GRAPHSKILLAA: ATTRIBUTION-GUIDED SKILL-GRAPH UPDATING WITH TARGETED VALIDATION AND ROLLBACK

Anonymous authors
Paper under double-blind review

ABSTRACT

External skills provide domain knowledge and procedures without updating model parameters, but flat collections obscure skill applicability, dependencies, and composition. Graphs organize skills into addressable nodes and explicit relations, supporting selection and composition. Yet existing skill-graph methods use this structure mainly for retrieval, rather than to localize updates, scope retesting, or precisely roll back rejected changes. We introduce GRAPHSKILLAA (*Graph-Skill Abductive Attribution*), which uses one addressable graph for skill selection, execution, failure attribution, targeted updating, validation, and rollback. Nodes separate applicability, execution, and exclusion conditions; typed edges encode prerequisite and enhancement relations. The frozen student records used nodes and edges, while the teacher contrasts related successes and failures to route each supported repair to the smallest relevant field or relation; execution lapses or insufficient evidence leave the graph unchanged. A Local Gate retests affected examples, while a Big Gate evaluates the merged graph on the complete update pool; rejected changes are rolled back. With GPT-5.6-sol, GRAPHSKILLAA reaches 81.5%, 66.7%, and 91.2% on SearchQA, LiveMath, and DocVQA, respectively, and attains the highest observed mean in every main setting. These results show that object-level attribution and graph-scoped validation make a skill graph a locally optimizable, testable, and reversible external state.

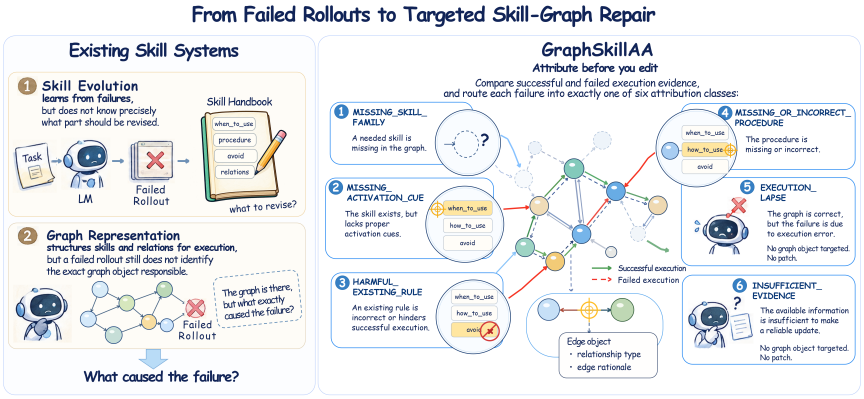

Figure 1: GRAPHSKILLAA: first localize the cause of failure, then update the skill locally. *Left:* Existing methods can update skills or organize a skill graph, but have difficulty determining exactly what should change after a failure. *Right:* GRAPHSKILLAA contrasts successful and failed execution traces, decides whether an update is warranted, localizes a specific skill field or relation, and makes only an evidence-supported local edit. If the failure is an execution lapse or the evidence is insufficient, the skill graph remains unchanged.

1 INTRODUCTION

Frozen language models increasingly rely on external tools and APIs to extend their capabilities without changing model parameters (Mialon et al., 2023; Schick et al., 2023; Patil et al., 2024;

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.

Qin et al., 2024). External skills are readable, portable, and inexpensive to update, but optimizing them is harder than simply adding more instructions. A failed rollout reveals that the final behavior is wrong, yet it does not determine whether the reusable cause is missing knowledge, a missed activation cue, an incorrect procedure or relation, or a one-off execution lapse. Editing a skill directly from such coarse feedback can overfit the source failure and regress cases that were previously correct. Reliable skill optimization therefore requires both diagnostic localization—what reusable component should change—and controlled validation—whether the proposed change helps without damaging established behavior.

A graph representation provides a natural interface for this problem. Node fields can separate when a skill applies, how it executes, and when it should be avoided; stable IDs make these fields addressable; typed edges expose prerequisite and enhancement relations; and dependency paths describe how several skills compose. Recent systems use these properties to retrieve relevant skills and construct executable bundles (Zheng et al., 2026; Liu et al., 2026; Bai et al., 2026; Xu et al., 2026; Li et al., 2026c;b). However, existing methods typically use the graph only before inference to retrieve and compose skills. Once the model answers incorrectly, the update stage often treats the skill library as an undifferentiated text artifact: it cannot tell whether the problem lies in a skill's applicability conditions, execution steps, or relations to other skills, and it does not use graph dependencies to decide which previously correct cases should be retested. The graph therefore helps the model use skills, but does not yet clearly determine what to edit or how to check that an edit introduces no new errors.

To close this gap, we introduce GRAPHSKILLAA (*Graph-Skill Abductive Attribution*). It first represents the skill library and its composition process as an addressable procedural graph: nodes separate when a skill applies, how it executes, and when it should be avoided, while typed edges describe how skills compose. When an execution fails, the teacher no longer rewrites the entire skill from that rollout. Instead, it contrasts the failure with related successes, reattributes the error to a specific node field or relation, and edits only the smallest graph object supported by the evidence. If the comparison does not show that a skill or relation should change, the graph is left untouched (`NO_PATCH`). Before commitment, each edit undergoes affected-case retesting and whole-graph validation, with failed edits rolled back. GRAPHSKILLAA thus replaces coarse "summarize-and-rewrite" updates with a localized attribute–edit–validate process.

Across SearchQA, LiveMath, and DocVQA, GRAPHSKILLAA attains the highest observed mean in every main setting; with GPT-5.6-sol, it reaches 81.5%, 66.7%, and 91.2%, respectively. Progressive comparisons and ablations support the roles of structured node semantics, typed relations, object-level attribution, and two-level validation. These results show that a skill graph is most useful not merely as a retrieval structure, but as a shared, auditable state for localized optimization.

Our contributions are:

1. **A skill-graph representation for localized updating:** skill rules and compositional relations are converted into optimization objects that can be precisely localized and edited.
2. **Object-level abductive attribution:** successful and failed evidence is contrasted to identify the smallest repair object, while execution lapses or insufficient evidence produce no graph update.
3. **Dependency-aware two-level validation and rollback:** related modifications are first retested locally and then validated jointly; only verified updates with positive net gain are committed atomically.
4. **Evaluation across models and task types:** experiments and ablations on open-domain QA, mathematical reasoning, and document understanding establish the end-to-end gains and clarify the contribution and boundary of each mechanism.

2 RELATED WORK

Graph-structured skills. Graphs organize document evidence, memory, and tool relations: ControlLLM models tool dependencies; CRAFT builds and retrieves specialized toolsets; ToolRerank uses hierarchical relations for reranking; and GraphRAG and HippoRAG connect distributed knowledge through relational graphs (Liu et al., 2024; Yuan et al., 2024; Zheng et al., 2024; Edge et al.,

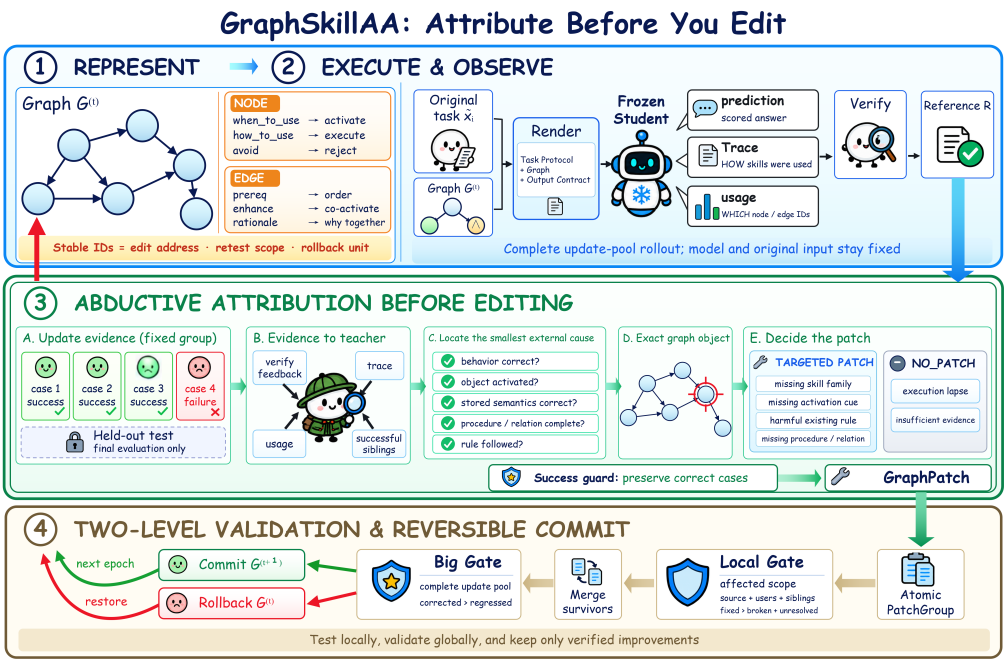

Figure 2: The GRAPHSKILLAA pipeline: localize, edit, and update only after validation. ① Organize skills: nodes encode applicability, execution, and exclusions; edges encode inter-skill relations. ② Execute and record: the frozen student solves tasks and records its trace and graph usage. ③ Localize and repair: success–failure contrasts identify a node field or relation for an evidence-supported edit; execution lapses or insufficient evidence yield no update. ④ Validate and update: affected cases are retested, edits are merged, and the candidate is evaluated on the complete update pool. Only positive-gain changes are retained; otherwise they are rolled back. The held-out test set is used only for final evaluation.

