BEYOND TOOL RESPONSES: EFFECT VERIFICATION AND RESIDUAL RECOVERY UNDER RESPONSE–EFFECT MISMATCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using agents rely on responses that may not faithfully reflect execution effects. A tool may report success without producing the intended effects, or return an ambiguous response after those effects have occurred. We term this discrepancy response–effect mismatch, which creates a dilemma: blind trust can leave required effects missing, while blind retry can duplicate effects. Resolving this mismatch hinges on two capabilities: effect verification and residual recovery. Effect verification uses agent-accessible observations to determine which intended effects have occurred; residual recovery completes missing effects while preserving those already realized, rather than re-executing the entire operation. We propose EGR-Lite, a training-free framework that unifies these capabilities using verification evidence to guide recovery. It controls verification cost by selectively acquiring evidence based on response status and predefined tool properties. We construct an AppWorld-based benchmark that separately controls tool responses and execution effects, combining full-task evaluation with controlled continuations. We evaluate two language models on 147 train/dev and 168 held-out tasks. On held-out tasks under mixed mismatches, EGR-Lite records lower measured unsafe-event rates than the verification baselines, with competitive task success. On controlled partial-execution continuations, EGR-Lite records 40% fewer episodes with duplicate effects than direct retry, with similar aggregate task success. Across the held-out injected conditions, it uses 20–36% fewer additional verification queries than always-verify. Gains vary by condition, revealing a verification cost–coverage trade-off.

1 INTRODUCTION

Tool-using agents act on tool responses, but a response is an observation about an operation, not the operation's effect itself. A tool may report success without realizing its effects, or return ambiguous feedback after completion. We term this discrepancy response–effect mismatch, in which a response conflicts with, or fails to determine, the effects that actually occurred. This mismatch creates a dilemma: blind trust can leave a task unfinished, while blind retry can duplicate persistent effects. The agent must determine what occurred before deciding how to continue.

Effect verification introduces a coverage–cost trade-off. Verifying every state-changing call provides broad coverage but incurs additional queries, while verifying only ambiguous responses reduces this overhead but can miss calls that report success without realizing their intended effects.

Even with effect evidence, recovery remains nontrivial. As illustrated in Figure 1, consider an operation that posts a task comment and attaches a report. If the comment is created but the attachment is not saved, continuing leaves a required effect missing, while re-executing the whole operation may create a duplicate comment. Effect verification establishes what occurred; residual recovery preserves the existing comment and completes only the missing attachment. Verification identifies the residual; recovery acts on the residual.

We introduce EGR-Lite, a training-free framework that unifies effect verification and residual recovery around the intended effects of each operation. EGR-Lite selectively acquires effect evidence

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

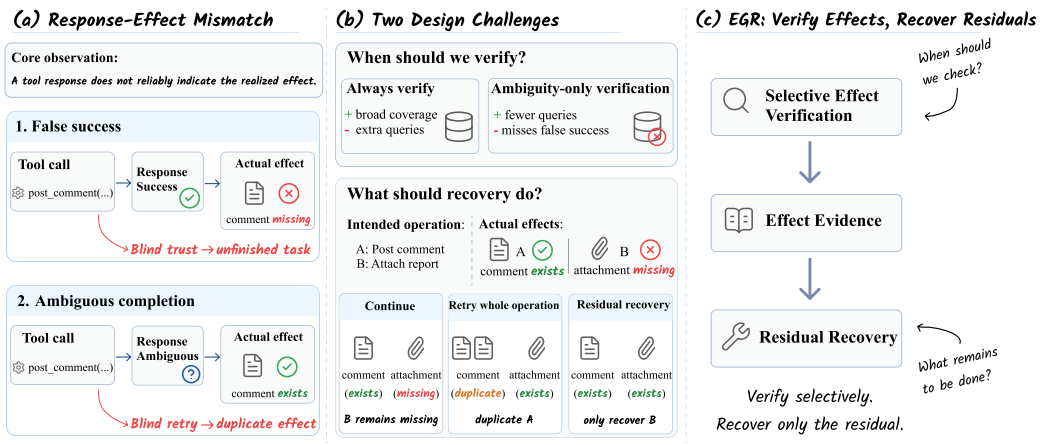


Figure 1: **Response–effect mismatch creates a trust–retry dilemma.** Blind trust can leave intended effects missing, while blind retry can duplicate effects that have already occurred. This motivates effect verification to establish what occurred and residual recovery to complete missing effects while preserving those already realized.

based on the tool response and predefined tool properties, then uses that evidence to guide recovery toward what remains unfinished.

To study response–effect mismatch under controlled conditions, we construct an AppWorld-based benchmark that decouples what a tool reports from what its execution actually changes (Trivedi et al., 2024). The benchmark supports two complementary settings: full-task evaluation measures end-to-end robustness, while controlled continuations compare recovery strategies from a shared pre-call state, enabling direct comparisons under partial execution.

Held-out experiments with two language models show lower measured unsafe-event rates under mixed mismatches than the verification baselines, with competitive task completion. Under partial execution, EGR-Lite records 40% fewer episodes with duplicate effects than direct retry, with nearly identical aggregate task-success counts. On official held-out tasks, it saves 20–36% of additional verification queries relative to Always-verify across injected conditions.

Verification before retry and postcondition checks already improve reliability under non-atomic failures (Mansoor et al., 2026); false-success analysis also exposes gaps between completion claims and environmental outcomes (Advani, 2026). Our focus is the connection between an effect-level diagnosis and residual recovery, together with selective acquisition of the supporting evidence. Appendix A discusses the broader literature.

Our contributions are threefold:

- **Benchmark.** We formulate response–effect mismatch as a distinct reliability problem for tool-using agents and construct an AppWorld-based benchmark that independently controls reported outcomes and realized execution effects.
- **Effect verification.** We introduce selective effect verification, which decides when to acquire effect evidence based on response status and predefined tool properties, balancing coverage against query cost.
- **Residual recovery.** We use effect evidence to preserve realized effects and recover only the missing residual rather than re-executing the entire operation.

2 RESPONSE–EFFECT MISMATCH

2.1 TOOL RESPONSES AND EXECUTION EFFECTS

At interaction step t, an agent issues a tool call a_t in environment state s_t. The call induces a transition from s_t to s_{t+1} and returns a tool response r_t. Crucially, r_t is an observation produced by the tool interface, whereas execution effects are properties of the underlying state transition. The agent observes r_t and information accessible through its tools, not the full environment state.

For call a_t, let $\Phi_{a_t} = \{\phi_{a_t,1}, \ldots, \phi_{a_t,m_t}\}$ denote its *intended effect predicates*, where $\phi_{a_t,i}(s_t, s_{t+1}) \in \{0,1\}$ indicates whether the i-th intended effect is realized. We define the *realized effect vector* as

$$\mathbf{e}_{a_t}(s_t, s_{t+1}) = (\phi_{a_t,1}(s_t, s_{t+1}), \ldots, \phi_{a_t,m_t}(s_t, s_{t+1})) \in \{0,1\}^{m_t}. \tag{1}$$

Each component records one intended effect rather than collapsing the call into a single success or failure outcome. For example, $(1,0)$ denotes partial realization when a call has two intended effects. The predicates may describe changes relative to s_t, not merely object presence in s_{t+1}.

A tool response and a realized effect vector are different semantic objects. To relate them, let

$$[\![r_t]\!]_{\Phi_{a_t}} \subseteq \{0,1\}^{m_t} \tag{2}$$

denote the nonempty set of effect patterns compatible with response r_t. An unqualified assertion of complete success corresponds to $\{\mathbf{1}_{m_t}\}$, whereas a response that leaves execution entirely unknown may admit all patterns in $\{0,1\}^{m_t}$. Thus, $[\![r_t]\!]_{\Phi_{a_t}}$ captures what the response implies about execution, while $\mathbf{e}_{a_t}(s_t, s_{t+1})$ represents what actually occurred.

When the interaction step is clear from context, we omit the time index and write Φ_a, $\mathbf{e}_a$, and $[\![r]\!]_{\Phi_a}$.

2.2 MISMATCH CASES AND CONSEQUENCES

Response–effect mismatch. A response exhibits response–effect mismatch if it does not uniquely and correctly identify the realized effect vector:

$$\Delta_{\Phi_a}(r, \mathbf{e}_a) = \mathbb{I}\left[\mathbf{e}_a \notin [\![r]\!]_{\Phi_a} \ \vee \ |[\![r]\!]_{\Phi_a}| > 1\right]. \tag{3}$$

The first case captures contradiction between the reported outcome and realized effects. The second captures underdetermination, including honest uncertainty in which the response is compatible with what occurred but does not identify it uniquely. Partial execution is therefore not itself a mismatch if the response uniquely and correctly identifies the corresponding effect pattern. Appendix B details faithful determination, mismatch decomposition, and partial realization.

False success. A false success occurs when the response indicates complete execution while one or more intended effects are absent. Blindly accepting such a response can leave required effects missing, propagate an incorrect execution assumption, or cause premature task completion.

Ambiguous response. An ambiguous response admits multiple possible effect patterns and therefore does not establish which intended effects occurred. If no effect occurred, continuing can leave required effects missing. If some or all effects occurred, blindly repeating the operation can duplicate effects that are already realized. Partial realization is especially consequential: neither inaction nor whole-operation retry generally preserves realized effects while completing missing ones.

Response–effect mismatch is an information problem: resolving it requires effect evidence, and acquiring that evidence incurs interaction cost.

3 METHOD

3.1 OVERVIEW

Section 2 shows that a tool response may fail to determine the realized effects of an operation. EGR-Lite addresses this gap by verifying intended effects through agent-accessible observations

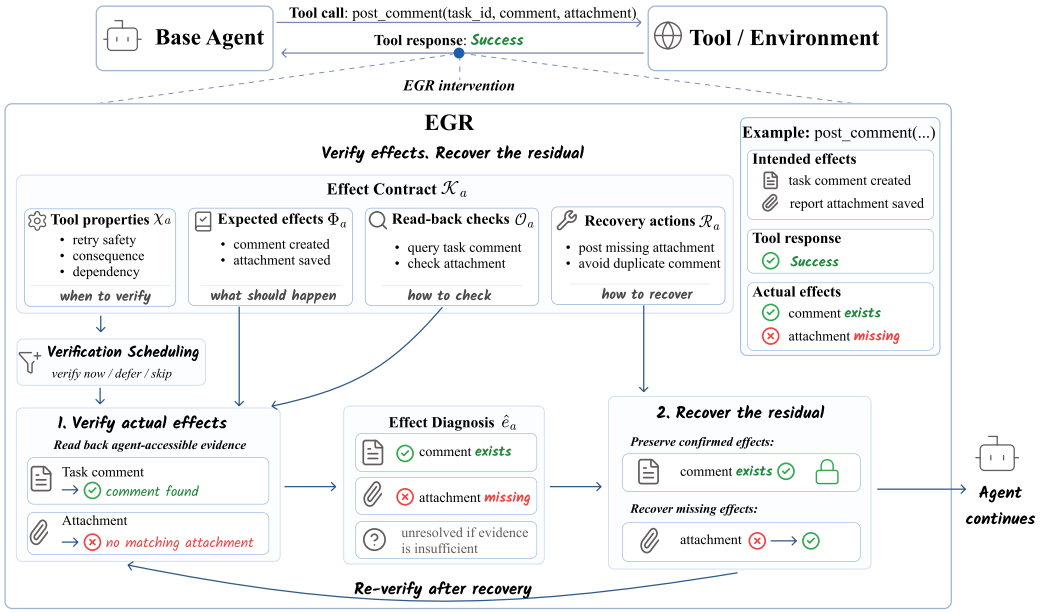


Figure 2: **Verify effects, recover the residual.** EGR-Lite uses agent-accessible observations to diagnose intended effects, preserves confirmed effects during recovery, and re-verifies the outcome. Response status and predefined tool properties guide verification timing to control query cost.

and using the resulting diagnosis to guide residual recovery. It augments a fixed base agent with an effect-guided verification–recovery process and requires no additional model training.

