

Verifying Stop Scope in Agent Workflows

Huseyin Buldurgan

Independent Researcher

huseyin@latentatlas.ai

With

Apart Research

Abstract

This paper compares cancellation, waiting for worker completion, and origin-scoped control in an Open-SWE workflow with programmed model responses and real file effects. A 26-condition comparison was repeated in a fresh environment on the same host. When a worker was held before writing, cancellation allowed the already-started worker to write the final report. Waiting allowed the worker to finish before acknowledging completion; origin-scoped control prevented the pending write. Existing file effects were not reversed. Summary runs could still reread records and recompute results, distinguishing a stop on report writing from a stop on computation. A broader catalogue defines 286 draft cases across 24 families, but those tests remain unexecuted. The contribution is an auditable local comparison of intervention scope, file effects, and the results that remain available.

1 Introduction

1.1 Why this study

I wanted to start with Track 2 - What happened, what breaks next -, a cost-benefit type of approach to evidence at certain points on the timeline, a matrix which involves date-evidence related to possible benefits of stopping at the date and cost of stopping at the date. But especially on the cost side, I realized that we do not have enough information to process the actual cost which I clearly do not mean monetary cost, a potential loss of research knowledge. A certain knowledge about phases and dates on OpenAI's work would help [12,13] but clearly there was not enough public evidence for a healthy computation on cost-benefit. And then, I thought maybe a research on the ideal time to stop, what was the ideal time that optimal cost-benefit balance. What was lacking for a computation of cost-benefit was still same. I did not consider a generic shutdown protocol a sufficiently original contribution.

As the sprint continued, statements from both OpenAI and Anthropic [1,2] made me question if there is a consensus on AI safety, implementation of third party oversight and a set of rules and obligations could that bring an end to the threat of autonomous AI attacks and could this be the solution of concerns about AI-safety that many see but less talk. I genuinely think not. Slowing it down is a positive development, but it does not directly prevent the incidents we study. Even though the statements from companies that admit pace of the developments, and safety and security of AI is very far from each other. That race among the companies and states was the problem. So AI will be safer when every party slows the development down. I am very skeptical about it and even if these statements come to a point, an agreement, I still will be. These ongoing talks could be only round one of a lengthy-process. This research and paper is a preparation before round two. My concern is that future incidents could be followed by claims such as "We

stopped it, but it did not stop”. Round one is the understanding that stopping is needed. This is the round of common understanding that it should not be limitless. Round two, I think, could concern proper ways of stopping it. I argue these questions should be answered; before the settlement of proper ways of stopping: what are the evidence-outcome relations for stopping? Can it still be written when we stop, when an AI agent could still be working even when we stopped, is it possible even if we stop child-agents be spawned? What is required to stop completely? And causality among these should be studied; parent-agents and sub-agents, ongoing-tool calls and background workers, queued tasks and scheduled jobs, retries and follow-up actions, requests already sent to external systems, file writes, summaries and final reports.

1.2 Questions and contributions

I examine how existing controls meet stop obligations under matching local timing, what output remains, and what additional evidence is needed for queued and delegated tasks. I contribute a native effect comparison, a verifiable audit package, and an open ended scope list.

2 Related Work

CPython distinguishes cancellation of pending futures from a work which is still running [3]. A DeerFlow change drains synchronous skill work before lifecycle cleanup [4]. Chubby’s sequencers illustrate receiver-side protection against stale authority [5]; Our control blocks writes from a revoked task within one trusted process.

VERA studies how to revoke authority in agent workflows [6]. Evidence-Carrying Termination examines what evidence is needed to support a claim that a task has been successfully completed [7]. I reviewed both studies but did not test their methods in my experiment. This study is a comparison within one reproducible local workflow. I examine how different stopping methods affect work already accepted by the system, actual file writes, and the results that remain available. I do not claim to be the first to study these issues. Celery distinguishes revoking a task from terminating it. Kubernetes distinguishes suspending a schedule from stopping jobs that have already started [8,9]. These distinctions show why queued tasks and future work need separate checks.

