RGA: EVIDENCE-GROUNDED APPLICABILITY AUDITING FOR PROCEDURAL MEMORY REUSE

Anonymous authors
Paper under double-blind review

ABSTRACT

A procedure retrieved from memory can be relevant to a language agent's task yet inapplicable under current conditions. Reliable reuse therefore requires distinguishing evidence of inapplicability from insufficient evidence. We introduce the Rule-Grounded Applicability Auditor (RGA) to assess retrieved procedures before execution, given explicit applicability rules and an evidence policy. RGA combines learned evidence routing and three-valued semantic verification with deterministic evidence qualification and action aggregation. Unresolved conditions remain UNKNOWN, allowing the controller to distinguish REUSE, ADAPT, REJECT, and DEFER. Targeted counterfactual supervision trains the auditor to produce rule-level audits near applicability boundaries without directly supervising the final action. We also introduce RGABENCH, a controlled offline benchmark covering state conflicts, public-task transfer, source qualification, and evidence-representation shifts. With equal weighting across its four tracks, RGA achieves 84.90% action accuracy and 5.17% Harm, defined as the rate of incorrect executable decisions, compared with 44.97% and 45.68%, respectively, for the strongest evaluated mechanism-level control adapted to the same interface. Under this contract, ablations and compute-matched binary training show that collapsing unresolved evidence into true or false reduces action accuracy or increases incorrect executable decisions. These results support evidence-grounded applicability auditing as an explicit decision layer between procedural-memory retrieval and execution.

1 INTRODUCTION

Language-model agents increasingly turn experience into reusable procedures, skills, and workflows. Voyager builds an executable skill library (Wang et al., 2023), ExpeL distills guidance from prior experience (Zhao et al., 2024), and Agent Workflow Memory induces workflows from successful trajectories (Wang et al., 2024). Recent systems also learn skill conditions and refine procedural memory (Mi et al., 2026; Belikova et al., 2026). Yet retrieving a relevant procedure does not establish that the conditions supporting its previous success still hold in the current state.

Consider an agent that retrieves a workflow for modifying a retail order. Although the workflow may match the request, its applicability depends on the current order status, inventory, payment method, and policy window. A stale record or evidence referring to another order cannot establish these conditions. Moreover, a confirmed violation and unavailable evidence require different responses: treating both as failure can discard useful procedures, while treating missing evidence as permission can authorize unsupported execution.

We study *runtime applicability auditing*: the decision between procedure retrieval and execution. This requires separating several judgments that are often conflated. Retrieval determines whether a historical procedure is relevant; semantic verification determines what an observation establishes; evidence qualification determines whether that observation is admissible for establishing a condition; and the controller decides whether the resulting conditions support REUSE, ADAPT, REJECT, or DEFER. At the rule level, UNKNOWN denotes that admissible evidence establishes neither a condition nor its negation, rather than merely reflecting low model confidence.

We propose the *Rule-Grounded Applicability Auditor* (RGA), which implements this separation using a procedure-associated rule ledger and an explicit evidence policy. The ledger specifies condi-

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.

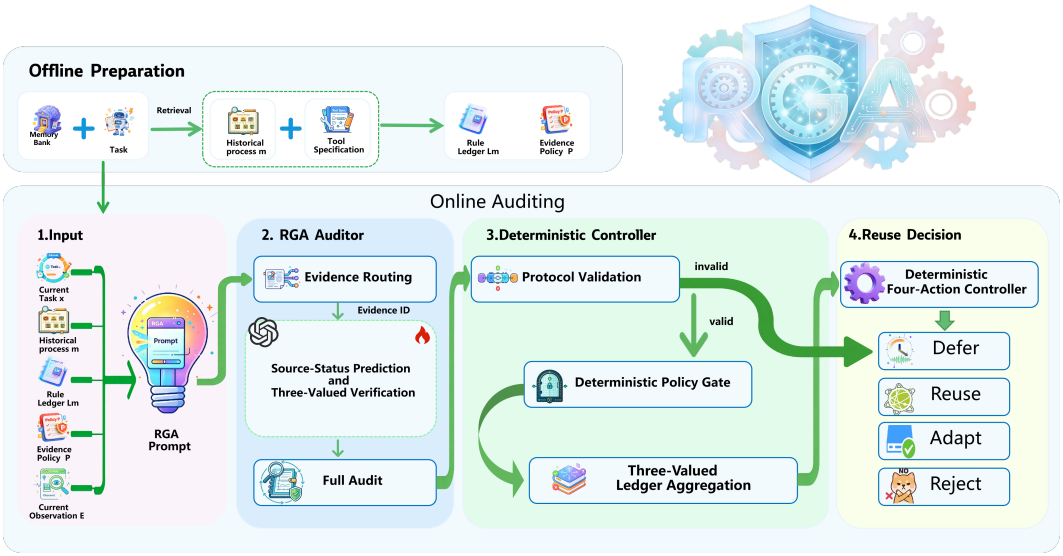

Figure 1: **Overview of the RGA pipeline.** The retrieved procedure, rule ledger, evidence policy, current task, and observations form the audit input. The learned auditor routes evidence and predicts rule-level semantics; deterministic qualification then enforces evidence admissibility before ledger aggregation maps the resulting (T, D, R) states to REUSE, ADAPT, REJECT, or DEFER.

tions for target compatibility, direct applicability, and recovery feasibility. The learned auditor routes observations to rules and predicts their semantic status, while a deterministic qualification module independently checks source, binding, temporal, and completeness requirements. Only qualified evidence may establish a rule; model-predicted source status does not itself authorize the evidence. The qualified rule verdicts are then aggregated into the final four-action decision (Figure 1).

We train the learned audit with structured supervision and minimal counterfactual examples that vary decision-critical conditions while preserving the surrounding task and procedure. Training targets evidence selection and rule verification rather than the final action, covering applicability boundaries, source qualification, entity binding, and heterogeneous evidence representations.

We introduce RGABENCH, a four-track benchmark covering state conflicts, transfer to public task settings, source qualification, and evidence-representation shifts. Across the four tracks, RGA achieves 84.90% track-averaged Action accuracy with 5.17% Harm, compared with 44.97% Action and 45.68% Harm for the strongest evaluated mechanism-level control under the same interface. Further analyses show that structured auditing is most useful when decisions depend on evidence admissibility, binding, and representation; frozen-output replay isolates the contribution of deterministic qualification, while matched verdict-space training shows the value of preserving unresolved evidence. Evaluation on 40 AppWorld task-generator families further examines transfer to an independent task source.

Our contributions are:

- We formulate procedural-memory reuse as a runtime applicability interface that separates retrieval relevance, semantic interpretation, evidence qualification, and execution decision, yielding actionable signals for downstream execution, repair, and evidence acquisition.
- We develop structured counterfactual supervision for learning rule-level evidence audits near applicability boundaries without directly supervising the final action.
- We introduce RGABENCH, addressing the lack of systematic evaluation for reliable procedural-memory reuse across state, source, and representation shifts. Together, RGA and RGABENCH establish an evaluation and decision framework for safer, more effective reuse of agent experience.

2 Related Work

Experience and procedural memory. Language agents store experience as self-feedback (Shinn et al., 2023), reusable guidance (Zhao et al., 2024), executable skills (Wang et al., 2023), and workflows (Wang et al., 2024). Subsequent work studies hierarchical memory (Forouzandeh et al., 2025), procedural-memory control and adaptation (Belikova et al., 2026), skill conditions and evolution (Mi et al., 2026), verified skill synthesis (Ma et al., 2026), trajectory-to-skill distillation (Ni et al., 2026), and executable skill programs (Liu et al., 2026a). SkillLens provides state-dependent applicability cues for GUI skills (Liu et al., 2026b), while Generative Skill Composition studies skill selection and composition (Zhao et al., 2026). RGA complements these memory mechanisms by auditing a retrieved procedure against current evidence.

Grounded reasoning and agent auditing. ReAct grounds reasoning in environment interaction (Yao et al., 2022). ToolSandbox (Lu et al., 2025) and τ-bench (Yao et al., 2024) evaluate tool use under state and policy constraints, while PrivacyPeek audits sensitive information acquired during execution (Zhang et al., 2026). These settings motivate treating runtime evidence as part of agent evaluation. Our focus is the applicability decision that precedes execution of a retrieved procedure.

Evidence verification and uncertainty. FEVER classifies claims as supported, refuted, or lacking sufficient evidence (Thorne et al., 2018), and selective prediction permits abstention when prediction risk is high (Geifman & El-Yaniv, 2019). RGA applies evidence-sensitive verification to procedural prerequisites: evidence must satisfy an explicit policy before it can establish a condition, and unresolved conditions must propagate to an appropriate reuse decision.

Skill conditions and runtime checks. SkillWrapper learns interpretable predicates and operators (Yang et al., 2025), while Skill-Pro learns reusable skills together with selection and execution conditions (Mi et al., 2026). Execution monitoring has also studied insufficient observations (Bouguerra et al., 2007), while Runtime Skill Audit probes reusable skills for security-sensitive behavior (Lan & Xiao, 2026). Our contribution integrates procedure-level conditions with independent evidence qualification and deterministic Reuse/Adapt/Reject/Defer boundaries. Appendix M compares these interfaces.

3 RGABench

3.1 Task Definition

Given the current task x, a retrieved procedure m, its rule ledger L_m, and available tool observations $E = \{e_1, \ldots, e_n\}$, the auditor determines whether current evidence supports procedure reuse. An evidence policy P specifies admissible sources, entity or selector bindings, temporal validity, and completeness requirements. Each instance is

$$\mathcal{I} = (x, m, L_m, E, P, \tau), \tag{1}$$

where decision time τ is supplied in the input context for temporal checks, with a target action

$$a \in \mathcal{A} = \{\textsc{Reuse}, \textsc{Adapt}, \textsc{Reject}, \textsc{Defer}\}. \tag{2}$$

Reuse denotes direct applicability, Adapt a supported recovery or rebinding path, Reject established inapplicability, and Defer an unresolved prerequisite. The task assumes a supplied rule ledger and evidence policy and evaluates the applicability decision before procedure execution.