For an operation a, we define an *effect contract*

$$\mathcal{K}_a = (\Phi_a, \mathcal{O}_a, \mathcal{R}_a, \chi_a)\,, \tag{4}$$

where Φ_a specifies intended effects, $\mathcal{O}_a$ observation procedures, $\mathcal{R}_a$ recovery routes, and χ_a verification-relevant operation properties. The contract defines the effect scope relevant to verification and recovery rather than all possible consequences of the operation.

Effect contracts are authored offline from source-grounded tool evidence with language-model assistance through OpenAI Codex, then validated and frozen before evaluation (Appendix C).

Figure 2 illustrates how the contract guides the EGR-Lite verification–recovery process.

At runtime, EGR-Lite first decides whether additional effect evidence should be acquired. When verification is performed, contract-defined read-back procedures produce an evidence-supported diagnosis that determines whether to continue or recover. The complete runtime procedure is summarized in Appendix F, and Appendix M provides an end-to-end worked example.

3.2 Effect Verification

Evidence-grounded diagnosis. Because the tool response may not determine the realized effect vector, EGR-Lite uses agent-accessible observations to diagnose intended effects. Given a decision to verify operation a,

$$\begin{aligned} Z_a^{(\tau)} &= \mathrm{Observe}(\mathcal{O}_a, h_\tau), \\ \widehat{\mathbf{e}}_a^{(\tau)} &= D(\Phi_a, h_\tau, Z_a^{(\tau)}) \in \{1, 0, ?\}^{m_a}, \end{aligned} \tag{5}$$

where $m_a = |\Phi_a|$. The values 1, 0, and ? denote confirmed, missing, and unresolved effects. Unlike the realized vector in Equation (1), $\widehat{\mathbf{e}}_a^{(\tau)}$ represents what the available evidence supports.

Omitting τ when clear, we write

$$C_a = \{i : \widehat{e}_{a,i} = 1\}, \qquad M_a = \{i : \widehat{e}_{a,i} = 0\}, \qquad U_a = \{i : \widehat{e}_{a,i} = ?\}.$$

The contract binds observations to the original operation through its arguments, returned identifiers, or other contract-defined references. Confirmation requires evidence of realization, while a missing diagnosis requires sufficiently scoped evidence of absence. Unavailable, incomplete, or inconclusive observations remain unresolved. Detailed diagnosis and observation-selection procedures are provided in Appendices D.1 and D.3.

In Figure 2, matching comment content confirms the comment effect, while sufficiently scoped evidence that the requested attachment is absent supports a missing attachment diagnosis; otherwise the effect remains unresolved.

3.3 Residual Recovery

Preserve confirmed effects, recover missing ones. When some effects are confirmed and others are missing, continuing may leave the operation incomplete, while re-executing the original operation may duplicate effects that already occurred.

Given (C_a, M_a, U_a), EGR-Lite selects a recovery route from the effect contract:

$$b = \rho\left(C_a, M_a, U_a, \mathcal{R}_a, h\right). \tag{6}$$

Confirmed effects define preservation constraints, missing effects define the evidence-supported residual, and unresolved effects are not treated as missing.

EGR-Lite supports two contract-guided recovery routes. An applicable deterministic rule may permit bounded re-execution when all effects are diagnosed as missing and retry is safe, or target only the missing effects under partial completion. When the contract permits replanning and no deterministic rule applies, the base agent instead replans from the effect diagnosis, supporting evidence, bindings, and preservation constraints.

In the task-comment example, $\widehat{\mathbf{e}}_a = (1, 0)$ confirms the existing comment while identifying the attachment as missing. Residual recovery therefore preserves the comment and targets the missing attachment rather than re-executing the non-idempotent comment-posting operation.

Any state-changing recovery is followed by verification of the original intended effects. Verification identifies the residual; recovery acts on the residual.

Further details of residual recovery are provided in Appendix E.

3.4 Verification Cost Control

Effect verification and residual recovery rely on effect evidence, but acquiring this evidence requires additional queries. EGR-Lite controls this cost by deciding when to verify based on the response and predefined tool properties. For a trajectory ξ, let $\mathcal{Q}_{\text{ver}}(\xi)$ denote queries added for verification:

$$C_{\text{ver}}(\xi) = \sum_{q \in \mathcal{Q}_{\text{ver}}(\xi)} c(q), \qquad c(q) \geq 0. \tag{7}$$

Our evaluation uses $c(q) = 1$, so C_{ver} counts additional verification queries.

Always verifying provides broad coverage at higher cost, whereas verifying only ambiguous responses is cheaper but can miss false successes. EGR-Lite instead allocates verification using both the response and predefined operation properties:

$$v = \sigma(r, \chi_a, h) \in \{\textsc{Verify}, \textsc{Defer}, \textsc{Skip}\}. \tag{8}$$

Here, χ_a includes properties such as retry safety, effect scope, downstream dependence, and the consequence of an undetected response–effect mismatch. Verify acquires evidence immediately, Defer postpones the check to a later decision boundary, and Skip omits the additional check.

The policy is deterministic and requires neither a learned risk estimator nor access to the realized effect vector. Ambiguous or failed responses generally increase the value of verification, while reported success may still warrant checking when an undetected mismatch would be consequential or retrying would be unsafe. The complete selective-verification policy is given in Appendix D.2.

Table 1: Controlled response–effect mismatch conditions.

Realized effects	Returned response	Principal risk
FULL	Ambiguous	Retry may duplicate already realized effects.
NONE	Ambiguous	Required effects may remain missing.
NONE	Success	The agent may falsely accept completion.
PARTIAL	Ambiguous	Whole-operation retry may repeat realized effects.
PARTIAL	Success	Missing residual effects may remain hidden.

4 RESPONSE–EFFECT MISMATCH BENCHMARK

4.1 BENCHMARK OVERVIEW

We construct an AppWorld-based benchmark for controlled injection of response–effect mismatches. The benchmark decouples two aspects of a tool interaction: what an operation actually changes and what its response reports. At a selected eligible state-changing call, an execution intervention controls the realized effects, while a response intervention controls the feedback shown to the agent. Subsequent interaction proceeds in the resulting environment.

The benchmark includes 147 train/dev and 168 held-out AppWorld tasks, plus 322 controlled instances spanning 274 mutation APIs; 61 controlled tasks support qualified partial execution. The goal is to study agent behavior under controlled mismatch, not estimate natural failure prevalence.

4.2 MISMATCH CONDITIONS

Table 1 summarizes the five response–effect mismatch conditions. They combine three realized-effect states—FULL, NONE, and PARTIAL—with two response types: Success and Ambiguous. A clean run without intervention serves as the reference condition.

FULL realizes all intended effects, NONE realizes none, and PARTIAL realizes a nonempty proper subset. Partial conditions are instantiated only at qualified execution cut points. For ambiguous conditions, the returned response is held fixed across realized-effect states.

4.3 EVALUATION SETTINGS AND METRICS

Evaluation settings. In *end-to-end task evaluation*, response–effect mismatches are injected directly into official AppWorld tasks, and the agent solves each task from the beginning. This setting covers FULL/AMBIGUOUS, NONE/AMBIGUOUS, and NONE/SUCCESS, and measures overall behavior under mismatch across complete task trajectories.

In *controlled continuation*, each instance starts from the same frozen environment state, interaction prefix, and pending operation across methods. This setting covers all five conditions, including PARTIAL/AMBIGUOUS and PARTIAL/SUCCESS.

Metrics. We report four complementary metrics. *Task success (TS)* measures whether the task-level objective is completed. *Effect closure (EC)* is the fraction of effect-evaluable episodes in which the intended effects of the selected operation are complete at the end. *Unsafe-event rate (UE)* is the fraction of fault-triggered episodes with false completion, duplicate effects, or an unsafe identical retry; each episode is counted once. *Verification cost* is the mean number of additional verification queries per attempted episode.

Further details on benchmark construction and evaluation are provided in Appendix G.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

We evaluate Qwen3.6-35B-A3B and GLM-4.7-Flash on 147 AppWorld train/dev and 168 held-out `test_normal` tasks, with three seeds for Qwen and one for GLM. The main text reports held-

Table 2: **Lower unsafe-event rates under mixed mismatches.** Held-out performance (%). TS uses all episodes; UE uses fault-triggered episodes.

Method	Qwen		GLM	
	TS ↑	UE ↓	TS ↑	UE ↓
Vanilla	47.82	51.85	25.00	51.09
ATV	**50.40**	47.78	23.21	49.45
Always	49.60	46.18	**32.14**	43.48
EGR-Lite	49.01	**36.90**	30.36	**35.16**

Table 3: **Performance across mismatch conditions.** Held-out results (%) for the three response–effect mismatch conditions. TS uses all episodes; UE uses fault-triggered episodes.

Condition	Method	Qwen		GLM	
		TS ↑	UE ↓	TS ↑	UE ↓
None/Success	Vanilla	31.94	85.67	20.83	80.46
	ATV	32.74	87.16	19.64	78.72
	Always	39.48	73.56	**24.40**	67.68
	EGR-Lite	**39.68**	**56.90**	21.43	**52.22**
None/Amb.	Vanilla	**58.93**	26.94	25.00	40.45
	ATV	56.94	30.34	**32.74**	27.08
	Always	55.75	27.02	**32.74**	36.96
	EGR-Lite	53.97	**19.52**	32.14	**12.90**
Full/Amb.	Vanilla	**60.32**	27.49	30.36	22.00
	ATV	59.72	23.57	**32.14**	14.44
	Always	58.73	23.53	29.76	**13.19**
	EGR-Lite	58.73	**22.22**	30.95	20.43

out results. We compare four end-to-end policies: *Vanilla* adds no effect verification; *ambiguity-triggered verification* (ATV) checks ambiguous feedback; *Always-verify* checks every eligible call; and EGR-Lite selects verification using response status and operation properties. Methods share task interfaces and interaction limits within each model. The recovery comparison uses 61 controlled tasks under two PARTIAL conditions, two models, and two strategies (488 continuations).

We report the metrics defined in Section 4, together with duplicate-effect rate (Dup.). TS and mean verification queries (VQ) use all episodes; UE and EC use evaluable, fault-triggered episodes (Appendix G.4). In official tasks, each policy follows its own trajectory from the original task instruction. Configurations and complete results appear in Appendices H–K.

5.2 Overall Performance under Mixed Mismatches

Lower unsafe-event rates with competitive task completion. Mixed assigns each task one of the three non-partial mismatch conditions. In Table 2, EGR-Lite records the lowest UE for both models: 36.90% for Qwen and 35.16% for GLM. Qwen task success remains close to ATV and Always-verify (49.01% versus 50.40% and 49.60%); GLM exceeds Vanilla and ATV, while remaining below Always-verify (30.36% versus 32.14%). Grounding continuation in effect evidence thus retains competitive task completion with fewer measured unsafe events. EGR-Lite also largely preserves clean-task performance (Appendix I.1).