2024; Gutiérrez et al., 2024). Skill libraries extend this idea: SkillRouter retrieves from skill-body signals; Graph-of-Skills returns dependency-aware bundles; SkillGraph and SkillDAG evolve typed relations; and HyperSkill and CaSKG model higher-order or directed composition (Zheng et al., 2026; Liu et al., 2026; Li et al., 2026b; Bai et al., 2026; Xu et al., 2026; Li et al., 2026c). These methods show that relations aid compositional execution, but focus on graph construction and retrieval rather than using graph objects for attribution, targeted updating, validation, and rollback.

Skill evolution and controlled optimization. Language-artifact optimization has progressed from prompt search to trajectory-based revision. ProTeGi, OPRO, TextGrad, and GEPA optimize prompts or agent configurations from critiques, textual feedback, or trajectories (Pryzant et al., 2023; Yang et al., 2024; Yuksekgonul et al., 2025; Agrawal et al., 2026). Skill systems similarly move from experience accumulation (Voyager and ExpeL) to persistent workflows, skills, or memories (Agent Workflow Memory, SkillWeaver, ReasoningBank, and Trace2Skill) (Wang et al., 2023; Zhao et al., 2024; Wang et al., 2025; Zheng et al., 2025; Ouyang et al., 2026b; Ni et al., 2026). SkillsBench and related surveys define skills as reusable procedures with applicability conditions (Li et al., 2026a; Xu & Yan, 2026), while SkillOpt applies bounded edits and multilevel controls to a persistent skill document (Yang et al., 2026). Yet these approaches largely optimize flat text or whole candidates, limiting field-level routing and independent updates or rollback of skill relations.

3 METHOD

3.1 OVERVIEW: SKILL OPTIMIZATION AS GRAPH-STATE EDITING

GRAPHSKILLAA builds on the outer optimization framework of SKILLOPT (Yang et al., 2026), retaining its model access, data and rollout organization, and epoch loop while converting the flat skill

document into an addressable skill graph. This representation allows updates to target specific rules or relations instead of rewriting the entire document. Figure 2 shows the overall GRAPHSKILLAA workflow. The system takes an initial skill specification, an update pool, frozen student and teacher models, and an optimization budget E as input, and returns the final committed graph $G^{(E)}$. During optimization, only the skill graph changes; model parameters, original task inputs, and the output contract remain fixed, and $\mathcal{D}_{\text{test}}$ never enters updating. Appendix C.3 gives the complete algorithm.

Let the frozen student M_{student} execute tasks and the frozen teacher M_{teacher} attribute failures and propose edits. The update pool is

$$\mathcal{D}_{\text{update}} = \{(x_i, y_i)\}_{i=1}^{N},$$

where i indexes one of N examples and y_i is a gold answer or verifiable feedback. The final-evaluation set $\mathcal{D}_{\text{test}}$ is disjoint. The initial skill is deterministically converted into $G^{(0)}$; $G^{(t)}$ denotes the graph committed at epoch t. Inputs x_i, including images, are never rewritten.

A deterministic renderer combines the current graph with the fixed task protocol and output contract:

$$P^{(t)} = \text{Render}\Big(\text{TaskProtocol},\ G^{(t)},\ \text{OutputContract}\Big).$$

TaskProtocol fixes task requirements and OutputContract the answer format; only $G^{(t)}$ changes. RENDER follows fixed serialization rules and neither learns parameters nor edits or retrieves skills.

3.2 ADDRESSABLE SKILL GRAPH

GRAPHSKILLAA treats the graph as persistent state for selection, editing, validation, and restoration. The skill library is $G = (\text{Nodes}, \text{Edges})$. A node

$$v = \big(\text{id},\ \text{title},\ \texttt{when_to_use},\ \texttt{how_to_use},\ \texttt{avoid}\big)$$

denotes a conditional reusable procedure: `when_to_use` states applicability, `how_to_use` specifies execution, and `avoid` excludes confusable cases. Conceptually,

$$\text{Use}(v, x) \approx \text{Match}(x, \texttt{when_to_use}(v)) \wedge \neg\, \text{Conflict}(x, \texttt{avoid}(v)).$$

Here, $\approx$ denotes a conceptual judgment; MATCH and CONFLICT are in-context judgments by the frozen student, not a separate classifier or keyword test. Separating activation, execution, and exclusion makes these failure surfaces independently addressable.

An edge

$$e = \big(\text{source},\ \text{target},\ \text{type},\ \text{strength},\ \text{rationale}\big)$$

describes composition. Source and target specify direction; PREREQ is required, ENHANCE is beneficial, and CO-OCCUR records joint use without order or causality. Strength and rationale record priority and evidence.

At inference, GRAPHSKILLAA renders all active nodes and edges in stable order, without external embedding-based Top-K pruning (Appendix F.2). The student selects nodes from `when_to_use` and `avoid`, executes `how_to_use`, and follows edge semantics. Stable IDs make the same objects usable for selection, attribution-guided editing, impact analysis, and rollback.

3.3 EXECUTION EVIDENCE AND ABDUCTIVE ATTRIBUTION

A failed answer reveals that the current execution is inadequate, but not which persistent state, if any, should change. The cause may be a missing skill, missed activation, harmful rule, incomplete procedure or relation, or a one-off execution lapse; editing before diagnosis can therefore modify the wrong object. GRAPHSKILLAA instead contrasts the failure with related successes, seeks the smallest reusable graph-level explanation supported by the evidence, and routes the initial repair to its corresponding edit surface.

To support this diagnosis, attribution observes both outcome correctness and graph participation. On each original input, the student returns

$$(\text{prediction}_i^{(t)},\ \text{trace}_i^{(t)},\ \text{usage}_i^{(t)}) = M_{\text{student}}(x_i; P^{(t)}),$$
$$\text{correct}_i^{(t)} = \text{Verify}(\text{prediction}_i^{(t)}, y_i).$$

Table 1: Attribution-to-update routing. Each route constrains its initial graph edit.

Root-cause code	Failure condition	Authorized update
`MISSING_SKILL_FAMILY`	No node covers the required independent, reusable procedure.	Atomically add a specialist node and its connecting `enhance` edge.
`MISSING_ACTIVATION_CUE`	A correct, applicable skill exists but is absent from validated `usage`.	Extend only its `when_to_use`.
`HARMFUL_EXISTING_RULE`	A used node or edge contains harmful semantics.	Replace only the harmful field; atomically replace a faulty edge.
`MISSING_OR_INCORRECT_PROCEDURE`	The correct node is active but lacks a step, exclusion boundary, or relation.	Complete only `how_to_use` or `avoid`, or add the required `prereq`/`enhance` edge.
`EXECUTION_LAPSE`	The graph is correct and active, but execution diverges.	Return `NO_PATCH`; do not persist a one-off lapse.
`INSUFFICIENT_EVIDENCE`	Evidence cannot localize a field or edge.	Return `NO_PATCH`; preserve the graph.

`prediction` is the submitted answer. `trace` records the semantic path, for example, *Q3 determines relation direction, E2 invokes Q7, and Q7 normalizes the answer*; `usage` lists the same path by stable ID, for example, `used_nodes=[Q3,Q7]` and `used_edges=[E2]`. The former explains *how* skills were used for attribution; the latter identifies *which* objects may be affected. They are auditable external records, not hidden reasoning, and must agree (Appendix C.4).

The verifier compares only `prediction` with y_i, yielding $\text{correct}_i^{(t)} \in \{0, 1\}$. A full-pool rollout—predictions, traces, usage, and correctness—forms the *reference* for same-example candidate comparisons.

Before optimization, GRAPHSKILLAA partitions only the update pool into fixed, disjoint quadruples using task-type constraints and question-text similarity:

$$\text{Group}_m = \{\text{update}_{m,1}, \text{update}_{m,2}, \text{update}_{m,3}, \text{update}_{m,4}\}.$$

Each group contains four update examples. Successful members provide local contrast paths; members of relevant groups enter the Local Gate. The disjoint test set is excluded from grouping, attribution, patch synthesis, and both Gates.

For each failure, the teacher combines verifier feedback, `trace`, validated `usage`, and successful group members. It locates the earliest unrecoverable decision, then follows this diagnostic chain:

$$\text{behavior covered?} \rightarrow \text{relevant object activated?} \rightarrow \text{stored semantics correct?}$$
$$\rightarrow \text{steps or relations complete?} \rightarrow \text{rule followed?}$$

Table 1 maps the diagnosis to six routes. The first four constrain the initial object-level GRAPHPATCH; execution lapse and insufficient evidence return `NO_PATCH`. Here, *abductive* attribution selects the smallest graph-level cause whose repair can be tested by paired executions. Appendix C.6 gives the subsequent joint synthesis, validation constraints, and examples.