3.2 Benchmark Construction

RGABench provides four controlled offline tracks (Table 1), illustrated in Figure 2. Examples form counterfactual groups that keep the task and retrieved procedure largely fixed while varying decision-critical conditions, such as temporal validity, entity binding, evidence availability, source qualification, and recovery feasibility. Evaluation and statistical testing preserve these groups.

RGABench contains four complementary tracks. **State**, derived from ToolSandbox task settings, tests conflicts in target compatibility, temporal validity, observation availability, entity binding, tool

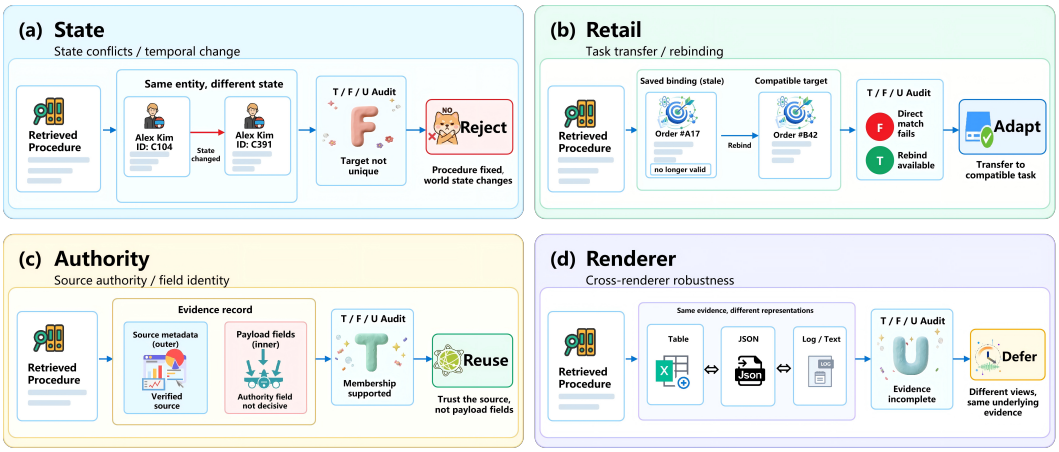

Figure 2: **Four complementary tracks in RGABENCH.** All tracks share the procedure-applicability auditing interface. (a) State tests changing world conditions and target ambiguity. (b) Retail tests transfer to public task settings and supported rebinding. (c) Authority separates source qualification from payload-local authority claims. (d) Renderer varies evidence representations while preserving their underlying semantics. The panels show illustrative cases; the displayed actions are examples, rather than the complete action distribution of each track.

Table 1: Overview of the four RGABENCH tracks. Scale denotes model examples / independent groups.

Track	Primary question	Provenance	Scale
State	Does the current state support execution?	ToolSandbox-derived	960 / 24
Retail	Does applicability reasoning transfer to public tasks?	τ-bench Retail-derived	128 / 32
Authority	Which evidence may legitimately establish a fact?	controlled held-out	384 / 96
Renderer	Does the same reasoning survive representation shifts?	controlled held-out	768 / 192

state, and recovery feasibility. **Retail** transfers the applicability task to public τ-bench Retail settings, pairing source tasks with procedures that support exact reuse, parameter rebinding, incompatibility, or unresolved applicability; evaluation uses our common offline auditing interface rather than native τ-bench interaction and reward. **Authority** isolates source qualification, where admissibility is determined by the external evidence policy rather than payload-local trust cues, and focuses on the REUSE/ADAPT boundary. **Renderer** varies raw evidence representations while preserving the task, procedure, rule semantics, and expected action, testing robustness across heterogeneous formats. State, Retail, and Renderer balance all four actions, whereas Authority balances REUSE and ADAPT; these distributions are designed as controlled stress tests of decision boundaries. Primary evaluations are separated from training and model selection at the relevant task, family, source-task, or group level, with split details and data checks provided in Appendices J and L. We additionally evaluate external-source transfer on 40 AppWorld task-generator families, reported separately from the four-track macro average (Section 5.6).

4 RULE-GROUNDED APPLICABILITY AUDITING

4.1 OVERVIEW

Figure 1 summarizes the preparation and online auditing pipeline. RGA separates three responsibilities: selecting relevant observations, interpreting their semantic content, and enforcing the evidence and action contract. The first two are learned; evidence qualification and action aggregation are deterministic. A procedure's rule ledger is instantiated from reusable applicability templates and specifies checkpoints for target compatibility, direct reuse, and recovery. It contains the checks and their composition, but no instance-specific verdicts or final action.

Unqualified evidence yields an effective UNKNOWN verdict; qualified rule verdicts are aggregated into the four-action decision.

4.2 RULE-GROUNDED APPLICABILITY REPRESENTATION

Let $L_m = \{r_1, \ldots, r_k\}$ denote the ordered rule ledger for procedure m. Semantic verification uses

$$y_i \in \mathcal{Y} = \{\text{TRUE}, \text{FALSE}, \text{UNKNOWN}\}, \tag{3}$$

where TRUE and FALSE represent evidence establishing a condition or its negation, and UNKNOWN represents an unresolved condition. The qualified verdicts $\bar{\mathbf{y}} = (\bar{y}_1, \ldots, \bar{y}_k)$ are composed into target compatibility T, direct applicability D, and recovery feasibility R:

$$T = \phi_T(\bar{\mathbf{y}}), \qquad D = \phi_D(\bar{\mathbf{y}}), \qquad R = \phi_R(\bar{\mathbf{y}}). \tag{4}$$

The ledger-defined functions use Kleene-style three-valued conjunction, disjunction, negation, and threshold operators. They preserve unresolved states when the available verdicts do not determine the expression's value. A deterministic controller maps (T, D, R) to REUSE for supported direct applicability, ADAPT for supported recovery when direct reuse is ruled out, REJECT for established inapplicability, and DEFER for unresolved prerequisites. Target compatibility is checked before direct reuse or recovery. The complete hierarchical mapping $g(T, D, R)$ is specified in Appendix A.

4.3 EVIDENCE ROUTING AND SOURCE QUALIFICATION

For each rule r_i, the learned router selects candidate observation indices:

$$I_i = R_\theta(x, m, r_i, E, P, \tau). \tag{5}$$

The evidence policy guides routing, while a separate deterministic function checks the selected observations:

$$q_i = Q(x, m, r_i, E_{I_i}, P, \tau) \in \{\texttt{verified}, \texttt{invalid}, \texttt{missing}\}. \tag{6}$$

Qualification checks source authority, entity or selector binding, temporal validity, and required completeness. A `verified` result means that the selected evidence satisfies these requirements; `invalid` and `missing` indicate violated or unavailable qualifying evidence.

Qualification uses provenance and binding information supplied by the interface and policy. Payload-local claims such as `authority:true` cannot authorize their own source. The learned verifier handles the semantic content of qualified observations, including conflicts; completeness checks apply when the interface specifies which evidence must be present.

4.4 THREE-VALUED VERIFICATION AND DETERMINISTIC AGGREGATION

Given the routed observations, the learned verifier predicts

$$\tilde{y}_i = V_\theta(x, m, r_i, E_{I_i}, P, \tau). \tag{7}$$

The qualification result determines whether this semantic prediction may establish an effective verdict:

$$\bar{y}_i = \begin{cases} \tilde{y}_i, & q_i = \texttt{verified}, \\ \text{UNKNOWN}, & q_i \in \{\texttt{invalid}, \texttt{missing}\}. \end{cases} \tag{8}$$

Thus, effective TRUE and FALSE verdicts require qualified evidence, while the learned verifier determines what that evidence establishes. The ledger aggregates effective verdicts into T, D, and R, and the controller mapping in Appendix A (Equation 10) produces the action.

Before qualification, the controller validates the structured audit, including rule coverage, evidence identifiers, and verdict values. Protocol-invalid outputs fall back to DEFER. We evaluate final action correctness and structured-audit validity separately (Appendices A and N). The resulting audit exposes the selected evidence, its qualification status, and the semantic verdict supporting each decision.

4.5 COUNTERFACTUAL TRAINING AND BOUNDARY TUNING

Training uses structured supervision generated from typed latent applicability states. Counterfactual groups preserve the task, procedure, ledger, entities, and non-decisive context while varying a decision-critical condition: target compatibility, temporal validity, entity binding, evidence availability or completeness, source qualification, recovery feasibility, or the decisive semantic claim. The targets are evidence identifiers and TRUE/FALSE/UNKNOWN semantic verdicts, trained with autoregressive supervised fine-tuning.

Minimal pairs concentrate supervision near applicability boundaries. Training progresses from clear cases to single-condition changes, irrelevant observations, source distractors, and representation shifts. Focused source-qualification pairs hold distracting payload-local fields fixed while changing the decisive semantic or qualification relation. Retention replay preserves the learned audit behavior under a stable output schema.

The counterfactual examples address both unsupported executable decisions and unnecessary deferral. Unresolved cases remain valid DEFER targets; training does not impose a global penalty on deferral. Evidence qualification and action aggregation remain deterministic throughout. Appendix B specifies the construction and optimization procedure.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Models and inference. All generative systems use Qwen3-8B (Team et al., 2025). RGA generates a structured rule audit followed by deterministic qualification and aggregation. Primary results report the main seed, with final-stage data-order sensitivity in Appendix I. The Direct-4A comparison and qualification replay use a separately rerun matched RGA reference, so their results differ slightly from the primary RGA evaluation.

Comparisons. The primary evaluation compares three mechanism-level controls under the same backbone and applicability interface. *Direct reuse* always selects REUSE. *AWM/ExpeL-style semantic reuse* directly predicts one of the four actions from the task, procedure, and evidence. *Skill-Pro-style binary gate* predicts binary rule conditions without explicit UNKNOWN or evidence provenance and uses deterministic aggregation. These controls adapt representative reuse mechanisms to the common task. We additionally evaluate Direct-4Action SFT (Direct-4A), which receives the complete decision context but directly predicts the final action, together with controlled qualification and verdict-space ablations.

Direct-action SFT. Direct-4A receives the task, retrieved procedure, rule ledger, tool evidence, evidence policy, and decision time, and predicts a strict JSON action from $\{\text{REUSE}, \text{ADAPT}, \text{REJECT}, \text{DEFER}\}$. Appendix C provides training details and resources.