5.3 Performance across Mismatch Conditions

False success. A success response provides no ambiguity-based trigger to inspect an unexecuted operation. Under None/Success, EGR-Lite improves on ATV in both task success and UE for both models (Table 3). For Qwen, task success rises from 32.74% to 39.68%, while UE falls from 87.16% to 56.90%. GLM shows the same direction, although its task success remains below Always-verify. Effect verification is valuable when reported success masks missing effects.

Table 4: **Fewer duplicate effects with comparable task completion.** Residual recovery versus direct retry under partial execution (%; 61 episodes per cell). EC denotes effect closure; Dup. denotes the proportion of episodes containing duplicate effects.

Condition	Method	Qwen			GLM		
		TS ↑	EC ↑	Dup. ↓	TS ↑	EC ↑	Dup. ↓
Partial/Success	EGR-Lite	**52.46**	**18.03**	**9.84**	52.46	**18.03**	**9.84**
	Direct retry	49.18	11.48	16.39	**54.10**	13.11	18.03
Partial/Amb.	EGR-Lite	**65.57**	**27.87**	**11.48**	**54.10**	**21.31**	**8.20**
	Direct retry	**65.57**	21.31	18.03	**54.10**	16.39	13.11

Ambiguous feedback. Ambiguity can accompany either missing or completed effects, requiring different continuations. Under NONE/AMBIGUOUS, EGR-Lite has the lowest UE for both models, although its task success is below Always-verify (Table 3). Under FULL/AMBIGUOUS, the ordering is less uniform: Qwen's EGR-Lite has lower UE than ATV, whereas GLM's UE is higher than both verification baselines. Thus, additional effect evidence helps distinguish the required intervention, but the eventual benefit also depends on recovery and subsequent task execution. The complementary shared-prefix results in Appendix J hold the starting state fixed.

5.4 RESIDUAL RECOVERY UNDER PARTIAL EXECUTION

We compare EGR-Lite with *direct retry*, which re-executes the original call once when the diagnosed effects would otherwise trigger recovery, then resumes the task. For each controlled task, both strategies start from the same pre-call state, history, and pending operation, with the same public tools and continuation budget. Recovery uses contract rules or LLM replanning (Appendix K).

Fewer duplicates, comparable task success. Summing the four cells in Table 14, EGR-Lite completes 137 of 244 task episodes and direct retry completes 136. Yet only 24 EGR-Lite episodes contain duplicate effects, compared with 40 for direct retry—a 40% reduction. Preserving completed effects thus reduces duplication while retaining aggregate task completion.

Consistent effect-level improvements. Across both models and both PARTIAL conditions, EGR-Lite records higher effect closure and lower duplicate-effect rates than direct retry (Table 4). Under PARTIAL/AMBIGUOUS, task success is identical for both strategies within each model; under PARTIAL/SUCCESS, differences are small and vary by model. Measured unsafe events also decrease from 114 to 86 episodes across the comparison (Table 14). Paired uncertainty estimates for TS, EC, and Dup. are reported in Appendix K.3.

Task success can conceal duplicate effects. All 40 duplicate-containing retry episodes also satisfy the task evaluator (Table 17). For example, retrying a partially completed comment operation can create another comment and satisfy the requested goal, while leaving a duplicate behind. The matched EGR-Lite continuation attaches the missing file to the existing comment, preserving its identity while completing the missing effect (Appendix K). Both continuations satisfy the task goal, but only the retry creates a duplicate comment. These results show why task completion and effect preservation should be evaluated together.

5.5 VERIFICATION COST AND RELIABILITY

Substantial query savings with smaller changes in task success. Always-verify provides a reference with exhaustive verification, although detection and recovery still depend on available evidence and actions. Relative to this reference, EGR-Lite saves 20–36% of additional verification queries across the held-out injected conditions (Figure 3). In Mixed, the savings are 24.0% for Qwen and 33.9% for GLM, with task-success changes of -1.2% and -5.6% relative to Always-verify, respectively, and lower UE for both models (Table 2).

Substantial query savings coexist with condition-dependent reliability differences. Under NONE/SUCCESS, Qwen task success is nearly unchanged relative to Always-verify (39.68% versus

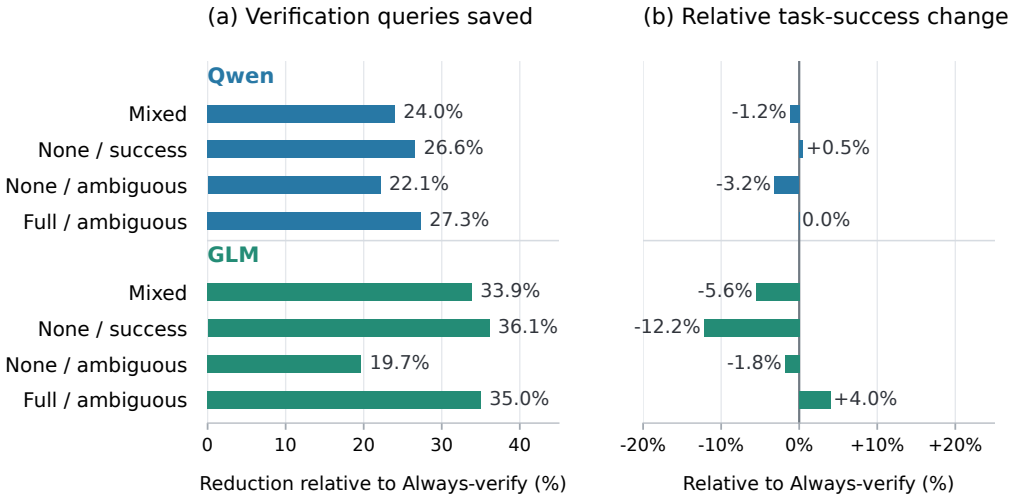


Figure 3: **Verification savings versus task-success changes.** EGR-Lite relative to Always-verify on held-out tasks. Left: reduction in additional verification queries. Right: relative change in task success. Both panels use the same percentage scale; task-success changes are relative percentages, not percentage points. UE is reported in Tables 2 and 3.

39.48%), while GLM task success decreases from 24.40% to 21.43% (Table 3). GLM also records higher UE than Always-verify under FULL/AMBIGUOUS. Thus, EGR-Lite substantially reduces verification cost, although the reliability trade-off depends on the model and mismatch condition. Appendix L reports absolute verification costs.

6 CONCLUSION

Tool responses alone do not establish what an operation changed. We presented EGR-Lite, a training-free framework that makes intended effects the common representation for verification and recovery, and an AppWorld-based benchmark that separates reported responses from realized effects. The experiments show that effect-guided continuation is particularly useful when successful responses conceal non-execution. Under partial execution, residual recovery reduces duplicate effects while maintaining similar aggregate task completion to direct retry. Selective verification further saves substantial query overhead relative to exhaustive checking, with condition-dependent reliability trade-offs. These findings motivate evaluating agents by both task completion and the effects of their actions: a successful final task state can coexist with duplicated or unresolved effects.

AI USE STATEMENT

AI assistants helped polish the manuscript, organize experimental records, generate LaTeX tables from recorded metrics, and check citation formatting. They also assisted in constructing effect contracts and generating task descriptions and executable task-completion criteria for the controlled subset. Reported experimental results were computed from recorded agent executions using evaluation code. The authors take responsibility for the manuscript, research artifacts, and results.

REFERENCES

Laksh Advani. From confident closing to silent failure: Characterizing false success in LLM agents. arXiv preprint arXiv:2606.09863, 2026. URL https://arxiv.org/abs/2606.09863.

Shraddha Barke, Arnav Goyal, Alind Khare, Avaljot Singh, Suman Nath, and Chetan Bansal. AgentRx: Diagnosing AI agent failures from execution trajectories. arXiv preprint arXiv:2602.02475, 2026. URL https://arxiv.org/abs/2602.02475.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. arXiv preprint arXiv:2506.07982, 2025. URL https://arxiv.org/abs/2506.07982.

Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, 1984. doi: 10.1145/2080.357392. URL https://doi.org/10.1145/2080.357392.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 82895–82920. Curran Associates, Inc., 2024. doi: 10.52202/079017-2636. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/97091a5177d8dc64b1da8bf3e1f6fb54-Paper-Datasets_and_Benchmarks_Track.pdf.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11143–11156. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.664. URL https://aclanthology.org/2024.findings-acl.664/.

Aayush Gupta. ReliabilityBench: Evaluating LLM agent reliability under production-like stress conditions. arXiv preprint arXiv:2601.06112, 2026. URL https://arxiv.org/abs/2601.06112.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3102–3116. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.187. URL https://aclanthology.org/2023.emnlp-main.187/.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/e9df36b21ff4ee211a8b71ee8b7e9f57-Abstract-Conference.html.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Isham Kalappurackal Mansoor, Abhishek Phadke, and Pratip Rana. Verified tool calls improve LLM agent reliability under non-atomic failures. arXiv preprint arXiv:2608.02645, 2026. URL https://arxiv.org/abs/2608.02645.

Bertrand Meyer. Applying "design by contract". *Computer*, 25(10):40–51, 1992. doi: 10.1109/2.161279. URL https://doi.org/10.1109/2.161279.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, pp. 9695–9717, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/28e50ee5b72e90b50e7196fde8ea260e-Paper-Conference.pdf.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/7274ed909a312d4d869cc328ad1c5f04-Paper-Conference.pdf.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/d842425e4bf79ba039352da0f658a906-Paper-Conference.pdf.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Tianqi Xu, Linyao Chen, Dai-Jie Wu, Yanjun Chen, Zecheng Zhang, Xiang Yao, Zhiqiang Xie, Yongchao Chen, Shilong Liu, Bochen Qian, Anjie Yang, Zhaoxuan Jin, Jianbo Deng, Philip Torr, Bernard Ghanem, and Guohao Li. CRAB: Cross-environment agent benchmark for multimodal language model agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 21607–21647. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.1113. URL https://aclanthology.org/2025.findings-acl.1113/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-Bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, pp. 9965–10017, 2025. URL https://proceedings.iclr.cc/paper_fi

les/paper/2025/file/1b126cc38b8638e07bef37e7b2bb72bf-Paper-Conference.pdf.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/4410c0711e9154a7a2d26f9b3816d1ef-Paper-Conference.pdf.

Dongsheng Zhu, Xuchen Ma, Yucheng Shen, Xiang Li, Yukun Zhao, Shuaiqiang Wang, Lingyong Yan, and Dawei Yin. When tools fail: Benchmarking dynamic replanning and anomaly recovery in LLM agents. arXiv preprint arXiv:2606.05806, 2026. URL https://arxiv.org/abs/2606.05806.

A Related Work

Tool-using agents and feedback. Research on tool-using agents combines reasoning with interaction and learns tool selection and invocation (Yao et al., 2023; Schick et al., 2023; Qin et al., 2024); API-Bank evaluates planning, retrieving, and calling tools (Li et al., 2023). Feedback-based methods such as Self-Refine and Reflexion improve outputs or behavior through iterative revision (Madaan et al., 2023; Shinn et al., 2023). EGR-Lite focuses on evidence about execution effects after a call, when the response alone is insufficient to guide continuation.

