BEYOND TOOL-CALL SUCCESS: DYNAMIC EVIDENCE GRAPHS FOR RUNTIME EFFECT CERTIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We study runtime certification of state-changing tool use: whether an agent's claimed write effects are supported by execution-specific evidence in the realized post-state. Existing methods improve call-level consistency or select among candidate trajectories, but do not explicitly relate the grounded target, realized write, and qualified post-state witness. Consequently, apparently successful executions may modify the wrong object or produce no effect. We propose DEG, a dynamic evidence graph representation for runtime effect certification. DEG turns certification from separate checks into graph queries over explicit, dynamically maintained evidence relations: paths capture attribution and reachability, connected subgraphs support compound certificates, and evolving evidence state maintains temporal validity. Building on this representation, we certify effects over connected target–write–witness subgraphs, requiring grounded target provenance, realized-target consistency, and qualified post-state evidence attributable to the same write and valid for its post-write state. Shared graph identity further enables evidence reuse and localized maintenance across effects, while candidate selection remains decoupled from certification. Across six matched AppWorld and τ-bench settings, DEG achieves 0% observed selective certificate risk (SCR), while weaker evidence tiers reach up to 28.6% SCR. The complete framework improves end-to-end execution over five baselines.

1 INTRODUCTION

Tool-augmented language models can create, delete, and update persistent objects (Trivedi et al., 2024; Yao et al., 2025; Lu et al., 2025). This capability makes state-changing tool use fundamentally different from read-only generation, because execution errors can produce lasting changes in the external environment. For example, when asked to delete every song by a given artist, an agent may issue a valid tool call and receive a successful payload, yet some target songs may remain, and unrelated songs may be removed. The key problem, therefore, is whether the claimed write effect is actually supported by the realized environment state.

Existing methods address adjacent parts of this problem. Contract-grounded execution (Liu et al., 2026) checks whether tool calls satisfy registered interfaces and constraints; state-grounded evaluation (Trivedi et al., 2024; Yao et al., 2025) assesses task outcomes in the resulting environment; and test-time methods (Wang et al., 2023; Shinn et al., 2023; Kang et al., 2025; Lifshitz et al., 2025) select among sampled executions using consistency, feedback, confidence, or learned verification. They can determine whether a call is admissible, whether a final outcome is correct, or which candidate appears preferable, but they do not provide an online, execution-specific certificate linking the grounded target, realized write, and qualified post-state witness. We characterize this gap through three observations about the evidence required for effect certification.

Observation I: Execution success is weak evidence of realized effect. Existing works (Ross et al., 2025; Yao et al., 2023; Liu et al., 2026) improve when tools are invoked, how actions are composed, and whether calls are valid. Yet these execution-level signals can disagree with the resulting world state. To quantify this mismatch, we conduct a diagnostic on 50 held-out AppWorld tasks and report results in Fig. 1. The agent claims commit-blind completion on 46 tasks, while the task-level state evaluator supports only 7; separately, a payload-based verifier issues certificates for 25

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.

task executions, none supported by the realized post-state. Thus, successful calls and payloads are insufficient evidence for a claimed mutation.

Observation II: A correct post-state observation can still be insufficient evidence. A natural response is to inspect the state after execution, as in state-grounded evaluation and proxy-state methods (Trivedi et al., 2024; Yao et al., 2025; Chuang et al., 2026). However, a plausible post-state observation does not by itself establish a particular write effect. If the user asks to delete all songs by a given artist, observing that one target song is absent establishes neither that the complete target set is removed nor that the observation verifies the same realized write. Certification therefore depends not only on what is observed, but on how that evidence is related to the grounded target and realized write.

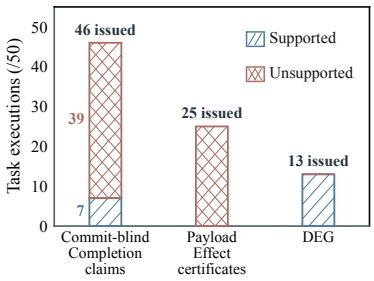


Figure 1: AppWorld completion claims and effect certificates.

Observation III: Selecting a better execution does not establish its effects. A complementary approach is to generate multiple trajectories and select among them using consistency, reflection, or verifier scores (Wang et al., 2023; Shinn et al., 2023; Kang et al., 2025; Lifshitz et al., 2025). However, selection compares candidates rather than establishing the effects of the selected execution. The highest-scoring candidate may still modify the wrong object when candidates share the same binding error. Thus, selection determines which execution to return, whereas certification requires evidence specific to the realized state changes.

Together, these observations reveal a structure: runtime certification of state-changing tool use is relational rather than a collection of independent checks. Evidence supports a claim only through explicit relations to the execution that produces it. These relations must preserve write-specific attribution, evidence reachability, and temporal validity for the current state. Relational structure and evolving validity naturally induce a dynamic graph: paths encode evidence relations, connected subgraphs compose compound certificates, and node state maintains validity across state transitions.

We therefore propose DEG, a dynamic evidence graph representation for runtime effect certification. DEG represents runtime evidence and effect instances as nodes; object-flow paths encode target provenance, verification associations encode write-specific attribution, and scope/version metadata maintain temporal validity. For each committed write, provenance and verification relations meet at the same realized write to form a connected target–write–witness subgraph, which serves as the evidence boundary for graph-grounded certification. Building on this representation, we develop effect certification as graph queries over these connected subgraphs, requiring each certificate to establish grounded target provenance, realized-target consistency, and a qualified write-specific witness on which the declared post-condition holds (**Observations I-II**). We further exploit shared graph identity to reuse compatible evidence across effects and localize maintenance to affected evidence scopes rather than replicated effect records. In this way, DEG turns certification evidence into an explicit graph representation, certificate decisions into graph queries, and shared evidence into runtime efficiency, while providing a common representation for broader certification objectives. We also decouple candidate selection from certification, allowing native backups without transferring certificates across executions (**Observation III**). Across six matched AppWorld and τ-bench certification settings, DEG achieves 0% observed SCR at 8.0–78.6% coverage, while weaker evidence tiers reach up to 28.6% SCR. The complete framework also improves end-to-end AppWorld execution over five baselines.

2 Related Work

Safe tool execution and state validation. Existing work approaches reliable tool use through contract checking, trajectory safety, state management, and environment-grounded evaluation. ToolGate (Liu et al., 2026) validates typed execution with registered pre-/post-conditions; ToolSword, ToolEmu, and SABER (Ye et al., 2024; Ruan et al., 2024; Cuadron et al., 2025) study unsafe or error-prone interactions; transactional and proxy-state methods (Chang & Geng, 2025; Chuang et al., 2026) improve consistency or infer structured state from traces. T-Eval and BFCL (Chen et al., 2024; Patil et al., 2025) evaluate tool use, while benchmarks including AppWorld and τ-

bench (Trivedi et al., 2024; Yao et al., 2025; Barres et al., 2025; Lu et al., 2025; Li et al., 2026b) use environment-state or execution-grounded outcomes. SafeCommit (Akewar & Ranjan, 2026) studies whether side-effectful actions are safe to commit under uncertainty, while Cordon (Chen et al., 2026b) uses semantic transactions and result lineage to validate mutations and external effects. In contrast, DEG casts runtime certification as queries over an evolving evidence graph that explicitly relates grounded targets, realized writes, and live-state witnesses; we instantiate this representation for effect certification.

Test-time selection and verifier-guided execution. Test-time methods (Wang et al., 2023; 2024; Shinn et al., 2023) improve candidates through sampling, revision, or ranking, while verifier-guided and process-reward methods (Kang et al., 2025; Zhang et al., 2025a; Li et al., 2026a; Lifshitz et al., 2025) rank executions using confidence, generative verification, or multiple judgments. These methods improve which execution is returned but do not by themselves establish its realized effects. DEG therefore separates selection from execution-specific certification.

Graphs for agents. Existing works (Zhuge et al., 2024; Chen et al., 2026a; Zhu et al., 2025; Zhang et al., 2025b; Luo et al., 2026; Feng et al., 2026; Xu et al., 2026) use graphs to structure agent computation, tool dependencies, execution plans, reusable tools, and routing. DEG instead represents certification evidence as a dynamic graph, turning effect certification into graph queries over provenance, attribution, and validity relations. Shared graph identity further enables evidence reuse and localized maintenance across effects.

3 PRELIMINARIES

Stateful Tool Use and Effect Certification. We consider an agent interacting with a mutable environment through read-only and state-changing tools, following settings such as AppWorld and τ-bench (Trivedi et al., 2024; Yao et al., 2025). Let $\mathcal{E} = (\mathcal{W}, \mathcal{T}^r, \mathcal{T}^w)$, where $\mathcal{W}$ is the space of world states and $\mathcal{T}^r$ and $\mathcal{T}^w$ denote read-only and state-changing tools. Given instruction u and initial state $W_0 \in \mathcal{W}$, an execution trace with m invocations is $\tau = ((t_1, \beta_1, o_1), \ldots, (t_m, \beta_m, o_m))$, where t_j, β_j, and o_j denote the tool, arguments, and observation at step j. Let the n committed writes occur at steps $j_1 < \cdots < j_n$. Effect instance a_i invokes $w_i = t_{j_i} \in \mathcal{T}^w$ with realized arguments $\alpha_i = \beta_{j_i}$, inducing $W_{i-1} \xrightarrow{a_i} W_i$. We index states by committed writes and assume no unmodeled external mutation between them. For a registry-covered write with bound target S_i and post-condition φ_i^{post}, effect certification asks whether the realized write targets S_i and whether φ_i^{post} holds in W_i. Complete task success additionally requires correct target binding, coverage of requested effects, preservation of relevant state, and satisfaction of the final task goal.