Metrics and statistics. *Action* measures final-action accuracy. *Harm* is the proportion of examples assigned an incorrect executable action (REUSE or ADAPT), and *false deferral* (FD) is the proportion assigned DEFER when another action is required. Four-track averages weight tracks equally. Paired analyses use independent families, source tasks, or counterfactual groups rather than individual model calls. Complete metric and statistical definitions are given in Appendices J, K, and N.

5.2 OVERALL PERFORMANCE ON RGABENCH

RGA achieves 84.90% macro Action accuracy and 5.17% Harm (Table 2). Relative to the strongest evaluated mechanism-level control, the Skill-Pro-style binary gate, this improves Action by 39.93 percentage points and reduces Harm by 40.51 points.

The gains extend across independent evaluation units: RGA improves 21/24 State families, 22/32 Retail source tasks, 94/96 Authority groups, and 188/192 Renderer groups over the strongest control on each track. All four Action comparisons are significant under the grouped sign test ($p \leq 2.77 \times$

Table 2: Primary results on RGABENCH. Entries are Action/Harm/FD (%). Avg. is the equal-weight macro average across tracks.

System	State	Retail	Authority	Renderer	Avg.
Direct reuse	25.00/75.00/0.00	25.00/75.00/0.00	50.00/50.00/0.00	25.00/75.00/0.00	31.25/68.75/0.00
AWM/ExpeL-style	25.62/72.19/1.04	67.19/6.25/0.78	50.00/50.00/0.00	25.13/74.87/0.00	41.99/50.83/0.46
Skill-Pro-style	28.85/58.75/0.00	75.00/**0.00**/0.00	51.04/48.96/0.00	25.00/75.00/0.00	44.97/45.68/0.00
RGA	**53.54/18.33/3.54**	**92.19**/0.78/0.00	**97.66/1.56/0.78**	**96.22/0.00/3.78**	**84.90/5.17/2.02**

Table 3: Audit-versus-action comparison. Entries are Action/Harm/FD (%); Direct-4A averages seeds 11, 23, and 37. RGA is separately rerun under the matched supplementary comparison setting, so its values differ slightly from the primary results in Table 2.

Track	Direct-4A (three-seed mean)	RGA
State	**57.05**/19.48/10.42	52.92/**19.38/3.65**
Retail	**97.14**/2.60/0.26	92.97/**0.78/0.00**
Authority	47.14/46.18/2.26	**97.66/1.30/1.04**
Renderer	35.11/35.29/21.92	**96.22/0.13/3.65**

10^{-4}; Appendix K). The following experiments examine when structured auditing is useful and which parts of the decision interface account for its behavior.

5.3 WHEN DOES STRUCTURED AUDITING HELP?

Table 3 compares Direct-4A with RGA. Both receive the complete decision context, but Direct-4A learns the final four-way action directly, whereas RGA predicts an intermediate evidence-and-rule audit before deterministic aggregation.

Direct-4A attains higher Action on State and Retail by 4.13 and 4.17 points. In contrast, RGA improves Authority and Renderer Action by 50.52 and 61.11 points, reducing Harm from 46.18% to 1.30% and from 35.29% to 0.13%, respectively. Explicit auditing thus offers its largest gains when decisions depend on source admissibility, binding, and evidence representation. RGA also has lower mean FD on all four tracks.

5.4 HOW DOES QUALIFICATION CHANGE DECISIONS?

We isolate deterministic qualification by replaying the same frozen RGA outputs through two controllers. *Full-Q* maps semantic verdicts to UNKNOWN when selected evidence is missing or disqualified, whereas *No-Q* passes the same semantic verdicts directly to ledger aggregation. All learned outputs and downstream controller components are otherwise fixed.

As shown in Table 4, on Renderer, qualification increases Action from 81.77% to 96.22% and reduces Harm from 14.58% to 0.13%. It overrides 141 non-UNKNOWN rule verdicts, changing 111 of 768 final actions. Because the learned predictions are fixed, this directly shows that evidence admissibility can independently alter the decision: semantic verification determines what an observation appears to establish, while qualification determines whether it may establish that condition. Authority produces no final-action changes under the replay despite the large audit-versus-Direct-4A gap. Its improvement therefore arises upstream of the terminal gate, from the structured learned audit rather than qualification alone. Appendix E gives the complete replay details.

5.5 WHY PRESERVE AN UNRESOLVED STATE?

The role of UNKNOWN is not merely to add a third label. Under the four-action controller, an established contradiction and an unresolved condition can require different downstream decisions; collapsing them therefore removes decision-relevant information.

Complementing the qualification replay in Table 4, we test this by holding model predictions and qualification fixed while replacing each effective UNKNOWN before aggregation. Mapping UN-

Table 4: Frozen-output qualification replay. Full-Q applies deterministic qualification, whereas No-Q bypasses it with the same frozen model outputs. Entries are Action/Harm/FD (%); Changed actions counts altered final decisions.

Track	Full-Q	No-Q	Changed actions
State	52.92/19.38/3.65	52.92/19.38/3.65	0/960
Retail	92.97/0.78/0.00	92.97/0.78/0.00	0/128
Authority	97.66/1.30/1.04	97.66/1.30/1.04	0/384
Renderer	**96.22/0.13**/3.65	81.77/14.58/3.65	111/768

Table 5: AppWorld offline external-source transfer over 40 distinct task-generator families. All values are percentages. Direct-4A averages three frozen seeds.

System	Action ↑	Harm ↓	FD ↓	Coverage	Validity
Direct-4A mean	52.50	18.33	17.50	45.00	100.00
RGA	62.50	20.00	7.50	62.50	100.00

KNOWN to FALSE reduces Action by 25.00 points on Retail and 21.35 points on Renderer, whereas mapping UNKNOWN to TRUE increases Harm by 25.00 and 22.40 points, respectively. The two collapses fail in different ways. Collapsing UNKNOWN into FALSE treats unresolved conditions as established blockers and reduces applicability utility, whereas collapsing UNKNOWN into TRUE treats them as established support and increases incorrect executable decisions.

We then ask whether native binary training can compensate for this information loss. Full-3V, Binary-UF, and Binary-UT are trained from the same base with matched data, ordering, update budget, scheduler, and seed. Full-3V reaches 100.00% Action on Retail versus 75.00% for both binary variants, and 94.01% on Renderer versus 64.32% and 74.87%. State shows the same qualitative pattern. Thus, adapting the learner to binary targets does not recover the distinction required by the four-action interface. Authority contains only REUSE and ADAPT labels, and Binary-UT performs best there. The value of preserving UNKNOWN therefore depends on whether the action boundary distinguishes unresolved applicability. Full results appear in Appendix F.

Counterfactual supervision sharpens audit boundaries. In a matched component study, targeted tuning improves renderer-diagnostic Action/Harm from 85.00/12.19 to 93.75/1.88 while retaining 100% Action, zero Harm, and zero FD on the anti-over-defer split. Source-qualification tuning further improves Authority and transfers across evidence formats (Appendices G and H).

5.6 OFFLINE EXTERNAL-SOURCE TRANSFER TO APPWORLD

We first test whether the observed behavior extends beyond the controlled RGABENCH sources. We freeze 40 AppWorld 0.2.0 tasks (Trivedi et al., 2024) from distinct task-generator families, with ten instances per action. Inputs use the original task instructions, required APIs, and task-instance relations; models and checkpoints are frozen with no AppWorld training or model selection. Appendix D documents construction and contamination checks.

As shown in Table 5, relative to Direct-4A, RGA increases Action and execution coverage by 10.00 and 17.50 points and reduces FD by 10.00 points. The results suggest transfer of decision accuracy and coverage, without a corresponding reduction in incorrect executable decisions.

5.7 END-TO-END AGENT EXECUTION ON APPWORLD

We next test whether applicability auditing translates into improved downstream task completion. We construct 196 controlled AppWorld execution episodes from 49 public `train`/`dev` task-generator families, with four matched memory conditions per target: a same-instance procedure, a same-family sibling procedure, a different-family procedure, and an initially unresolved procedure. Within each family, the target task and initial environment are fixed across conditions. Both pipelines use Qwen3-8B and receive the same candidate memory available in each episode. Appendix O details the construction and information controls.

Table 6: End-to-end execution over 196 controlled AppWorld episodes (49 task-generator families × four matched memory conditions) with Qwen3-8B. Branch success is conditional on episodes routed to the corresponding RGA execution branch.

Pipeline / branch	Successful / evaluated	Success (%)
Agent without RGA	39 / 196	19.9
Agent with RGA	58 / 196	29.6
Direct reuse in the RGA pipeline	36 / 45	80.0
Memory repair in the RGA pipeline	22 / 45	48.9

Without RGA, the agent decides whether to execute the supplied historical trajectory or generate its own, optionally using that history as a reference. With RGA, REUSE executes the trajectory directly; ADAPT triggers repair followed by re-auditing; DEFER acquires the permitted missing evidence and repeats the audit; and REJECT falls back to the same planning path as the unaudited agent. This experiment therefore evaluates the integrated audit-and-execution loop, complementary to the 40-family offline Action evaluation in Section 5.6. Because the four episodes for a target share the same task-generator family, statistical inference treats the 49 families, rather than the 196 episodes, as independent units.

Table 6 shows a 9.7-point increase in final task success, corresponding to 19 additional completed episodes. Within the audited pipeline, 36 episodes are completed through direct reuse and 22 through repaired memory execution. The remaining 106 episodes were routed through non-executable/fallback branches and yielded no successful completions. The latter result shows that procedures unsuitable for direct execution can remain useful after adaptation, while its 48.9% conditional success rate also exposes downstream repair and execution as a remaining bottleneck.

6 DISCUSSION

Qualification and semantic learning play different roles. Qualification replay improves Renderer decisions with fixed predictions but leaves Authority actions unchanged. Together with matched verdict-space training, this supports complementary roles: learning interprets evidence, while the gate determines whether it may establish a condition.

Transfer and scope. The 40-family offline AppWorld evaluation broadens task provenance and shows favorable accuracy, false-deferral, and coverage trends, with Harm and paired statistics delimiting those trends. The complementary 196-episode execution study connects auditing to higher final task success and successful memory repair. Together, the studies evaluate both the decision interface under supplied specifications and its use in an agent loop, where ADAPT invokes repair and DEFER acquires evidence before re-auditing.