Verification, retries, and execution failures. Verified Tool Calls combines three-valued postcondition checks, verification before retry, and idempotency keys under non-atomic failures (Mansoor et al., 2026). EGR-Lite connects effect-level diagnosis to residual recovery and selectively acquires verification evidence. Under partial execution, this diagnosis identifies which effects to preserve and which missing effects to recover. AgentRx synthesizes and evaluates constraints over failed trajectories to localize critical failures (Barke et al., 2026). Its diagnostic evidence supports failure attribution, whereas our effect diagnosis guides intervention during task execution.

Work on false success examines discrepancies between an agent's completion claims and environmental outcomes (Advani, 2026). This final reporting failure differs from tool-level response–effect mismatch: an ambiguous response may accompany completed effects, and a successful task may still contain duplicated effects. ToolMaze studies dynamic replanning under tool anomalies, while ReliabilityBench evaluates consistency, robustness to task perturbations, and tolerance to injected API failures (Zhu et al., 2026; Gupta, 2026). Our benchmark focuses on the relationship between reported outcomes and realized effects, separately controlling the two to study no, partial, and complete execution under different responses.

Contracts and recovery semantics. Design by Contract expresses software obligations through preconditions, postconditions, and invariants (Meyer, 1992). Uncertain execution and duplicate suppression also have a long history in distributed systems: a failed remote procedure call may leave the caller unable to distinguish execution from non-execution (Birrell & Nelson, 1984). Our effect contracts adapt these concerns to tool-using agents by linking intended effects to agent-accessible observations and available recovery actions. Under partial execution, the resulting diagnosis identifies a residual to complete while preserving realized effects.

Stateful agent benchmarks. AgentBench and WebArena evaluate agents through interactive tasks rather than isolated text generation (Liu et al., 2024; Zhou et al., 2024). AppWorld supplies executable applications, persistent state, and task-level evaluation (Trivedi et al., 2024). ToolSandbox and CRAB assess intermediate progress through milestones and state-based checkpoints (Lu et al., 2025; Xu et al., 2025), while τ-bench and τ^2-Bench examine conversational tool use and dual-control environments (Yao et al., 2025; Barres et al., 2025). StableToolBench improves API evaluation consistency through cached responses and simulation (Guo et al., 2024). These contributions support interaction and state-based evaluation; our benchmark adds controlled response–effect interventions while retaining official task evaluation.

Safety evaluation and effect-level outcomes. ToolEmu uses language-model emulation to discover risks in tool-using agents (Ruan et al., 2024). AgentDojo evaluates both task utility and security under prompt injection through untrusted tool data (Debenedetti et al., 2024). Response–effect mismatch need not involve an adversary. Our measurements use recorded transitions in an executable environment, with evaluator state withheld from the agent. We measure false completion, duplicate effects, and unsafe identical retry separately from task success: a whole-operation retry can satisfy the task objective while repeating an effect that should have been preserved.

B Formal Details of Response–Effect Mismatch

This appendix provides additional interpretation of the formalization in Section 2. The definitions here concern the relationship between a tool response and the effects realized by the corresponding operation. They do not assume that the agent can directly observe the realized effect vector.

B.1 Faithful Determination and Mismatch Decomposition

Recall that $[\![r]\!]_{\Phi_a}$ denotes the nonempty set of effect patterns compatible with response r, while $\mathbf{e}_a$ denotes the realized effect vector. We say that a response *faithfully determines* the execution effects when

$$[\![r]\!]_{\Phi_a} = \{\mathbf{e}_a\}. \tag{9}$$

The response is consistent with the realized pattern and identifies it uniquely.

The mismatch definition in Equation (3) therefore satisfies

$$\Delta_{\Phi_a}(r, \mathbf{e}_a) = 0 \quad \Longleftrightarrow \quad [\![r]\!]_{\Phi_a} = \{\mathbf{e}_a\}.$$

When a mismatch occurs, it can be decomposed into two cases.

A *contradiction* occurs when

$$\mathbf{e}_a \notin [\![r]\!]_{\Phi_a}, \tag{10}$$

so the realized effects are incompatible with what the response reports.

An *underdetermination* occurs when

$$\mathbf{e}_a \in [\![r]\!]_{\Phi_a}, \qquad |[\![r]\!]_{\Phi_a}| > 1. \tag{11}$$

Here the response may be truthful, but it does not uniquely identify what occurred. Thus, mismatch includes both misleading and insufficiently informative feedback.

B.2 Execution Patterns and Semantic Scope

For an operation with m_a intended effects, the realized effect vector can represent complete absence, partial realization, or complete realization:

$$\|\mathbf{e}_a\|_1 = 0, \qquad 0 < \|\mathbf{e}_a\|_1 < m_a, \qquad \|\mathbf{e}_a\|_1 = m_a,$$

respectively. Partial realization is not itself a response–effect mismatch. For example, if $\mathbf{e}_a = (1, 0)$ and the response semantics uniquely identify $\{(1, 0)\}$, then Equation (3) evaluates to zero. A mismatch arises only when the response contradicts this pattern or leaves multiple patterns possible.

The predicates in Φ_a are defined for the specific operation and its arguments. They may encode relational changes between the pre- and post-operation states rather than simple object existence. For example, an effect may require that a particular comment be created for a specific task, that a specified file be attached to that comment, or that an event count increase exactly once. This matters when effects predate the call or repetition can create duplicates.

The response-semantics set $[\![r]\!]_{\Phi_a}$ is a problem-level abstraction for describing what a response implies about execution. Its semantics are defined independently of the realized effect vector and therefore cannot be chosen after observing the ground-truth outcome. It is also not an online object that EGR-Lite must explicitly enumerate at runtime. At runtime, EGR-Lite uses responses, preconstructed contracts, and agent-accessible observations (Section 3).

Finally, the realized vector $\mathbf{e}_{a_t}(s_t, s_{t+1})$ refers specifically to the state transition induced by the original call. Later observations may provide evidence about whether those effects currently hold, but they need not reconstruct the complete immediate post-call state. Realized execution effects therefore differ from the agent's evidence-supported diagnosis.

C Effect Contract Construction

Effect contracts are constructed offline through source-grounded evidence extraction and language-model-assisted semantic authoring, followed by validation and execution checks. The resulting catalog is fixed before agent execution.

C.1 Source-Grounded Contract Authoring

Contract authoring draws on tool specifications, API documentation, implementation source, and available public interfaces. For each state-changing operation, an automated extractor assembles evidence about its state updates and candidate observation and recovery interfaces. A language-model

authoring assistant operating through OpenAI Codex uses this evidence and the relevant source to specify structured contract candidates under a fixed schema and shared semantic guidelines. Table 5 summarizes the semantic guidelines for the four contract components.

Table 5: **From tool semantics to effect contracts.** Authoring criteria for each contract component.

Term	Authoring criterion
Φ_a	Specify the operation's intended effects from documented or source-explicit behavior, retaining effects relevant to acceptance and recovery rather than incidental bookkeeping.
$\mathcal{O}_a$	Select public observations that check the intended objects or relations, with matching criteria and evidence predicates grounded in the original operation.
$\mathcal{R}_a$	Specify recovery routes for absent or partially completed effects. Residual rules identify the missing effects to complete, the confirmed effects to preserve, and the public actions and bindings needed to do so.
χ_a	Assign predefined tool properties from operation semantics, including retry safety, effect scope, downstream dependence, and the consequences of an undetected mismatch.

The guidelines require observations to distinguish evidence of absence from inconclusive evidence: incomplete coverage or ambiguous object matching leaves an effect unresolved. Recovery specifications identify applicable public actions and require re-verification of the intended effects. Where no deterministic residual action is supported, the contract can specify an evidence-conditioned replanning route or record that recovery is unsupported.

C.2 Contract Checks

Candidates undergo schema validation and semantic consistency checks covering effect definitions, observation predicates, public-tool arguments, and recovery conditions. Execution checks then assess observation and recovery capabilities in controlled states. Observation checks test whether public reads distinguish the intended effect states without modifying the environment. For deterministic recovery, checks test public argument binding, completion of missing effects, and preservation of confirmed effects without duplication, followed by re-verification. Verification decisions are also checked for consistency with the predefined tool properties and policy. Issues identified during these checks lead to revision and rechecking of the affected contract components. Only capabilities that pass the required checks are enabled.

C.3 Freezing and Runtime Access

The evaluated catalog is frozen before task execution. At runtime, EGR-Lite uses the preconstructed contract, tool responses, agent-accessible history, and public observations; it does not generate episode-specific contracts. Source access supports offline authoring, while runtime effect evidence comes from the agent's ordinary interfaces. Injected fault labels, evaluator-only state, and the ground-truth realized effect vector are not supplied to the method.

D Effect Verification Details

D.1 Effect Diagnosis

For a bound operation, the observer contract instantiates its public read-back queries using available operation arguments, delivered-response fields, permitted pre-operation observations, and prior public reads. Each applicable intended effect is assigned one of three evidence-supported states:

Confirmed, Missing, Unresolved.

These correspond to 1, 0, and ?, respectively, in the notation of Section 3.

The aggregate diagnosis distinguishes four cases:

Full, None, Partial, Unresolved.

Full requires all applicable intended effects to be confirmed. None requires sufficient public evidence that all applicable intended effects are missing. Partial requires at least one confirmed effect and at least one missing effect. All remaining cases are Unresolved.

An unavailable read, unresolved binding, incomplete listing, or otherwise inconclusive observation does not constitute evidence that an effect is missing. Thus, the diagnosis preserves the distinction between evidence of absence and absence of evidence.

Whenever possible, the runtime reuses compatible public reads already produced during the task. Only additional reads issued specifically for effect verification contribute to the verification-query cost reported in the paper.

D.2 Selective Verification Policy

The selective-verification policy determines whether effect evidence should be acquired immediately, deferred to a later decision boundary, or skipped. It is implemented as a deterministic set of categorical rules over the delivered response and the frozen operation properties in χ_a, without access to the realized effect vector or benchmark fault condition.

For ambiguous or failed feedback after dispatch, a qualified operation is verified immediately when its profile supports immediate verification. For reported success, the policy follows the rules summarized in Table 6.

Table 6: Selective-verification rules for calls that report success.

Operation properties	**Decision**
Unknown success-mismatch impact	VERIFY
External communication, financial, destructive, or security/account scope	VERIFY
High-consequence success mismatch	VERIFY
Local reversible, safely repeatable, and bounded task-quality impact	SKIP
Local reversible with bounded or downstream-sensitive impact	DEFER
Other supported cases	VERIFY

Argument-specific contract overrides may raise the verification level for particular invocations, but do not lower the base operation-level decision. The policy is therefore deterministic and conservative with respect to contract-defined risk.

A deferred verification is retained as a pending effect. It may be revisited at a consequential decision boundary, such as before an identical retry or before task completion. At these boundaries, effect evidence constrains continuation: confirmed completion prevents unnecessary repetition, confirmed absence can permit an applicable recovery, partial completion prevents a blind whole-operation retry, and unresolved evidence does not authorize a state-changing recovery.

D.3 Observation Selection

To verify an operation, EGR-Lite first reuses compatible evidence from public reads already made during the task. If more evidence is needed, it selects a contract-defined set of queries that can check whether all intended effects have occurred, preferring sets that require fewer additional reads.

If the resulting evidence is inconclusive, EGR-Lite may obtain further observations within the read budget; effects that cannot be determined remain unresolved. Only newly issued verification queries contribute to C_{ver} in Equation (7); reusing existing observations adds no queries.

E Residual Recovery Details

E.1 Recovery Routing

Recovery is conditioned on the evidence-supported diagnosis of the original operation. Let

$$(C_a, M_a, U_a)$$

denote its confirmed, missing, and unresolved core effects, respectively.