3.4 TARGETED UPDATES AND TWO-LEVEL ROLLBACK

Once attribution opens an initial edit surface, the teacher synthesizes an evidence-constrained change and jointly synchronizes connected proposals against successful-use guards. The resulting operations form an atomic PATCHGROUP J_k that records its evidence and is accepted or rolled back as a unit. Attribution constrains the initial target; joint synthesis forms the executable candidate; the Gates decide acceptance.

A targeted patch may still break other users of the same object, and individually useful patches may conflict after merging. The Local Gate therefore tests each atomic group on its affected scope; the Big Gate tests all survivors as one executable graph before epoch-level commitment. This ties validation cost and rollback granularity to graph structure.

Local Gate: graph-scoped retesting. For the k-th atomic group J_k, GRAPHSKILLAA reruns only update examples that may be affected:

$$\text{Retest}_k = \text{UpdateMembers}\big(\mathcal{Q}^k_{\text{src}} \cup \mathcal{Q}^k_{\text{use}} \cup \mathcal{Q}_{\text{unk}}\big)\,.$$

$\mathcal{Q}^k_{\text{src}}$ contains source groups; $\mathcal{Q}^k_{\text{use}}$ contains groups whose validated `usage` includes the edited node or, for an existing edge, the edge or either endpoint; and $\mathcal{Q}_{\text{unk}}$ contains groups with invalid or missing `usage`. UpdateMembers expands these groups to examples. A new node has no prior users, so its retest scope starts from source and unknown-usage groups.

Let n^k_{fixed}, n^k_{broken}, and $n^k_{\text{unresolved}}$ count corrected errors, broken successes, and unresolved source failures. Retain the group only if

$$n^k_{\text{fixed}} > n^k_{\text{broken}} + n^k_{\text{unresolved}}.$$

Otherwise, all of J_k is rolled back, restoring old fields and jointly reversing new-node–edge pairs or edge replacements. Graph objects thus determine retest scope, and structural dependencies determine rollback units (Appendix C.7).

Big Gate: epoch-level commit. Merge all local survivors into the uncommitted candidate $\widetilde{G}^{(t+1)}$. After a full-pool rollout, let $n^{(t)}_{\text{corrected}}$ and $n^{(t)}_{\text{regressed}}$ count incorrect-to-correct and correct-to-incorrect transitions relative to $G^{(t)}$. Then

$$G^{(t+1)} = \begin{cases} \widetilde{G}^{(t+1)}, & n^{(t)}_{\text{corrected}} > n^{(t)}_{\text{regressed}}, \\ G^{(t)}, & \text{otherwise.} \end{cases}$$

Positive net gain commits the candidate and its rollout as the next reference; otherwise, both graph and reference are restored. The Big Gate evaluates one executable candidate rather than splicing local outcomes.

Thus, one graph connects semantic activation, targeted updating, validation, and rollback.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Evaluation objectives. We evaluate GRAPHSKILLAA as a graph-space optimizer for frozen language models. The experiments ask whether optimized graphs improve over controlled skill-representation stages, how their results compare with source-protocol references, which components contribute to downstream utility, whether the approach extends to sequential interaction, and how the graph evolves across epochs.

Models. We evaluate OpenAI's GPT-5.6-sol, GPT-5.5, and GPT-5.4-mini (OpenAI, 2026c;b;a); Google's Gemini-3.5-Flash (Google AI for Developers, 2026); and Alibaba's Qwen3.8-Flash (Alibaba Cloud Model Studio, 2026) and Qwen3.6-35B-A3B (Qwen Team, 2026).

Benchmarks. We use three main benchmarks spanning open-domain question answering, mathematical reasoning, and document understanding:

1. **SearchQA** (Dunn et al., 2017): open-domain question answering from retrieved evidence.
2. **LiveMathematicianBench** (He et al., 2026), abbreviated LiveMath: research-level multi-step mathematical reasoning.
3. **DocVQA** (Mathew et al., 2021): document-image question answering combining visual localization and text understanding.

A separate ALFWorld experiment (Shridhar et al., 2021) evaluates sequential interaction.

Model	Benchmark	No skill	Flat skill	Structured skill G0	G0 + Trace/ Usage	GraphSkillAA
GPT-5.6-sol	SearchQA	$66.0_{\pm 2.5}$	$72.5_{\pm 0.5}$	$75.0_{\pm 1.8}$	$76.2_{\pm 1.5}$	$\mathbf{81.5}_{\pm 1.3}$
	LiveMath	$28.2_{\pm 1.7}$	$42.2_{\pm 3.0}$	$44.2_{\pm 2.6}$	$51.3_{\pm 3.4}$	$\mathbf{66.7}_{\pm 2.1}$
	DocVQA	$68.0_{\pm 3.3}$	$87.7_{\pm 0.5}$	$88.0_{\pm 1.3}$	$88.2_{\pm 2.0}$	$\mathbf{91.2}_{\pm 1.0}$
Gemini-3.5-Flash	SearchQA	$75.0_{\pm 0.5}$	$77.5_{\pm 0.5}$	$77.5_{\pm 0.5}$	$77.5_{\pm 1.0}$	$\mathbf{79.0}_{\pm 0.3}$
	LiveMath	$34.5_{\pm 2.1}$	$49.6_{\pm 2.1}$	$52.4_{\pm 0.4}$	$53.3_{\pm 3.4}$	$\mathbf{62.1}_{\pm 2.6}$
	DocVQA	$92.8_{\pm 0.5}$	$94.0_{\pm 1.0}$	$93.8_{\pm 0.5}$	$93.5_{\pm 0.3}$	$\mathbf{95.0}_{\pm 0.3}$
Qwen3.8-Flash	SearchQA	$70.7_{\pm 1.3}$	$74.2_{\pm 1.8}$	$74.5_{\pm 1.5}$	$72.2_{\pm 0.8}$	$\mathbf{76.5}_{\pm 0.8}$
	LiveMath	$30.2_{\pm 2.1}$	$31.6_{\pm 4.3}$	$56.7_{\pm 0.4}$	$55.3_{\pm 3.4}$	$\mathbf{60.1}_{\pm 3.4}$
	DocVQA	$93.5_{\pm 0.5}$	$92.8_{\pm 0.5}$	$94.2_{\pm 0.8}$	$93.3_{\pm 0.3}$	$\mathbf{95.0}_{\pm 0.8}$

Table 2: Progressive skill-graph construction and optimization (%).

Benchmark	Model	No skill	Trace2Skill	TextGrad	GEPA	SkillOpt	GraphSkillAA
SearchQA	GPT-5.5	77.7	82.4	81.4	84.8	82.6	$\mathbf{87.0}_{\pm 0.6}$
	GPT-5.4-mini	75.9	78.6	77.5	79.4	81.3	$\mathbf{82.6}_{\pm 1.0}$
	Qwen3.6-35B-A3B	72.7	75.4	76.4	75.8	80.7	$\mathbf{83.4}_{\pm 0.5}$
LiveMath	GPT-5.5	37.6	51.9	49.2	43.3	51.1	$\mathbf{58.6}_{\pm 2.4}$
	GPT-5.4-mini	14.8	32.8	27.2	27.2	32.8	$\mathbf{36.3}_{\pm 2.0}$
	Qwen3.6-35B-A3B	31.2	29.6	7.3	31.2	28.0	$\mathbf{51.9}_{\pm 1.2}$
DocVQA	GPT-5.5	78.8	90.6	87.2	89.1	89.7	$\mathbf{91.1}_{\pm 0.7}$
	GPT-5.4-mini	71.4	88.5	84.0	83.7	82.2	$\mathbf{89.0}_{\pm 3.2}$
	Qwen3.6-35B-A3B	87.6	90.4	84.5	88.0	91.4	$\mathbf{92.7}_{\pm 1.1}$

Table 3: Comparison with skill-evolution and text-optimization methods under the SkillOpt direct-chat protocol (%).

Evaluation metrics and repeated runs. SearchQA and LiveMath use exact-match accuracy; DocVQA is correct only when ANLS$= 1$. Unless noted otherwise, we report the mean and half-range, $(\max - \min)/2$, over three runs. For graph-optimization settings, these are independent optimization-and-evaluation runs with seeds 42, 43, and 44; for frozen settings, the same seeds control evaluation only. Models remain frozen and test results never update the graph. Full settings are in Appendix E.

4.2 Main Results and Progressive Analysis

Table 2 progressively compares five settings under the same data splits and frozen model parameters: *No skill* uses no external skill; *Flat skill* directly adds the original SkillOpt text; *Structured skill G0* deterministically converts the same text into a graph with applicability conditions, execution steps, and typed relations, but does not update it; *G0 + Trace/Usage* keeps G0 fixed and additionally records the execution path and the node and edge IDs used; and *GraphSkillAA* further updates the graph from successful–failed contrasts and validates modifications with the Local and Big Gates. All three benchmarks use disjoint update/test splits, and the held-out test set never enters graph initialization, updating, or selection. Full split and grouping details are in Appendix E.