7 CONCLUSION

We introduced RGA, an applicability auditor combining learned evidence routing and three-valued verification with deterministic qualification and action aggregation. On RGABENCH, the primary RGA reaches 84.90% macro Action accuracy with 5.17% Harm. Supplementary comparisons against three-seed direct-action SFT locate the audit's strongest advantages in source qualification and heterogeneous evidence. Frozen-output replay isolates the qualification gate's Renderer contribution without increasing false deferrals, while matched training supports preserving unresolved evidence at four-action boundaries. AppWorld evaluation combines offline transfer across 40 task families with end-to-end execution on 196 episodes, where the audited pipeline improves task success from 19.9% to 29.6% and recovers useful memories through repair. These findings support applicability auditing as a decision layer that helps agents both reuse and recover procedural memory during execution.

AI USE STATEMENT

Generative AI tools assisted with literature search, code debugging, and manuscript editing and organization. The research problem, methodology, benchmark design, data construction, experimental setup, and scientific conclusions were developed and determined by the authors. The authors reviewed and verified AI-assisted content, including references, code suggestions, and edited text, and take responsibility for the final paper.

REPRODUCIBILITY STATEMENT

Section 4 and Appendix A specify the audit, qualification, aggregation, and protocol semantics. Appendices B and C document structured training, direct-action supervision, seeds, checkpoint selection, and measured Direct-4A training resources. Appendix E specifies the frozen-output replay of RGA. AppWorld provenance and the 156-file contamination audit appear in Appendix D for the 40-family offline study. Section 5.7 specifies the complementary execution comparison and reports task-level and branch-conditional success. Benchmark units, paired inference, and metrics are defined in Appendices J, K, and N. Each comparison identifies its reference configuration.

REFERENCES

Julia Belikova, Rauf Parchiev, Evgeny Egorov, Grigorii Davydenko, Gleb Gusev, Andrey Savchenko, and Maksim Makarenko. Managing procedural memory in llm agents: Control, adaptation, and evaluation. *arXiv preprint arXiv:2606.23127*, 2026.

Abdelbaki Bouguerra, Lars Karlsson, and Alessandro Saffiotti. Active execution monitoring using planning and semantic knowledge. In *ICAPS Workshop on Planning and Plan Execution for Real-World Systems*, 2007.

Saman Forouzandeh, Wei Peng, Parham Moradi, Xinghuo Yu, and Mahdi Jalili. Learning hierarchical procedural memory for llm agents through bayesian selection and contrastive refinement. *arXiv preprint arXiv:2512.18950*, 2025.

Yonatan Geifman and Ran El-Yaniv. Selectivenet: A deep neural network with an integrated reject option, 2019. URL https://arxiv.org/abs/1901.09192.

Tu Lan and Chaowei Xiao. Runtime skill audit: Targeted runtime probing for agent skill security. *arXiv preprint arXiv:2606.11671*, 2026.

Hongjun Liu, Yifei Ming, Shafiq Joty, and Chen Zhao. Harnessing llm agents with skill programs. *arXiv preprint arXiv:2605.17734*, 2026a.

Zhou Liu, Ligang Huang, Zeli Su, Zewei Pan, Zhaoyang Han, Xing Chen, Yuanfeng Song, and Wentao Zhang. Skilllens: Visual skill cards for retrieval-augmented gui action prediction and on-policy distillation. *arXiv preprint arXiv:2608.10775*, 2026b.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. 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, 2025.

Yuchen Ma, Yue Huang, Han Bao, Haomin Zhuang, Swadheen Shukla, Michel Galley, Xiangliang Zhang, and Stefan Feuerriegel. Skillgen: Verified inference-time agent skill synthesis. *arXiv preprint arXiv:2605.10999*, 2026.

Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. Skill-pro: Learning reusable skills from experience via non-parametric ppo for llm agents. *arXiv preprint arXiv:2602.01869*, 2026.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Qwen Team et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 6(7):13, 2025.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: a large-scale dataset for fact extraction and verification, 2018. URL https://arxiv.org/abs/1803.05355.

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.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Ziyi Yang, Benned Hedegaard, Ahmed Jaafar, Yichen Wei, Skye Thompson, Shreyas Sundara Raman, Haotian Fu, Stefanie Tellex, George Konidaris, David Paulius, et al. Skillwrapper: Generative predicate invention for skill abstraction. 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Mingxuan Zhang, Jiahui Han, Dadi Guo, Songze Li, Guanchu Wang, Na Zou, Dongrui Liu, and Xia Hu. Privacypeek: Auditing what llm-based agents acquire, not just what they say. *arXiv preprint arXiv:2606.00152*, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Xinyu Zhao, Zhen Tan, Vaishnav Tadiparthi, Nakul Agarwal, Kwonjoon Lee, Ehsan Moradi Pari, Hossein Nourkhiz Mahjoub, and Tianlong Chen. Generative skill composition for llm agents. *arXiv preprint arXiv:2606.32025*, 2026.

A Deterministic Aggregation and Protocol Semantics

The rule ledger uses Kleene-style three-valued logic over {TRUE, FALSE, UNKNOWN} and supports `and`, `or`, `not`, and `at_least(k)` operators.

For conjunction, the result is FALSE if any operand is FALSE, UNKNOWN if no operand is FALSE but at least one is UNKNOWN, and TRUE otherwise. Disjunction is defined dually: it is TRUE if any operand is TRUE, UNKNOWN if no operand is TRUE but at least one is UNKNOWN, and FALSE otherwise. Negation exchanges TRUE and FALSE while preserving UNKNOWN.

For a threshold expression `at_least(k)`, let n_{TRUE} and n_{UNKNOWN} denote the numbers of TRUE and UNKNOWN operands. Its value is

$$\text{at_least}(k) = \begin{cases} \text{TRUE}, & n_{\text{TRUE}} \geq k, \\ \text{FALSE}, & n_{\text{TRUE}} + n_{\text{UNKNOWN}} < k, \\ \text{UNKNOWN}, & \text{otherwise.} \end{cases} \tag{9}$$

Thus, a composite expression remains UNKNOWN whenever the currently established effective verdicts are insufficient to determine its value.

These operators instantiate the ledger expressions ϕ_T, ϕ_D, and ϕ_R in Section 4. They compose the policy-gated effective verdicts $\bar{\mathbf{y}} = (\bar{y}_1, \ldots, \bar{y}_k)$ into target compatibility T, direct applicability D, and recovery feasibility R.

Given (T, D, R), the controller uses the following deterministic mapping, without further language-model generation:

$$g(T, D, R) = \begin{cases} \textsc{Defer} & T = \textsc{Unknown}, \\ \textsc{Reject} & T = \textsc{False}, \\ \textsc{Defer} & T = \textsc{True}, D = \textsc{Unknown}, \\ \textsc{Reuse} & T = \textsc{True}, D = \textsc{True}, \\ \textsc{Defer} & T = \textsc{True}, D = \textsc{False}, R = \textsc{Unknown}, \\ \textsc{Adapt} & T = \textsc{True}, D = \textsc{False}, R = \textsc{True}, \\ \textsc{Reject} & T = \textsc{True}, D = \textsc{False}, R = \textsc{False}. \end{cases} \tag{10}$$

The controller follows an ordered applicability semantics: target compatibility is treated as a prerequisite for direct applicability, and recovery feasibility is consulted for action selection only when the target has been established as compatible and direct reuse has been established as false. This ordering distinguishes established inapplicability, supported recovery, and unresolved prerequisites under the applicability contract.

Equation 10 defines a hierarchical contract: an unresolved prerequisite yields DEFER even when all Boolean completions would imply the same downstream action. The controller follows this prerequisite ordering without enumerating completions.

At inference time, the structured model output is validated before evidence qualification and aggregation. Each ledger rule must appear exactly once, all referenced rule and evidence identifiers must be valid, and every semantic verdict must belong to $\{\textsc{True}, \textsc{False}, \textsc{Unknown}\}$. Malformed audit structures, missing or duplicated rules, invalid identifiers, and illegal verdict values are treated as protocol-invalid outputs.

Protocol-invalid outputs fall back to DEFER. Such a prediction is Action-correct only when DEFER is the expected label, while the underlying audit remains protocol-invalid. Action accuracy and protocol validity are therefore reported as separate measures.

B Implementation and Training Details

B.1 Model Configuration

All trainable systems use Qwen3-8B with rank-32 LoRA adaptation. Table 7 summarizes the structured-audit training and inference configuration. The directly supervised action baseline is detailed separately in Appendix C.

All reported generative evaluations use greedy decoding with model thinking disabled. For RGA, the language model produces a structured audit containing rule-level evidence identifiers and semantic verdicts. Evidence qualification, policy gating, ledger aggregation, and the final REUSE, ADAPT, REJECT, or DEFER action are computed deterministically as described in Section 4. For RGA, the final action is therefore computed from the audit. Direct-4A instead learns the final action directly (Appendix C).

B.2 Rule-Ledger Construction

Rule ledgers are instantiated from reusable applicability templates rather than manually tailored to individual benchmark instances or expected solutions. The templates define generic checkpoints such as target compatibility, direct usability, temporal validity, entity or selector binding, evidence availability, and recovery feasibility, together with their target, direct, and recovery aggregation structure.

Table 7: Structured-audit model, optimization, and inference configuration.

Configuration	Value
Backbone	Qwen3-8B
LoRA rank / alpha	32 / 64
LoRA dropout	0.05
LoRA target modules	q/k/v/o and gate/up/down projections
Precision	bfloat16
Optimizer	AdamW
Weight decay	0.01
Global batch size	16
Scheduler	cosine decay
Gradient checkpointing	enabled
Inference decoding	greedy
Model thinking	disabled

The data builder instantiates these reusable templates using the retrieved procedure and relevant tool or interface metadata. This process specifies *what must be checked* and how the checks are composed, but not *whether the conditions currently hold*. In particular, a ledger contains neither runtime TRUE/FALSE/UNKNOWN verdicts nor the expected final REUSE, ADAPT, REJECT, or DEFER action.

The instantiated ledger is stored with each training and evaluation example and supplied at inference time. This shared specification defines the conditions whose applicability must be established from runtime evidence.

