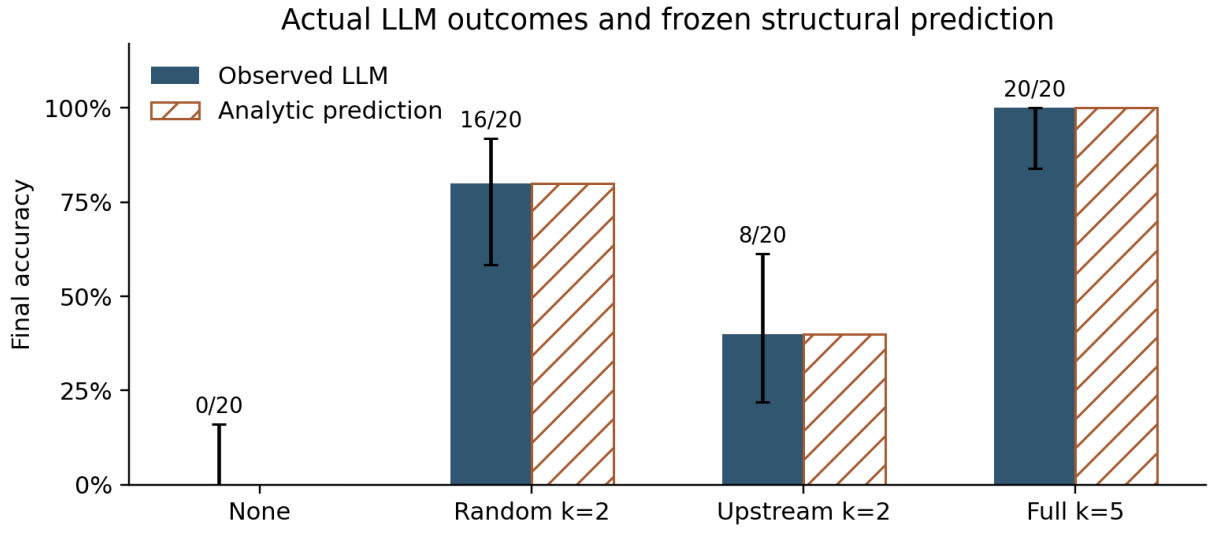


Figure 2. Main observed LLM final accuracy with Wilson intervals, alongside the perfect-execution prediction frozen before main generation. Prediction bars are analytic, not additional LLM observations.

4.3 Structural prediction and discrepancies

The frozen perfect-execution prediction is 0/20 correct for no verification, 16/20 for random, 8/20 for upstream and 20/20 for full. Under correct arithmetic and prefix verification, success occurs when the last

verified position is at or after the injection. Any two distinct random positions have maximum at least 2, so random weakly dominates checking positions 1 and 2 for this terminal outcome. This is a property of the protocol, not a discovery about LLM preferences or natural error recovery. Observed final outcomes match 80/80 frozen predictions. The observed per-policy correct counts are 0, 16, 8, 20, against predictions 0, 16, 8 and 20. Thus any reported policy advantage here must be assessed as execution of the specified coverage mechanism rather than independent evidence of stochastic LLM contagion. No discrepant outcome is removed or changed.

4.4 Simulation retained and corrected

Policy	Simulated final accuracy	Simulated wrong-stage count after correction
None	53.0%	2.3223
Random k=2	78.5%	1.1157
Upstream k=2	66.5%	1.1901
Full k=5	97.0%	0.1736

Table 2. Archived probability simulation, not LLM observations. Accuracy uses all 200 scenarios; wrong-stage means use the 121 injected-error scenarios. Its random-minus-upstream accuracy difference is +12 percentage points with 31 versus 7 discordant outcomes. The historical continuity-corrected statistic is 13.9211. Random also has the lower post-correction wrong-stage count; the historical statement that upstream has the lower count is erroneous. Independent scenario-index random draws do not establish empirical within-question correlation or justify calling the tests conservative. The stored scenario_seed field is unused by behavioral draws. These limitations remain documented without changing the archived results.