GraphSkillAA achieves the highest observed mean for every model–benchmark combination, and the progression clarifies where the gains arise. Flat skill generally improves over No skill, showing that the initial skills provide useful task procedures. From Flat skill to Structured skill G0, the structured representation separates applicability conditions, execution steps, and skill relations, making retrieval and activation conditions clearer while explicitly specifying how skills compose. G0 + Trace/Usage further requires the student to report its execution process and the graph objects used, thereby constraining the model to organize its solution around the skills. Finally, GraphSkillAA improves over G0 + Trace/Usage for every model and benchmark, demonstrating the end-to-end value of attribution-guided updates and validation. The especially large gains on LiveMath highlight the benefit of localized editing and regression control for multi-step skill composition; Table 4 further identifies the contribution of each mechanism through targeted ablations.

Category	Component	GPT-5.6-sol			GPT-5.4-mini		
		SearchQA	LiveMath	DocVQA	SearchQA	LiveMath	DocVQA
Graph structure ablation	Flat skill	$72.5_{\pm 0.5}$	$44.2_{\pm 2.6}$	$85.5_{\pm 1.3}$	$74.2_{\pm 1.0}$	$24.8_{\pm 2.1}$	$67.5_{\pm 15.5}$
	Nodes only	$77.5_{\pm 0.3}^{+5.0}$	$62.1_{\pm 2.1}^{+17.9}$	$90.0_{\pm 1.0}^{+4.5}$	$77.0_{\pm 0.8}^{+2.8}$	$21.4_{\pm 0.4}^{-3.4}$	$86.5_{\pm 1.8}^{+19.0}$
Retrieval mechanism ablation	No when-to-use & avoid	$70.5_{\pm 1.8}^{-2.0}$	$42.7_{\pm 4.3}^{-1.5}$	$84.3_{\pm 1.0}^{-1.2}$	$73.5_{\pm 1.5}^{-0.7}$	$19.9_{\pm 0.4}^{-4.9}$	$78.2_{\pm 1.8}^{+10.7}$
	No edge-rationale	$78.5_{\pm 0.8}^{+6.0}$	$63.5_{\pm 6.0}^{+19.3}$	$89.0_{\pm 0.5}^{+3.5}$	$76.5_{\pm 0.3}^{+2.3}$	$26.5_{\pm 0.9}^{+1.7}$	$88.0_{\pm 0.5}^{+20.5}$
	S–Q retrieval	$73.5_{\pm 2.3}^{+1.0}$	$42.5_{\pm 1.7}^{-1.7}$	$86.0_{\pm 0.8}^{+0.5}$	$72.5_{\pm 1.0}^{-1.7}$	$21.1_{\pm 2.6}^{-3.7}$	$82.8_{\pm 0.5}^{+15.3}$
	Nearest-Q reuse	$77.7_{\pm 1.3}^{+5.2}$	$58.1_{\pm 1.7}^{+13.9}$	$88.5_{\pm 1.0}^{+3.0}$	$76.5_{\pm 1.5}^{+2.3}$	$25.4_{\pm 3.4}^{+0.6}$	$87.7_{\pm 0.8}^{+20.2}$
Update mechanism ablation	No structured attribution	$73.2_{\pm 1.5}^{+0.7}$	$38.5_{\pm 5.6}^{-5.7}$	$86.7_{\pm 4.3}^{+1.2}$	$73.0_{\pm 1.0}^{-1.2}$	$18.5_{\pm 1.7}^{-6.3}$	$85.7_{\pm 2.5}^{+18.2}$
	No correct-case protection	$80.5_{\pm 0.5}^{+8.0}$	$62.4_{\pm 3.8}^{+18.2}$	$89.2_{\pm 0.5}^{+3.7}$	$75.5_{\pm 1.3}^{+1.3}$	$22.5_{\pm 0.4}^{-2.3}$	$87.3_{\pm 0.8}^{+19.8}$
	No Local Gate	$78.0_{\pm 2.3}^{+5.5}$	$49.0_{\pm 5.6}^{+4.8}$	$89.5_{\pm 1.8}^{+4.0}$	$74.7_{\pm 2.3}^{+0.5}$	$20.2_{\pm 3.4}^{-4.6}$	$84.5_{\pm 2.8}^{+17.0}$
	No Big Gate	$80.0_{\pm 1.3}^{+7.5}$	$62.1_{\pm 11.1}^{+17.9}$	$90.0_{\pm 1.0}^{+4.5}$	$76.8_{\pm 0.8}^{+2.6}$	$25.9_{\pm 1.3}^{+1.1}$	$87.2_{\pm 0.5}^{+19.7}$
Teacher-transfer evaluation	GPT-5.4-mini teacher	$78.2_{\pm 0.5}^{+5.7}$	$64.7_{\pm 1.7}^{+20.5}$	$90.0_{\pm 1.3}^{+4.5}$	$75.7_{\pm 0.5}^{+1.5}$	$23.9_{\pm 3.0}^{-0.9}$	$86.3_{\pm 1.3}^{+18.8}$
	GPT-5.6-sol teacher	$\mathbf{81.5}_{\pm 1.3}^{+9.0}$	$\mathbf{66.7}_{\pm 2.1}^{+22.5}$	$\mathbf{91.2}_{\pm 1.0}^{+5.7}$	$\mathbf{78.2}_{\pm 0.8}^{+4.0}$	$\mathbf{29.1}_{\pm 2.1}^{+4.3}$	$\mathbf{89.0}_{\pm 1.0}^{+21.5}$

Table 4: Component ablations and mechanism analysis (%). Colored superscripts show differences from Flat skill: red denotes an increase and green denotes a decrease. Teacher effects are obtained by comparing the two teacher rows for the same fixed student.

4.3 Comparison with Existing Methods

Table 3 compares GraphSkillAA with No skill, Trace2Skill (Ni et al., 2026), TextGrad (Yuksekgonul et al., 2025), GEPA (Agrawal et al., 2026), and SkillOpt (Yang et al., 2026) on GPT-5.5, GPT-5.4-mini (OpenAI, 2026b;a), and Qwen3.6-35B-A3B (Qwen Team, 2026). The comparison follows the direct-chat protocol of SkillOpt Table 1, using the same released instances, test IDs, original inputs, inference budget, scorer, and approximately 2:1:7 train/validation/test split. GraphSkillAA updates only on train/validation and reserves test for final evaluation; exact split and protocol details are in Appendix E.

The baselines cover four complementary optimization paradigms. Trace2Skill consolidates execution trajectories into reusable skills; TextGrad propagates textual feedback to optimize language-based variables; GEPA evolves prompts by reflecting on complete trajectories; and SkillOpt applies bounded edits and validation to a persistent skill document. GraphSkillAA advances beyond these flat or candidate-level optimization units by exposing node fields and typed relations as explicit edit targets. Successful and failed executions identify the implicated object, and the Local and Big Gates validate both the targeted repair and the merged graph.

GraphSkillAA ranks first in all nine model–benchmark combinations. It exceeds the strongest competing method in every row by 1.3–2.7 *pp* on SearchQA, 3.5–20.7 *pp* on LiveMath, and 0.5–1.3 *pp* on DocVQA. This consistent lead across three evaluated models and three task types demonstrates the advantage of optimizing skills as addressable graph objects rather than as undifferentiated text. Compared with trajectory distillation and prompt/text optimization, GraphSkillAA preserves reusable knowledge in explicit node fields and relations and routes each repair to the implicated object; compared with flat skill-document optimization, it additionally models composition dependencies and validates changes at both affected-object and merged-graph scopes. The largest gains occur on LiveMath, where multi-step reasoning benefits most directly from typed relations, localized repair, and dependency-aware validation.

4.4 Component Ablations and Mechanism Analysis

Table 4 examines four factors under the same Table-A benchmarks, splits, and metrics. Flat skill is the common baseline; colored superscripts show changes from it (red for increases and green for decreases). The graph-structure rows test structured nodes and explicit relations. Retrieval rows apply read-only interventions to the matching self-teacher terminal graph, covering activation boundaries, edge rationales, S–Q retrieval, and Nearest-Q reuse. Update rows remove structured attribution, correct-case protection, the Local Gate, or the Big Gate. The final two rows form a separate **Teacher-transfer evaluation**, holding the student, data, and three-epoch schedule fixed while changing only the patch teacher. Thus, teacher effects should be read by comparing the two teacher rows for the same student, not their superscripts. Appendix F defines all interventions and comparisons in detail.

Graph-structure ablation. Under the self-teacher reference, removing all edges (Nodes only) lowers performance by $1.8\,pp$ on average, with the clearest gains from relations appearing for the GPT-5.6-sol student. The optimized nodes remain $7.6\,pp$ above Flat skill on average even without edges, showing that structured node fields provide the primary representation gain, while typed edges add complementary support for coordinating already-relevant procedures.

Retrieval-mechanism ablation. These results strengthen the activation-boundary explanation. Relative to each student's self-teacher Full graph, removing `when_to_use` and `avoid` causes the largest retrieval-side loss ($9.4\,pp$ on average), and fixed S–Q retrieval also trails Full ($7.8\,pp$). Nearest-Q reuse narrows but does not close this gap, while edge rationales provide a smaller complementary signal. The strongest and most consistent retrieval mechanism is therefore the node-level contract that tells the student when a skill applies and when it should be rejected.