The evidence policy P is specified separately from the ledger. It defines the admissibility requirements that can be checked from the available tool/interface representation, including permitted sources, entity or selector bindings, temporal constraints, and required completeness conditions. Neither the ledger nor P encodes whether the observations in a particular instance satisfy those requirements.

Deterministic qualification enforces properties represented by the policy and evidence interface. Semantic interpretation is handled by the learned verifier; evidence completeness is checked against requirements exposed by the interface.

B.3 STRUCTURED AUDIT SUPERVISION

Training uses a stable structured-audit contract throughout. For each procedure, the target output contains every ledger rule together with the evidence identifiers selected for that rule and its semantic verdict $\tilde{y}_i \in \{\text{TRUE}, \text{FALSE}, \text{UNKNOWN}\}$. The language model is not supervised to directly generate the final applicability action.

The target semantic verdict $\tilde{y}_i$ represents what the selected observations semantically establish for the rule. Final evidence qualification q_i, the policy-gated effective verdict $\bar{y}_i$, and the final action are computed through the deterministic mechanisms described in Section 4.4. Thus, the training target does not replace the independent evidence-policy enforcement performed at inference time.

Training begins with factorized supervision for evidence routing and rule-level semantic verification. The factorized stage uses 1,050 optimizer updates with a learning rate of 2×10^{-5}, followed by 1,920 full-audit updates with a learning rate of 1×10^{-5}. Both stages use 3% warmup, and rule-level semantic supervision is balanced across TRUE, FALSE, and UNKNOWN.

Factorized supervision covers complementary rule semantics including existence, quantification, temporal validity, logical composition, entity binding, and nested-record reasoning. Evidence-routing targets teach the model which observations should be associated with each rule given the task, procedure, ledger, evidence, and evidence policy P. Semantic-verification targets teach what the selected observations establish with respect to the rule. Full-audit supervision combines routing and verification predictions under the same output contract, avoiding changes to the decoder interface between training stages.

B.4 COUNTERFACTUAL TRAINING AND BOUNDARY TUNING

Subsequent boundary tuning uses structured counterfactual supervision generated from typed latent applicability states. Rather than constructing independent examples, we form counterfactual groups that preserve the task, retrieved procedure, ledger structure, entities, and non-decisive context while varying one or a small number of decision-critical conditions.

The interventions cover target compatibility, temporal validity, argument compatibility, entity or selector binding, evidence availability and completeness, source qualification, recovery feasibility, and the decisive semantic claim. The resulting supervision covers four complementary categories:

- **Rule-level supervision**, which trains evidence routing and TRUE/FALSE/UNKNOWN semantic verification;
- **Action-boundary supervision**, which varies target, direct, and recovery conditions so that the learned rule audit supports distinctions among REUSE, ADAPT, REJECT, and DEFER after deterministic aggregation;
- **Source-qualification supervision**, which varies the relationship between observations and the evidence policy P, including source authority, entity or selector binding, temporal validity, and completeness requirements; and
- **Representation-diverse supervision**, which renders equivalent latent semantics through heterogeneous evidence structures.

Minimal counterfactual pairs concentrate supervision near applicability boundaries. Such pairs preserve the surrounding task, procedure, ledger, and non-decisive records while changing only a decision-critical condition, for example whether an observation is current, whether its entity binding matches, whether qualifying evidence is available, whether a supported recovery path exists, or whether the decisive semantic claim holds. This construction encourages the learned audit to respond to decision-critical variables rather than incidental lexical or structural correlations.

Counterfactual supervision is also organized by increasing ambiguity. Easier examples contain clearly supported conditions or multiple obvious blockers; intermediate examples isolate a single decision-critical condition or compose several aligned rules; harder examples introduce irrelevant observations yielding UNKNOWN, source-level distractors, representation shifts, and minimal action-changing pairs. The progression exposes the model to both clear cases and fine-grained applicability boundaries.

Targeted boundary tuning further includes balanced action examples and focused authority pairs. Balanced action examples provide supervision against both unsupported executable decisions and unnecessary deferral. Focused authority pairs separate the decisive semantic claim from neighboring payload-local fields that merely resemble authority indicators, discouraging field-identity shortcuts.

Retention replay is mixed with targeted boundary tuning to preserve previously learned evidence routing, semantic verification, four-action boundaries, and the stable structured-audit contract. We do not impose a global penalty against DEFER: unresolved cases remain valid DEFER targets, while unnecessary deferral on answerable cases is corrected through explicit counterfactual supervision.

All generated supervision is validated using the same deterministic qualification and aggregation semantics used during evaluation. We first construct the gold rule-level semantic audit, apply the evidence policy P and deterministic qualification rules to obtain effective verdicts, and then derive the expected action using Equation 10. Examples whose gold audit, policy-gated effective verdicts, and expected action are inconsistent are excluded.

Action-boundary examples supervise evidence selection and semantic verdicts in cases where small changes alter the deterministically derived action. They use the same audit objective as other training examples.

B.5 Training Objective

For a structured audit target $z = (z_1, \ldots, z_{|z|})$, the model is optimized with the standard autoregressive negative log-likelihood

$$\mathcal{L}_{\mathrm{SFT}} = -\sum_{t=1}^{|z|} \log p_\theta \left(z_t \mid z_{<t}, x, m, L_m, E, P, \tau \right). \tag{11}$$

The evidence policy P is therefore part of the model context during both training and inference. The target sequence z contains the structured audit, including rule identifiers, selected evidence identifiers, and semantic verdicts $\tilde{y}_i$, but not the final applicability action.

Final evidence qualification is enforced independently by the deterministic function Q. The controller combines q_i with the model-predicted $\tilde{y}_i$ through Equation 8 to obtain the effective verdict $\bar{y}_i$. The ledger expressions then produce T, D, and R, after which Equation 10 deterministically determines the final action.

Counterfactual boundary tuning and retention replay use the same autoregressive objective and the same structured output contract. No separate action-prediction objective, probability-calibration loss, or global anti-DEFER objective is used in the final training pipeline.

C Direct-4Action SFT: Supervision and Resources

Direct-4A is the direct-action SFT model. Its input contains the full task, retrieved procedure, rule ledger, tool evidence, evidence policy, and decision time, so the comparison supplies the same decision-relevant information to both approaches. Its target is a strict JSON action from {REUSE, ADAPT, REJECT, DEFER}. The backbone is Qwen3-8B with LoRA rank 32 and alpha 64.

The training set contains 23,040 complete instances, balanced at 5,760 per action. Each of seeds 11, 23, and 37 uses 1,440 optimizer updates and global batch size 16. All three development-set selections choose step 1,440; test results are not used for checkpoint or seed selection. The reported three-seed mean includes all three trained models. Direct-4A generates an average of seven output tokens per evaluated example.

Table 8: Measured Direct-4A training resources. Aggregate GPU-hours sum device-hours across the GPUs used by a run. The total sums all three seeds.

Seed	Training tokens	Wall time (s)	Aggregate GPU-hours
11	36,486,310	5,647.6	6.275
23	36,486,310	5,324.4	5.916
37	36,486,310	5,328.1	5.920
Total	109,458,930	16,300.1	18.111

Direct-4A and RGA share the backbone, adapter rank, and full input interface but optimize different targets through their respective training schedules. The compute-matched experiment is the separate Full-3V/Binary-UF/Binary-UT comparison in Appendix F. This distinction separates the practical direct-action comparison from the controlled verdict-space study.

D AppWorld Offline External-Source Transfer

D.1 Provenance and Independent Evaluation Units

The task source is AppWorld 0.2.0; the recorded source license is Apache-2.0. The official repository, pinned commit, and abbreviated dataset archive fingerprint are:

`https://github.com/StonyBrookNLP/appworld`
Commit: `42b5bcf3cd334fee33f0c37c02070a9f5807add5`
SHA-256 (abbreviated): `c9299e6c...b2f532e`.

The frozen set contains 25 tasks from the public AppWorld training split and 15 from its public development split. These split names describe the upstream source: all 40 tasks are evaluation-only for the models studied here. Each task comes from a different native task-generator family. One audit instance is retained per family, yielding 40 distinct statistical units and ten examples for each action. Counterfactual views are not counted as additional cases.

Audit construction uses the original task instruction, official required APIs, and task-instance relations. These supply the task context and API/entity relations for the retrieved procedure, ledger, evidence, and evidence-policy interface. Models predict the resulting applicability action under the same four-action contract as RGABENCH. This is an AppWorld-derived offline applicability evaluation, distinct from AppWorld's native end-to-end execution score.

Model parameters, checkpoints, and decoding settings are frozen; AppWorld is used for evaluation only, with no training, tuning, or model selection. External-source transfer claims refer to this 40-family evaluation.

D.2 CONTAMINATION AND LABEL-LEAKAGE AUDIT

We audit 156 existing training/tuning JSONL files against the frozen AppWorld evaluation. Table 9 summarizes the checks, including action-label leakage in the evaluation inputs.

Table 9: AppWorld contamination audit over the 156 audited training/tuning JSONL files and evaluation inputs.

Check	Result
AppWorld canary matches	0
AppWorld name or source-marker matches	0
Exact matches to the 40 normalized task instructions	0
Maximum five-word-gram overlap ratio	0
Gold action-label leakage in model inputs	0

D.3 ALL-SEED RESULTS AND INFERENCE COST

Table 10: AppWorld results for every evaluated seed. Metrics other than tokens are percentages over the 40 independent tasks. Direct-4A mean averages the three seeds; RGA uses the frozen seed-0 model.

System	Action	Harm	FD	Coverage	Validity	Output tokens
Direct-4A seed 11	47.50	20.00	20.00	45.00	100.00	7.00
Direct-4A seed 23	60.00	25.00	10.00	55.00	100.00	7.00
Direct-4A seed 37	50.00	10.00	22.50	35.00	100.00	7.00
Direct-4A mean	52.50	18.33	17.50	45.00	100.00	7.00
RGA seed 0	62.50	20.00	7.50	62.50	100.00	149.35

For RGA, 25 of 40 decisions are correct, eight are incorrect executable decisions, and three are false deferrals. It executes on 25 tasks, compared with a mean of 18 for Direct-4A. Its structured audit is longer than the direct-action response, averaging 149.35 versus 7.00 generated tokens. All systems achieve 40/40 protocol-valid outputs.

