Simulation Fidelity and Runtime-Driven Convergence in Output-Capped Tool-Using Agents

A mechanism study, a failed fidelity assumption, and an evidence boundary

Aleksandr Tikhonov

July 16, 2026

Abstract

Output caps can leave a tool-using language-model agent with an incomplete artifact and no reliable transition from writing to finalization. We study a bounded runtime driver that detects a lack of byte-producing mutations and can force an append or finalize action. In a 27-run runtime-mirror ablation over three fixed task families, operational completion—explicit finalization plus a task-specific byte floor—was 1/9 for baseline recovery, 7/9 for forced runtime action, and 2/9 for identical recovery wording without forcing. A subsequent fidelity audit found that the mirror could materialize partial tool arguments at a boundary unavailable to the parser and runtime path. The ablation therefore establishes mechanism behavior only inside the mirror; it is not a production-effect estimate. A faithful parser/loop runtime-path validation found zero driver engagements in nine enabled small-artifact controls. In the large-artifact arm, bytes landed after recovery in all eight runs that latched a target-path truncation, but the arm lacked a matched driver-off comparison and five of nine runs reached the wall-time limit. The aggregate does not establish recovery of every individual truncation event. This case shows why agent-runtime experiments must validate pathway fidelity, artifact state, and termination separately before transferring a result from a simulator to a deployed system.

1 Introduction

Tool-using language-model agents interleave generated reasoning with externally executed actions (Yao et al. 2023; Schick et al. 2023). When the intended output is a large file, the orchestration runtime has an additional responsibility: it must preserve partial work, keep the interaction within context and turn limits, and decide when the artifact is ready to seal. An output-token cap can interrupt this sequence, but an agent may also fail without truncation by repeatedly inspecting an already assembled file or by never issuing a finalization action.

These behaviors make evaluation unusually sensitive to the harness. A simulator can reproduce tool schemas and result shapes while still placing one state transition on the wrong side of the model, stream parser, or dispatcher boundary. That discrepancy can create a recovery path the deployed runtime cannot execute. Simulation verification and validation must therefore address both returned values and whether the same path is reachable (Sargent 2013).

This paper reports a mechanism study and the fidelity failure discovered after it. We ask three questions:

1. Within a controlled runtime mirror, does bounded forced action separate from baseline recovery and wording-only guidance on three fixed tasks?
2. Does the mirror exercise a pathway reachable through the actual stream parser and orchestration loop?
3. What does a faithful parser/loop path validation establish without a matched treatment arm and without production persistence boundaries?

Our contribution is an evidence boundary. The 27-run ablation provides descriptive mechanism evidence under the mirror's transition semantics. The pathway audit invalidates transfer of its apparent effect to the runtime. A smaller faithful path validation establishes two narrower observations: the driver remained inert in the tested small-artifact control, and bytes landed in each run that latched a target-path truncation. It leaves the causal runtime effect unresolved.

2 Runtime model and outcome definitions

2.1 Artifact lifecycle

The evaluated agent can initialize a file, append content, inspect the workspace, and explicitly finalize the artifact. The mirror recorded three outcomes:

- **size floor met:** persisted bytes reached a fixed task-specific threshold;
- **finalized:** the runner observed the explicit sealing action; and
- **operational completion:** both the size floor and finalization conditions held.

The byte floors were 12,000 bytes for the long-form document, 8,000 bytes for the code module, and 4,096 bytes for structured notes. The mirror did **not** validate semantic correctness or code syntax. Its term "completion" must therefore not be read as full task success. The later parser/loop path test used a different large- artifact outcome: Python `ast.parse` succeeded and the file contained at least 4,096 bytes. That still did not validate the requested program's behavior.

Separating artifact state from runtime status follows the artifact-oriented direction of agent evaluation (Liu et al. 2024; Jimenez et al. 2024). A model can cross a size floor and exhaust its turns without finalizing; a driver can also seal an artifact that is large enough but wrong.

2.2 Bounded convergence driver

The candidate driver observes successful byte-producing mutations rather than the number of tool calls. While the artifact is below a plausible floor, a stalled turn can trigger a forced append action. Once the artifact reaches the floor, either a byte plateau or a completion-like response can trigger a forced finalize action. The mirror bounded execution to 20 turns, eight forced appends, and two forced finalizations.

The mirror used three conditions:

- **baseline recovery:** recovery guidance without the candidate driver;
- **forced runtime action:** incomplete-artifact guidance plus stall detection and forced append/finalize actions; and
- **wording only:** the same guidance and stall detection as the forced condition, but prose re-prompting instead of forced tool selection.