Update-mechanism ablation. Under the same-student self-teacher reference, replacing structured attribution with a generic failure summary causes the largest update-side drop ($8.3\,pp$ on average), establishing attribution-conditioned object routing as the main source of repair utility among the tested update mechanisms. Removing the Local Gate causes the next-largest average loss ($4.9\,pp$), confirming the value of affected-scope regression testing, while correct-case protection contributes a further $1.3\,pp$ on average. The Big Gate adds complementary epoch-level control by admitting only merged candidates with positive update-pool gain; for GPT-5.6-sol, it also improves SearchQA, LiveMath, and DocVQA by $1.5/4.6/1.2\,pp$. Together, structured attribution, success guards, and the two Gates turn each proposed edit into a localized, tested, and reversible graph update.

Teacher-transfer evaluation. The Teacher-transfer rows test cross-model patch transfer rather than component removal. Holding the student fixed, they replace only the teacher used for attribution and patch generation; the colored superscripts remain Flat-skill gains, so teacher effects are read by comparing the two teacher rows for the same student. The results show that a stronger teacher is consistently better, while the weaker GPT-5.4-mini teacher still yields substantial gains for the stronger GPT-5.6-sol student, indicating that the graph-patch interface does not require an exact teacher–student match. Appendix F.4 lists the four pairings, and Appendix F.5 reports the fixed-student deltas.

4.5 Additional Evaluations and Analysis

ALFWorld evaluation. ALFWorld shows that not every problem can be solved through skills. Although all methods solve every task within 100 steps, the 50-step budget exposes failures driven by exploration or execution efficiency rather than graph defects (Appendix C.1). In such cases, preserving the current graph through `NO_PATCH` is the correct decision.

Optimization stopping behavior. Epoch trajectories show that the Gates reject patches without net improvement, preventing unproductive proposals from changing executable state and providing a stopping signal (Appendix C.2). Non-graph reasoning, perception, or execution failures should instead be routed to the model or an appropriate tool.

5 Conclusion

We introduced GraphSkillAA, which improves external skills from execution feedback while keeping model parameters frozen. Its graph organizes skill use and provides stable addresses for locating and tracking changes. By contrasting successful and failed executions, it edits only evidence-supported rules or relations instead of rewriting an entire skill. The Local Gate retests affected cases, while the Big Gate commits the merged graph only when overall performance improves; otherwise, it rolls back. Across three benchmarks and multiple model families, GraphSkillAA achieves the highest mean in every main setting and ranks first in all nine method comparisons. Progressive experiments and ablations show that explicit applicability, execution, and composition rules, targeted attribution, success protection, and whole-graph validation all contribute. Skill graphs therefore support not only selection and composition, but also precise repair and regression control. Future work can further explore combining skill updating with model training.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0e9e708b6f48e14fd0ac29e167413f76-Abstract-Conference.html.

Alibaba Cloud Model Studio. Qwen3.8-Flash model information. Alibaba Cloud Model Studio documentation, 2026. URL https://www.alibabacloud.com/help/en/model-studio/qwen3-8-flash. Accessed September 26, 2026.

Tong Bai, Zhenglin Wan, Pengfei Zhou, Xingrui Yu, Yang You, and Ivor W. Tsang. SkillDAG: Self-evolving typed skill graphs for LLM skill selection at scale. arXiv preprint arXiv:2606.03056, 2026. URL https://arxiv.org/abs/2606.03056.

Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Guney, Volkan Cirik, and Kyunghyun Cho. SearchQA: A new Q&A dataset augmented with context from a search engine. arXiv preprint arXiv:1704.05179, 2017. URL https://arxiv.org/abs/1704.05179.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A Graph RAG approach to query-focused summarization. arXiv preprint arXiv:2404.16130, 2024. URL https://arxiv.org/abs/2404.16130.

Google AI for Developers. Gemini 3.5 Flash model documentation. Google AI for Developers model documentation, 2026. URL https://ai.google.dev/gemini-api/docs/models/gemini-3.5-flash. Accessed September 26, 2026.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 59532–59569, 2024. doi: 10.52202/079017-1902. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/6ddc001d07ca4f319af96a3024f6dbd1-Abstract-Conference.html.

Linyang He, Qiyao Yu, Hanze Dong, Baohao Liao, Xinxing Xu, Micah Goldblum, Jiang Bian, and Nima Mesgarani. LiveMathematicianBench: A live benchmark for research-level mathematical reasoning with proof sketches. arXiv preprint arXiv:2604.01754, 2026. URL https://arxiv.org/abs/2604.01754.

Zhiyuan Jiang, Fangrui Huang, Hanwen Xing, Xander Wu, Yipeng Gao, Rui Cao, Mengdi Wang, Shilong Liu, and Yijiang Li. Demystifying agent skills: Why they work-until they don't. arXiv preprint arXiv:2608.14036, 2026. URL https://arxiv.org/abs/2608.14036.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. SkillsBench: Benchmarking how well agent skills work across diverse tasks. arXiv preprint arXiv:2602.12670, 2026a. URL https://arxiv.org/abs/2602.12670.

Xiaoyuan Li, Moxin Li, Keqin Bao, Yubo Ma, Wenjie Wang, Dayiheng Liu, and Fuli Feng. SkillGraph: Skill-augmented reinforcement learning for agents via evolving skill graphs. arXiv preprint arXiv:2605.12039, 2026b. URL https://arxiv.org/abs/2605.12039.

Zhiyuan Li, Linyuan Gao, Xuechun Ding, Hongwei Chen, Yuan Wu, and Yi Chang. CaSKG: Counterfactual-causal skill graphs for scalable agent skill retrieval. arXiv preprint arXiv:2608.25500, 2026c. URL https://arxiv.org/abs/2608.25500.

Zihan Lin, Zhenyu Chen, Jiawen Wei, Xiaohan Wang, Jie Cao, Jiajun Chai, Wei Lin, Guojun Yin, and Ran He. When not to imitate: Boundary-aware skill memory for reliable tool-use LLM agents. In *Findings of the Association for Computational Linguistics: EMNLP 2026*, 2026. URL https://arxiv.org/abs/2608.22339.

Dawei Liu, Zongxia Li, Hongyang Du, Xiyang Wu, Shihang Gui, Yongbei Kuang, and Lichao Sun. Graph-of-Skills: Dependency-aware structural retrieval for massive agent skills. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, 2026. URL https://arxiv.org/abs/2604.05333.

Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Ziheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, and Wenhai Wang. ControlLLM: Augment language models with tools by searching on graphs. In *Computer Vision – ECCV 2024*, Lecture Notes in Computer Science, pp. 89–105. Springer Nature Switzerland, 2024. doi: 10.1007/978-3-031-73254-6_6. URL https://doi.org/10.1007/978-3-031-73254-6_6.

Minesh Mathew, Dimosthenis Karatzas, and C. V. Jawahar. DocVQA: A dataset for VQA on document images. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 2199–2208. IEEE, 2021. doi: 10.1109/WACV48630.2021.00225. URL https://openaccess.thecvf.com/content/WACV2021/html/Mathew_DocVQA_A_Dataset_for_VQA_on_Document_Images_WACV_2021_paper.html.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. Augmented language models: a survey. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=jh7wH2AzKK.

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. URL https://arxiv.org/abs/2603.25158.

OpenAI. GPT-5.4 Mini model. OpenAI API documentation, 2026a. URL https://developers.openai.com/api/docs/models/gpt-5.4-mini. Accessed September 26, 2026.

OpenAI. GPT-5.5 model. OpenAI API documentation, 2026b. URL https://developers.openai.com/api/docs/models/gpt-5.5. Accessed September 26, 2026.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition. OpenAI release article, 2026c. URL https://openai.com/index/gpt-5-6/. Accessed September 26, 2026.

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, Maohao Shen, Vishy Tirumalashetty, George Lee, Jiawei Han, Tomas Pfister, and Chen-Yu Lee. SkillOS: Learning skill curation for self-evolving agents. arXiv preprint arXiv:2605.06614, 2026a. URL https://arxiv.org/abs/2605.06614.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/980ea04d23d1f6908964eba2a74afe45-Abstract-Conference.html.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 126544–126565, 2024. doi: 10.52202/079017-4020. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/e4c61f578ff07830f5c37378dd3ecb0d-Abstract-Conference.html.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.494. URL https://aclanthology.org/2023.emnlp-main.494/.

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

Qwen Team. Qwen3.6 model collection and technical documentation. Official Qwen model collection, 2026. URL https://huggingface.co/collections/Qwen/qwen36. Accessed September 26, 2026.

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

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

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. In *2nd Workshop on Agent Learning in Open-Endedness (ALOE) at NeurIPS 2023*, 2023. URL https://openreview.net/forum?id=pAMNKGwja6.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. arXiv preprint arXiv:2602.08234, 2026. URL https://arxiv.org/abs/2602.08234.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. In *Agent Skills '26 Workshop at the ACM Conference on AI and Agentic Systems*, 2026. URL https://arxiv.org/abs/2602.12430.