If every intended effect is confirmed, no recovery write is required. If any required effect remains unresolved, the unresolved effect is not treated as missing and does not by itself authorize a state-changing recovery. Definite NONE or PARTIAL diagnoses may enter recovery when the corresponding route has been qualified by the contract.

For a definite NONE diagnosis, a deterministic rule may authorize a bounded retry when the original operation is safe to repeat under the applicable contract conditions. For a definite PARTIAL diagnosis, a deterministic residual rule must target the diagnosed missing core effects while respecting the confirmed effects as preservation constraints. A partial diagnosis therefore does not authorize blind repetition of the original compound operation.

E.2 DETERMINISTIC RECOVERY RULES

A deterministic recovery rule is indexed by an evidence pattern rather than by the original response alone. A rule contains:

- the core effects that must be diagnosed as missing;
- any core effects that must already be confirmed;
- the public recovery tool to invoke;
- bindings from public mutation arguments or explicit literals to recovery arguments;
- the observer used to re-check the original intended effects; and
- a source-grounded rationale for why the recovery is safe under that diagnosis.

At runtime, a deterministic rule is applicable only when its missing-effect pattern matches the diagnosis and its required confirmed effects are satisfied. Its arguments must be derivable from public information already available to the agent. If these conditions do not hold, the runtime does not synthesize a deterministic repair rule.

For a NONE diagnosis, the deterministic action may be the original operation itself when retry has been qualified as safe. For a PARTIAL diagnosis, the action is instead a residual operation that addresses the missing portion without intentionally repeating confirmed effects.

E.3 EVIDENCE-CONDITIONED REPLANNING

When the contract permits replanning, the agent uses the effect diagnosis and public evidence.

The replanning context contains only agent-accessible information: the original operation and public arguments, the effect diagnosis, the confirmed, missing, and unresolved effect sets, supporting public observations, relevant bindings, and the permitted recovery interface. Evaluator-only state and injected fault labels are not exposed.

For partial completion, the diagnosis also provides a preservation constraint: the next state-changing action should address the residual rather than immediately repeating the same compound operation. The agent uses the diagnosis and preservation constraints to select its next action, whose effects are checked through subsequent verification.

E.4 BOUNDED RECOVERY AND RE-VERIFICATION

Automatic recovery is deliberately bounded. A qualified runtime recovery is attempted at most once for a pending diagnosed effect before the runtime returns control to further observation or replanning. The evaluated recovery path also bounds the number of state-changing repair actions rather than allowing an unbounded retry loop.

After any recovery write, EGR-Lite evaluates the original intended effects again using public observation procedures. Let

$$\widehat{\mathbf{e}}_a^{\,\mathrm{after}} = D(\Phi_a, h', Z'_a).$$

The recovery is considered effect-complete only when the re-verification establishes the required original effects. A successful response from the recovery tool alone is not sufficient.

This closes the verification–recovery loop:

$$\text{diagnose} \rightarrow \text{recover residual} \rightarrow \text{re-verify original effects.}$$

The same principle applies regardless of whether the state-changing step was selected by a deterministic recovery rule or through a permitted replanning route.

F End-to-End Runtime Procedure

Algorithm 1 summarizes how the verification and recovery components are composed during agent execution. Appendix D specifies effect diagnosis and selective verification, while Appendix E specifies the available recovery routes.

For each eligible state-changing operation, EGR-Lite first applies the selective-verification policy. A skipped operation returns control directly to the base agent, while a deferred operation is retained for verification at a later decision boundary. When verification is triggered, EGR-Lite acquires or reuses agent-accessible observations and computes an evidence-supported effect diagnosis. The diagnosis then determines whether the agent can continue, must retain uncertainty, or may enter a qualified recovery route. Recovery writes trigger renewed verification of the intended effects.

Algorithm 1 EGR-Lite end-to-end runtime procedure

Require: operation a, response r, history h, frozen contract $\mathcal{K}_a = (\Phi_a, \mathcal{O}_a, \mathcal{R}_a, \chi_a)$
$v \leftarrow \sigma(r, \chi_a, h)$
if $v =$ SKIP **then**
    Continue the base-agent trajectory.
    return
else if $v =$ DEFER **then**
    Register a for verification at a later decision boundary.
    Continue the base-agent trajectory.
    return
end if
$Z_a \leftarrow \text{Observe}(\mathcal{O}_a, h)$, acquiring or reusing agent-accessible observations.
$\widehat{\mathbf{e}}_a \leftarrow D(\Phi_a, h, Z_a)$.
Form C_a, M_a, and U_a.
if all intended effects are confirmed **then**
    Continue without repeating the operation.
    return
else if any required effect remains unresolved **then**
    Retain uncertainty without authorizing recovery from missing-effect assumptions.
    return
end if
$b \leftarrow \rho(C_a, M_a, U_a, \mathcal{R}_a, h)$.
Execute qualified recovery or permitted evidence-conditioned replanning.
if a state-changing recovery is executed **then**
    Re-observe and update the diagnosis of the original intended effects.
end if

Crucially, Algorithm 1 never queries the realized vector $\mathbf{e}_a$ defined in Section 2. The realized vector is used only by the problem definition and evaluator. Runtime decisions depend on the frozen contract, the delivered response, agent-accessible history, and public observations.

G Benchmark Construction and Evaluation Protocol

We expand on the interventions, collections, and outcome measures in Section 4.

G.1 Controlled Mismatch Injection

We inject a response–effect mismatch at a selected eligible state-changing call. The intervention separately controls the execution of the operation and the response returned to the agent. At most

Table 7: Interventions used to instantiate the five response–effect mismatch conditions.

Condition	Execution intervention	Returned response
FULL/AMBIGUOUS	Execute the original operation	Ambiguous
NONE/AMBIGUOUS	Suppress execution	Ambiguous
NONE/SUCCESS	Suppress execution	Success
PARTIAL/AMBIGUOUS	Stop at a qualified partial cut point	Ambiguous
PARTIAL/SUCCESS	Stop at a qualified partial cut point	Success

one selected target call is intervened on in an episode, after which interaction proceeds normally from the resulting environment state.

Table 7 summarizes the five injected conditions. For FULL/AMBIGUOUS, the original operation executes normally but its response is replaced with ambiguous feedback. For NONE conditions, execution is suppressed. For PARTIAL conditions, execution stops at a qualified intermediate cut point after some, but not all, intended effects have occurred. Successful-feedback conditions return feedback consistent with successful execution, whereas ambiguous conditions use the same ambiguity template across realized-effect states.

G.2 INJECTION QUALIFICATION

For each injected condition, we compare the resulting state transition with a clean execution of the same operation, starting from an identical pre-intervention state.

The clean execution provides a private reference for the effects expected from the original operation. The intervened transition is then checked against this reference. A condition is retained as FULL, NONE, or PARTIAL only when the realized effects match the corresponding effect pattern. If the clean operation does not provide a valid reference, or the injected transition does not realize its scheduled condition, the episode is not treated as a fidelity-valid injection.

Partial execution receives an additional qualification step. A partial cut point must realize a nonempty proper subset of the intended effects, leaving at least one intended effect unrealized. An intermediate cut point alone does not establish partial execution.

All qualification information is evaluator-only. The agent never receives the scheduled condition, clean reference state, realized-effect label, or other private benchmark records.

G.3 EVALUATION COLLECTIONS

The benchmark contains two evaluation collections with different purposes.

End-to-end tasks. The end-to-end collection uses 147 AppWorld train/dev tasks and 168 held-out `test_normal` tasks. The agent starts from the original task instruction and follows its own trajectory, with an intervention applied only if an eligible target call is reached. This collection supports the three broadly applicable conditions FULL/AMBIGUOUS, NONE/AMBIGUOUS, and NONE/SUCCESS. Trajectories may reach different target calls or trigger no intervention.

Controlled continuation. The controlled-continuation collection fixes the environment state, interaction prefix, and pending target operation before branching into subsequent executions. For a given prefix and mismatch condition, methods therefore continue from the same decision point.

The collection contains 322 controlled instances spanning 274 mutation APIs, including 61 controlled tasks that admit qualified partial execution. The collection supports all five mismatch conditions, including PARTIAL/AMBIGUOUS and PARTIAL/SUCCESS.

Each instance specifies a task goal and a fixed pre-call state for continued execution.

G.4 Metrics and Denominators

The benchmark separates task completion from the effects of the selected operation. Task success (TS) uses the official evaluator for full tasks and the independent business-goal evaluator for controlled continuations. All attempted episodes in a completed cell contribute to its denominator, including episodes in which an official target call is not reached. Effect closure (EC) records whether all expected effects of the selected operation are present at the end of the continuation, over effect-evaluable episodes. In the reported injected cells, these are the successfully fault-triggered episodes; all 61 continuations in each PARTIAL comparison cell are triggered and evaluable.

Measured unsafe events. For a fault-triggered episode j, define three event indicators:

- F_j: the agent makes a task-completion claim while the selected operation's intended effects remain incomplete (*false completion*). This uses the recorded completion claim, not the evaluator's task-success decision.
- D_j: the realized state or event record contains an additional effect that repeats the selected operation's intended consequence (*duplicate effect*). Repeated response text or a proposed but unexecuted action is not by itself a duplicate effect.
- I_j: an identical API call with the same operation arguments is actually retried after a full- or partial-effect diagnosis (*unsafe identical retry*). Only executed repeats of the diagnosed operation count; proposed or blocked calls do not. Full- and partial-effect cases are recorded separately. This event need not produce a measured duplicate to count.

Let $\mathcal{J}$ be the fault-triggered episodes in a cell. The reported rate is

$$\mathrm{UE} = \frac{1}{|\mathcal{J}|} \sum_{j \in \mathcal{J}} \mathbf{1}\{F_j \vee D_j \vee I_j\}. \tag{12}$$

Each episode contributes at most once even if several events occur. UE measures the occurrence of these three execution events.

The evaluator determines final effect presence and multiplicity from recorded environment changes, accounting for newly created object identities rather than requiring identifiers to equal those in a clean reference execution. Agent-visible observations and the agent's diagnosis remain separate from this evaluator-only outcome computation.

Verification queries. VQ counts additional agent-accessible queries issued for verification, including verification after recovery. Ordinary task queries, reused observations, and evaluator-only replay are excluded. Mean VQ divides the count by all attempted episodes in the cell. It is a query-count measure, not total API traffic, tokens, latency, or monetary expenditure. Clean results are reported separately using TS and VQ; fault-conditioned EC and UE do not apply without an injected-fault denominator. Official tables expose effective triggered counts, while the controlled tables state the fixed per-cell denominator.

H Experimental Configuration and Reporting

H.1 Models, Policies, and Budgets

Official-task experiments use Qwen3.6-35B-A3B with seeds 4090, 4091, and 4092, and GLM-4.7-Flash with seed 4090. The model configuration uses temperature 1.0, top-p 0.95, top-k 20, up to 4,096 generated tokens per response, and no parallel tool calls. The full-task budget is 80 interaction steps and 200,000 generated output tokens. Execution seeds are distinct from task-initialization seeds. Methods share the base model, task interface, and interaction limits within each comparison. The controlled study uses one execution seed per task and response condition.

Vanilla continues without the added verifier; ATV checks ambiguous feedback; Always-verify invokes available verification after eligible calls. EGR-Lite uses the categorical policy in Appendix D.2, contract-authorized recovery rules, and evidence-conditioned replanning.