Dynamic Graph Definition. Following the snapshot view of dynamic graphs (Kazemi et al., 2020), a dynamic graph is a sequence $\mathcal{G} = \{G_k\}_{k=0}^{L}$ with $G_k = (V_k, E_k, X_k)$, where L is the number of graph updates, $k \in \{0, \ldots, L\}$ indexes the resulting $L + 1$ graph snapshots, and V_k, E_k, and X_k denote the node set, edge set, and node attributes.

4 DYNAMIC EVIDENCE GRAPHS FOR RUNTIME EFFECT CERTIFICATION

We operationalize this graph view with DEG, shown in Fig. 2. The key insight is representational: graph structure encodes which runtime evidence can support an effect claim, while evolving graph state determines whether that support remains valid. Effect certification is then expressed as graph-local queries over these maintained relations. DEG comprises four components, which are presented below.

4.1 DYNAMIC EVIDENCE GRAPH REPRESENTATION

Runtime effect certification is both relational and dynamic. Grounded targets, realized writes, and post-state witnesses matter through relations tying them to the same execution, while execution can materialize new relations and invalidate previously acquired evidence. DEG therefore maintains a dynamic evidence graph whose structure records certification relations and whose runtime state records their validity.

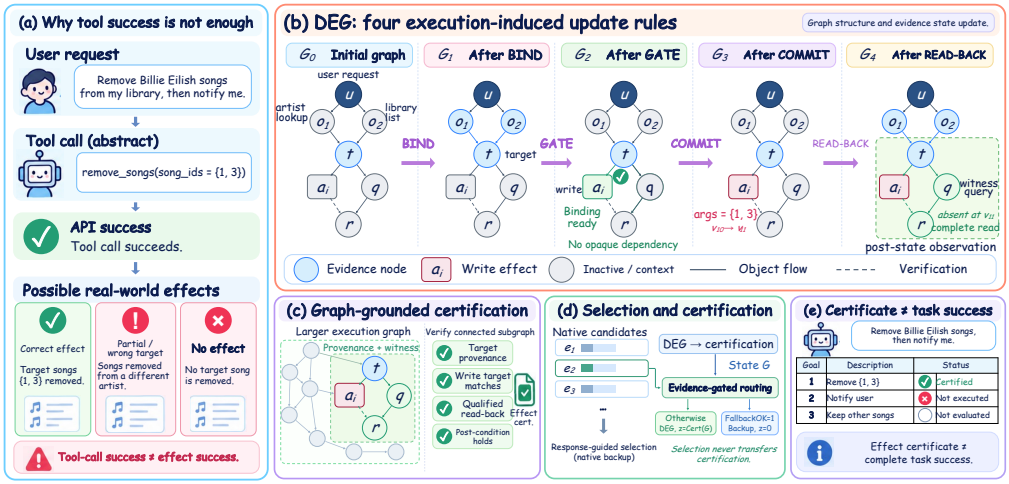


Figure 2: Overview of the DEG framework with graph construction and effect certification.

Certification topology. At snapshot $G_k = (V_k, E_k, X_k)$, DEG organizes certification evidence with two node classes and two edge classes:

$$V_k = V_k^{\text{evid}} \dot{\cup} V_k^{\text{eff}}, \qquad E_k = E_k^{\text{obj}} \dot{\cup} E_k^{\text{ver}}, \tag{1}$$

where $\dot{\cup}$ denotes disjoint union. We omit k when unambiguous. Evidence nodes V^{evid} denote non-mutating evidence operations, including post-state read-back, while effect nodes V^{eff} denote planned state-changing effect instances; only committed nodes instantiate the committed-effect sequence in Section 3. Their stage-dependent attributes are stored in X_k and specified in Appendix A. An object-flow edge $(v, v') \in E^{\text{obj}}$ means that v' consumes an object or value produced by v, while a verification association in E^{ver} links a write to evidence designated to verify its post-condition. Eq. (1) provides the relational structure for certification: object-flow edges define provenance and reachability, while verification associations encode write-specific attribution. Shared evidence nodes can be reused across effect-local regions when their scope/version requirements are compatible. In our implementation, object-flow dependencies are compiled from variable references in an LLM-generated task specification.

Execution-induced updates. Topology alone is insufficient because execution progressively resolves targets, writes, and witnesses, while committed mutations change world state. DEG updates the graph through $\mathcal{S} = \{\text{bind}, \text{gate}, \text{commit}, \text{read}\}$:

$$G_{k+1} = \mathcal{U}_{\xi_k}(G_k, \Delta_k), \qquad \xi_k \in \mathcal{S}, \quad 0 \le k < L, \tag{2}$$

where ξ_k is the update type, Δ_k is newly available execution information, $\mathcal{U}_{\xi_k}$ the corresponding update operator, and $G_{k+1} = (V_{k+1}, E_{k+1}, X_{k+1})$ the resulting graph snapshot. Binding resolves targets and may materialize effect nodes and object-flow relations; gating records pre-commit admissibility; commit records realized arguments; and read-back records witness evidence, verification associations, scope, completeness, and version metadata. Eq. (2) is the dynamic counterpart to Eq. (1): topology determines which evidence is structurally relevant, while evolving state determines whether that evidence remains usable. Scope/version metadata support local invalidation and refresh in Appendix A.1, preserving unaffected evidence as the environment evolves. Fig. 2 illustrates these updates.

> **Insight.** DEG separates certification structure from evidence validity: graph relations determine which evidence can support a claim, while evolving scope/version state determines whether that support remains valid as the environment changes. This separation enables local maintenance: state changes need only refresh affected evidence, while unrelated evidence remains reusable.

4.2 PATH-LOCAL GROUNDING FOR WRITE ADMISSION

A pending write should be checked against the evidence paths that ground its target, not unrelated execution context. DEG therefore expresses pre-commit grounding as a path query over the evidence graph: reachability identifies relevant evidence, while freshness and mutation-relevant opacity determine write admission.

Target provenance path. Consider an effect node $a_i \in V^{\text{eff}}$ with bound target set S_i. For every $s \in S_i$, DEG records a typed object-flow path:

$$P_i(s) = (v_0, \ldots, v_{\ell_s}), \qquad v_{\ell_s} = a_i, \tag{3}$$

where ℓ_s is the number of edges in $P_i(s)$. An admissible path starts from task-grounded evidence, follows only edges in E^{obj}, and preserves target identity until the write; otherwise $P_i(s) = \perp$, where $\perp$ denotes unresolved or absent structure. Identifying each path with its vertex–edge subgraph, let $V(H)$ denote the node set of H and define $H_i^{\text{prov}} = \bigcup_{s \in S_i : P_i(s) \neq \perp} P_i(s)$. Eq. (3) turns target grounding into a graph reachability condition: every bound target must be traceable through typed object flow to the pending write, rather than merely appear in its arguments. Missing or stale provenance triggers binding repair only for implicated evidence operations; Appendix A.2 gives the freshness condition.

Path-local validity. Let $\mathbf{1}\{\cdot\}$ denote the indicator function and $V^{\text{opaque}} \subseteq V$ the nodes whose outputs remain structurally traceable to typed upstream objects but whose selection rule cannot be deterministically audited from those inputs (e.g., free-form semantic selection). We define provenance coverage and effect-local opacity:

$$\text{ProvOK}_i = \mathbf{1}\{|S_i| > 0 \wedge \forall s \in S_i,\ P_i(s) \neq \perp\}, \tag{4}$$

$$\text{OpaqueCut}_i = \mathbf{1}\{V(H_i^{\text{prov}}) \cap V^{\text{opaque}} \neq \varnothing\}. \tag{5}$$

Eq. (4) requires provenance for each bound target, whereas Eq. (5) evaluates opacity only on the provenance region, so unrelated opaque evidence cannot affect the pending write. Binding repair additionally requires compatibility with the current pre-write state; Appendix A.2 defines this as FreshProv_i. Status `bind_ready` means $\text{ProvOK}_i = \text{FreshProv}_i = 1$. We admit the write by

$$\text{Gate}_i = \mathbf{1}\{\text{status}(a_i) = \texttt{bind_ready}\}\big(1 - \text{OpaqueCut}_i\big). \tag{6}$$

Thus a write is admitted only when its target paths are complete, current, and free of mutation-relevant opaque dependence. Under gate-enforced DEG, every committed effect has $\text{Gate}_i = 1$; we retain this factor in the certificate to make pre-commit admissibility explicit. DEG separately records Verifiable_i, indicating whether contract-supported read-back can cover the state required for certification; it does not affect admission.

4.3 GRAPH-GROUNDED EFFECT CERTIFICATION

Post-commit certification is compound: no target, write, or witness check alone establishes the certificate. The required relations must meet at the same realized write. We therefore develop effect certification based on the graph structure, using the connected target–write–witness subgraph of each committed effect.

Post-commit consistency. For committed effect a_i invoking write tool w_i with realized arguments α_i, let Target_{w_i} extract the addressed target and ν_i^+ denote the version of post-write state W_i. Let Q_i be contract-supported read-back procedures, $q_i \in Q_i$ the instantiated procedure, and r_i its witness node with observation $y(r_i)$. Let P_i^{ver} be the resolved verification path, possibly a single edge, associating a_i with r_i; if absent, $P_i^{\text{ver}} = \perp$. Finally, $\text{Qual}_i(r_i)$ requires contract support, sufficient scope, completeness, post-write version compatibility, and independence from the mutation payload; Appendix A.3 gives its definition. We define three post-commit predicates:

$$\text{WriteOK}_i = \mathbf{1}\big\{\text{Target}_{w_i}(\alpha_i) = S_i\big\}, \tag{7}$$

$$\text{WitnessOK}_i = \mathbf{1}\{P_i^{\text{ver}} \neq \perp\}\ \text{Qual}_i(r_i), \tag{8}$$

$$\text{EffectOK}_i = \mathbf{1}\big\{\varphi_i^{\text{post}}(y(r_i)) = \text{true}\big\}. \tag{9}$$

Eqs. (7)–(9) require target consistency, a qualified witness explicitly attributed to the realized write, and satisfaction of the declared post-condition. Eq. (8) is crucial: a valid observation is not effect-specific evidence unless the graph relates it to the write being certified. Appendix A details them.