The forced-versus-wording contrast isolates forced selection more directly than the forced-versus-baseline contrast, which changes both wording and runtime action.

3 Experiments

3.1 Runtime-mirror mechanism ablation

The mirror crossed three fixed synthetic task families—a long-form document, a code module, and structured notes—with the three conditions above. Each cell had three sequential stochastic repetitions, for 27 runs total. Run order was not randomized and no explicit seed was set. All conditions used the same tool schemas, byte-reporting result shapes, history compaction, and 4,096-token output cap. The largest observed prompt was 9,101 tokens within a 49,152-token context window.

The runner explicitly set temperature 0.2. Its top-p and top-k values were not captured. The exact model checkpoint, checkpoint digest, tokenizer revision, and serving-engine revision were also not captured. We therefore call the serving target **M1**, an unidentified local checkpoint, rather than treating its stale service alias as a scientific model identifier.

The three tasks are fixed cases, not a sample from a task population. Repetitions may also share unrecorded serving state. We consequently show every task-condition cell and pooled descriptive counts without confidence intervals or hypothesis tests.

3.2 Pathway-fidelity audit

After the ablation, we traced output-cap handling across the streamed response, parser, orchestration loop, and workspace dispatch. The audit found that the mirror could recover a partial mutation by materializing incomplete tool arguments before the boundary used by the runtime path. The actual stream path could

instead yield an absent or structurally incomplete argument requiring a smaller first-chunk retry. The mirror reproduced result shapes but not reachability of the transition that created them.

Persisting partial bytes changes every later stall, append, and finalization decision. We therefore classify all 27 ablation runs as **mechanism evidence in the mirror**, not as an estimate of the driver's effect in the runtime.

3.3 Faithful parser/loop runtime-path validation

The follow-up exercised the actual stream parser, forced-tool translation, and orchestration loop at a 4,096-token output cap. It used byte-reporting in-memory workspace and event-persistence doubles; it did not exercise production storage, cross-process persistence, or resume behavior. This is a runtime-path validation, not a full-system validation.

The validation contained nine small-artifact runs with the driver enabled, nine with it disabled, and nine large-artifact runs with the driver enabled. The small task checked whether a truncation-gated driver remained inert when no target-path truncation was observed. The large task checked whether target-path cap events were latched and whether bytes existed after recovery. It had no driver-disabled arm, so it cannot estimate a causal driver effect.

This runner also targeted M1's stale serving alias, but the immutable checkpoint and serving state were not preserved. It did not explicitly set sampling temperature, top-p, or top-k. We cannot verify that the mirror and runtime-path studies used identical model or decoding state, despite targeting the same alias on the same date.