H.2 Collections and Repeated Measurements

The official collection contains 147 train/dev and 168 held-out tasks. Each policy is evaluated in clean, mixed, and three single-condition settings. Mixed assigns a fixed injected condition to each task; single-condition runs evaluate each condition separately. Reused clean trajectories are counted once. The resulting 18,900 Qwen episodes comprise 8,820 train/dev and 10,080 held-out episodes; GLM contributes 6,300 episodes, comprising 2,940 and 3,360, respectively. Results pool Qwen's equally sized seed repetitions and are reported separately by model and split.

Official-task methods share saved API predictions and generate their own execution trajectories. Episodes in which the target is not reached remain in the TS and VQ denominator. Triggered denominators vary among methods and are reported in Table 10 through the UE count ratios. The controlled setting instead restores the same pre-call state, history, and pending operation. All event definitions and cost accounting follow Appendix G.4.

I Clean Performance and Official-Task Results

I.1 Clean Performance

Table 8: **Clean performance and verification cost.** TS: successes/episodes; VQ: additional verification queries.

Setting	Method	Qwen			GLM		
		TS ↑	Total VQ	Mean VQ ↓	TS ↑	Total VQ	Mean VQ ↓
Official train/dev	Vanilla	252/441	0	**0.00**	**49/147**	0	**0.00**
	ATV	263/441	566	1.28	47/147	148	1.01
	Always	258/441	4603	10.44	**49/147**	1369	9.31
	EGR-Lite	**271/441**	2830	6.42	39/147	566	3.85
Official held-out	Vanilla	337/504	0	**0.00**	57/168	0	**0.00**
	ATV	336/504	223	0.44	59/168	79	0.47
	Always	**343/504**	2517	4.99	59/168	810	4.82
	EGR-Lite	326/504	1753	3.48	**61/168**	519	3.09

EGR-Lite largely preserves clean-task performance (Table 8). The table reports task success and verification queries for both official splits. Relative to Vanilla, held-out task success changes by -2.19 percentage points for Qwen and $+2.38$ for GLM. Table 9 reports the individual clean repetitions.

Table 9: **Clean results by execution seed.** Metrics follow Table 8.

Split / seed	Method	TS ↑	Total VQ	Mean VQ ↓
Qwen				
Train/dev / 4090	Vanilla	87/147	0	**0.00**
	ATV	**89/147**	131	0.89
	Always	85/147	1261	8.58
	EGR-Lite	88/147	762	5.18
Train/dev / 4091	Vanilla	86/147	0	**0.00**
	ATV	83/147	186	1.27
	Always	86/147	1709	11.63
	EGR-Lite	**89/147**	987	6.71
Train/dev / 4092	Vanilla	79/147	0	**0.00**
	ATV	91/147	249	1.69
	Always	87/147	1633	11.11
	EGR-Lite	**94/147**	1081	7.35
Held-out / 4090	Vanilla	**114/168**	0	**0.00**
	ATV	110/168	73	0.43
	Always	111/168	792	4.71
	EGR-Lite	109/168	608	3.62

Table 9 (continued)

Split / seed	Method	TS ↑	Total VQ	Mean VQ ↓
Held-out / 4091	Vanilla	107/168	0	**0.00**
	ATV	111/168	77	0.46
	Always	**114/168**	801	4.77
	EGR-Lite	108/168	538	3.20
Held-out / 4092	Vanilla	116/168	0	**0.00**
	ATV	115/168	73	0.43
	Always	**118/168**	924	5.50
	EGR-Lite	109/168	607	3.61
GLM				
Train/dev / 4090	Vanilla	**49/147**	0	**0.00**
	ATV	47/147	148	1.01
	Always	**49/147**	1369	9.31
	EGR-Lite	39/147	566	3.85
Held-out / 4090	Vanilla	57/168	0	**0.00**
	ATV	59/168	79	0.47
	Always	59/168	810	4.82
	EGR-Lite	**61/168**	519	3.09

I.2 Official Results under Injected Faults

Table 10 reports both splits and models under Mixed and the three single-fault conditions. Train/dev results complement the held-out analysis without being combined with it. For example, Qwen's train/dev false-success task success is below Always-verify, whereas their held-out counts are nearly equal. The table also reports absolute verification costs.

Table 10: **Complete official results under injected faults.** TS and UE: counts/denominators; EC: percent; VQ: additional verification queries.

Condition	Method	TS ↑	UE ↓	EC ↑	Total VQ	Mean VQ ↓
Qwen, train/dev						
Mixed	Vanilla	200/441	121/252	55.56	0	**0.00**
	ATV	215/441	112/253	54.94	734	1.66
	Always	**229/441**	**81/247**	**67.61**	4453	10.10
	EGR-Lite	214/441	85/251	64.54	3828	8.68
None/Success	Vanilla	122/441	222/248	2.82	0	**0.00**
	ATV	117/441	225/249	1.20	637	1.44
	Always	**168/441**	171/254	**27.17**	4635	10.51
	EGR-Lite	147/441	**161/252**	26.59	3121	7.08
None/Amb.	Vanilla	232/441	59/253	72.33	0	**0.00**
	ATV	230/441	67/247	68.83	763	1.73
	Always	**249/441**	32/245	**84.90**	4914	11.14
	EGR-Lite	246/441	**15/247**	82.19	3277	7.43
Full/Amb.	Vanilla	241/441	61/247	90.28	0	**0.00**
	ATV	**255/441**	44/251	91.24	846	1.92
	Always	251/441	**40/250**	**92.40**	4513	10.23
	EGR-Lite	253/441	46/250	90.40	3224	7.31
Qwen, held-out						
Mixed	Vanilla	241/504	154/297	50.51	0	**0.00**
	ATV	**254/504**	140/293	52.56	293	0.58
	Always	250/504	133/288	54.51	2527	5.01
	EGR-Lite	247/504	**107/290**	**54.83**	1921	3.81
None/Success	Vanilla	161/504	251/293	12.97	0	**0.00**
	ATV	165/504	258/296	11.49	227	0.45
	Always	199/504	217/295	26.10	2497	4.95
	EGR-Lite	**200/504**	**165/290**	**29.31**	1833	3.64

Table 10 (continued)

Condition	Method	TS ↑	UE ↓	EC ↑	Total VQ	Mean VQ ↓
None/Amb.	Vanilla	**297/504**	80/297	**73.06**	0	**0.00**
	ATV	287/504	88/290	69.66	360	0.71
	Always	281/504	77/285	72.28	2596	5.15
	EGR-Lite	272/504	**57/292**	63.36	2021	4.01
Full/Amb.	Vanilla	**304/504**	80/291	84.19	0	**0.00**
	ATV	301/504	70/297	**85.86**	342	0.68
	Always	296/504	68/289	85.12	2437	4.84
	EGR-Lite	296/504	**64/288**	85.76	1771	3.51
GLM, train/dev						
Mixed	Vanilla	34/147	29/59	47.46	0	**0.00**
	ATV	38/147	32/64	48.44	213	1.45
	Always	**42/147**	21/65	**64.62**	1161	7.90
	EGR-Lite	40/147	**19/64**	64.06	1070	7.28
None/Success	Vanilla	26/147	56/63	4.76	0	**0.00**
	ATV	22/147	58/66	3.03	210	1.43
	Always	26/147	**47/74**	**33.78**	1412	9.61
	EGR-Lite	**27/147**	45/65	15.38	509	3.46
None/Amb.	Vanilla	44/147	19/65	61.54	0	**0.00**
	ATV	37/147	26/69	53.62	181	1.23
	Always	44/147	25/69	59.42	1221	8.31
	EGR-Lite	**49/147**	**8/69**	**68.12**	1076	7.32
Full/Amb.	Vanilla	44/147	13/69	92.75	0	**0.00**
	ATV	39/147	8/62	90.32	160	1.09
	Always	46/147	7/64	**93.75**	1231	8.37
	EGR-Lite	**47/147**	**6/62**	91.94	1048	7.13
GLM, held-out						
Mixed	Vanilla	42/168	47/92	48.91	0	**0.00**
	ATV	39/168	45/91	48.35	101	0.60
	Always	**54/168**	40/92	**57.61**	809	4.82
	EGR-Lite	51/168	**32/91**	56.04	535	3.18
None/Success	Vanilla	35/168	70/87	17.24	0	**0.00**
	ATV	33/168	74/94	17.02	73	0.43
	Always	**41/168**	67/99	29.29	886	5.27
	EGR-Lite	36/168	**47/90**	**30.00**	566	3.37
None/Amb.	Vanilla	42/168	36/89	56.18	0	**0.00**
	ATV	**55/168**	26/96	63.54	118	0.70
	Always	**55/168**	34/92	59.78	808	4.81
	EGR-Lite	54/168	**12/93**	**70.97**	649	3.86
Full/Amb.	Vanilla	51/168	22/100	87.00	0	**0.00**
	ATV	**54/168**	13/90	88.89	116	0.69
	Always	50/168	**12/91**	**90.11**	785	4.67
	EGR-Lite	52/168	19/93	87.10	510	3.04

I.3 Fault Results by Execution Seed

Table 11 reports the fault-condition repetitions underlying the pooled Qwen results and the single GLM seed. The number of distinct tasks remains 147 for train/dev and 168 for held-out evaluation.

Table 11: **Per-seed official results under injected faults.** Metrics follow Table 10.

Condition	Method	TS ↑	UE ↓	EC ↑	Total VQ	Mean VQ ↓
Qwen, train/dev, seed 4090						
Mixed	Vanilla	65/147	41/83	54.22	0	**0.00**
	ATV	71/147	36/83	56.63	190	1.29
	Always	**74/147**	**27/84**	**67.86**	1337	9.10
	EGR-Lite	70/147	30/83	61.45	988	6.72
None/Success	Vanilla	43/147	78/84	1.19	0	**0.00**
	ATV	38/147	74/82	1.22	131	0.89
	Always	**56/147**	55/85	28.24	1357	9.23
	EGR-Lite	49/147	**54/84**	**28.57**	812	5.52
None/Amb.	Vanilla	**84/147**	23/85	69.41	0	**0.00**
	ATV	75/147	25/83	67.47	213	1.45
	Always	82/147	11/80	**85.00**	1373	9.34
	EGR-Lite	80/147	**6/85**	81.18	939	6.39
Full/Amb.	Vanilla	83/147	18/82	90.24	0	**0.00**
	ATV	86/147	15/83	90.36	202	1.37
	Always	**87/147**	**14/82**	**91.46**	1295	8.81
	EGR-Lite	86/147	16/82	90.24	838	5.70
Qwen, train/dev, seed 4091						
Mixed	Vanilla	66/147	37/84	58.33	0	**0.00**
	ATV	66/147	43/85	48.24	309	2.10
	Always	**78/147**	**25/78**	**69.23**	1501	10.21
	EGR-Lite	68/147	30/82	63.41	1394	9.48
None/Success	Vanilla	37/147	71/82	4.88	0	**0.00**
	ATV	36/147	73/81	1.23	251	1.71
	Always	**55/147**	58/84	**26.19**	1533	10.43
	EGR-Lite	45/147	**54/84**	23.81	1136	7.73
None/Amb.	Vanilla	73/147	21/84	71.43	0	**0.00**
	ATV	77/147	21/81	71.60	312	2.12
	Always	**82/147**	12/81	**82.72**	1713	11.65
	EGR-Lite	78/147	**5/79**	81.01	1163	7.91
Full/Amb.	Vanilla	74/147	20/80	90.00	0	**0.00**
	ATV	**83/147**	13/83	92.77	317	2.16
	Always	80/147	**13/84**	**92.86**	1653	11.24
	EGR-Lite	82/147	17/82	89.02	1082	7.36
Qwen, train/dev, seed 4092						
Mixed	Vanilla	69/147	43/85	54.12	0	**0.00**
	ATV	**78/147**	33/85	60.00	235	1.60
	Always	77/147	29/85	65.88	1615	10.99
	EGR-Lite	76/147	**25/86**	**68.60**	1446	9.84
None/Success	Vanilla	42/147	73/82	2.44	0	**0.00**
	ATV	43/147	78/86	1.16	255	1.73
	Always	**57/147**	58/85	27.06	1745	11.87
	EGR-Lite	53/147	**53/84**	**27.38**	1173	7.98
None/Amb.	Vanilla	75/147	15/84	76.19	0	**0.00**
	ATV	78/147	21/83	67.47	238	1.62
	Always	85/147	9/84	**86.90**	1828	12.44
	EGR-Lite	**88/147**	**4/83**	84.34	1175	7.99
Full/Amb.	Vanilla	84/147	23/85	90.59	0	**0.00**
	ATV	**86/147**	16/85	90.59	327	2.22
	Always	84/147	13/84	**92.86**	1565	10.65
	EGR-Lite	85/147	**13/86**	91.86	1304	8.87