Connected evidence subgraph. The path-level relations above must compose through the same realized write. When $\mathrm{ProvOK}_i = 1$ and $P_i^{\mathrm{ver}} \neq \bot$, we define the effect-local connected evidence subgraph H_i as

$$H_i = H_i^{\mathrm{prov}} \cup P_i^{\mathrm{ver}}. \tag{10}$$

We identify P_i^{ver} with its node–edge subgraph, with connectivity in the underlying undirected graph. Since H_i^{prov} and P_i^{ver} meet at a_i, Eq. (10) composes provenance and post-state verification into one effect-specific evidence region. This subgraph is the certification boundary: given fixed contract metadata, the certificate reads evidence from H_i, while unrelated regions cannot affect it except through relevant version compatibility. Effect subgraphs may share evidence nodes without merging certificate identity. If either prerequisite fails, no such subgraph is asserted and $C_i = 0$.

Effect certificate query. With supporting evidence localized to the effect-specific subgraph H_i, we formulate effect certification as a graph query that combines the pre-commit gate with post-commit predicates over this region:

$$C_i = \mathrm{Gate}_i \cdot \mathrm{WriteOK}_i \cdot \mathrm{WitnessOK}_i \cdot \mathrm{EffectOK}_i. \tag{11}$$

Here Gate_i is recorded immediately before a_i commits. Eq. (11) is the certificate query over the same write-centered evidence region: admissible grounding, realized-target consistency, qualified write-specific witnessing, and the declared post-condition must all hold. The graph localizes supporting evidence; the factorization specifies what that evidence must establish. Once issued, C_i is fixed to the graph snapshot in which its witness is qualified; later graph updates do not recompute the historical certificate. For snapshot G_k, let $M_k \subseteq V_k^{\mathrm{eff}}$ denote effects committed by stage k. We aggregate their fixed certificates as

$$\mathrm{Cert}(G_k) = \mathbf{1}\{|M_k| > 0\} \prod_{a_i \in M_k} C_i. \tag{12}$$

Eq. (12) returns one only when the snapshot contains at least one committed effect and every committed effect is certified. The guarantee remains relative to each bound target S_i and does not establish that the bound targets exhaust the user's intended target set. Thus, Eqs. (10)–(12) instantiate runtime effect certification as graph queries over connected subgraphs of a dynamic evidence graph.

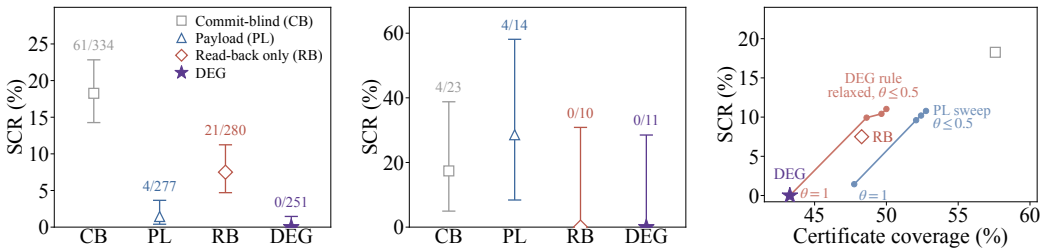
> **Insight.** Runtime effect certification is compositional: paths establish individual evidence relations, while their connection at the same realized write forms the effect-local subgraph required by the compound certificate. This subgraph defines the certificate's evidence boundary, preserving write-specific attribution while excluding unrelated execution context.

4.4 Selection as a Separate Control Layer

Candidate ranking may improve utility but cannot create evidence relations required for certification. Selection is comparative, certification execution-specific; we therefore keep response-guided selection and evidence-gated routing outside the certificate query. For instruction u, let $\mathcal{P}_u = \{e_1, \ldots, e_K\}$ denote the candidate pool. The selector ranks lexicographically by response-heuristic successes, execution errors, and interaction length from observable responses, excluding task ground truth and DEG certificate semantics; Eq. (19) in Appendix A.4 defines it. Selected e^{sel} serves as backup under the state-dependent fallback rule against DEG state. Routing chooses the returned execution; certification remains tied to the execution supported by recorded graph evidence. A backup may recover utility but cannot inherit another execution's certificate.

5 Experiments

Experimental setup. We evaluate DEG on AppWorld and τ-bench using six backbones across the main comparisons and diagnostic studies. For certificate reliability, we compare four criteria on identical realized executions: commit-blind (CB), payload (PL), read-back-only (RB), and full DEG, using commit status, response-level success, target-keyed post-state evidence, and graph-grounded target–write–witness evidence, respectively. We report certificate coverage (Cov.) and selective certificate risk (SCR). The main AppWorld and pooled τ-bench analyses use Qwen3.5-122B and Gemini-3.5-Flash. For end-to-end AppWorld execution, we compare against COLT (Qu



(a) Pooled τ-bench SCR. (b) Pooled AppWorld SCR. (c) τ-bench risk–coverage.

Figure 3: Certificate reliability under different criteria. Lower SCR is better. In (c), lines sweep threshold θ for PL and for DEG's read-back rule; CB and RB are fixed points.

et al., 2024), ToolRerank (Zheng et al., 2024), Self-Healing (Bholani, 2026), AutoTool (Jia & Li, 2026), ToolExpNet (Zhang et al., 2025b), and a native-pool oracle, reporting official sub-test pass rate and replay errors. Additional backbones and protocols are provided in Appendix B.1.

5.1 Certificate Reliability Evaluation

To investigate certificate reliability, we examine whether it depends not only on what evidence says, but on how that evidence is related to the realized execution. We compare DEG with three controlled verifier variants introduced in this work. Commit-blind (CB) uses only mutation commit status; payload (PL) additionally requires response-level success; and read-back-only (RB) checks the declared post-condition on target-keyed post-state evidence but omits write-specific attribution and DEG's provenance/version qualification. We pool nine τ-bench cohorts and AppWorld dev50 runs from Qwen3.5-122B and Gemini-3.5-Flash. Figs. 3a–3b report SCR across the four criteria, while Fig. 3c examines how PL/DEG risk changes as their acceptance threshold is relaxed.

Finding 1: Certificate reliability depends on graph-grounded evidence relations, not observed values alone. On pooled τ-bench, DEG yields 0/251 observed contradictions (exact 95% CI upper bound: 1.5%), compared with 18.3%, 1.4%, and 7.5% SCR for CB, PL, and RB, respectively (Fig. 3a). As shown in Fig. 3b, AppWorld shows the same weakness of response-level evidence, with 17.4% and 28.6% SCR for CB and PL, although RB and DEG are not separated at the available sample size. The τ-bench RB–DEG gap is particularly informative: both inspect post-state evidence, but only DEG relates that evidence to the grounded target and realized write through the graph. This supports the representation view that certification evidence derives its value from its execution relations, not its observed value alone.

Finding 2: Certifiability should expand through richer graph-grounded evidence, not weaker certification queries. We can observe from Fig. 3c that relaxing the PL threshold admits 29 executions, all falsely certified; relaxing DEG's read-back requirement admits 39, with 32 false. RB removes write-specific attribution: its 43 certificates beyond DEG contain all 21 false certificates, while none of the 237 shared with DEG are false. Thus, additional coverage comes from weakening evidence relations or query requirements, not from expanding the effects supported by the same graph-grounded criterion. Certifiability should instead grow by acquiring richer grounded evidence. A graph-independent held-out audit is reported in Table 5.

5.2 Evaluation of Graph-Structured Requirements

We next isolate the three structural requirements encoded by DEG. Each intervention isolates one graph-structured requirement while preserving the remaining evidence: attribution changes only the write–witness association; reachability adds an unrelated opaque branch while preserving write-relevant evidence; and temporal validity applies controlled state changes to compare static reuse, global invalidation, and scope-aware versioning. Table 1 reports false-certification rate (FCR), false-abstention rate (FAR), decision retention (Retained), pass-rate drop (pass drop), stale acceptance (Stale), and false abstention (false abst.) under controlled interventions.

Table 1: Controlled tests of graph-structured requirements, with DEG results highlighted.

Property	Controlled comparison	Representative result
Attribution	RB / permuted association / **DEG**	FCR: 45.7% / 84.2% / **0%** ; FAR: 0% / 84.2% / **0%**
Reachability	**Path-local DEG** / global FLAT	Retained: **262/262** / 0/262; pass drop: **0** / 25.2–36.0%
Temporal validity	Static / global / **scope-aware DEG**	Stale: 11 / 0 / **0** ; false abst.: 0 / 40 / **0**

Finding 3: Attribution relations carry effect-specific certificate information. *RB*, which omits write-specific association, has 45.7% FCR. *Permuted association* raises both FCR and FAR to 84.2%, whereas DEG remains at 0% on both metrics (Table 1). Crucially, the permutation keeps the witness observations fixed and changes only which write they support. The degradation therefore isolates information encoded by the relation itself. This directly supports E^{ver}: a valid post-state witness is not effect-specific evidence unless explicitly attributed to the realized write being certified.