Ruiyao Xu, Tiankai Yang, and Wei-Chieh Huang. HyperSkill: Self-evolving LLM agents via hypergraph-structured skill memory. arXiv preprint arXiv:2608.16114, 2026. URL https://arxiv.org/abs/2608.16114.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/3339f19c5fcee3ad74502947a32be9e6-Abstract-Conference.html.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills. arXiv preprint arXiv:2605.23904, 2026. URL https://arxiv.org/abs/2605.23904.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R. Fung, Hao Peng, and Heng Ji. CRAFT: Customizing LLMs by creating and retrieving from specialized toolsets. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/af31604708f3e44b4de9fdfa6dcaa9d1-Abstract-Conference.html.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. Optimizing generative AI by backpropagating language model feedback. *Nature*, 639:609–616, 2025. doi: 10.1038/s41586-025-08661-4. URL https://doi.org/10.1038/s41586-025-08661-4.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue (Steve) Liu, Xiaoxiao Li, and Philip S. Yu. CoEvoSkills: Self-evolving agent skills via co-evolutionary verification. In *Conference on Language Modeling (COLM)*, 2026a. URL https://arxiv.org/abs/2604.01687.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. MemSkill: Learning and evolving memory skills for self-evolving agents. arXiv preprint arXiv:2602.02474, 2026b. URL https://arxiv.org/abs/2602.02474.

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. doi: 10.1609/aaai.v38i17.29936. URL https://doi.org/10.1609/aaai.v38i17.29936.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web agents can self-improve by discovering and honing skills. arXiv preprint arXiv:2504.07079, 2025. URL https://arxiv.org/abs/2504.07079.

YanZhao Zheng, ZhenTao Zhang, Chao Ma, YuanQiang Yu, JiHuai Zhu, Yong Wu, Tianze Xu, Baohua Dong, Hangcheng Zhu, Ruohui Huang, and Gang Yu. SkillRouter: Skill routing for LLM agents at scale. arXiv preprint arXiv:2603.22455, 2026. URL https://arxiv.org/abs/2603.22455.

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. doi: 10.18653/v1/2024.lrec-main.1413. URL https://aclanthology.org/2024.lrec-main.1413/.

APPENDIX CONTENTS

A Reproducibility Statement . . . 15

B AI Use Statement . . . 15

C Additional Evaluations and Method Specification . . . 15

C.1. Sequential Interaction under Step Budgets . . . 15
C.2. Epoch-Level Commit Dynamics . . . 15
C.3. End-to-End GraphSkillAA Pipeline . . . 15
C.4. Epoch-Level Execution and Usage Records . . . 17
C.5. Allowed Graph Edits and Atomic Groups . . . 17
C.6. Attribution Routing and Case Studies . . . 18
C.7. Gate Scope Expansion and Rollback Mechanism . . . 18
C.8. Validation Scope and Evidence Levels . . . 20

D Prompt Templates . . . 20

D.1. GraphSkillAA Graph Workflow on SearchQA . . . 21
D.2. Abbreviated SearchQA SkillGraph Example . . . 22
D.3. Student Execution and Trace/Usage Specification . . . 23
D.4. Abductive Attribution Prompt . . . 24
D.5. Epoch-Level Meta-Auditor Prompt . . . 25
D.6. Patch Synthesis and Gate Audit Prompts . . . 26

E Experimental Protocol . . . 27

E.1. Dataset Splits and Group Construction . . . 27
E.2. Model Access, Inference, and Optimization Settings . . . 27
E.3. Evaluation Protocol . . . 28
E.4. Result Provenance and Comparison Protocol . . . 28

F Ablation Protocols and Retrieval Definitions . . . 29

F.1. Shared Settings and Graph-Structure Ablation . . . 29
F.2. Retrieval-Mechanism Ablations . . . 29
F.3. Update-Mechanism Ablations . . . 30
F.4. Teacher-Transfer Evaluation . . . 30
F.5. Comparison Protocol for Table C . . . 30

G Residual Error Analysis . . . 30

H Discussion, Limitations, and Future Work . . . 31

H.1. Interpretation of Experimental Findings . . . 31
H.2. Limitations . . . 32
H.3. Near-Term Improvements . . . 32
H.4. Long-Term Research Directions . . . 33

Setting	S@50	S@100	ΔS	Steps@50	TO@50
SkillOpt Markdown	$89.6_{\pm 1.1}$	$\mathbf{100.0}_{\pm 0.0}$	+10.4	$9.7_{\pm 0.1}$	$10.4_{\pm 1.1}$
Initial Graph skill	$\mathbf{95.5}_{\pm 0.4}$	$\mathbf{100.0}_{\pm 0.0}$	**+4.5**	$\mathbf{9.5}_{\pm 0.2}$	$\mathbf{4.5}_{\pm 0.4}$

Table 5: ALFWorld step-budget evaluation. S@50/S@100: success within 50/100 steps; ΔS: absolute percentage-point gain; Steps@50: steps among successful runs; TO@50: timeout rate.

A Reproducibility Statement

The supplementary material reports the main settings needed to reproduce the results, including the graph schema, allowed edits, attribution and Gate rules, data splits, model settings, and scorers. We will release the code, experiment configurations, data splits, final skill graphs, and per-example predictions so that the results can be checked and reproduced.

B AI Use Statement

Generative AI tools assisted with language editing, code-level consistency checks, literature search, and LaTeX conversion. The authors designed the method and experiments, selected the cited literature, and checked the technical statements and numerical results against the original experiment records. The authors take responsibility for the manuscript.

C Additional Evaluations and Method Specification

This appendix first uses a step-budget stress test and epoch-level commit trajectories to examine when graph updating helps and when it should stop. It then describes the update pipeline and its validation rules in detail.

C.1 Sequential Interaction under Step Budgets

ALFWorld tests whether the optimizer can distinguish a reusable skill defect from an execution-efficiency problem. Table 5 follows the SkillOpt ALFWorld protocol independently of the three main benchmarks.

Both settings reach 100% success within 100 steps. Under the stricter 50-step budget, Initial Graph skill achieves a higher success rate, shorter successful paths, and fewer timeouts than SkillOpt Markdown. The remaining gap between the 50- and 100-step results shows that some failures arise from exploration cost or execution efficiency rather than a reusable graph defect. Such cases should not be written into the skill graph; `NO_PATCH` preserves the initial graph when the evidence points to budget-induced truncation or inefficient execution.

C.2 Epoch-Level Commit Dynamics

Figure 3 presents the GPT-5.6-sol runs summarized in Table 2 and shows when candidate graphs were accepted or rolled back.

The released default launcher disables in-loop epoch testing and records the terminal held-out evaluation; consequently, it reproduces the optimization trajectory and final score but does not regenerate the dashed intermediate test series in Figure 3. Those intermediate values are read-only diagnostics and never enter attribution or either Gate.

SearchQA and LiveMath obtain their best graphs at epoch 2; their epoch-3 candidates fail the Big Gate, so the previous graphs remain in use. DocVQA rolls back the epoch-2 candidate and accepts a different repair set at epoch 3. Thus, rejected candidates do not change the current skill graph, and held-out test results never affect the commit decision.

C.3 End-to-End GraphSkillAA Pipeline

Optimization can be viewed as a sequence of transitions between committed graphs. At epoch t, the current state consists of graph $G^{(t)}$ and its results R on the complete update pool. Attribution and

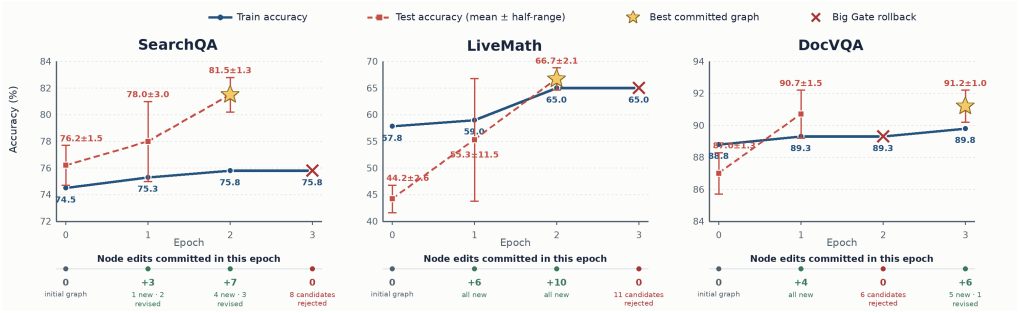

Figure 3: GPT-5.6-sol GraphSkillAA epoch trajectories (%). Solid blue lines show hard accuracy on the complete update pool; dashed red lines show held-out test means, with error bars denoting half-ranges over three runs. Gold stars mark the best committed graph, red crosses mark Big-Gate rollback, and the lower strip reports committed node edits. No test is rerun when rollback leaves the graph unchanged; node counts exclude edge edits.