5. Discussion and Limitations

The experiments demonstrate real LLM participation and correction-sensitive handoffs. They do not provide an independent behavioral confirmation of the simulation's random-allocation finding. Workers are explicitly told to use the most recent value, so this measures compliance with a prescribed relay rather than spontaneous susceptibility to misinformation. The task's restricted worker information and global-prefix verifier give late checks substantial recovery power. A measured random advantage must be interpreted against the frozen structural baseline. Agreement demonstrates faithful protocol execution on simple arithmetic; disagreement would reveal model or harness behavior requiring inspection. Neither is an estimate of a universal contagion rate.

Important limitations are one model family, one deliberately simple task template, forced faults in every main scenario, no clean main controls, one generation per stage and condition, no repeated stochastic runs, and no hard filesystem isolation. The same model serves as worker and verifier, so errors can be correlated. Equal call budgets do not guarantee equal monetary or token budgets. The native pilot and CLI main study use distinct transports with different standing context, so they are reported separately. Deterministic evaluation makes accuracy inspectable but does not make the task representative of coding, negotiation, open-ended research or adversarial swarms.

A stronger follow-up should vary verifier information access, use localized checks that cannot automatically reconstruct a corrupted prefix, include uncorrupted counterfactuals and clean scenarios, and run several heterogeneous task families and models. Both early checks that prevent costly downstream work and late checks that repair terminal outputs should be measured, including token costs and latency. Such experiments must be specified before their outcomes; we do not retrofit this dataset to claim a broader result.

6. Conclusion

This project replaces an exclusively simulated evidence base with actual native-agent and fresh-session LLM executions while preserving the original simulation. The instrumented records show whether real workers propagated a controlled fault and whether model-proposed verification changed later behavior. The observed policy comparison is constrained by an analytically favorable architecture for late verification. The defensible result is an inspectable control demonstration and a narrowly scoped empirical stress test, with the structural confound made explicit.

Code and Data

Code, evidence and the report are supplied in the local submission package; nothing has been published. Native evidence is under results_empirical and evidence_audit; programmatic model records and analysis are under results_llm. The archived simulation remains under experiment and results_v2. Reproduction commands are in Appendix B. Original README and research summary copies and a 42-file pre-edit inventory are retained. The original v1/v2 study predated this takeover; newly added experiment tooling, actual responses, audits and this manuscript constitute the present work. We make no unsupported claim about which earlier files were created before the hackathon weekend.

Author Contributions

Louis Wren Hernandez David is the submitting author and provided the research brief and constraints. Codex assisted with implementation, experimental orchestration, auditing and drafting. The report is an AI-assisted draft for author review; this wording does not imply that the author has already personally reviewed every claim.

References

[1] Greenblatt, R., Shlegeris, B., Sachan, K., and Roger, F. (2024). AI Control: Improving Safety Despite Intentional Subversion. ICML. https://arxiv.org/abs/2312.06942

[2] Hammond, L., et al. (2025). Multi-Agent Risks from Advanced AI. Cooperative AI Foundation Technical Report 1. https://arxiv.org/abs/2502.14143

[3] OpenAI (2026). Subagents. ChatGPT and Codex documentation. https://learn.chatgpt.com/docs/agent-configuration/subagents

[4] Apart Research (2026). Swarm Alignment Hackathon, official guidelines. https://apartresearch.com/sprints/swarm-alignment-2026

[5] Apart Research (2026). Swarm Alignment official submission template. https://docs.google.com/document/d/1_1_27EyZriMlc8n8EBE2YScaeyMjm-8xjR097nmHXAE/edit

Appendix A Limitations and Dual-Use / Ethical Considerations

All empirical participants are actual model invocations; deterministic evaluator calculations, analytic predictions and researcher mutations are separately labeled. No personal or third-party research data are included in task inputs. Native export excludes account identifiers, credentials and encrypted reasoning. The participant no-tools instruction and audit establish observed behavior, not a secure sandbox guarantee. We did not test adversarial attempts to escape the environment.

Studying verification gaps can help a malicious operator choose where to introduce a fault. This report uses harmless arithmetic perturbations and does not develop an attack against a third-party service. Random allocation retains nonzero blind spots in a single trial; it does not close every coverage gap or guarantee safety. Prefix reconstruction and identical-model verification may also fail in realistic workflows. The defensive purpose is to expose these dependencies and improve evidence standards, not to claim a validated deployment safeguard. Existing ChatGPT/Codex usage is consumed; no paid external API key is used. No external publication, submission or third-party messaging occurred.