The observed differences relative to the direct-baseline mean are $+10.00$ points in Action, -10.00 in FD, $+17.50$ in coverage, and $+1.67$ in Harm. These support a favorable accuracy–deferral–coverage trend, with no observed reduction in Harm. None of the nine corrected seed-by-metric paired tests is significant (Appendix K).

E FROZEN-OUTPUT QUALIFICATION REPLAY

Let z_j be the raw output string produced by the frozen RGA model for instance j. The replay parses this same string in both conditions to obtain the selected evidence I_{ji} and raw semantic verdict $\tilde{y}_{ji}$ for each rule. The two effective verdicts are

$$\bar{y}_{ji}^{\text{Full}} = \begin{cases} \tilde{y}_{ji}, & q_{ji} = \texttt{verified}, \\ \textsc{Unknown}, & q_{ji} \in \{\texttt{invalid}, \texttt{missing}\}, \end{cases} \qquad \bar{y}_{ji}^{\text{NoQ}} = \tilde{y}_{ji}. \tag{12}$$

Both are passed to the same ledger expressions and the same controller $g(T, D, R)$. Protocol-invalid strings receive the identical DEFER fallback in both conditions. No model inference, evidence selection, or decoding is repeated between the conditions.

Intervention invariants. The checkpoint, input records, evaluation labels, evidence selection, raw output strings, semantic verdicts, strict parser, four-action controller, and format-error fallback are identical. The sole intervention is whether qualification overrides raw semantic verdicts before aggregation. Because the aggregation is hierarchical, neither the number of overridden rule verdicts nor the number of changed actions is a count of newly deferred examples.

Track-specific execution paths. Renderer exercises the gate and has 141 non-UNKNOWN semantic verdicts changed to UNKNOWN under Full-Q, leading to 111 changed actions among 768 examples. Authority exercises the qualification interface but has no changed final actions in this replay. State retains its historical compatibility path; Retail's frozen protocol has no generic Q gate. The latter two tracks are explicit no-op strata. Their identical results therefore describe the replay implementation rather than an estimate of a qualification effect.

Table 4 gives all three metrics for every track. On Renderer, the Action gain and Harm reduction are each 14.45 percentage points, while FD is unchanged at 3.65%. This directly measures the gate's incremental value for the frozen RGA outputs and observed evidence.

F VERDICT-SPACE ABLATIONS

F.1 FROZEN-OUTPUT VERDICT ABLATION

This analysis uses the RGA evaluation from Table 2.

Table 11 isolates the role of unresolved verdicts while holding the final RGA predictions, evidence policy, qualification results, and model parameters fixed. Each policy-gated UNKNOWN verdict is replaced by either FALSE or TRUE before ledger aggregation. The four-action controller, expected labels, and Harm definition are unchanged.

Table 11: Frozen-output verdict ablation. Entries are Action/Harm (%). Predictions and evidence qualification are fixed; only effective UNKNOWN verdicts are replaced before aggregation.

Variant	State	Retail	Authority	Renderer
Full RGA	**53.54/18.33**	**92.19/0.78**	**97.66/1.56**	**96.22/0.00**
UNKNOWN → FALSE	46.98/18.44	67.19/0.78	97.66/1.56	74.87/5.99
UNKNOWN → TRUE	44.69/29.69	67.19/25.78	97.66/1.56	74.35/22.40

The two replacements expose different failure modes. Mapping UNKNOWN to FALSE turns unresolved evidence into an established blocker and primarily sacrifices applicability utility. Mapping UNKNOWN to TRUE instead treats unresolved conditions as established support and substantially increases the Harm proxy on several tracks. The result therefore shows that the unresolved state carries decision-relevant information under the current applicability contract.

Authority contains only REUSE and ADAPT labels and therefore probes source qualification separately from the full four-action boundary. The frozen-output analysis measures information lost from the specified verdict space; the following experiment additionally tests native binary training.

Table 12: Compute-matched comparison of native verdict spaces. Entries are Action/Harm (%). All three variants share the backbone, training data, and update budget. Full-3V is the matched reference for this comparison; the final RGA in Table 2 includes additional tuning.

Verdict space	State	Retail	Authority	Renderer
Full-3V	**56.04/24.79**	**100.00/0.00**	88.02/11.98	**94.01/2.47**
Binary-UF	46.77/25.42	75.00/0.00	75.00/25.00	64.32/15.76
Binary-UT	40.21/36.98	75.00/25.00	**94.79/5.21**	74.87/25.13

F.2 Training in a Native Binary Verdict Space

To isolate verdict representation during learning, we train three variants from the same Qwen3-8B base under matched compute and data conditions. The results appear in Table 12. *Full-3V* retains the native TRUE/FALSE/UNKNOWN semantic verdict space, *Binary-UF* replaces training-time UNKNOWN targets with FALSE, and *Binary-UT* replaces them with TRUE.

All three variants use the same training examples, example order, update budget, batch size, scheduler, and random seed. Binary-UF and Binary-UT use their corresponding binary prompts and targets from the beginning of training, so the comparison is not a post-hoc transformation of a three-valued model.

Full-3V is the matched reference for this experiment and shares the from-base structured training pipeline with Binary-UF and Binary-UT. The final RGA additionally receives boundary, source-qualification, and representation tuning. Its difference from Full-3V reflects those later stages jointly.

Training-time compensation does not recover the information removed by the binary target spaces on the three tracks that exercise all four actions. On Retail, both binary variants regress relative to Full-3V on all 32 independent source tasks ($p = 4.66 \times 10^{-10}$). On Renderer, Full-3V outperforms Binary-UF in 183 groups versus seven regressions and Binary-UT in 164 groups versus nine regressions ($p \leq 5.47 \times 10^{-38}$). Both binary variants also incur substantially higher Harm on Renderer, while Binary-UT increases Harm on State and Retail.

These results complement the qualification replay: preserving an explicit unresolved state also matters when the verdict space is chosen during training. The comparison isolates the specified verdict spaces; alternative ways of encoding evidence insufficiency remain compatible with this finding.

Authority has 192 REUSE and 192 ADAPT examples, so its label distribution tests source qualification rather than all four action boundaries. Binary-UT performs best on this track, consistent with the distinct role of affirmative verdicts in this two-action setting.

G Counterfactual Boundary Tuning and Retention

G.1 Matched Counterfactual Supervision

The matched reference and counterfactually tuned variant start from the same model and differ only in the targeted supervision studied here. Both are evaluated on diagnostic and transfer examples excluded from their tuning pairs. This design isolates the counterfactual intervention within the broader training pipeline.

Table 13: Matched analysis of counterfactual tuning and retention. Entries are Action/Harm (%).

Split	Matched reference	Counterfactual tuning
Renderer diagnostic	85.00/12.19	**93.75/1.88**
Renderer OOD	98.13/1.88	**100.00/0.00**
Anti-over-defer	100.00/0.00	**100.00/0.00**
Retail retention	97.50/0.00	**98.13/0.00**

Counterfactual tuning improves the multi-renderer diagnostic while maintaining performance on the transfer and retention evaluations. On the anti-over-defer split, both variants retain 100% Action

Table 14: Intermediate-model applicability diagnosis on the 96-example RGABENCH-STATE diagnostic split. FD denotes false deferral and Evidence denotes exact evidence selection.

System	Action	Harm	FD	Evidence
Prompt-only	25.00	0.00	73.96	0.00
Boolean-SFT	30.21	0.00	65.62	–
Grounded-SFT	21.88	11.46	21.88	0.00
Factorized RGA	**61.46**	28.13	**1.04**	**97.92**

with zero Harm and zero false deferral, distinguishing the improvement from a general shift toward deferral. Table 20 provides the paired statistical comparisons.

G.2 FACTORIZATION IDENTIFIES THE BOTTLENECK

Factorized RGA reaches 97.92% evidence exactness and 61.46% Action, with 28.13% Harm. The gap between routing and decision quality localizes the remaining errors to verification, particularly FALSE →TRUE and UNKNOWN →TRUE confusions. This diagnostic motivates the targeted semantic and boundary supervision used in the subsequent training stages.

H SOURCE QUALIFICATION AND REPRESENTATION ANALYSES

H.1 SOURCE-QUALIFICATION TRANSFER

We evaluate whether targeted source-qualification tuning transfers beyond the evidence layouts emphasized during training.

Table 15: Source-qualification tuning on held-out evaluations. Entries are Action/Harm (%). Harm is the execution-risk proxy defined in Appendix N.

Evaluation	Matched reference	RGA	ΔAction
RGABench-Authority	87.50/11.98	**97.66/1.56**	+10.16
Authority transfer	95.44/2.60	**97.53/0.39**	+2.09
RGABench-Renderer	93.36/2.99	**96.22/0.00**	+2.86

The gains on Authority and Renderer extend across independent counterfactual groups (Table 20). The additional Authority transfer evaluation tests held-out evidence layouts, with its scale and composition reported in Table 17. These results support learning the relationship between observations and the external evidence contract, while semantic verification interprets the qualified content.

H.2 AUDIT QUALITY ON RENDERER

The final RGA reaches 96.22% Action with zero Harm on Renderer; all 29 remaining Action errors are false deferrals. Its structured-audit protocol validity is 93.62%. Action accuracy is measured after deterministic protocol handling: an invalid audit falls back to DEFER and is Action-correct when DEFER is expected, while the audit itself remains invalid (Appendix N).

I FINAL-STAGE TUNING SENSITIVITY

We repeat the final-stage tuning with three data-order seeds from the same pre-tuning model. Each run uses the identical training multiset, optimizer, batch composition, and number of updates; only the ordering of complete global batches changes. No seed-specific checkpoint selection is performed.

The results are stable across the three tested data orderings. These standard deviations measure final-stage batch-order sensitivity conditional on the shared pre-tuning model.

Table 16: Three-seed final-stage tuning order sensitivity. Entries are mean ± sample standard deviation in percentage points.