3 Methods

3.1 Completed comparison

The experiment used Open-SWE at commit e0d9aff59925a4da55ec8d6c31fa723651c24909 [10]. It used the actual local components for agent creation, middleware, tool routing, the Agent Server interface, and the Store, together with isolated local files. Model responses were programmed in advance and were deterministic. The experiment did not include calls to external model providers, production authentication, remote sandboxes, or live LLM behavior. The task analyzed 24 existing synthetic incident records. It produced a partial report covering the first 12 records and a final report covering all 24.

The experiment compared four setups:

- 1 Cancellation with a summary.
- 2 Waiting for worker completion, also called draining, with a summary.

- 3 Control based on the task an operation originated from, called origin-scoped control, with a summary.
- 4 The same origin-scoped control, but with ordinary tools available to the original task's completion run.

The fourth setup removed a tool restriction; it did not remove all summarization.

Each setup was tested under three timing conditions: holding an already-started worker before its file write, intervening after the final file write, and running without the timing barrier. These were combined with two fixed dataset seeds, 17 and 41, producing 24 conditions. Two additional tests examined what happened when waiting for worker completion timed out, bringing the total to 26. Each condition ran in a fresh process. The order of the four setups was reversed for the second seed. The seeds provided fixed test data, not samples of unpredictable model behavior.

Three stop requirements were assessed separately:

- **R:** No final report write from the original task after the stop request.
- **V:** No file write from the targeted task origin starting after its local authority was revoked.
- **C:** No further file effect or unfinished worker within the stated scope after worker completion was acknowledged.

The availability of results and any interference with legitimate new work were recorded separately. In the cancellation and waiting setups, the revocation event was recorded for comparison, but no control enforced it. Failing V in these setups therefore did not mean that Open-SWE had broken a revocation guarantee. C was assessed only when the setup acknowledged worker completion.

The origin-scoped setup used the same lock for revoking authority, checking task origin, and writing the file. This prevented revocation from occurring between the check and the write. The mechanism operated within a local runtime whose identity and authority records were assumed to be trustworthy.

Each file write was observed through two events: `native_write_attempt` and `native_write_result`. The second event recorded the file hash after the write returned. A worker starting was not treated as the file write starting. The experiment did not measure the exact physical commit time or whether the data would survive a crash. If the observed write interval overlapped the stop boundary, that observation alone was not classified as definitely before or after it.

A separate audit checked that requests and responses, caller identities, task-origin records, worker events, and actual file contents agreed. A control that failed could still provide valid evidence if its behavior was recorded consistently. Missing evidence was not counted as successful stopping.

3.2 Broader design and completion criteria

The broader test catalogue contains 24 families, 91 variants, 146 sub-scenarios, and 286 draft cases: 260 direct tests and 26 tests of interactions between components or controls. Each case describes its scope, intervention, identity checks, observations, checks of normal operation, fault controls, and cleanup.

Listing a case does not mean that it has been tested. The full test list is not yet finalized or connected to a complete test runner. The 26 completed conditions were developed separately and are not a numbered subset of the 286 draft cases. A completion rate of 26/286 would therefore be misleading. Appendix B describes the planned scope and estimated workload. These estimates are not measurements of time already spent.

4 Results

4.1 File effects and useful work

All 26 conditions produced valid evidence records. Each condition was repeated in a fresh environment at a different filesystem location on the same computer. This produced 52 executions of 26 conditions, not 52 different conditions or an independent laboratory replication. The outcome classifications matched in all 26 comparisons.

Each environment recorded 156 accepted platform runs, 220 local workers, 204 report writes, and 130 verified numerical outputs in final responses. These counts describe activity within the experiment; they are not independent experimental samples. The same 45 quality-assurance tests passed in both environments.

Table 1 shows the results when the worker was held before writing. Each fraction refers to the two fixed dataset inputs in the initial environment. The repeat produced the same outcomes.