Appendix B Reproduction and Evidence

Evidence replay needs Python and no model inference. Run python experiment/test_experiment.py to check the preserved simulation, python -m unittest empirical_cli.test_runner for transport tests, python evidence_tools/test_audit_native.py for native auditor tests, python evidence_tools/verify_native_exports.py for offline replay of supplied safe native exports, and python evidence_tools/audit_cli.py to independently check programmatic evidence. Re-extraction with evidence_tools/audit_native.py requires the original local native rollouts. Offline replay validates supplied exports and checksums, not a provider signature or an independent re-extraction.

Run python -m empirical_cli.runner analyze to recompute statistics from recorded responses. Fresh inference requires Codex CLI with ChatGPT authentication and consumes included account usage; it is not an API-free computation. Preserve the supplied frozen evidence and run in a separate copy with fresh output directories. The native orchestration procedure and exact prompts are in empirical/protocol.md, native.py, requests.jsonl and events.jsonl. The programmatic protocol is empirical_cli/protocol.md; its runner uses subprocess stdin with no conversation resume. Output schema and raw stream accompany every call. A pre-main hardening amendment records parser, timeout-evidence and inventory changes without changing treatments or task prompts.

The independent CLI audit verifies 639 actual completed model calls: 580 main calls, 58 pilot calls and one transport control. The native study contains 58 worker/verifier calls in eight completed pilot condition trials; its initial context control is separate. Checks cover fresh identities, raw model response equivalence, tool-use abstention, prompt reconstruction, planned budgets, injections, model-proposed corrections and integer scoring. Exact native prompt plaintext remains an attestation boundary. See evidence_audit/audit_cli.json, audit.json, limitations.md and SHA256 manifests. Safe native exports allow offline response inspection without account secrets; rerunning native export itself requires the local source rollouts.

The original native schedule retains 80 planned main-trial entries that were not executed natively. They are not empirical observations. The main study is the separately identified programmatic backend with its own completed records and pre-main gate.

One native verifier explanation contained multiplication and minus glyphs that were damaged by PowerShell logging. The original event remains append-only; a separate erratum restores the explanation against the independently exported native response. Its numeric value and flag were unchanged. Two prepared downstream requests were superseded before dispatch; they are retained but are not counted as model calls. The corrected explanation was present before the actual next worker ran. A source amendment records explicit UTF-8 reading. This is a logging repair, not an outcome correction.

Appendix C Model Configuration and Usage

All study calls request gpt-6.1-sol with low reasoning effort. Native rollout configuration is independently exported; CLI records preserve requested configuration and actual turn usage. The installed programmatic CLI is 0.167.0; native rollout metadata reports 0.166.0. The initial native context control uses xhigh and is excluded from study outcomes.

Scope / role	Calls	Input tokens	Cached input tokens	Output tokens
main / worker	400	4,316,341	3,144,960	14,113
main / verifier	180	1,957,258	1,182,720	10,282
pilot / worker	40	431,519	89,600	1,403
pilot / verifier	18	195,556	35,840	973
control / worker	1	10,718	0	32

Across the main study, random verification uses 40 verifier calls, 435,532 input tokens and 2,562 output tokens; upstream verification uses 40 calls, 433,152 input tokens and 1,945 output tokens. Cached input is a subset of input, not an additional total. Equal verification counts do not establish equal reasoning work. Full raw usage fields remain available per call. Native study usage across 58 independently exported participant threads is 1,592,508 input tokens, including 1,470,592 cached input tokens, and 2,334 output tokens; the separate context-control thread is excluded.

Unavailable fields include exact model checkpoint, temperature, random generation seed, hidden server-side instructions and a monetary charge breakdown. The scenario seed controls schedule generation, not LLM sampling. Token fields are backend-reported and are not interchanged across transports. Native system/developer context is inherited; fork-none removes substantive parent conversation, not standing instructions. The template source is retained, with single-author layout and Times New Roman substituted for unavailable Old Standard TT. All other unchanged DOCX package parts are byte-identical to the official template before PDF export.

LLM Usage Statement

Codex was used for research assistance, code, audit tooling, orchestration and manuscript drafting. Experimental participants are separately identified worker and verifier invocations; development assistants are excluded from empirical outcomes. Deterministic evaluators and independent transcript checks substantiate the quantitative claims within their stated scope. This draft still requires the submitting author's review before external submission.