4 Results

4.1 The forced condition separated inside the mirror

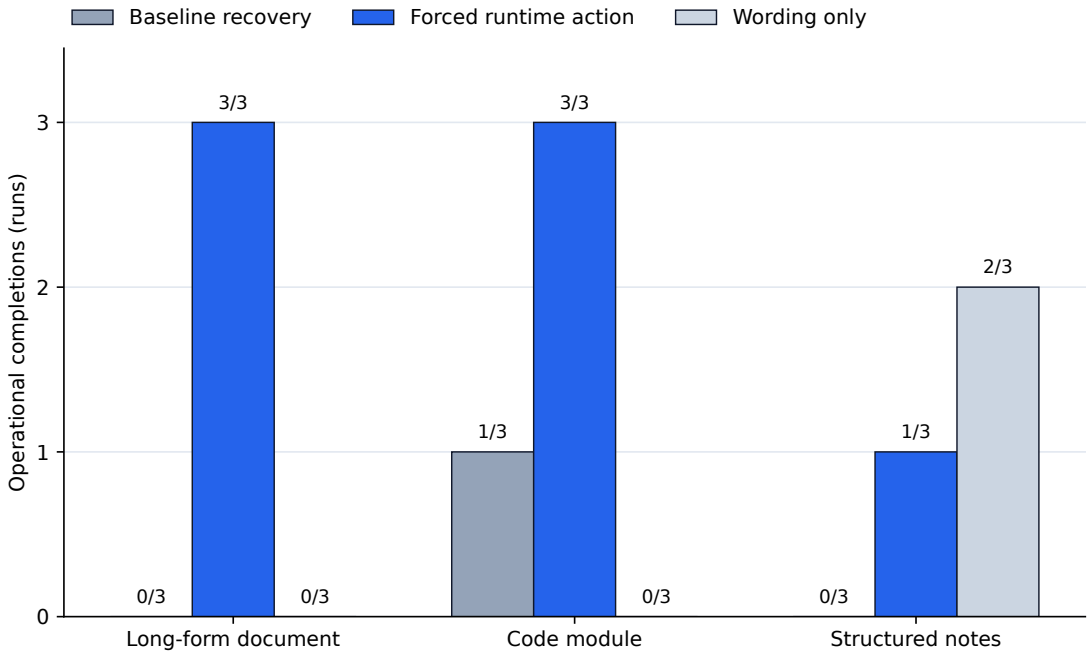

Figure 1: Operational completions for every fixed task-condition cell in the 27-run runtime-mirror ablation.

Fixed task family	Baseline recovery	Forced runtime action	Wording only
Long-form document	0/3	3/3	0/3
Code module	1/3	3/3	0/3
Structured notes	0/3	1/3	2/3

Fixed task family	Baseline recovery	Forced runtime action	Wording only
Pooled descriptive count	**1/9**	**7/9**	**2/9**

The forced condition produced 7/9 operational completions, compared with 1/9 for baseline and 2/9 for wording only. At the fixed-task level, forcing was 3/3 on the long-form document and code module but only 1/3 on structured notes. Wording only was 0/3, 0/3, and 2/3 respectively. The heterogeneity is visible and is not reduced to an inferential interval.

Because prompt text matched between forced and wording-only conditions, this pattern is consistent with forced selection being an active ingredient **inside the mirror**. It is not a population estimate. The structured-notes result is also an important guardrail: all three forced runs finalized, but only one reached the 4,096-byte floor. The study provides no evidence that any finalized artifact was semantically correct.

4.2 Runtime-path execution narrowed the claim

Scenario	Outcome criterion	Outcome met	Explicit finalize	Driver engaged	Target-path truncation latched	Wall timeout
Small-artifact control, driver off	runtime reported completed	9/9 runs	not collected	n/a	not collected	0/9 runs
Small-artifact control, driver on	runtime reported completed	9/9 runs	not collected	0/9 runs	0/9 runs	0/9 runs
Large-artifact truncation, driver on	Python AST parses and at least 4,096 bytes	6/9 runs	3/9 runs	3/9 runs	8/9 runs	5/9 runs

With the driver enabled, the runtime reported all nine small-artifact controls as completed and the driver engaged zero times; it also reported all nine disabled controls as completed. Explicit finalization was not collected for this control, and target-path truncation was not collected in the disabled arm. This is an inertness observation for one control task, not a general zero-collateral claim.

In the large-artifact arm, eight of nine **runs** latched at least one target-path truncation. Bytes existed after recovery in all eight latched runs. This is a run-level association: the aggregate neither counts every individual truncation event nor shows that every event recovered successfully. Six of nine runs produced a Python file that parsed and exceeded 4,096 bytes; three finalized, three used a forced action, and five reached the wall-time limit. Without a matched driver-off arm, none of these counts estimates the effect of forcing.

5 Discussion

5.1 Result-shape fidelity is insufficient

The mirror returned the same byte counters as the workspace test double used in the parser/loop validation, yet it was still unfaithful. The divergence occurred earlier: which partial representation survived stream parsing and who could persist it. A fidelity checklist for tool-using agent experiments should trace:

1. the model's streamed termination signal;

2. partial argument parsing and structural repair;
3. the boundary at which a mutation becomes dispatchable;
4. persisted artifact state after an interrupted action;
5. the observation consumed by the next model turn; and
6. storage, snapshot, and resume behavior when claims cover those boundaries.

A harness that matches only tool schemas and final result JSON can disagree at any earlier transition.

5.2 Runtime forcing is a policy intervention

Forced selection can convert an unproductive inspection loop into an append or finalize action, but it also constrains model autonomy. A production policy should remain bounded, gated on observed mutation state, and paired with independent syntax, semantic, and task-specific validators. Its metrics should separately count engagements, latched truncation runs, individual recovery events, premature finalizations, wall-time exhaustion, and validator failures.

5.3 Evidence hierarchy

The study suggests a practical hierarchy for runtime claims:

1. a component test establishes decision logic;
2. a mirror experiment explores a mechanism;
3. a pathway audit verifies transition reachability;
4. a matched runtime-path experiment estimates mechanism effect;
5. a system test adds persistent workspace and resume boundaries; and
6. a broad matched suite, recording mechanism exposure, assesses external validity.