Intervention	Original task's final report written after the stop request?	Interpretation
Cancellation with summary	Yes, 2/2	Cancelling the asynchronous task did not prevent the already-started worker from writing.
Waiting for worker completion with summary	Yes, 2/2	The worker finished before completion was acknowledged.
Origin-scoped control with summary	No, 0/2	The pending write from the revoked task origin was rejected.
Origin-scoped control with ordinary completion tools	No, 0/2	Access to ordinary tools did not bypass the file-write control.

Table 1. The same timing conditions distinguish cancellation, waiting for completion, and preventing a pending write. The controls are not ranked as though they all promised the same outcome.

All six normal waiting conditions satisfied requirement C: no further effect or unfinished worker within the stated scope after completion was acknowledged. The two timeout conditions remained pending or unknown. In the eight conditions where intervention followed the final write, no control reversed that write. In the eight conditions without the timing barrier, the final write occurred before the stop request. These observations do not estimate how often such events occur in practice.

The tests also included work representing a new human request, its completion run, and an independent task. All produced correct results and files. None of these 78 selected tasks per

environment was rejected by the origin-scoped write control. This does not establish a general zero false-blocking rate.

Early origin-scoped intervention prevented the original final report from being written while preserving the partial report. However, summary runs could still read all 24 original records and recompute the analysis. Blocking the final report therefore did not stop all computation. The summaries were not simply repeating an answer already produced.

The experiment used controlled delays, so the results do not establish production performance. Financial cost, CPU use, and the value of potential scientific discoveries were not measured.

4.2 Preliminary extension tests

Four queue tests, each run once, showed that the implemented stop paths had different scopes:

- **No stop:** The current run processed queued work.
- **Owner stop:** The current run was interrupted, but an ordinary replacement run started and processed pending work.
- **Admin stop:** No replacement started during the short observation period, but the queue remained. A later explicit human request caused the pending work to be processed.
- **Slack stop:** The queue was cleared and a summary run with restricted tools started.

These differences reflect the implemented control paths, not evidence that a model resisted stopping. New and independent test tasks ran sequentially in these checks. Their isolation during concurrent execution was not tested.

A fifth combined test verified that a parent task could delegate to a child, which then called `write_file` and produced an actual local file. The child remained within the same platform run and was tracked using subtask identities. Stopping the child was not tested.

A scheduled job was also created, but an error in how the runtime handled time values prevented it from running during the 150-second observation. The 27 error entries came from one trial, not 27 trials. Normal scheduled execution was therefore not established, and no successful scheduler stop can be claimed.

The broader 286-case experiment and the proposed smaller extension were not executed.

5 Discussion and Limitations

The main finding is that different stopping methods make different commitments. Waiting for worker completion can allow accepted work to finish and then correctly acknowledge that it has ended. Preventing a pending effect is a different requirement. The local origin-scoped control prevented the pending write under the tested timing conditions, but it did not reverse a write that had already occurred.

A stop report should therefore identify which task origin was targeted, where the control was applied, which effects it covered, and whether queued or future work was included. It should also state the observation period and any unresolved questions. These recommendations combine established ideas. The experiment supports them only within the scope tested.

The original cost–benefit question can now be addressed in a narrower form: which output was prevented, which analysis remained available, and whether selected legitimate work continued. These observations cannot determine whether the historical research was “worth it.”

The summary results also matter when defining success. Preventing a report write is not the same as stopping computation. An institution seeking to stop both would need controls and evidence covering both.

I have limited confidence that voluntary attention to safety alone will lead to adequate action. In my view, institutions should face binding, lawful requirements to demonstrate control, supported by public scrutiny and independent oversight where appropriate. This is my position on accountability, not an effect measured by the experiment. The aim is to make a claim of attempted shutdown answerable to specific requirements, rather than treating the intention to stop as sufficient.

Limitations

Model responses were deterministic. The observer, operating system, and local authority records were assumed to be trustworthy. The experiment did not test frontier-model intent, forged evidence from a hostile runtime, tracking task origins across distributed systems, computation at external model providers, reliable recovery after crashes, or reversal of external effects.