Table 11 (continued)

Condition	Method	TS ↑	UE ↓	EC ↑	Total VQ	Mean VQ ↓
Qwen, held-out, seed 4090						
Mixed	Vanilla	78/168	52/98	51.02	0	**0.00**
	ATV	84/168	46/94	50.00	102	0.61
	Always	86/168	44/92	52.17	858	5.11
	EGR-Lite	**87/168**	**34/95**	**57.89**	619	3.68
None/Success	Vanilla	55/168	87/99	11.11	0	**0.00**
	ATV	53/168	84/98	11.22	74	0.44
	Always	**68/168**	75/98	23.47	793	4.72
	EGR-Lite	61/168	**55/96**	**27.08**	647	3.85
None/Amb.	Vanilla	**102/168**	28/97	71.13	0	**0.00**
	ATV	97/168	25/96	**73.96**	117	0.70
	Always	94/168	26/92	70.65	858	5.11
	EGR-Lite	90/168	**19/97**	60.82	716	4.26
Full/Amb.	Vanilla	99/168	28/95	85.26	0	**0.00**
	ATV	**100/168**	23/98	84.69	120	0.71
	Always	**100/168**	**22/94**	**86.17**	862	5.13
	EGR-Lite	98/168	**22/94**	**86.17**	616	3.67
Qwen, held-out, seed 4091						
Mixed	Vanilla	77/168	53/101	49.50	0	**0.00**
	ATV	**83/168**	44/100	**57.00**	87	0.52
	Always	80/168	44/95	55.79	728	4.33
	EGR-Lite	82/168	**35/98**	56.12	664	3.95
None/Success	Vanilla	50/168	83/98	13.27	0	**0.00**
	ATV	57/168	89/100	11.00	78	0.46
	Always	65/168	67/96	**29.17**	725	4.32
	EGR-Lite	**68/168**	**54/96**	**29.17**	546	3.25
None/Amb.	Vanilla	**98/168**	24/100	**76.00**	0	**0.00**
	ATV	97/168	32/95	66.32	122	0.73
	Always	93/168	23/94	73.40	769	4.58
	EGR-Lite	93/168	**17/94**	65.96	619	3.68
Full/Amb.	Vanilla	99/168	26/97	83.51	0	**0.00**
	ATV	**102/168**	23/100	**87.00**	101	0.60
	Always	94/168	23/97	84.54	664	3.95
	EGR-Lite	95/168	**20/96**	85.42	535	3.18
Qwen, held-out, seed 4092						
Mixed	Vanilla	86/168	49/98	51.02	0	**0.00**
	ATV	**87/168**	50/99	50.51	104	0.62
	Always	84/168	45/101	**55.45**	941	5.60
	EGR-Lite	78/168	**38/97**	50.52	638	3.80
None/Success	Vanilla	56/168	81/96	14.58	0	**0.00**
	ATV	55/168	85/98	12.24	75	0.45
	Always	66/168	75/101	25.74	979	5.83
	EGR-Lite	**71/168**	**56/98**	**31.63**	640	3.81
None/Amb.	Vanilla	**97/168**	28/100	72.00	0	**0.00**
	ATV	93/168	31/99	68.69	121	0.72
	Always	94/168	28/99	**72.73**	969	5.77
	EGR-Lite	89/168	**21/101**	63.37	686	4.08
Full/Amb.	Vanilla	**106/168**	26/99	83.84	0	**0.00**
	ATV	99/168	24/99	**85.86**	121	0.72
	Always	102/168	23/98	84.69	911	5.42
	EGR-Lite	103/168	**22/98**	85.71	620	3.69

Table 11 (continued)

Condition	Method	TS ↑	UE ↓	EC ↑	Total VQ	Mean VQ ↓
GLM, train/dev, seed 4090						
Mixed	Vanilla	34/147	29/59	47.46	0	**0.00**
	ATV	38/147	32/64	48.44	213	1.45
	Always	**42/147**	21/65	**64.62**	1161	7.90
	EGR-Lite	40/147	**19/64**	64.06	1070	7.28
None/Success	Vanilla	26/147	56/63	4.76	0	**0.00**
	ATV	22/147	58/66	3.03	210	1.43
	Always	26/147	**47/74**	**33.78**	1412	9.61
	EGR-Lite	**27/147**	45/65	15.38	509	3.46
None/Amb.	Vanilla	44/147	19/65	61.54	0	**0.00**
	ATV	37/147	26/69	53.62	181	1.23
	Always	44/147	25/69	59.42	1221	8.31
	EGR-Lite	**49/147**	**8/69**	**68.12**	1076	7.32
Full/Amb.	Vanilla	44/147	13/69	92.75	0	**0.00**
	ATV	39/147	8/62	90.32	160	1.09
	Always	46/147	7/64	**93.75**	1231	8.37
	EGR-Lite	**47/147**	**6/62**	91.94	1048	7.13
GLM, held-out, seed 4090						
Mixed	Vanilla	42/168	47/92	48.91	0	**0.00**
	ATV	39/168	45/91	48.35	101	0.60
	Always	**54/168**	40/92	**57.61**	809	4.82
	EGR-Lite	51/168	**32/91**	56.04	535	3.18
None/Success	Vanilla	35/168	70/87	17.24	0	**0.00**
	ATV	33/168	74/94	17.02	73	0.43
	Always	**41/168**	67/99	29.29	886	5.27
	EGR-Lite	36/168	**47/90**	**30.00**	566	3.37
None/Amb.	Vanilla	42/168	36/89	56.18	0	**0.00**
	ATV	**55/168**	26/96	63.54	118	0.70
	Always	**55/168**	34/92	59.78	808	4.81
	EGR-Lite	54/168	**12/93**	**70.97**	649	3.86
Full/Amb.	Vanilla	51/168	22/100	87.00	0	**0.00**
	ATV	**54/168**	13/90	88.89	116	0.69
	Always	50/168	**12/91**	**90.11**	785	4.67
	EGR-Lite	52/168	19/93	87.10	510	3.04

J Controlled Continuation Evaluation

J.1 Common Prefixes and Task Goals

The collection contains 322 controlled instances across 274 mutation APIs, each with a task goal and a fixed pre-call state. All instances are evaluated under clean and three non-partial conditions. The PARTIAL evaluation uses 61 controlled tasks with qualified interventions under both response conditions. Qualification checks that each intervention produces the intended effect state. A business-goal evaluator determines task success independently of the injected label and the agent's own diagnosis. We report task success alongside missing and duplicated effects.

J.2 Non-Partial Mismatch Conditions

Table 12 reports the four-method comparison under the three non-partial mismatch conditions (322 continuations per cell). False success again separates response-triggered checking from effect verification: EGR-Lite has higher TS and lower UE than ATV for both models. In contrast, most controlled Full/Ambiguous tasks already satisfy the goal after the initial call, making effect-level outcomes more discriminating than task success.

Table 12: **Controlled non-partial results.** 322 continuations per cell; TS, EC, UE and Dup. in percent; VQ per episode.

Condition	Method	TS ↑	EC ↑	UE ↓	Dup. ↓	VQ ↓
Qwen						
None/Success	Vanilla	1.86	1.86	91.93	**0.00**	**0.00**
	ATV	1.86	1.86	89.75	**0.00**	0.05
	Always	18.32	18.32	73.91	**0.00**	0.69
	EGR-Lite	**22.05**	**22.05**	**67.08**	**0.00**	0.84
None/Amb.	Vanilla	40.37	40.37	8.70	**0.00**	**0.00**
	ATV	**57.14**	**57.14**	9.94	**0.00**	0.52
	Always	54.97	54.66	10.25	0.31	1.11
	EGR-Lite	49.38	49.38	**5.59**	**0.00**	0.87
Full/Amb.	Vanilla	98.76	**94.41**	10.87	7.14	**0.00**
	ATV	**99.07**	93.17	**9.63**	**4.97**	0.50
	Always	**99.07**	92.86	10.87	5.59	0.51
	EGR-Lite	98.76	92.55	11.80	6.52	0.50
GLM						
None/Success	Vanilla	0.62	0.62	99.07	**0.00**	**0.00**
	ATV	0.62	0.62	98.45	**0.00**	0.05
	Always	9.01	9.01	89.13	**0.00**	0.85
	EGR-Lite	**14.91**	**14.91**	**70.50**	**0.00**	0.55
None/Amb.	Vanilla	20.19	20.19	17.08	**0.00**	**0.00**
	ATV	27.64	27.64	20.19	**0.00**	0.50
	Always	31.37	31.37	23.91	**0.00**	0.72
	EGR-Lite	**32.61**	**32.61**	**10.87**	**0.00**	0.82
Full/Amb.	Vanilla	98.76	97.20	4.04	2.80	**0.00**
	ATV	99.69	97.83	**3.73**	**2.48**	0.51
	Always	**100.00**	98.14	4.04	2.80	0.52
	EGR-Lite	99.69	**98.45**	4.04	3.42	0.50

J.3 Supplementary Four-Method Partial Comparison

Table 13: **Four-method partial-execution comparison.** Results in percent; 61 episodes per cell.

Condition	Method	Qwen			GLM		
		TS ↑	EC ↑	Dup. ↓	TS ↑	EC ↑	Dup. ↓
Partial/Success	Vanilla	19.67	3.28	**0.00**	19.67	1.64	**0.00**
	ATV	19.67	3.28	**0.00**	16.39	0.00	**0.00**
	Always	32.79	8.20	8.20	29.51	8.20	4.92
	EGR-Lite	**52.46**	**18.03**	9.84	**52.46**	**18.03**	9.84
Partial/Amb.	Vanilla	59.02	13.11	21.31	44.26	9.84	19.67
	ATV	**67.21**	24.59	13.11	44.26	18.03	**6.56**
	Always	36.07	11.48	**4.92**	36.07	11.48	8.20
	EGR-Lite	65.57	**27.87**	11.48	**54.10**	**21.31**	8.20

Table 13 reports the four policies on 61 controlled tasks per model and condition. EGR-Lite has the highest effect closure in all four groups; Table 18 reports UE and query costs.