A large unrelated regression suite cannot replace exposure-aware measurement of the mechanism. We therefore exclude whole-suite pass rates from this paper.

6 Provenance and reproducibility

The public package contains two aggregate CSV files, structured claims, a complete record of preserved configuration and explicit unknowns, and a SHA-256 provenance lock for six retained private inputs: the two runners, two reports, and two fixed task definitions. It also records two full core revisions described by the runtime- path report. The adapter revision was not captured, and both runners were retained outside version control; their content digests, rather than a Git revision, are the immutable identifiers.

The validator enforces exact schemas, unique task/condition and scenario/driver keys, fixed sample sizes, count ranges, byte-landing counts no greater than latched- truncation run counts, structured-claim equality, configuration/source consistency, and current generated assets. Authorized reviewers can supply a private source map and verify every retained input against the public digest lock. Run:

```
python research-data/convergence/generate.py
python research-data/convergence/generate.py --check
python research-data/convergence/generate.py \
  --verify-private-sources /path/outside/repository/source-map.json
```

The first two commands reproduce public arithmetic, tables, and vector figures. The third verifies private-source integrity for an authorized reviewer. Exact study replication is not possible from this package: the checkpoint and several serving parameters were never captured, while prompts, runners, and traces are not public. The package makes that limitation machine-readable instead of substituting a dated description for provenance.

7 Threats to validity and limitations

- The mirror used three fixed task families and three non-randomized, unseeded repetitions per cell. No independent-trial or task-population inference is made.
- Operational completion in the mirror was only finalization plus a byte floor; semantic correctness and code syntax were not checked.

- The mirror had a pathway-fidelity defect. Its 1/9, 7/9, and 2/9 counts must not be represented as runtime treatment outcomes.
- The faithful large-artifact path test had no driver-off arm, and five of nine runs reached a wall-time limit. Outcome and censoring are entangled.
- The faithful path test used actual parser/loop components but in-memory workspace and persistence doubles. Cross-process persistence and resume were not exercised.
- The exact M1 checkpoint, serving revision, tokenizer revision, and several decoding parameters were not captured. Equality across studies is unverifiable.
- Public artifacts contain aggregates rather than raw traces. Authorized reviewers can verify source integrity, but readers cannot independently reanalyze traces.

8 Recommended confirmatory design

A confirmatory study should randomize matched driver-on and driver-off conditions within every task, immutable model checkpoint, output cap, and recorded seed. It should pre-register artifact validators and wall-time handling, collect enough tasks and repetitions for task-clustered analysis, and record both run-level truncation exposure and individual recovery events. The path should include persistent workspace and resume behavior. Forced finalization should count as task success only when syntax, semantic, and task-specific validators pass.

9 Conclusion

Runtime-driven forcing produced a descriptive separation in a controlled mirror: 7/9 operational completions versus 1/9 for baseline recovery and 2/9 for identical wording without forcing. The task-level cells reveal substantial heterogeneity, and operational completion did not establish artifact correctness. A later audit showed that the mirror admitted a partial-write transition unavailable to the parser and runtime path. The correct response is to narrow the claim.

Faithful parser/loop validation showed that truncation-gated logic remained inert in the tested small control and that bytes landed after recovery in every run that latched a target-path truncation. It did not establish recovery of every truncation event or the causal benefit of forcing. For output-capped tool-using agents, simulation fidelity is a property of reachable state transitions, not merely matching prompts, tools, and result shapes.

Jimenez, Carlos E., John Yang, Alexander Wettig, et al. 2024. "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" *International Conference on Learning Representations*. https://openreview.net/forum?id=VTF8yNQM66.

Liu, Xiao, Hao Yu, Hanchen Zhang, et al. 2024. "AgentBench: Evaluating LLMs as Agents." *International Conference on Learning Representations*. https://openreview.net/forum?id=zAdUB0aCTQ.

Sargent, Robert G. 2013. "Verification and Validation of Simulation Models." *Journal of Simulation* 7 (1): 12–24. https://doi.org/10.1057/jos.2012.20.

Schick, Timo, Jane Dwivedi-Yu, Roberto Dessì, et al. 2023. "Toolformer: Language Models Can Teach Themselves to Use Tools." *Advances in Neural Information Processing Systems* 36. https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Yao, Shunyu, Jeffrey Zhao, Dian Yu, et al. 2023. "ReAct: Synergizing Reasoning and Acting in Language Models." *International Conference on Learning Representations*. https://openreview.net/forum?id=WE_vluYUL-X.