The repeat used the same computer and base Python version. There was no independent human reviewer. Two fixed dataset inputs cannot establish a statistical safety rate. The 24-family catalogue is a structured proposal for further testing, not a complete classification of every possible system.

Future Work

The next tests should examine whether queued work retains a trustworthy task origin and whether its authority is still valid when it starts. They should also test child-agent stopping and distinguish summaries that only report existing results from summaries that perform new calculations.

Scheduled stopping first requires a working baseline in which the timer successfully starts a run. Preserving stop decisions across restarts and enforcing controls at a separate service that performs an action also requires further implementation and fault tests.

A task starting successfully, or one preliminary test passing, is not enough to mark these stop requirements as verified.

6 Conclusion

In the tested Open-SWE workflow, cancellation, waiting for worker completion, and preventing a pending write produced different outcomes. Cancelling did not prevent an already started worker from writing. Waiting allowed the worker to finish before acknowledging completion, while origin scoped control blocked the pending write. Summary runs could still recompute the analysis. These results distinguish ending a run, completing accepted work, preventing a file write, and stopping computation.

A claim that an agent workflow has stopped should therefore identify which activities were targeted, what the intervention achieved, and what evidence supports that conclusion. Activities outside the tested scope should remain explicitly unresolved. The completed experiment provides a local comparison; the broader catalogue still requires testing.

I see this study as preparation for Round 2. Recognizing that limits are needed is one step. Defining those limits operationally and verifying their effects is another. The purpose is to prepare these distinctions before a future incident leaves institutions explaining, “We stopped it, but it did not stop.” A decision or an attempt to stop should not be seen as guaranteed stop of set of actions.

Code and Data

The evidence supplement, `stop-scope-supplement-v1.zip`, is available from the [versioned GitHub release](#). It includes canonical and repeat records, native file evidence, source snapshots, an offline auditor, and separately labelled preliminary and unexecuted-design material. After extraction, run `python3 -B verify_bundle.py` in the archive directory to check retained records and hashes. This is an offline evidence audit, not a turnkey setup for running new experiments. The sharing copy minimizes unnecessary local paths and records original-to-export digests; original evidence remains retained unchanged.

Acknowledgements

I am grateful to Apart Research and to organizations working on AI-safety education and public awareness. I also wish to acknowledge Alex Turner, whom I came to know through his writing, and whose efforts on ethical responsibility I respect [11].

References

- [1] Altman, S. 2026. [I agree with Dario that we need to pace the frontier](#). X post, 12 September.
- [2] Amodei, D. 2026. [We Must Pace the Frontier](#). September. Author’s website.
- [3] CPython contributors. [concurrent.futures base implementation, v3.14.3](#). Python Software Foundation.
- [4] DeerFlow contributors. [Drain skill sync before cancellation](#). Merged change 76d072c. Source implementation.
- [5] Burrows, M. 2006. [The Chubby lock service for loosely-coupled distributed systems](#). OSDI.
- [6] Liu, L., Yu, H., and Jiang, X. 2026. [VERA: Authority-Preserving Edge Revocation for Federated AI-Agent Workflows](#). arXiv:2608.30091v1.
- [7] Liu, J. 2026. [When May an Agent Stop? Evidence-Carrying Termination for Tool-Using LLMs](#). arXiv:2608.23623v1.
- [8] Celery project. [Workers Guide](#). Observed version 5.6.3. Accessed 13 September 2026.
- [9] Kubernetes project. [CronJob documentation](#). Accessed 13 September 2026.
- [10] LangChain. [Open-SWE source at the study pin](#). Commit e0d9aff59925a4da55ec8d6c31fa723651c24909.
- [11] Turner, A. 2026. [Why I Left Google DeepMind](#). 15 July; updated 6 September. Author’s website.

[12] Hugging Face. [Agent intrusion technical timeline](#). Public incident account.

[13] OpenAI. [Hugging Face model evaluation security incident](#). Public incident account.