Track	Action	Harm
State	53.51 ± 0.06	18.30 ± 0.06
Retail	91.93 ± 0.45	1.04 ± 0.45
Authority	97.92 ± 0.26	1.22 ± 0.30
Renderer	96.05 ± 0.20	0.00 ± 0.00

J RGABENCH SCALE AND STATISTICAL UNITS

Several examples in RGABENCH are counterfactual variants of the same underlying case and are therefore statistically correlated. We report model calls and independent statistical units separately and perform the primary paired tests at the family, source-task, or counterfactual-group level rather than treating individual model calls as independent samples.

State uses procedure families as independent units, Retail uses source tasks, and Authority and Renderer use counterfactual groups. The offline AppWorld study uses one task from each of 40 distinct native task-generator families. State, Retail, and Renderer balance the four expected actions by construction to expose applicability boundaries. Authority intentionally isolates the REUSE/ADAPT boundary. These distributions are controlled stress tests and should not be interpreted as estimates of action prevalence in natural agent traffic.

Table 17: RGABENCH composition and supporting diagnostic splits. Examples denote model calls; groups denote the independent units used in paired statistical analyses.

Evaluation	Role	Examples	Groups	Source tasks/ scenarios	Domains/ families	Renderers	Actions
State diagnostic	diagnostic	96	6	6	3 domains	5	4
RGABench-State	primary held-out	960	24	24	4 domains	5	4
Retail diagnostic	selection/ retention	80	20	20	1 domain	1	4
RGABench-Retail	primary held-out	128	32	32	1 domain	1	4
Renderer diagnostic	diagnostic	640	160	–	8 families	3	4
RGABench-Authority	primary held-out	384	96	–	6 families	2	2
Authority transfer	transfer	768	192	–	8 families	8	4
RGABench-Renderer	primary held-out	768	192	–	12 families	8	4
AppWorld transfer	external-source	40	40	40	40 families	–	4

Some matched component analyses use subsets of the complete diagnostic pools. In particular, the paired counterfactual-tuning comparison in Table 20 is computed over the 80 counterfactual groups shared by both matched variants, whereas the complete Renderer diagnostic pool contains 160 groups. The reported denominator for each paired test therefore refers to the common matched units available to that comparison rather than necessarily to the full diagnostic pool.

K PAIRED STATISTICAL TESTS

K.1 PRIMARY EXTERNAL-CONTROL COMPARISON

Table 18 reports group-level paired tests against the mechanism-level control with the highest Action accuracy on each track under the common applicability interface.

Table 18: Group-level paired tests for the primary RGABENCH comparison. Entries report improved/regressed/tied independent units for RGA relative to the named mechanism-level control. Harm improvement denotes a reduction in the Harm proxy within the corresponding unit.

Track / reference	Action $+/-/=$ (exact p)	Harm $+/-/=$ (exact p)	Groups
State / Skill-Pro-style	21/3/0 (2.77×10^{-4})	22/2/0 (3.59×10^{-5})	24
Retail / Skill-Pro-style	22/0/10 (4.77×10^{-7})	0/1/31 (1.00)	32
Authority / Skill-Pro-style	94/0/2 (1.01×10^{-28})	94/0/2 (1.01×10^{-28})	96
Renderer / AWM/ExpeL-style	188/0/4 (5.10×10^{-57})	192/0/0 (3.19×10^{-58})	192

All four Action comparisons favor RGA at the independent-group level (two-sided exact sign test, $p \leq 2.77 \times 10^{-4}$). Harm is significantly reduced on State, Authority, and Renderer. Retail has one incorrect executable decision for RGA and none for the binary control ($p = 1.00$).

K.2 SUPPLEMENTARY DIRECT-4A VERSUS AUDIT COMPARISONS

Each direct-baseline seed is paired with the same frozen RGA outputs on complete test units: 24 State families, 32 Retail source tasks, 96 Authority groups, and 192 Renderer groups. A whole-unit bootstrap samples these units with replacement, retaining every example within a sampled unit; 10,000 resamples produce the 95% confidence intervals. The exact two-sided sign test uses the signs of within-unit metric differences and excludes ties. Holm's procedure controls the familywise error rate across all $3 \times 4 \times 2 = 24$ Action/Harm comparisons. Seeds are analyzed separately, not pooled as additional independent test cases.

Table 19 uses $\Delta = \text{Direct-4A} - \text{RGA}$: positive Action differences favor the direct baseline, whereas positive Harm differences favor RGA.

Table 19: Seed-specific paired comparison with RGA. Deltas are percentage points. Significance refers to the 24-test Holm correction. CI denotes a 95% whole-unit bootstrap confidence interval.

Track	Metric	Seed 11	Seed 23	Seed 37
State	Action / Harm	not significant	not significant	not significant
Retail	ΔAction	$+5.47$	0.00	$+7.03$
	Holm p	0.654	1.000	0.0469
	Action CI	–	–	$[3.13, 10.94]$
Authority	ΔAction	-49.74	-50.26	-51.56
	Action / Harm	significant	significant	significant
Renderer	ΔAction	-62.63	-62.24	-58.46
	ΔHarm	$+40.36$	$+35.29$	$+29.82$
	Action / Harm	significant	significant	significant

State's Action and Harm comparisons are not significant for any seed after correction. Retail's seed-37 Action comparison favors Direct-4A, whereas seeds 11 and 23 do not show significant Action differences. Authority and Renderer favor RGA on both Action and Harm for every seed after correction.

K.3 OFFLINE APPWORLD PAIRED COMPARISONS

The statistical unit is a native AppWorld task-generator family, with exactly one evaluation instance per family. For each of the three Direct-4A seeds, we pair Action, Harm, and FD with the fixed RGA

predictions on the same 40 tasks. Holm correction is applied over these nine comparisons. None is significant after correction. Consequently, the differences in Table 5 are reported as observed transfer trends. The three-seed baseline mean summarizes repeated training runs over the same 40 tasks and does not increase the number of independent evaluation units.

K.4 Matched Component Comparisons

The following tests compare the matched reference models and tuning variants used in the component analyses.

Table 20: Group-level paired tests for supporting component analyses.

Comparison	Action $+/-/=$ (exact p)	Harm $+/-/=$ (exact p)	Groups
Counterfactual tuning / matched reference	29/3/48 (2.56×10^{-6})	31/0/49 (9.31×10^{-10})	80
Source-qualification tuning / matched audit	40/1/55 (3.82×10^{-11})	40/0/56 (1.82×10^{-12})	96
Representation tuning / semantic audit	27/4/161 (3.40×10^{-5})	23/0/169 (2.38×10^{-7})	192

The 80-group counterfactual comparison uses the common subset for which both matched variants have paired predictions; it is therefore smaller than the 160-group Renderer diagnostic pool summarized in Table 17.

L Data Provenance and Leakage Control

All primary RGABench evaluations are isolated from training and model selection at the corresponding task, family, entity, evidence-ID, source-task, or counterfactual-group level.

Development and diagnostic splits are separated from the primary held-out evaluations. Diagnostic data may be used for aggregate failure analysis and, where applicable, to identify abstract error categories such as unauthorized sources, incomplete observations, entity-binding conflicts, or field-identity shortcuts. The primary held-out evaluations are frozen before final model selection and are not used to define subsequent repair categories, construct training examples, select tuning interventions, or update model parameters.

For State, the diagnostic split is used for aggregate diagnosis and error categorization, while the primary evaluation is frozen before model inference and undergoes automated consistency checks together with a stratified manual semantic review of 192 examples, all of which are accepted. Retail diagnostic and primary evaluations are separated by source-task identity. Controlled Authority and Renderer evaluations use disjoint counterfactual groups and evidence identities.

Rule ledgers do not encode instance-specific solutions. They are instantiated from reusable applicability templates shared across examples rather than manually tailored to individual benchmark instances or expected outcomes. A ledger specifies the conditions to be audited and their target, direct, and recovery aggregation structure, but contains neither runtime TRUE/FALSE/UNKNOWN verdicts nor the expected REUSE, ADAPT, REJECT, or DEFER action. The evidence policy P similarly specifies admissibility requirements, including source, entity or selector binding, temporal validity, and completeness, without revealing whether the observations in a particular instance satisfy those requirements. Applicability must therefore still be established from runtime evidence.

No primary evaluation row, entity, identifier, evidence instance, or surface form is copied, paraphrased, or transformed into training supervision. Expected actions, gold rule verdicts, and gold audits are never included in model inputs.

Generated benchmark and training artifacts are checked for aggregation consistency, duplicate prompts, latent-label exposure, source-task overlap, evidence-identity overlap, and token truncation. Primary evaluation examples and their model outputs are excluded from subsequent training and model selection.

Table 21: Representative procedural-memory systems from the perspective of runtime applicability. "Implicit" denotes applicability handled primarily through retrieval or subsequent model reasoning.

Method	Memory form	Applicability interface	Evidence handling	Explicit UNKNOWN
Reflexion / ExpeL	textual guidance	implicit	task context	no
Voyager / AWM	skills/workflows	implicit	task/environment context	no
Skill-Pro / AFTER	procedures/skills	learned conditions/control	runtime context	no
SkillWrapper	predicates/operators	symbolic conditions	state predicates	no
RGA	procedure + rule ledger	**audited**	**policy-qualified evidence**	**yes**

M PROCEDURAL-MEMORY INTERFACE COMPARISON

Table 21 summarizes representative procedural-memory systems from the perspective of runtime applicability. The comparison focuses on how each system represents reusable experience, whether applicability is made explicit, and how runtime evidence is incorporated before reuse. This view is intentionally narrower than a full comparison of the original systems: it highlights the interface relevant to our setting, where a retrieved procedure must be checked against current evidence before execution.

N METRIC DEFINITIONS

For example j, let $a_j^\star$ denote the expected applicability action and let $\hat{a}_j$ denote the final controller output after deterministic protocol handling. Action accuracy is

$$\mathrm{Action}_j = \mathbb{1}[\hat{a}_j = a_j^\star]. \tag{13}$$

Action therefore measures controller-level decision correctness rather than strict validity of the model-produced structured audit. In particular, a protocol-invalid audit falls back to DEFER before Action is computed. If $a_j^\star = \textsc{Defer}$, such an invalid audit may still receive a correct Action label despite failing the structured protocol.