Algorithm 1 Complete GraphSkillAA pipeline

Require: frozen M_{student}, frozen M_{teacher}, update pool $\mathcal{D}_{\text{update}}$, fixed update groups $\mathcal{Q}$, initial graph $G^{(0)}$, epochs E
Ensure: final committed graph $G^{(E)}$

$R \leftarrow \textsc{Rollout}(M_{\text{student}}, G^{(0)}, \mathcal{D}_{\text{update}})$
for $t = 0, \ldots, E-1$ **do**
  $F \leftarrow$ failed update examples in R
  $D \leftarrow \textsc{Attribute}(M_{\text{teacher}}, F, G^{(t)}, R, \mathcal{Q})$ ▷ Algorithm 2
  $P \leftarrow \textsc{TargetedPatch}(D, R, \mathcal{Q})$ ▷ apply route constraints and success guards
  $P^{\star} \leftarrow \textsc{JointSemanticSynthesis}(P, D, R)$ ▷ synchronize connected edits
  $\{J_k\}_{k=1}^{K} \leftarrow \textsc{AtomicGroups}(P^{\star})$
  $S \leftarrow \emptyset$
  for each J_k **do**
    $J'_k \leftarrow J_k$
    $b \leftarrow \textsc{LocalGate}(G^{(t)}, J'_k, R)$ ▷ Appendix C.7
    if $\neg b$ and J'_k is refinable **then**
      $J'_k \leftarrow \textsc{RefineRejectedPatch}(J'_k)$ ▷ at most once
      $b \leftarrow \textsc{LocalGate}(G^{(t)}, J'_k, R)$
    end if
    if b **then**
      $S \leftarrow S \cup \{J'_k\}$
    end if
  end for
  $\widetilde{G} \leftarrow \textsc{ApplyAll}(G^{(t)}, S)$
  $\widetilde{R} \leftarrow \textsc{Rollout}(M_{\text{student}}, \widetilde{G}, \mathcal{D}_{\text{update}})$
  if $\textsc{NumCorrected}(R, \widetilde{R}) > \textsc{NumRegressed}(R, \widetilde{R})$ **then**
    $G^{(t+1)} \leftarrow \widetilde{G}$; $R \leftarrow \widetilde{R}$
  else
    $G^{(t+1)} \leftarrow G^{(t)}$ ▷ keep the current reference R
  end if
end for
return $G^{(E)}$

local validation may create temporary candidates, but only a candidate that passes the Big Gate can replace the current state. Algorithm 1 gives the complete procedure; Algorithm 2 details attribution, and Appendix C.7 defines the retest set.

A complete rollout on the initial graph $G^{(0)}$ produces the first reference R. If the Big Gate accepts a candidate, both the graph and its reference results are updated; otherwise, both remain unchanged. Held-out test outcomes never enter this process.

C.4 EPOCH-LEVEL EXECUTION AND USAGE RECORDS

To localize failures and determine the retest scope reliably, each epoch first runs the complete update pool while keeping the graph fixed—800 examples for SearchQA and DocVQA and 468 for LiveMath. Requests may run concurrently, but the graph cannot change until all results have been collected. Fixed groups provide successful contrasts and expand the retest scope. Previous results are reused only when the graph, rendered prompt, sample set, and run status all match.

`usage` validation. `usage` is the deduplicated set of graph objects reported as changing an intermediate state or affecting the submitted answer. The generation contract requires every listed object to appear in `trace` as an explicit input–operation–output transition. Deterministic validation certifies student provenance, ID existence, valid edge endpoints, and exact agreement between the ID sets in `trace` and `usage`; it does not independently parse the semantic correctness of the stated transition. A missing or unknown ID, an ID-set mismatch, or an invalid relation marks usage as unknown. Valid records identify which examples may be affected by an edit; incomplete records can only expand, never narrow, the retest set.

C.5 ALLOWED GRAPH EDITS AND ATOMIC GROUPS

Object-level attribution is useful only when each repair is confined to a clearly defined scope. GRAPHSKILLAA therefore permits three classes of graph operations:

1. revise the relevant part of a node's `when_to_use`, `how_to_use`, or `avoid`, including replacement of a localized harmful rule;
2. add a complete specialist node with a supporting edge, or add or correct a PREREQ/ENHANCE edge, including its direction, type, or strength; and
3. delete a node or edge only when deletion is enabled and supported by evidence.

Dependent operations are atomic: a new node and its connecting edge, or an old edge and its replacement, are accepted or rolled back together. Existing titles and categories normally remain fixed; unsupported observations and execution lapses produce `NO_PATCH` at the attribution stage.

Joint semantic synthesis. The route-specific proposals are initial edit constraints rather than independently committed patches. Before structural grouping, every connected node/edge component with a concrete proposal is regenerated once from its complete failure evidence and related successful-use guards. This stage may synchronize a touched node's `when_to_use`, `how_to_use`, and `avoid` fields and may introduce up to four evidence-linked supporting edits, for example a specialist node and its connecting edge. It cannot create an edit without source evidence, revive a `NO_PATCH` execution lapse, or bypass subsequent structural checks and both Gates. The resulting component, rather than each preliminary field edit in isolation, is the candidate tested by the Local Gate.

Deterministic merge and structural validation. Candidate edits are processed in a fixed order. Exact duplicates are removed, and edits that share evidence or touch the same node are grouped before Local-Gate testing so that they cannot overwrite one another. Changes are first applied to a copy of the graph. An atomic group is rejected if it creates a duplicate relation, conflicting PREREQ/ENHANCE types on one directed pair, a missing endpoint, a self-loop, or a PREREQ cycle. If a conflict appears only after locally accepted groups are merged, the merged candidate does not enter the Big Gate.

Fixed co-occurrence relations. Teachers may propose PREREQ and ENHANCE relations, but they cannot directly add, delete, or relabel CO-OCCUR. In the reported experiments, CO-OCCUR edges are inherited from $G^{(0)}$ and remain fixed throughout the three optimization epochs. The released implementation includes an optional updater that retains a pair after at least two joint successes when $n_{ab}/\min(n_a, n_b) \geq 0.25$, but that updater is inactive in the reported protocol. Compact serialization keeps a canonical source–target order for every edge; for CO-OCCUR, the arrow is representational and does not imply execution order or causality. Edge strength is either `medium` or

Algorithm 2 Abductive attribution and edit routing

Require: failure, current graph, verifier feedback, validated skill-use record, successful siblings
Ensure: root cause and an edit target or `NO_PATCH`

Locate the earliest observable error and the smallest successful contrast
if neither can be localized **then**
    return `INSUFFICIENT_EVIDENCE`
end if
$C \leftarrow$ graph objects covering the required behavior
if $C = \varnothing$ **then**
    return `MISSING_SKILL_FAMILY`
end if
if C is absent from validated usage **then**
    return `MISSING_ACTIVATION_CUE`
end if
if stored semantics of a used object are harmful **then**
    return `HARMFUL_EXISTING_RULE`
end if
if a step, boundary, or required relation is missing **then**
    return `MISSING_OR_INCORRECT_PROCEDURE`
end if
if the graph is correct but execution diverges **then**
    return `EXECUTION_LAPSE`
end if
return `INSUFFICIENT_EVIDENCE`

`strong`: the default `medium` is omitted from compact rendering, while `strong` signals greater importance. Strength does not change applicability or ordering; only PREREQ imposes order.

C.6 Attribution Routing and Case Studies

Attribution routing answers where a repair should be considered; later validation decides whether that repair is kept. This subsection explains how the six routes correspond to specific node fields or relations.

Figure 4 illustrates the six mutually exclusive attribution outcomes. The first four permit changes only to the corresponding node field or relation; `EXECUTION_LAPSE` and `INSUFFICIENT_EVIDENCE` produce no graph edit. Every candidate that contains an edit must still pass both Gates.

Benchmark labels determine whether an answer is correct but do not identify a unique repair location; behaviorally equivalent repairs may also target different local spans. We therefore evaluate whether attribution selects a reasonable location rather than requiring it to match a pre-annotated text span. An edit may touch only the declared field or relation, is compared with the original graph on its source and affected cases, and persists only after both Gates accept it. This tests whether object-level attribution yields an effective repair without undue damage to established behavior.

C.7 Gate Scope Expansion and Rollback Mechanism

Once attribution selects an object, graph structure determines which examples may be affected and which dependent operations must be rolled back together. This section expands those rules into concrete sample sets and a recoverable update process.

For atomic group J_k, the Local-Gate scope in the main text expands into the following example-level sets, all drawn from the current update pool:

- $\mathrm{DirectSources}_k$: failed examples cited as source evidence by an operation in the group;
- $\mathrm{PriorUsers}_k$: reference examples whose valid skill-use record (`usage`) intersects an edited object—the node ID for a node-field edit, or the edge ID and both endpoint IDs for an existing-edge edit;
- $\mathrm{UnknownUsage}_k$: reference examples with missing or invalid skill-use records; and



Figure 4: Six attribution routes with verified examples and field-level edits. Each card pairs a SearchQA failure with its original clue, prediction, gold answer, validated semantic path, and the relevant skill field before and after attribution. *Candidate after* denotes a proposed edit before the Gates, not an automatically committed change; only the modified field is shown, and omitted fields remain unchanged. `EXECUTION_LAPSE` and `INSUFFICIENT_EVIDENCE` leave the graph unchanged.

- $\mathrm{GroupSiblings}_k$: other update examples in the same update group as any example in the preceding three sets.

The exact retest set is

$$\mathrm{Retest}_k = \mathrm{DirectSources}_k \cup \mathrm{PriorUsers}_k \cup \mathrm{GroupSiblings}_k \cup \mathrm{UnknownUsage}_k.$$

For a group containing only a newly added specialist and its connecting edge, touch is computed from the new node alone. Because that node has no reference users, $\mathrm{PriorUsers}_k$ is empty and retesting starts from direct sources, their update siblings, and usage-unknown examples. All prior users of the parent remain covered by the complete-pool Big Gate.