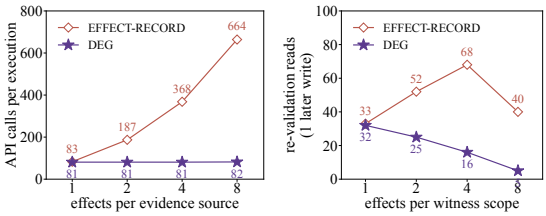
Finding 4: Reachability determines effect-local evidence relevance. As shown in Table 1, *Path-local* DEG retains all 262 challenged certificate decisions after adding an unrelated opaque branch, whereas *global* FLAT retains 0/262 and reduces passed sub-tests by 25.2%–36.0% across both backbones. The intervention changes only context outside the pending effect's provenance region while preserving all write-relevant evidence. This shows that relevance is determined by reachability to the effect, not by problematic evidence elsewhere in the execution. This directly supports the provenance paths $P_i(s)$ and H_i^{prov} as the structural boundary for pre-commit grounding and opacity checks.

Finding 5: Temporal validity must evolve with state. In Table 1, *Static* reuse accepts all 11 stale cases after relevant state changes, whereas *global* invalidation eliminates stale reuse but creates 40 false abstentions. *Scope-aware* DEG incurs neither error and requires only 22 re-reads for overlapping state changes. These failure modes show why graph structure alone is insufficient: the representation must also track whether supporting evidence remains valid under state changes. This directly supports the DEG separation between certification structure and evolving evidence validity.

Taken together, the three interventions support the graph representation underlying DEG: attribution is encoded by explicit write–witness relations, reachability defines effect-local evidence paths, and temporal validity is maintained by dynamic scope/version state. Together, these properties make certification a graph-local query over explicitly structured and dynamically maintained evidence.

5.3 Evidence Reuse and Maintenance Efficiency Evaluation

The previous section evaluates the certification semantics of graph structure. We now isolate a representation-level benefit of shared graph identity. We vary the number of effects per evidence source or witness scope and compare independent EFFECT-RECORD maintenance with shared DEG, while holding realized effects, certificate predicates, verifier logic, and execution controls fixed. Both variants produce identical certification decisions and realized effects, so the comparison isolates evidence acquisition and revalidation costs.



(a) Evidence acquisition. (b) Revalidation.

Figure 4: Evidence acquisition and revalidation under shared graph evidence.

Finding 6: Shared graph identity reduces repeated evidence-acquisition API calls. As effects per evidence source increase from one to eight, EFFECT-RECORD acquisition grows from 83 to 664 API calls per execution (8.0×), whereas DEG remains nearly constant at 81–82 (Fig. 4a). At eight effects, shared evidence reduces acquisition calls by 87.7%. Because certification semantics and realized effects are fixed, the reduction comes from representing common evidence once and reusing it across compatible effect-local subgraphs, yielding higher runtime efficiency without changing certificate semantics.

Table 2: AppWorld dev50 matched execution setting. The DEG row is highlighted.

	Qwen3.5-122B			Gemini-3.5-Flash			Average	
Method	Passed	Sub-tests (%) ↑	Replay ↓	Passed	Sub-tests (%) ↑	Replay ↓	Sub-tests (%) ↑	Replay ↓
ToolRerank	65	24.8	0	72	27.5	9	26.1	4.5
Self-Healing	78	29.8	0	82	31.3	14	30.5	7.0
COLT	83	31.7	0	111	42.4	8	37.0	4.0
AutoTool	94	35.9	0	118	45.0	5	40.5	2.5
ToolExpNet	90	34.4	0	128	48.9	3	41.6	1.5
Native-pool oracle	118	45.0	0	153	58.4	0	51.7	0.0
DEG	**127**	**48.5**	**0**	**154**	**58.8**	**0**	**53.6**	**0.0**

Finding 7: Shared scope state reduces repeated revalidation reads. As effects per witness scope increase, EFFECT-RECORD repeatedly revalidates duplicated records, whereas DEG maintains shared scope state. At eight effects, one later overlapping write requires 40 EFFECT-RECORD revalidation reads but only 5 with DEG, an 87.5% reduction (Fig. 4b). Both variants use the same temporal-validity rule and produce identical certificate outcomes, so the reduction comes from maintaining one shared scope/version state rather than refreshing duplicate records, improving revalidation efficiency without changing certificate semantics. Together, these results separate the two roles of the representation: paths and connected subgraphs determine certificate semantics, while shared graph identity reduces repeated evidence acquisition and revalidation across effects.

5.4 END-TO-END EXECUTION EVALUATION

Here, we test the control layer separated from certification in Section 4.4. On the matched AppWorld dev50 protocol, we evaluate Qwen3.5-122B and Gemini-3.5-Flash and compare the complete DEG framework against five representative tool-use baselines. We additionally include a native-pool oracle, which post-hoc returns the best realized trajectory in the matched candidate pool and isolates the utility available from fixed-pool selection alone. All returned trajectories are replayed in fresh environments and scored by the official AppWorld evaluator, reporting passed sub-tests (*Passed*, *Sub-tests (%)*) and replay errors (*Replay*).

Finding 8: Decoupled control improves execution utility while preserving certificate identity. In Table 2, DEG reaches 127 and 154 passed sub-tests on Qwen and Gemini, corresponding to relative improvements of 35.1% and 20.3% over the strongest external baseline on each backbone. It also exceeds the matched native-pool oracle by 9 and 1 sub-tests ($+3.4$ and $+0.4$ percentage points), yielding an average pass rate of 53.6% with zero replay errors. The oracle isolates utility recoverable by post-hoc selection from the same native pool; the remaining gain, particularly on Qwen, therefore lies beyond fixed-pool ranking alone. Because native backups cannot inherit DEG certificates by construction, these gains improve execution utility without conflating comparative selection with execution-specific certification.

6 CONCLUSION

We present a graph representation of certification evidence for state-changing tool use, replacing isolated checks with queries over explicit, dynamically maintained evidence relations. DEG separates relational structure from runtime validity: graph topology encodes attribution and reachability, while scope/version state determines whether supporting evidence remains valid as environments evolve. This representation turns effect certification into graph queries with a natural hierarchy: paths establish evidence relations, while connected target–write–witness subgraphs compose them into effect-specific certificates with explicit evidence boundaries. Shared graph identity enables evidence reuse and localized maintenance across effect-local regions, improving runtime efficiency without changing certificate identity. Across six matched settings, DEG achieves 0% observed SCR, while controlled interventions support write-specific attribution, path-local relevance, and dynamic evidence validity as distinct requirements for reliable certification. The complete framework improves end-to-end AppWorld execution over five baselines. Together, these results support dynamic evidence graphs as a principled representation for runtime effect certification, where certification is expressed as graph queries and shared evidence can be maintained efficiently across effects.

AI USE STATEMENT

We use generative AI tools solely to refine the writing of this manuscript, including improving language clarity, grammar, and presentation. We review all AI-assisted edits to ensure that they accurately reflect our technical content and claims. We take full responsibility for the final content of this work, including any text produced or revised with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Appendix A provides additional technical details of DEG, while Appendix B provides detailed experimental settings and evaluation protocols. We plan to release the implementation code upon acceptance.

REFERENCES

Mayur Akewar and Ravi Ranjan. SafeCommit: Certifying when memory-grounded agents may safely act. *arXiv preprint arXiv:2608.04289*, 2026.

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.

Neeraj Bholani. Graph-Based Self-Healing Tool Routing for Cost-Efficient LLM Agents. *arXiv preprint arXiv:2603.01548*, 2026.

Edward Y. Chang and Longling Geng. SagaLLM: Context Management, Validation, and Transaction Guarantees for Multi-Agent LLM Planning. *Proceedings of the VLDB Endowment*, 18(12):4874–4886, 2025. doi: 10.14778/3750601.3750611.

Wenjie Chen, Di Yao, Wenbin Li, Xuying Meng, Chang Gong, and Jingping Bi. GTool: Graph Enhanced Tool Planning with Large Language Model. In *International Conference on Learning Representations*, volume 2026, pp. 69247–69269, 2026a.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. T-Eval: Evaluating the Tool Utilization Capability of Large Language Models Step by Step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9510–9529. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.515.

Zheng Chen, Hanqing Liu, Duling Xu, Dong Dong, Jialin Li, Bangzheng Pu, and Jidong Zhai. Cordon: Semantic transactions for tool-using LLM agents. *arXiv preprint arXiv:2606.17573*, 2026b.

Yun-Shiuan Chuang, Chaitanya Kulkarni, Alec M. Chiu, Avinash Thangali, Zijie Pan, Shivani Shekhar, Yirou Ge, Yixi Li, Uma Kona, Linsey Pang, and Prakhar Mehrotra. Toward Scalable Verifiable Reward: Proxy State-Based Evaluation for Multi-turn Tool-Calling LLM Agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 1251–1264. Association for Computational Linguistics, 2026. ISBN 979-8-89176-394-4. doi: 10.18653/v1/2026.acl-industry.87.

Alejandro Cuadron, Pengfei Yu, Yang Liu, and Arpit Gupta. SABER: Small Actions, Big Errors – Safeguarding Mutating Steps in LLM Agents. *arXiv preprint arXiv:2512.07850*, 2025.

Tao Feng, Haozhen Zhang, Zijie Lei, Peixuan Han, and Jiaxuan You. GraphPlanner: Graph Memory-Augmented Agentic Routing for Multi-Agent LLMs. In *International Conference on Learning Representations*, 2026.

Jingyi Jia and Qinbin Li. AutoTool: Efficient Tool Selection for Large Language Model Agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 31265–31273, 2026. doi: 10.1609/aaai.v40i37.40389. Issue 37.

Zhewei Kang, Xuandong Zhao, and Dawn Song. Scalable Best-of-N Selection for Large Language Models via Self-Certainty. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 19720–19745. Curran Associates, Inc., 2025. doi: 10.52202/085713-0663.

Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

Dawei Li, Yuguang Yao, Zhen Tan, Huan Liu, and Ruocheng Guo. ToolPRMBench: Evaluating and Advancing Process Reward Models for Tool-using Agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 12378–12391. Association for Computational Linguistics, 2026a. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.602.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. The Tool Decathlon: Benchmarking Language Agents for Diverse, Realistic, and Long-Horizon Task Execution. In *International Conference on Learning Representations*, 2026b.

Shalev Lifshitz, Sheila A. McIlraith, and Yilun Du. Multi-Agent Verification: Scaling Test-Time Compute with Multiple Verifiers. In *Conference on Language Modeling*, 2025.

Yanming Liu, Xinyue Peng, Jiannan Cao, Xinyi Wang, Songhang Deng, Jintao Chen, Jianwei Yin, and Xuhong Zhang. ToolGate: Contract-Grounded and Verified Tool Execution for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 9653–9684. Association for Computational Linguistics, 2026. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.470.

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. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.65.

Jianwen Luo, Yiming Huang, Jinxiang Meng, Fangyu Lei, Shizhu He, Xiao Liu, Shanshan Jiang, Bin Dong, Jun Zhao, and Kang Liu. GATE: Graph-based Adaptive Tool Evolution Across Diverse Tasks. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1934–1961, 2026.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *International Conference on Machine Learning*, volume 267, pp. 48371–48392, 2025.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Towards Completeness-Oriented Tool Retrieval for Large Language Models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pp. 1930–1940, 2024.

Hayley Ross, Ameya Sunil Mahabaleshwarkar, and Yoshi Suhara. When2Call: When (not) to Call Tools. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3391–3409. Association for Computational Linguistics, 2025. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.174.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris Maddison, and Tatsunori Hashimoto. Identifying the Risks of LM Agents with an LM-Emulated Sandbox. In *International Conference on Learning Representations*, volume 2024, pp. 27031–27098, 2024.

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.

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. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.850.

Han Wang, Archiki Prasad, Elias Stengel-Eskin, and Mohit Bansal. Soft Self-Consistency Improves Language Models Agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 287–301. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-short.28.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *International Conference on Learning Representations*, 2023.

Yuanyuan Xu, Wenjie Zhang, Yin Chen, Xuemin Lin, and Ying Zhang. Self-evolving agents as dynamic graph transformation: A survey and new perspective. *arXiv preprint arXiv:2608.18104*, 2026.

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.

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*, 2025.

Junjie Ye, Sixian Li, Guanyu Li, Caishuang Huang, Songyang Gao, Yilong Wu, Qi Zhang, Tao Gui, and Xuanjing Huang. ToolSword: Unveiling Safety Issues of Large Language Models in Tool Learning Across Three Stages. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2181–2211, 2024.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Seyed Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative Verifiers: Reward Modeling as Next-Token Prediction. In *International Conference on Learning Representations*, volume 2025, pp. 12476–12505, 2025a.

Zijing Zhang, Zhanpeng Chen, He Zhu, Ziyang Chen, Nan Du, and Xiaolong Li. ToolExpNet: Optimizing Multi-Tool Selection in LLMs with Similarity and Dependency-Aware Experience Networks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 15706–15722. Association for Computational Linguistics, 2025b. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.811.

Yuanhang Zheng, Peng Li, Wei Liu, Yang Liu, Jian Luan, and Bin Wang. ToolRerank: Adaptive and Hierarchy-Aware Reranking for Tool Retrieval. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 16263–16273. ELRA and ICCL, 2024.

Dongsheng Zhu, Weixian Shi, Zhengliang Shi, Zhaochun Ren, Shuaiqiang Wang, Lingyong Yan, and Dawei Yin. Divide-Then-Aggregate: An Efficient Tool Learning Method via Parallel Tool Invocation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28859–28875, 2025.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language Agents as Optimizable Graphs. In *International Conference on Machine Learning*, volume 235, pp. 62743–62767. PMLR, 2024.

A Additional Technical Details of DEG

This section complements Section 4 with the runtime semantics and formal conditions underlying the graph view of DEG. We follow the main method: graph structure determines which evidence can support a claim, runtime state determines whether that evidence remains valid, and connected evidence regions define effect-specific certification.

A.1 Dynamic Evidence Graph Representation and Runtime State

Section 4.1 separates certification structure from runtime evidence validity. Graph structure records the relations through which evidence can support an effect claim, while node state records whether that evidence remains usable as execution changes the environment. We make these runtime states and validity conditions explicit below.

Graph materialization. DEG materializes effect and evidence nodes and their relations as concrete identities become available during execution. Binding may expand an effect specification into multiple effect instances and instantiate their object-flow relations; read-back materializes witness evidence and verification associations.

Runtime records. We define the runtime state of an evidence node $v \in V^{\mathrm{evid}}$ as $x_k(v) = (y_k(v), \mathrm{type}(v), \Lambda_k(v), \chi_k(v), \nu_k(v))$, where $y_k(v)$ is the observed value, $\mathrm{type}(v)$ its evidence-operation category, $\Lambda_k(v)$ its state scope, $\chi_k(v)$ its completeness indicator, and $\nu_k(v)$ its state version. Resolved completeness indicators take values in $\{0, 1\}$; unresolved fields take $\perp$. Predicates with unresolved required inputs evaluate to 0, so no certificate is issued until all required fields are resolved. We omit the stage index k when it is unambiguous. The observed value records the read result, whereas scope, completeness, and version determine whether that observation can support certification. For an effect node $a_i \in V^{\mathrm{eff}}$, let ν_i^- denote its pre-write version and Λ_i^{mod} its modification scope. We list its certification data as $x_k(a_i) = (w_i, S_i, \alpha_i, \nu_i^-, \nu_i^+, \varphi_i^{\mathrm{post}}, \Lambda_i^{\mathrm{mod}}, Q_i, q_i)$. The runtime state additionally includes the control status and recorded gate outcome. The tuple records the bound target, realized write arguments and versions, declared post-condition and modification scope, and the available and instantiated read-back procedures. They connect the planned effect to the write that actually reaches the environment.

Tool contracts and required state. For each registry-covered write w, we define its contract as $\mathcal{C}_w = (\mathrm{Sig}_w, \mathrm{Target}_w, \mathrm{Pre}_w, \mathrm{Post}_w, \mathrm{Mod}_w, \mathrm{Frame}_w, Q_w)$. Here Sig_w, Pre_w, and Frame_w denote registry metadata for the tool signature, precondition, and frame condition, respectively; the default certificate below explicitly uses Target_w, Post_w, Mod_w, and Q_w. Target_w extracts the objects addressed by realized arguments, while Post_w, Mod_w, and Q_w specify the post-condition, modification scope, and supported read-back procedures. For effect a_i, let $\mathcal{R}_i$ denote the state coordinates sufficient to decide its instantiated post-condition, and let Π_Λ denote projection onto scope Λ. We require $\varphi_i^{\mathrm{post}}$ to be well defined for any complete or partial assignment whose domain contains $\mathcal{R}_i$. For any such assignments Z and Z', we formulate required-state sufficiency as

$$\Pi_{\mathcal{R}_i}(Z) = \Pi_{\mathcal{R}_i}(Z') \quad \Longrightarrow \quad \varphi_i^{\mathrm{post}}(Z) = \varphi_i^{\mathrm{post}}(Z'). \tag{13}$$

Eq. (13) states that a qualified witness need only cover $\mathcal{R}_i$; values outside this scope cannot change the declared post-condition.

Scope-aware version compatibility. A later write should invalidate previously acquired evidence only when it may change state relevant to that evidence. Let $W[\nu]$ denote the world state at version ν, with $W[\nu_i^-] = W_{i-1}$ and $W[\nu_i^+] = W_i$ for the pre- and post-write versions of a_i, and let $\prec$ denote temporal precedence. We treat $\sim_\Lambda$ as the runtime's operational version-compatibility relation over scope Λ and require it to satisfy the following soundness condition:

$$\nu \sim_\Lambda \nu' \quad \Longrightarrow \quad \Pi_\Lambda(W[\nu]) = \Pi_\Lambda(W[\nu']). \tag{14}$$

Thus, compatible versions must agree on the state covered by the evidence. For state-independent evidence, we set $\Lambda(v) = \varnothing$, retain its acquisition version, and define $\nu \sim_\varnothing \nu'$ for every pair of world versions; because projection onto the empty scope is identical across states, such evidence

is unaffected by world-state changes. For state-dependent evidence, the runtime preserves compatibility across writes disjoint from Λ; overlapping writes conservatively require revalidation before reuse. We define the corresponding conservative stale condition for evidence v after a subsequent effect $a_{i'}$ as

$$\mathrm{Stale}(v; a_{i'}) = \mathbf{1}\left\{\Lambda(v) \cap \Lambda_{i'}^{\mathrm{mod}} \neq \varnothing \wedge \nu(v) \prec \nu_{i'}^{+}\right\}. \tag{15}$$

Eq. (15) prevents potentially changed evidence from being reused for a later-state judgment. It does not imply that the recorded post-condition is false, nor does it retroactively invalidate a certificate issued for an earlier post-write state.

Snapshot semantics. Eq. (12) aggregates effect certificates recorded by stage k. Let κ_i denote the graph-update index at which the required inputs for effect a_i are resolved and its certificate is issued. Operationally, C_i is the stored value evaluated on G_{κ_i}, including its evidence relations, witness values, and versions. A committed effect whose required inputs are unresolved contributes $C_i = 0$. Later graph updates do not recompute an issued historical certificate under $G_{k'}$, allowing shared evidence to be refreshed for future judgments without changing previously issued certificates.