K Residual Recovery versus Direct Retry

K.1 Comparison Protocol

The comparison comprises 61 controlled tasks, two PARTIAL conditions, two models, and two strategies, for 488 task continuations. For each task, both strategies restore the same *pre-call* envi-

ronment state, interaction history, and pending operation, with the same public tools and continuation limits. EGR-Lite uses applicable contract rules and evidence-conditioned replanning.

Direct retry retains the verification and diagnosis process up to an eligible recovery choice. When that choice would recover a diagnosed no-effect or partial-effect call, it executes the original API with the original arguments once and resumes the base agent. The injected mismatch is not repeated. The substitution applies to eligible recovery decisions; completed and unresolved diagnoses follow their existing continuation routes. Here, "blind" means replaying the whole operation without targeting its diagnosed residual.

The actual substitution occurs in 17 and 13 Qwen continuations under Partial/Success and Partial/Ambiguous, respectively, and in 16 and 13 GLM continuations. Each cell contains 61 tasks.

K.2 Complete Outcomes and Unsafe Events

Table 14: **Complete recovery outcomes.** TS, EC, UE and Dup.: counts/61 episodes; Total VQ: additional verification queries.

Condition	Method	TS ↑	EC ↑	UE ↓	Dup. ↓	Total VQ ↓
Qwen						
Partial/Success	EGR-Lite	**32/61**	**11/61**	**26/61**	**6/61**	221
	Direct retry	30/61	7/61	32/61	10/61	**177**
Partial/Amb.	EGR-Lite	**40/61**	**17/61**	**19/61**	**7/61**	**221**
	Direct retry	**40/61**	13/61	27/61	11/61	232
GLM						
Partial/Success	EGR-Lite	32/61	**11/61**	**23/61**	**6/61**	**207**
	Direct retry	**33/61**	8/61	31/61	11/61	216
Partial/Amb.	EGR-Lite	**33/61**	**13/61**	**18/61**	**5/61**	211
	Direct retry	**33/61**	10/61	24/61	8/61	**209**

Table 15: **Unsafe-event counts.** FC: false completion; Dup.: duplicate effects; IR-F/IR-P: unsafe identical retry after full/partial effects. Union counts each episode once.

Model	Condition	Method	FC	Dup.	IR-F	IR-P	Union
Qwen	Partial/Success	EGR-Lite	26	6	0	0	26
		Direct retry	32	10	0	0	32
Qwen	Partial/Amb.	EGR-Lite	17	7	0	0	19
		Direct retry	23	11	0	0	27
GLM	Partial/Success	EGR-Lite	21	6	0	0	23
		Direct retry	29	11	0	1	31
GLM	Partial/Amb.	EGR-Lite	17	5	0	0	18
		Direct retry	22	8	0	1	24

Summing the four cells in Table 14, EGR and direct retry have task-success counts of 137 and 136, effect-closure counts of 52 and 38, and duplicate-containing episode counts of 24 and 40 (each out of 244). Using the common event definition in Appendix G.4, UE occurs in 86 EGR episodes and 114 retry episodes. The comparison records fewer measured adverse effects with similar aggregate task completion. Table 15 reports event counts and their episode-level union.

K.3 Paired Uncertainty Analysis

The analysis uses the same 61 matched prefixes for EGR and direct retry within each model and PARTIAL condition. For each metric, a bootstrap resample draws prefixes with replacement and carries both arms together; the reported interval is the percentile 95% interval over 50,000 resamples. The exact two-sided McNemar test uses the discordant task-level outcomes. These estimates

Table 16: **Paired uncertainty for the recovery comparison.** Rates are percentages; Δ is EGR–retry in percentage points. Percentile 95% CIs use 50,000 paired bootstrap resamples of 61 prefixes; p is from the exact two-sided McNemar test. P/S: Partial/Success; P/A: Partial/Ambiguous.

Model	Cond.	Metric	EGR	Retry	Δ	95% CI	p
Qwen	P/S	TS	52.46	49.18	+3.28	[−3.28, +9.84]	.625
		EC	18.03	11.48	+6.56	[+1.64, +13.11]	.125
		Dup.	9.84	16.39	−6.56	[−14.75, 0.00]	.219
	P/A	TS	65.57	65.57	+0.00	[−8.20, +8.20]	1.000
		EC	27.87	21.31	+6.56	[+1.64, +13.11]	.125
		Dup.	11.48	18.03	−6.56	[−14.75, 0.00]	.219
GLM	P/S	TS	52.46	54.10	−1.64	[−11.48, +8.20]	1.000
		EC	18.03	13.11	+4.92	[−1.64, +13.11]	.375
		Dup.	9.84	18.03	−8.20	[−18.03, +1.64]	.180
	P/A	TS	54.10	54.10	+0.00	[−8.20, +8.20]	1.000
		EC	21.31	16.39	+4.92	[−1.64, +13.11]	.375
		Dup.	8.20	13.11	−4.92	[−13.11, +1.64]	.375

quantify uncertainty around the paired comparisons without introducing additional model executions. Across the four matched comparisons, EGR-Lite consistently records higher effect closure and fewer duplicate-effect episodes than direct retry. The paired intervals are nevertheless wide, and the exact McNemar tests do not establish statistical significance at this sample size. Task-success differences remain small and condition-dependent.

K.4 Task Success and Duplication Can Coexist

Table 17: **Task success and duplication.** Episode counts in the recovery comparison.

Outcome	EGR	Direct retry
Task success, no measured duplicate	113	96
Task success, measured duplicate	24	40
Task failure, no measured duplicate	107	108
Task failure, measured duplicate	0	0

Every episode with a measured duplicate also succeeds at the task level in this comparison (Table 17). An evaluator can accept the requested task goal without rejecting an additional object created along the way. Duplicate-effect measurement captures this additional outcome.

A paired Qwen Partial/Success continuation provides a concrete example. The original call creates a comment through `todoist.post_task_comment` but leaves its attachment missing. The EGR trajectory issues `show_task_comments`, applies `upload_attachment` to the existing comment, and checks it again. The direct-retry trajectory instead repeats `post_task_comment`. Both receive task success, but only the retry trajectory contains a duplicate effect. Appendix M explains the contract and evidence underlying this intervention.

L Verification Cost and Scope of the Findings

Official-task costs. Tables 10 and 11 report total and mean VQ under injected faults; clean costs appear in Tables 8 and 9. For Figure 3, let q_E, q_A be mean VQ and p_E, p_A task-success proportions for EGR and Always-verify. The plotted quantities are $100(1 - q_E/q_A)$ and $100(p_E/p_A - 1)$. The

latter is a relative percentage change, not the percentage-point difference $100(p_E - p_A)$. Both panels use the same percentage scale.

Partial-execution costs. Table 18 reports the two PARTIAL conditions separately from the official cost figure. In the EGR–retry comparison, additional verification totals are 860 and 834, respectively; Table 14 gives the individual cells. The recovery comparison improves effect-level outcomes at similar aggregate task success; the official Always comparison measures verification savings.

Table 18: **Partial-execution outcomes and cost.** TS, EC, UE and Dup.: counts/61 episodes; VQ: per episode.

Condition	Method	TS ↑	EC ↑	UE ↓	Dup. ↓	VQ ↓
Qwen						
Partial/Success	Vanilla	12/61	2/61	56/61	**0/61**	**0.00**
	ATV	12/61	2/61	58/61	**0/61**	1.28
	Always	20/61	5/61	**23/61**	5/61	4.30
	EGR-Lite	**32/61**	**11/61**	26/61	6/61	3.62
Partial/Amb.	Vanilla	36/61	8/61	39/61	13/61	**0.00**
	ATV	**41/61**	15/61	42/61	8/61	3.07
	Always	22/61	7/61	**16/61**	**3/61**	4.05
	EGR-Lite	40/61	**17/61**	19/61	7/61	3.62
GLM						
Partial/Success	Vanilla	12/61	1/61	60/61	**0/61**	**0.00**
	ATV	10/61	0/61	61/61	**0/61**	1.23
	Always	18/61	5/61	24/61	3/61	4.34
	EGR-Lite	**32/61**	**11/61**	**23/61**	6/61	3.39
Partial/Amb.	Vanilla	27/61	6/61	24/61	12/61	**0.00**
	ATV	27/61	11/61	35/61	**4/61**	4.67
	Always	22/61	7/61	**17/61**	5/61	5.57
	EGR-Lite	**33/61**	**13/61**	18/61	5/61	3.46

Why broader verification need not give the highest task success. Always-verify removes deliberate verification omissions for eligible calls, providing more opportunities to discover a mismatch. Task completion also depends on the available observations and subsequent recovery. Additional verification consumes budget and changes the information available to the agent, which may help explain the condition-dependent task-success ordering.

Scope. We study response–effect mismatch in AppWorld using two models and controlled fault conditions. Evaluation in broader environments is an important next step.

M Worked Example: Recovering a Missing Attachment

We illustrate the complete EGR-Lite workflow with the task-comment example used in Figure 2. Consider a compound operation

$$a = \texttt{post_task_comment}(\texttt{task_id}, \texttt{content}, \texttt{attachment}).$$

Its effect contract is

$$\mathcal{K}_a = (\Phi_a, \mathcal{O}_a, \mathcal{R}_a, \chi_a), \qquad \Phi_a = \{\phi_{\text{comment}}, \phi_{\text{attachment}}\}.$$

Here, ϕ_{comment} requires the intended comment to exist on the target task, while $\phi_{\text{attachment}}$ requires the requested file to be attached to that comment.

Effect contract. The observation procedures in $\mathcal{O}_a$ use `show_task_comments` to retrieve task comments, identify the target comment, and check its content and attachments. The operation properties in χ_a characterize the compound comment-posting operation as non-idempotent and externally visible. The recovery routes in $\mathcal{R}_a$ specify how a diagnosed missing residual can be addressed while preserving confirmed effects.

Selective verification. Suppose the original call reports success. The response alone does not establish that both effects in Φ_a occurred, while re-executing the non-idempotent compound operation may create a duplicate comment. Using the response together with the properties in χ_a, the policy therefore chooses $\sigma(r, \chi_a, h) = \textsc{Verify}$.

Effect diagnosis. EGR-Lite applies the observation procedures in $\mathcal{O}_a$. The public read-back finds the intended comment with the requested content, but its attachment list is empty. For the single-attachment case considered here, the resulting diagnosis is

$$\widehat{\mathbf{e}}_a = (1, 0), \qquad C_a = \{1\}, \qquad M_a = \{2\}, \qquad U_a = \varnothing.$$

Recovery must preserve the confirmed comment and supply the missing attachment.

Residual recovery. Given this diagnosis, residual recovery targets the missing attachment on the existing comment. Using `todoist.upload_attachment`, EGR-Lite attaches the requested file to the existing comment rather than re-executing `post_task_comment`:

`upload_attachment(task_comment_id, requested_file, overwrite=false)`.

The residual action adds the attachment while preserving the comment's identity and content.

The example considers a single requested attachment, a uniquely identified comment with an empty attachment list, and an available source file.

Re-verification. After the state-changing recovery, EGR-Lite applies the original observation procedures in $\mathcal{O}_a$ again. If the comment remains confirmed and the requested attachment is now present, then $\widehat{\mathbf{e}}_a^{\,\text{after}} = (1, 1)$, and the original intended effects in Φ_a are effect-complete.

The example shows how a single effect contract connects selective verification, effect diagnosis, residual recovery, and re-verification through a common representation of intended effects.