We define Harm as an execution-risk proxy:

$$\mathrm{Harm}_j = \mathbb{1}\left[\hat{a}_j \in \{\textsc{Reuse}, \textsc{Adapt}\} \wedge \hat{a}_j \neq a_j^\star\right]. \tag{14}$$

The track-level Harm rate averages this indicator over all examples. It measures incorrect executable applicability decisions, rather than realized downstream damage or the success of a subsequent adaptation.

False reject is

$$\mathrm{FR}_j = \mathbb{1}\left[\hat{a}_j = \textsc{Reject} \wedge a_j^\star \in \{\textsc{Reuse}, \textsc{Adapt}\}\right], \tag{15}$$

and false deferral is

$$\mathrm{FD}_j = \mathbb{1}\left[\hat{a}_j = \textsc{Defer} \wedge a_j^\star \neq \textsc{Defer}\right]. \tag{16}$$

FD, FR, and Harm are averaged over all evaluation examples, rather than conditioned on the number of deferred, rejected, or executable predictions. Execution coverage is

$$\mathrm{Coverage} = \frac{1}{N}\sum_{j=1}^{N} \mathbb{1}[\hat{a}_j \in \{\textsc{Reuse}, \textsc{Adapt}\}]. \tag{17}$$

Coverage counts both correct and incorrect executable decisions. We report output length as the mean number of generated tokens per example.

For rule-level diagnostics, *evidence exactness* requires the complete selected evidence-ID set for a rule to match its expected evidence set. *Verdict exactness* requires the semantic rule verdicts to

Table 22: Controlled memory conditions in the AppWorld end-to-end evaluation. Each of the 49 target families contributes one episode to each condition.

Condition	Candidate procedure	Intended boundary	Count
Same instance	target instance	REUSE	49
Sibling instance	same-family instance	ADAPT	49
Different family	different-family instance	REJECT	49
Initially unresolved	hidden initially	DEFER $\rightarrow$ resolve	49

match their expected values. *Strict audit* requires both evidence selection and semantic verdicts to be correct under a protocol-valid structured output.

Protocol validity indicates whether the raw model output satisfies its required response schema before fallback: a complete structured audit for RGA or the strict four-action JSON schema for Direct-4A. Protocol validity and Action accuracy are separate metrics. An invalid response can receive a correct action only through the deterministic DEFER fallback when DEFER is expected.

For source-qualification diagnostics, *authority exactness* requires all rule-level qualification statuses to match their expected values; authority item accuracy is micro-averaged over individual rule items. Finally, *invalid/missing-to-*UNKNOWN *recall* measures whether observations rejected by deterministic qualification as invalid or missing are propagated to an effective UNKNOWN verdict through the policy gate, rather than merely testing whether an auxiliary qualification-status token is predicted correctly.

O END-TO-END APPWORLD EXECUTION PROTOCOL

This appendix describes the construction and evaluation protocol for the end-to-end AppWorld experiment in Section 5.7. The experiment is designed as a controlled procedural-memory reuse evaluation: for each target task, the task instruction and initial environment are held fixed while the relation between the target and the supplied historical procedure is varied. This produces matched cases for direct reuse, adaptation, rejection, and initially unresolved applicability.

O.1 TASK-FAMILY SELECTION

We construct the evaluation from the public `train` and `dev` splits of AppWorld 0.2.0. AppWorld task identifiers contain a task-generator-family identifier followed by an instance suffix; for example, `07b42fd_1`, `07b42fd_2`, and `07b42fd_3` belong to the same generator family `07b42fd`. We group all public `train` and `dev` tasks after removing this final instance suffix.

This procedure yields 49 distinct task-generator families, each containing at least three public task instances. Within each family, task identifiers are sorted deterministically. The first instance is selected as the target task, and the second instance is reserved as the same-family sibling candidate. Target selection is therefore fixed before model execution and does not depend on model predictions or execution outcomes.

The 49 target families comprise 30 families from the public training split and 19 from the public development split. Each target is evaluated under four memory conditions, yielding

$$49 \times 4 = 196$$

execution episodes, with 120 episodes originating from training-split families and 76 from development-split families. Although 196 episodes are executed, the 49 task-generator families are the independent statistical units.

O.2 CONTROLLED MEMORY CONDITIONS

For each target task, we construct four episodes that share the same target instruction and initial environment but differ in the candidate procedural memory supplied to the agent. Table 22 summarizes the construction.

Same instance. The candidate memory is the successful procedure associated with the target instance itself. This condition serves as a positive-control reuse case: the retrieved procedure is already instantiated for the current task and is intended for direct execution.

Sibling instance. The candidate memory is taken from the second sorted instance of the same task-generator family. The task template and high-level procedure are shared, but instance-specific entities, arguments, or environment-dependent values differ. These cases therefore test whether the agent can preserve the useful procedure structure while adapting it to the current target.

Different family. The candidate memory is drawn from a different task-generator family and is inapplicable to the target task. These cases test whether the agent avoids executing a procedurally valid but task-incompatible historical trajectory.

Initially unresolved. No candidate procedure or source record is visible at the initial audit. The agent is allowed one source-resolution step, after which the resolver reveals either a same-family sibling procedure or a different-family procedure. The initial state therefore lacks sufficient evidence for an applicability decision and is intended to exercise DEFER followed by re-auditing once the missing provenance and procedure become available.

All constructed cases were reviewed before evaluation to verify that their intended applicability relation is consistent with the corresponding REUSE, ADAPT, REJECT, or initially unresolved boundary. The review is based on the task–procedure relation and is completed independently of model outputs.

O.3 Initially unresolved cases

The 49 initially unresolved cases are assigned deterministically using random seed `20260918`. For 25 target families, the resolver returns a same-family sibling procedure; after resolution, these cases are expected to support adaptation. For the remaining 24 families, the resolver returns a different-family procedure and the resolved applicability is rejection.

Thus, the two possible post-resolution paths are

$$\text{DEFER} \rightarrow \text{source resolution} \rightarrow \text{ADAPT}$$

and

$$\text{DEFER} \rightarrow \text{source resolution} \rightarrow \text{REJECT}.$$

When adaptation is selected, the repaired procedure is audited again before execution. The 25/24 assignment is used only for dataset construction and post-hoc stratified analysis; it is never exposed to the model at runtime.

O.4 Procedural-memory extraction and model-visible information

Candidate memories are derived from the official AppWorld task artifacts. For a source task, we extract the body of `solution()` from `ground_truth/compiled_solution.py` and use it as the historical procedure. We additionally read `ground_truth/required_apis.json` for the *source* procedure and provide the corresponding public API documentation. These artifacts describe how the historical task was successfully completed and which public interfaces that procedure used.

The agent is not given the target task's evaluator, private task data, official success signal, or target-side gold solution, except that the same-instance condition intentionally uses the target's successful historical procedure as its candidate memory. Target-side required APIs are likewise not supplied as separate supervision; API information is attached only to the candidate memory. Consequently, in the same-instance positive-control condition, the candidate-side API metadata naturally coincides with that of the target task.

The evaluation therefore asks whether a supplied historical procedure should be reused, adapted, rejected, or deferred under the current task and environment, rather than asking the model to reconstruct AppWorld's hidden evaluator.

O.5 MATCHED CONSTRUCTION WITHIN A TARGET FAMILY

The four episodes associated with a target family form a matched counterfactual group. For example, for target task `07b42fd_1`, the four conditions may be instantiated as

Same:	`07b42fd_1` → `07b42fd_1`,	REUSE,
Sibling:	`07b42fd_1` → `07b42fd_2`,	ADAPT,
Different:	`07b42fd_1` → `4fab96f_1`,	REJECT,
Unresolved:	`07b42fd_1` → hidden source,	DEFER first.

For the unresolved condition, the hidden source is revealed only after the resolution step and may correspond either to a same-family sibling or to a different-family candidate according to the fixed assignment above.

This construction holds the target task and environment fixed while changing only the relationship between the target and available procedural memory. It therefore separates memory-applicability behavior from variation in target task difficulty.

O.6 EXECUTION COMPARISON

Section 5.7 compares an unaudited agent pipeline with the RGA-guided execution loop. Both use Qwen3-8B and are evaluated on the same 196 matched episodes. In the RGA-guided pipeline, the four decisions have operational semantics: REUSE executes the candidate procedure directly; ADAPT repairs the candidate before execution and subsequently re-audits it; REJECT does not execute the candidate and instead falls back to the ordinary planning path; and DEFER acquires the permitted missing source evidence before repeating the audit.

The unaudited pipeline receives the corresponding historical information but does not use the explicit RGA applicability controller. It may either use the candidate history or generate a task-specific execution trajectory. We report this experiment as an *integrated execution comparison*: its purpose is to test whether incorporating the complete RGA-guided audit-and-control loop improves final agent task completion. The component-level contributions of structured auditing, deterministic qualification, and the three-valued verdict space are isolated separately by the controlled experiments in Sections 5.3–5.5 and Appendices E–H.

O.7 STATISTICAL UNITS AND INFERENCE

The 196 execution episodes are not treated as statistically independent, because each target task appears under four controlled memory conditions. The independent statistical unit is the task-generator family, giving 49 independent units.

For an overall metric M, let $M^s_{f,c}$ denote the outcome for system s, family f, and memory condition c. We first aggregate the four conditions within each family,

$$M^s_f = \frac{1}{4}\sum_{c=1}^{4} M^s_{f,c},$$

and compare systems using these family-level values.

Confidence intervals for system differences are computed with a family-cluster bootstrap: each bootstrap replicate samples the 49 families with replacement and retains all four episodes belonging to every sampled family. This preserves the dependence induced by evaluating the same target under multiple memory conditions.

For paired significance testing, we additionally compare the two systems at the family level. A family is counted as an improvement, regression, or tie according to the sign of its aggregate paired difference, and a two-sided exact sign test is applied after excluding ties. Condition-specific paired analyses may additionally be reported over the 49 independent families in each individual condition. Episode-level analyses over all 196 rows are treated as descriptive or auxiliary and are not used as the primary independence assumption.

The main end-to-end metric is final AppWorld task success. Branch-conditional success rates, such as successful direct reuse or successful memory repair, are reported as diagnostics conditional on

the execution branch selected by the RGA pipeline and are not interpreted as independent task distributions.