A.2 PATH-LOCAL GROUNDING FOR WRITE ADMISSION

Section 4.2 expresses target grounding as a path-local verification query. Structural reachability determines which evidence belongs to the pending write, while freshness determines whether that evidence remains usable at execution time. Pre-commit verification therefore combines provenance coverage with path-local freshness.

Provenance freshness. We define freshness of the provenance supporting effect a_i as

$$\mathrm{FreshProv}_i = \mathbf{1}\left\{\forall v \in V(H_i^{\mathrm{prov}}) \cap V^{\mathrm{evid}},\ \nu(v) \sim_{\Lambda(v)} \nu_i^{-}\right\}. \tag{16}$$

Eq. (16) requires all evidence used to ground the pending target to remain compatible with the realized pre-write state. We use $\mathrm{status}(a_i) = \texttt{bind_ready}$ iff $\mathrm{ProvOK}_i = \mathrm{FreshProv}_i = 1$, matching the definition in the main text.

Binding repair. When required provenance is missing or stale, DEG re-runs only the implicated evidence operations and re-resolves the affected target identifiers. Repair succeeds when the refreshed binding satisfies $\mathrm{ProvOK}_i = \mathrm{FreshProv}_i = 1$; otherwise the effect remains blocked. Repair therefore restores the evidence responsible for failed grounding rather than rebuilding unrelated execution state.

Post-state verifiability. Let $\mathrm{Obs}(q)$ denote the state coordinates observable by read-back procedure q. We define an effect as verifiable when at least one contract-supported read-back can observe all state required by its post-condition:

$$\mathrm{Verifiable}_i = \mathbf{1}\{\exists q \in Q_i : \mathcal{R}_i \subseteq \mathrm{Obs}(q)\}. \tag{17}$$

Eq. (17) records whether the registry offers at least one supported way to observe the required state; it does not affect write admission. The actual instantiated procedure q_i is checked again when its witness is qualified below.

A.3 GRAPH-GROUNDED EFFECT CERTIFICATION

Section 4.3 composes path-level evidence relations through the same realized write. Passing the pre-commit gate establishes admissible grounding but not the declared post-condition. Certification therefore requires a qualified post-state witness that completes the target–write–witness evidence region, remains valid for the realized post-write state, and is obtained independently of the mutation response.

Witness qualification. Let $\mathrm{Independent}(r_i, a_i) = 1$ when the witness is obtained from an external post-state observation rather than solely from the mutation response. We define qualification of witness r_i as

$$\mathrm{Qual}_i(r_i) = \mathbf{1}\left\{\begin{array}{l}\mathrm{Verifiable}_i = 1 \wedge q_i \in Q_i \wedge \mathcal{R}_i \subseteq \mathrm{Obs}(q_i) \\ \wedge\ \mathcal{R}_i \subseteq \Lambda(r_i) \wedge \chi(r_i) = 1 \\ \wedge\ \nu(r_i) \sim_{\mathcal{R}_i} \nu_i^{+} \wedge \mathrm{Independent}(r_i, a_i)\end{array}\right\}. \tag{18}$$

Eq. (18) expands the witness-quality term used in Section 4.3. Contract membership and $\mathcal{R}_i \subseteq \mathrm{Obs}(q_i)$ tie qualification to the actual read-back that produced the witness; sufficient recorded scope and completeness ensure that the witness can decide the post-condition; version compatibility ensures temporal validity relative to the realized post-write state; and independence prevents the mutation payload from serving as its own verification evidence. Together with the explicit verification association in $\mathrm{WitnessOK}_i$, these conditions make the witness specific to the realized effect being certified. Here $\chi(r_i) = 1$ means that the read-back is exhaustive over its recorded scope under the environment-specific protocol. Paginated interfaces require all relevant pages, while identifier-based relations require the concrete bound identifiers.

Certificate interpretation and boundary. The effect-level certificate is conditional on three runtime assumptions: the registered contract correctly extracts the realized target and specifies the declared post-condition; the recorded write arguments, versions, and complete witness values faithfully reflect the environment; and $\sim_\Lambda$ satisfies Eq. (14). Under these conditions, $C_i = 1$ supports both $\mathrm{Target}_{w_i}(\alpha_i) = S_i$ and satisfaction of the declared post-condition in the realized post-write state. The guarantee remains relative to the bound target S_i: it does not establish that S_i exhausts the user's intended target set, that the write is causally necessary when the post-condition already holds, that omitted task obligations are completed, or that undeclared state is preserved.

When a contract explicitly includes a frame condition, DEG may additionally compare the corresponding pre- and post-write state and require all observed changes to lie within the declared modification scope. This stronger check is not part of the default certificate.

A.4 Selection as a Separate Control Layer

Candidate selection compares executions, whereas certification refers to the effects of one realized execution. We therefore define response-guided selection and state-dependent routing separately, preserving certificate identity throughout.

Response-guided candidate selection. For candidate $e \in \mathcal{P}_u$, let $N_{\mathrm{succ}}(e)$ count response-heuristic successes, $N_{\mathrm{err}}(e)$ execution errors, and $N_{\mathrm{turn}}(e)$ interaction turns. We define the fixed-pool selector used by the matched AppWorld experiments as

$$\rho(e) = \big(N_{\mathrm{succ}}(e), -N_{\mathrm{err}}(e), -N_{\mathrm{turn}}(e)\big), \qquad e^{\mathrm{sel}} = \operatorname*{arg\,max}^{\mathrm{lex}}_{e \in \mathcal{P}_u} \rho(e). \tag{19}$$

Eq. (19) uses observable response-level signals only; neither task ground truth nor DEG certificate semantics enter the ranking. Ties follow candidate traversal order, and $e^{\mathrm{sel}} = \bot$ when no candidate is available.

State-dependent fallback. Let G denote the current DEG snapshot and e^{DEG} its realized execution. Let $n_{\mathrm{mut}}(\cdot)$ count non-error completed mutation calls and define $n_{\mathrm{mut}}(G) := n_{\mathrm{mut}}(e^{\mathrm{DEG}})$. Given the selected candidate e^{sel}, we define backup eligibility as

$$\mathrm{FallbackOK}(G, e^{\mathrm{sel}}) = \begin{cases} 0, & e^{\mathrm{sel}} = \bot \ \vee\ \mathrm{Blocked}(G), \\ 1, & e^{\mathrm{sel}} \neq \bot \wedge \neg\mathrm{Blocked}(G) \\ & \wedge\ \mathrm{AnswerFallback}(G), \\ \mathbf{1}\left\{ \begin{array}{l} \neg\mathrm{Success}(G) \\ \wedge\ n_{\mathrm{mut}}(e^{\mathrm{sel}}) > n_{\mathrm{mut}}(G) \\ \wedge\ \big(N^{+}_{\mathrm{rb}}(G) = 0 \vee N^{-}_{\mathrm{rb}}(G) > 0\big) \end{array} \right\}, & \text{otherwise.} \end{cases} \tag{20}$$

Blocked denotes termination caused by opacity or failed binding repair, while Success denotes runtime status `COMMIT_SUCCESS`. $\mathrm{AnswerFallback}(G)$ denotes the branch for graphs without mutation operations. The counts $N^{+}_{\mathrm{rb}}(G)$ and $N^{-}_{\mathrm{rb}}(G)$ denote mutation nodes with recorded read-back passes and violations, respectively. Eq. (20) is a routing rule: it determines whether the selected native candidate may replace e^{DEG} after selection.

We define the returned execution $D(G, e^{\mathrm{sel}})$ and its certificate indicator $z \in \{0, 1\}$ as

$$D(G, e^{\mathrm{sel}}) = \begin{cases} e^{\mathrm{sel}}, & \mathrm{FallbackOK}(G, e^{\mathrm{sel}}) = 1, \\ e^{\mathrm{DEG}}, & \text{otherwise,} \end{cases} \qquad z = \mathbf{1}\big\{D(G, e^{\mathrm{sel}}) = e^{\mathrm{DEG}}\big\}\,\mathrm{Cert}(G). \tag{21}$$

Equality in Eq. (21) denotes identity of the execution instance. The equation makes certificate identity explicit: only the realized DEG execution whose graph evidence supports the certificate can carry it; a native fallback cannot inherit certification from another execution.

B Additional Experiments

This section provides protocols and complementary evidence for the claims in Section 5.

B.1 Detailed Experimental Settings

Benchmarks, backbones, and candidate pools. We evaluate on AppWorld (Trivedi et al., 2024) and τ-bench (Yao et al., 2025). AppWorld *dev50* contains 50 tasks and 262 official sub-tests; a disjoint 50-task split is reserved for the held-out graph-independent audit. For τ-bench, we use the retail and airline domains. The pooled τ-bench reliability analysis combines nine execution cohorts ($N = 580$); throughout the reliability tables, N denotes execution instances after pooling rather than unique benchmark tasks. In particular, the τ-airline/Qwen entry with $N = 184$ is an execution count, not 184 distinct questions. Across the complete evaluation, we use Qwen3-8B, Qwen3-14B, Qwen3-32B, Qwen3.5-122B, Gemini-2.5-Flash-Lite, and Gemini-3.5-Flash. The main AppWorld evaluation uses Qwen3.5-122B and Gemini-3.5-Flash. Multi-sample methods use $K = 4$ native candidates except retail–Qwen settings with $K \leq 3$. Full DEG additionally has its graph-guided execution e^{DEG} and may route to a candidate from the native pool. The native-pool oracle is therefore an upper bound only for methods restricted to that matched native pool; it does not include e^{DEG}. The "selector only" control always returns the response-guided selection from the same K-candidate native pool when one is available. The end-to-end external baselines include COLT (Qu et al., 2024), ToolRerank (Zheng et al., 2024), Self-Healing (Bholani, 2026), AutoTool (Jia & Li, 2026), and ToolExpNet (Zhang et al., 2025b). We conduct the experiments on a server equipped with 8× NVIDIA Tesla V100-SXM2 GPUs, each with 32 GB of memory, for a total of 256 GB of GPU memory.