Appendix A Limitations and Dual-Use Considerations

All experimental effects concern synthetic records and owned local processes and files. No intrusion into an external target, real credentials, live model service, or paid provider was used in these trials. Knowledge of incomplete stopping can be misused. A public package should retain the defensive verification purpose and avoid adding offensive persistence recipes. Publication review should remove unnecessary local metadata from a separate sharing copy, not alter retained raw evidence.

The trusted observation and authority assumptions are essential: without them, the present logs do not prove control over an adversarial runtime. Native write completion is not evidence of fsync or crash durability. Absence of an observed event outside the stated observation window is not proved. No jury or other human-participant experiment was conducted; the earlier interactive-app idea supplies no human-subject findings. No claim about model shutdown resistance or a production failure rate follows from deterministic fixtures.

Appendix B Full Scope and Workload

The scope catalogue has 24 families. F01 covers stop entry points; F02 target and origin; F03 request transport; F04 chat and client lifecycle; F05 natural-language and model/provider stopping; F06 asynchronous task trees; F07 native workers; F08 parallel or batched tools; F09 processes and sandboxes; F10 delegated agents; F11 pending runs; F12 queues; F13 scheduled work; F14 event and background triggers; F15 retry, delivery, and receipts; F16 checkpoint, crash, and restart; F17 tokens, sessions, and authority; F18 networks and separate receivers; F19 persistent and partial effects; F20 prior outputs and downstream consumers; F21 human approval and reauthorization; F22 observation and acknowledgement; F23 safe cleanup; and F24 legitimate work, shared resources, and costs.

The 91 variants expand into 146 sub-scenarios and 286 draft cases. Twelve implementation packages organize the work. Eight packages have low-confidence engineering estimates totaling 120–226 active person-hours; four remain unknown. This incomplete subtotal is neither measured effort, autonomous AI runtime, nor an estimate of the whole project’s completion time. Full completion time remains undetermined.

Mechanical expansion of draft case–arm entries across two seeds and a fresh repeat gives 12,548 live base executions; illustrative additions of healthy and fault controls give 13,676 and 26,224. These are sensitivity scenarios, not frozen requirements. Nine real-model cases still need an appropriate sampling design, and shared-control equivalence has not been established. A hypothetical ten seconds per execution gives serial totals of 34.9, 38.0, and 72.8 hours; these are not forecasts or lower bounds. The old matrix’s measured runtime, roughly five and a half minutes per environment, excludes construction and setup and cannot calibrate the unimplemented families.

Appendix C Reproduction and Research Development

The canonical v02 and clean-v01 comparison retained earlier development records rather than overwriting them. The repeat used a fresh environment and relocated sources on the same host with Python 3.14.3. Matching outcome statuses do not imply identical timing or log bytes. Earlier development runs were not added to the final sample. An additional receipt-bypass diagnostic in each environment sits outside the 26-condition matrix; it checks classification of a well-observed control failure, not natural incident frequency.

The research developed from public-record reconstruction and intervention cost–benefit exploration into the local control comparison and then the broader stop-scope design. Pre-sprint source and conceptual work is not presented as newly produced during the sprint. The local implementation and retained experiment records dated 12–13 September constitute the empirical work reported here. The later broad design and preflights extend the research agenda without converting unrun cases into findings. No unmeasured personal work hours are inferred from file timestamps.

LLM Usage Statement

I used LLM assistance for source discovery, literature comparison, experimental design, coding, debugging, analysis, and editing.

I defined the research motivation and the Round 1–Round 2 framing. Through discussion, I questioned whether the available evidence supported the proposed conclusions, rejected approaches that did not meet the intended contribution, and asked for a broader examination of what stopping should cover. The experimental design developed through these discussions. LLM assistance was used extensively to implement and run the experiments, develop audit checks, analyze the records, and revise substantial parts of the manuscript.

Reported results were checked against execution records, file contents, and automated audits. These checks were AI-assisted and do not constitute independent human validation. The experiment itself used programmed, deterministic model responses rather than a live LLM.