Certificate tiers and risk–coverage threshold. Commit-blind (CB) uses only the commit status of a state-changing call. Payload (PL) additionally requires the mutation response to satisfy its response-level success predicate. Read-back-only (RB) evaluates the declared post-condition on target-keyed post-state evidence but does not require a write-specific verification association or DEG's provenance/version qualification. DEG uses the full criterion in Eqs. (11)–(12). For the risk–coverage sweep in Fig. 3c, let $\gamma_F(e)$ be the fraction of committed effects satisfying the predicate of family $F \in \{\mathrm{PL}, \mathrm{DEG}\}$: response-level success, or confirmation by the effect's target-keyed witness (0 if none commits). Family F accepts an execution when $\gamma_F(e) \geq \theta$. Thus $\theta = 1$ is the PL tier or the DEG certificate (not tuned against task outcomes), while lower θ relaxes each rule within its verifier. The execution set is fixed; only θ changes. RB, a per-execution verdict without per-effect scores, is a fixed point.

Metrics and state reference. AppWorld trajectories are replayed in fresh environments and scored using the official `world.evaluate()` implementation. "Replay" counts errors during this fresh-environment execution. For certification, we report $\mathrm{Coverage} = N_{\mathrm{cert}}/N_{\mathrm{eval}}$ and, when $N_{\mathrm{cert}} > 0$, $\mathrm{SCR} = N_{\mathrm{false}}/N_{\mathrm{cert}}$, where N_{eval} counts evaluated execution instances, N_{cert} counts executions receiving a certificate, and N_{false} counts certified executions whose certificates are contradicted by an environment-level reference derived from benchmark state or evaluation artifacts.

Protocol separation. Table 3 summarizes the principal controls. The main AppWorld execution comparison uses 50 dev50 tasks and 262 official sub-tests per backbone. The cross-backbone control later in this appendix uses a separate matched 38-sub-test protocol. The reachability study has two evaluation levels: Table 6 reports task-level behavior on dev50, while the retention result in Table 1 aggregates 262 challenged certificate decisions across 17 matched clean settings, each re-evaluated after inserting one unrelated opaque branch. These 262 decisions are not the 262 AppWorld official sub-tests.

Table 3: Controlled comparisons and evaluation units.

Comparison	Unit	Held fixed	Varied component
External baselines vs. DEG	Sub-test	Task set, backbone, replay scorer	Execution strategy
Certificate tiers	Execution	Realized execution	Admissible evidence
Attribution intervention	Write effect	Witness values	Write–witness association
Reachability intervention	Certificate decision	Write-relevant evidence	Relevance scope
Temporal-validity intervention	Perturbation	Effect relations	Validity policy
DEG vs. EFFECT-RECORD	Decision / calls	Registry, router, predicates	Evidence sharing
Component ablation	Sub-test / diagnostic	Native pool, selector, evaluator	Runtime component
Native-pool oracle	Sub-test	Matched native pool	Post-hoc pool selection

B.2 RELIABILITY OF RELATIONALLY GROUNDED CERTIFICATES

Additional certificate reliability evaluation. To evaluate reliability across individual settings, the pooled τ-bench analysis includes nine cohorts with 580 executions and 251 issued certificates. Table 4 reports four cohorts (337 executions; 158 certificates) and two AppWorld settings (100 executions; 11 certificates); the remaining five τ-bench cohorts contribute 243 executions and 93 certificates.

Table 4: Certificate reliability across six matched settings; N counts executions.

Setting	N	Issued	Coverage	SCR	95% CI upper
AppWorld / Qwen3.5-122B	50	4	8.0%	0%	60.2%
AppWorld / Gemini-3.5-Flash	50	7	14.0%	0%	41.0%
τ-airline / Qwen3.5-122B	184	59	32.1%	0%	6.1%
τ-airline / Gemini-3.5-Flash	19	4	21.1%	0%	60.2%
τ-retail / Qwen3.5-122B	120	84	70.0%	0%	4.3%
τ-retail / Gemini-3.5-Flash	14	11	78.6%	0%	28.5%

Finding 9: No certificate contradiction is observed in the six reported matched settings. As shown in Table 4, all six reported settings have 0% observed SCR, with coverage from 8.0% to 78.6%. The corresponding exact two-sided 95% binomial upper bounds range from 4.3% to 60.2%, with the widest intervals where only a few certificates are issued. These intervals quantify execution-level binomial uncertainty; because N counts executions rather than necessarily unique tasks, they should not be interpreted as task-cluster-robust intervals.

Table 5: Graph-independent audit of the held-out Fig. 1 AppWorld run (Gemini-3.5-Flash).

Effect class	Certified	Confirmed
Text message	5	5
Transaction / message	4	4
Song like	70	70
Playlist addition	10	10
Artist follow	34	34
Total	**123**	**123**

Graph-independent effect validation. To assess certified effects with an evaluator that does not consume DEG evidence, we replay the 13 executions receiving DEG certificates from the held-out Fig. 1 diagnostic and compare database deltas with the invoked endpoint, committed arguments, and bound target identifiers. None of the 13 executions satisfies the complete-task success criterion under the task-level evaluator, emphasizing that effect certification does not imply complete task success. Table 5 reports their 123 certified writes by effect class.

Finding 10: Graph-independent state deltas corroborate the certified write outcomes. All 123 certified writes are confirmed by the database deltas (Table 5). Because the audit does not use the DEG graph, read-back observation, or certificate decision, it independently checks the realized state transition relative to the committed and bound identifiers; it does not establish that the binding exhausts the user's intended target set.

B.3 Controlled Evaluation for Graph Structure and Evidence Sharing

Table 1 reports the controlled tests that support the graph view in Section 5.2. We detail how each intervention isolates one component—write–witness attribution, path-local relevance, or temporal validity—while preserving the remaining evidence, and then specify the shared-identity controls used in Fig. 4.

Write–witness attribution. To isolate attribution from observation content, we begin with 121 pooled multi-write τ-bench episodes containing 335 realized writes and select the 32 episodes in which the independent reference establishes different outcomes across writes. The resulting subset contains 85 realized writes and 38 reference-established writes. We keep all post-state observations fixed and cyclically permute only the write–witness association. Read-back-only (RB) is a separate attempt-blind control that uses target-keyed post-state evidence without any write-specific association; the permuted variant follows the deliberately reassigned witness; DEG retains the realized association. At the write-effect level, we report false-certification rate (FCR) and false-abstention rate (FAR) as $\mathrm{FCR} = N^{\mathrm{eff}}_{\mathrm{false}}/N^{\mathrm{eff}}_{\mathrm{issued}}$ and $\mathrm{FAR} = N_{\mathrm{withheld}}/N_{\mathrm{established}}$. Here $N^{\mathrm{eff}}_{\mathrm{false}}$ and $N^{\mathrm{eff}}_{\mathrm{issued}}$ count false and total issued effect certificates, respectively; $N_{\mathrm{established}}$ counts reference-established write effects, and N_{withheld} counts those effects for which the verifier withholds certification. The raw FCR counts are 32/70 for RB, 32/38 for the permuted association, and 0/38 for DEG; FAR is 0/38, 32/38, and 0/38, respectively.

Effect-local reachability. To test whether unrelated execution context can alter certification, we append an unconsumed opaque `semantic_select` branch outside the pending write's provenance region while preserving all write-relevant evidence. We compare path-restricted DEG, EFFECT-RECORD, which preserves the same effect-local provenance and verification semantics without shared graph identity, and a global FLAT scan. EFFECT-RECORD and FLAT are controlled variants constructed in this work. Table 6 reports task-level clean-to-perturbed behavior on AppWorld dev50. The Abst. column counts task executions with pre-commit status ABSTAIN, once per execution; it does not count all executions without a certificate. The 11 clean DEG certificates in Table 6 are task-execution counts. Separately, the retention statistic in Table 1 aggregates 262 challenged certificate decisions across 17 matched clean settings; each decision is re-evaluated after the same irrelevant opaque-branch intervention.

Table 6: Detailed AppWorld reachability intervention; clean→perturbed.

	Qwen3.5-122B			Gemini-3.5-Flash		
Variant	Cert.	Abst.	Passed	Cert.	Abst.	Passed
DEG	4→4	0→0	127→127	7→7	6→6	154→154
EFFECT-RECORD	4→4	0→0	127→127	7→7	6→6	154→154
FLAT	4→0	13→40	127→95	7→0	10→50	150→96

Temporal-validity intervention. We apply 51 controlled state changes after evidence acquisition: 22 later writes overlap the witness scope and 29 are disjoint. Static reuse never invalidates evidence, global invalidation discards evidence after every later write, and scope-aware versioning revalidates only overlapping scopes. Of the 22 overlaps, 11 produce stale-acceptance errors under static reuse and 11 are outcome-preserving after refresh. Global invalidation therefore creates 40 false abstentions—the 29 disjoint cases plus the 11 benign overlaps—while scope-aware versioning re-reads the 22 overlaps and incurs neither error. This intervention isolates temporal-validity semantics from evidence-sharing effects.

Controlled fan-out protocol. For Fig. 4, we hold realized effects, certificate predicates, verifier support, and execution controls fixed while varying evidence fan-out $f \in \{1, 2, 4, 8\}$. EFFECT-RECORD materializes supporting evidence independently per effect, whereas DEG stores one shared evidence node and reuses it across compatible effect regions. For revalidation, a later write overlaps a shared witness scope: EFFECT-RECORD refreshes each affected record, whereas DEG refreshes the shared scope state once. The number of eligible challenged groups differs across fan-out bins, so revalidation totals are compared within each f, not as a monotone workload series. At $f = 8$, the paired comparison is 40 per-record reads versus 5 shared-scope reads.

Natural evidence-sharing control. To check that the same reuse benefit appears outside the controlled fan-out construction, we compare DEG with EFFECT-RECORD on the matched AppWorld executions. Both variants preserve effect-local provenance, realized arguments, witness scope/version metadata, post-conditions, verifier support, routing logic, and certificate predicates; only evidence sharing differs.

Finding 11: Shared identity reduces redundant maintenance without changing certificate semantics. Table 7 shows identical certificate and execution outcomes for EFFECT-RECORD and DEG, while independent records require 9.9% and 13.7% more evidence-acquisition calls on Qwen and Gemini, respectively. Together with Fig. 4, this separates the two roles of the representation: path/-subgraph structure determines certificate semantics, while shared graph identity amortizes acquisition and revalidation when multiple effects reuse compatible evidence.

Table 7: Natural AppWorld evidence-sharing control.

	Qwen3.5-122B			Gemini-3.5-Flash		
Variant	Cert.	Passed	Calls	Cert.	Passed	Calls
EFFECT-RECORD	4	127	1260	7	154	1266
DEG	4	127	1147	7	154	1113

B.4 END-TO-END EXECUTION EVALUATION AND ABLATIONS

Matched component ablation. To separate certifiability, admission control, candidate selection, and routing, we ablate components on the same AppWorld dev50 tasks, native candidate pools, fixed-pool selector, replay evaluator, and two backbones used in Table 2. The "w/o Read-back" row is a certificate-only ablation: read-back is removed from certificate issuance but the recorded read-back signals used by routing are retained, so routing is held fixed. "Silent zero-effect commits" are non-error mutation commits for which the relevant replayed post-state remains unchanged. "Guard violation" counts replacements that an ablated router makes even though the DEG execution has at least one mutation with a recorded read-back pass ($N^{+}_{\mathrm{rb}}(G) > 0$) and no recorded violation ($N^{-}_{\mathrm{rb}}(G) = 0$). Full evidence-gated routing has zero such replacements by construction under Eq. (20); this diagnostic is therefore a structural routing check, not an independent safety metric.

Table 8: Full matched AppWorld component ablation. Passed sub-tests are out of 262 per backbone; guard viol. denotes replacements despite $N^{+}_{\mathrm{rb}}(G) > 0$ and $N^{-}_{\mathrm{rb}}(G) = 0$.

Variant	Qwen Passed sub-tests / 262	Gemini Passed sub-tests / 262	Qwen diagnostic	Gemini diagnostic
Full DEG	127	154	Certs 4; guard viol. 0	Certs 7; guard viol. 0
w/o Gate	127	161	36 silent zero-effect commits	21 silent zero-effect commits
w/o Read-back (certificate only)	127	154	0 certificates	0 certificates
w/o Binding repair	127	154	17 triggers skipped; 0 decisions lost	3 skipped; 0 lost
Never swap (DEG-only)	119	127	–	–
Count-only routing	130	159	guard viol. 1	guard viol. 3
Selector only (always swap)	109	153	guard viol. 6	guard viol. 10
Native-pool oracle	118	153	–	–

Finding 12: DEG components serve distinct certification and execution roles. Table 8 shows that removing read-back from certificate issuance leaves the execution scores unchanged at 127 and 154 passed sub-tests on Qwen and Gemini, respectively, but eliminates all four Qwen certificates and all seven Gemini certificates, confirming that post-state witnessing is necessary for certificate issuance in this setting. Disabling swaps reduces the scores to 119 on Qwen and 127 on Gemini, showing that routing contributes execution recovery. Removing the gate leaves Qwen at 127 passed sub-tests and raises Gemini to 161, while producing 36 Qwen and 21 Gemini silent zero-effect commits. This illustrates the gate's role as conservative admission control rather than a score-maximizing component. These observations separate certification controls from execution-utility mechanisms.

Cross-backbone evaluation. To assess whether the AppWorld execution pattern is confined to the two main backbones, we use a separate matched protocol with 38 sub-tests across five backbones and compare full DEG with the native-pool oracle. Table 9 reports the results.

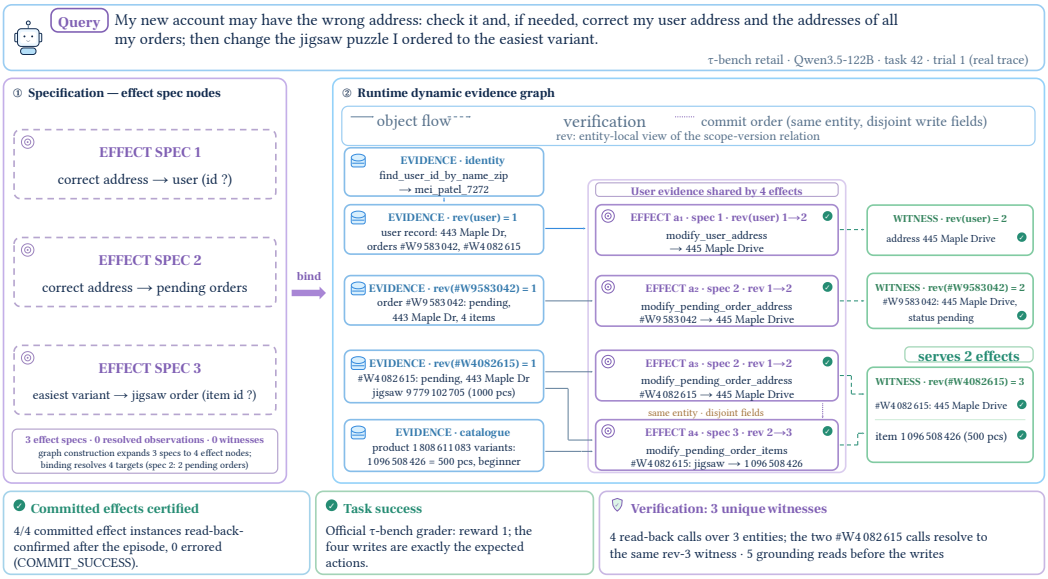


Figure 5: A real τ-bench execution illustrating graph-structured verification across four writes. Blue boxes denote grounding evidence, purple boxes effect instances, and green boxes post-state witnesses. Dashed links denote verification associations; dotted links show commit order between writes on the same entity.

Finding 13: The returned-execution advantage is not confined to the two primary backbones. Table 9 shows that DEG matches the native-pool oracle on Qwen3-8B and returns more passed sub-tests on each of the remaining four backbones, yielding 94/190 versus 83/190 in aggregate. Because the oracle is restricted to the matched native pool and this is a separate 38-sub-test protocol, the result should be read as a cross-backbone returned-execution comparison rather than a causal attribution to a single runtime component.

Table 9: Cross-backbone AppWorld evaluation.

Backbone	DEG	Native-pool oracle
Qwen3-8B	18/38	18/38
Qwen3-14B	18/38	15/38
Qwen3-32B	18/38	14/38
Gemini-2.5-Flash-Lite	18/38	15/38
Gemini-3.5-Flash	22/38	21/38
Total	**94/190**	83/190

Runtime profile. Finally, to quantify the absolute runtime footprint of evidence maintenance, we profile graph size and read-back overhead over 950 execution traces. Table 10 reports graph size, read-back frequency, and additional tool calls.

Table 10: DEG runtime profile.

Metric	Value
Execution traces	950
Graph nodes (median / mean / max)	4 / 4.6 / 15
Read-backs per task	2.3
Additional tool calls	≈15%

Finding 14: DEG maintains a small runtime graph with moderate evidence overhead. As shown in Table 10, the graph contains a median of 4 nodes (mean 4.6; maximum 15), while DEG performs 2.3 read-backs per task and incurs approximately 15% additional tool calls in the profiled executions. These measurements complement Fig. 4: the figure isolates how sharing changes maintenance cost, whereas Table 10 reports the absolute footprint in the evaluated traces.

B.5 Retail Case Study: Composing and Reusing Evidence Across Effects

Fig. 5 shows a real τ-bench retail query asking the agent to check and, if needed, correct the user's account address and the addresses of all orders, then change an ordered jigsaw puzzle to its easiest variant. The instruction induces three effect specifications (left), which DEG materializes as four effect instances that become four committed writes (right): one user-address update, two pending-

order address updates, and one item-variant update. During grounding and binding, the execution acquires five observations (blue boxes): the user identity, user record, two order records, and product catalogue. These observations resolve the four targets. The four effect instances (purple boxes) are then committed, followed by four read-back calls over three entities that yield qualified post-state evidence (green boxes).

The trace makes the graph structure explicit. Object-flow paths connect grounding evidence (blue boxes) to each realized write (purple boxes), preserving target provenance, while verification associations connect each write to qualified post-state evidence (green boxes). Each realized write therefore retains its own effect-local target–write–witness evidence region, even when nodes are shared. All four effects reuse user-level grounding evidence. On order #W4082615, the address write moves the entity from revision 1 to 2 and the later item-variant write from revision 2 to 3; because the second write changes a disjoint field, the revision-3 witness remains scope-compatible with the address post-condition and can support both effect-local regions without merging their certificate identities.

This case concretely illustrates the two roles of the representation: paths and connected subgraphs define the evidence used by graph-local certification queries, while shared graph identity and scope/version state allow compatible evidence to be reused and maintained across interacting writes. All four committed effects are certified, while the official τ-bench grader separately reports overall task success, keeping effect certification distinct from end-to-end task evaluation.