On this expanded set, the three counts in the main-text Local Gate are computed case by case against the current reference: n^k_{fixed} counts incorrect-to-correct examples, n^k_{broken} counts correct-

to-incorrect examples, and $n^k_{\text{unresolved}}$ counts direct source failures that remain incorrect. Thus,

$$n^k_{\text{fixed}} > n^k_{\text{broken}} + n^k_{\text{unresolved}}.$$

The decision applies to the complete J_k. For example, a new node and its connecting edge must be kept or removed together; the Gate cannot retain only the better-looking operation. Each corrected error contributes $+1$, each broken success contributes -1, and each motivating source failure that remains unresolved contributes another -1. The group proceeds only when the repair benefit strictly exceeds these costs; otherwise all of its operations are rolled back. This prevents a patch from hiding its original failure behind unrelated fixes while still allowing a small regression when the overall gain is positive.

The hard-transition count alone determines Local-Gate acceptance in the reported grouped full-pool protocol. A teacher produces a structured semantic explanation after the count for audit, but this explanation cannot override the numerical decision. If the first attempt fails without a structural materialization error and exposes ineffective cases, the rejected component may return once to a bounded refinement step and is then retested from scratch. The second decision again uses the same strict count; a second failure rolls back the complete component.

Controlled before–after comparison. Each evaluated example has a frozen pre-modification reference and a newly executed candidate result. The two sides use the same case identity and order, model route, reasoning effort, output limit, and response policy; the candidate changes the rendered graph. Provider generation does not expose a common random seed in the reported API path, so this is a matched-case comparison rather than a common-random-number experiment. The Local Gate compares the paired results on affected examples, while the Big Gate compares them on the complete update pool. At both levels, corrected cases must outnumber broken cases; the Local Gate additionally charges unresolved source failures, and ties are rolled back. The system stores the before–after result for every example and the final decision for later audit.

What the Big Gate restores. All Local-Gate survivors are first applied to the old graph and then evaluated together on the complete update pool. If the merged graph corrects no more cases than it breaks, the entire epoch is rejected: the graph, its rendered prompt, and the per-example results saved at the start of the epoch (the reference) are restored together. Results from different local candidates cannot be spliced into a synthetic baseline. The experiment uses one commit rule: on the complete update pool, the merged graph must correct more cases than it breaks. Commit and rollback always apply to the graph as a whole.

C.8 Validation Scope and Evidence Levels

Validation checks, in order, whether the execution record is complete, the attributed location is well supported, local behavior improves, and the merged graph produces an overall gain. The process advances only after the preceding level passes, as summarized in Table 6.

Here, *abductive* means identifying the smallest reusable defect whose repair can be tested through before–after behavior. Every candidate is compared with a frozen current-graph reference on the same sample set and under the same request settings; provider sampling itself is not seed-controlled. Uncertain `usage` expands rather than narrows the retest scope, and the complete update pool provides the final check. Because the datasets do not annotate a unique repair location, the evaluation asks whether the chosen location is explicit and whether editing it repairs failures while preserving established behavior.

D Prompt Templates

Before presenting the four prompt interfaces, this appendix first shows how a SearchQA query follows a shared graph path and enters specialist branches when their conditions apply. It then gives a compact graph representation and the main prompt templates. The prompt boxes reproduce the instructions used in the experiments and therefore retain their imperative, model-facing style. Only repeated formatting instructions and run-specific examples are omitted; all constraints that affect method behavior are preserved.

Level	Evidence checked	What this establishes
Record integrity	IDs in `trace` and `usage` agree, exist in the current graph, and have valid endpoints.	The record can be used to determine the affected scope; otherwise the example is treated as usage-unknown.
Routing validity	The earliest observable error, successful contrasts, graph coverage, activation, stored semantics, and execution are checked in order.	One localized component may be revised, or `NO_PATCH` preserves the graph when localization fails.
Local behavioral validity	The complete atomic group is rerun on direct sources, prior users, group siblings, and usage-unknown cases.	The proposed repair improves behavior over its affected scope under the Local-Gate rule.
Global behavioral validity	All locally retained groups are merged and rerun on the complete update pool.	The merged candidate has a positive hard-case net gain and may replace the committed graph.

Table 6: Evidence levels in attribution and Gate validation. Each level determines whether the process can proceed to the next decision.

D.1 GRAPHSKILLAA GRAPH WORKFLOW ON SEARCHQA

Figure 5 presents the graph execution process as a readable path. The upper panel shows eight nodes from the optimized graph: prerequisite edges connect the common steps of answer-type inference, evidence matching, relation checking, and answer normalization, while ENHANCE edges activate specialist checks only when needed. The three examples reuse this common path and branch according to their evidence. Stable node and edge IDs support both graph-guided execution and later failure attribution.

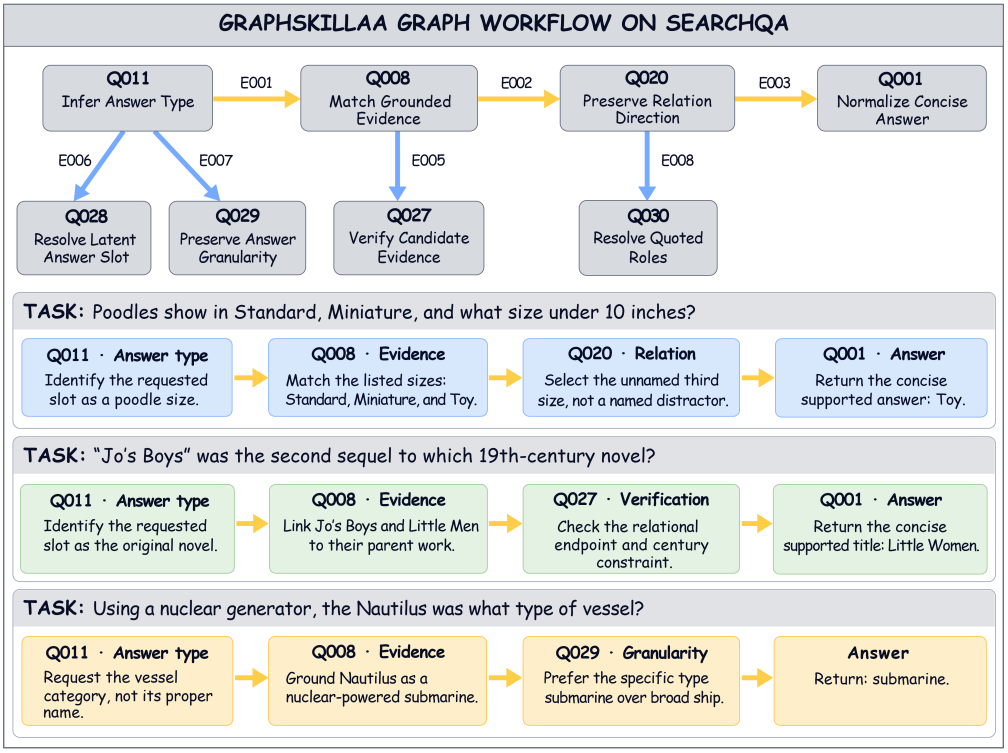

Figure 5: GraphSkillAA graph workflow on SearchQA. The upper panel shows the core prerequisite chain and specialist enhancement edges; the lower panels trace three examples from answer-type inference and evidence grounding to relation-specific checks and the final normalized answer.

D.2 ABBREVIATED SEARCHQA SKILLGRAPH EXAMPLE

The following excerpt shows the SearchQA SkillGraph rendered for the student. The two example nodes use the common three-field schema, and the edge rationales are abbreviated; ellipses mark omitted nodes and relations.

Abbreviated SearchQA SkillGraph example

```
Name: SearchQA SkillGraph
Task description: Answer a short open-domain question from the supplied
retrieved passages and return the shortest supported answer.

General principles (fixed task protocol):
- Answer only from the supplied context.
- Infer the requested answer type before extracting a candidate.
- Return reasoning_trace, graph_usage, and answer in the required format.
- Only the graph may evolve; never memorize a case ID or gold answer.

Node set V:
[Q001] Concise Answer Normalization 1
When to use: A supported candidate must be converted into the final
short-answer surface form.
How to use: Return the shortest unambiguous supported answer; remove only
generic descriptors that are not needed for identification.
Avoid: Do not shorten a full official name when it is explicitly requested
or when the shorter form would be ambiguous.

...

[Q027] Candidate Evidence Verification
When to use: Candidate evidence is ambiguous, partial, conflicting, or
supports a related entity of the wrong type.
How to use: Verify the requested type, endpoint, and every explicit
constraint before accepting the candidate.
Avoid: Do not replace a candidate unless another candidate satisfies the
complete constraint set with direct evidence.

Edge set E:
[E004] Q005 -[enhance; strength=strong]-> Q026
  Rationale: Keep the evidence-backed exception attached to Q005 without
  changing the base-node semantics globally.
[E005] Q008 -[enhance; strength=strong]-> Q027
  Rationale: Confine candidate verification to observable ambiguity instead
  of changing Q008 globally.
...
```

D.3 STUDENT EXECUTION AND TRACE/USAGE SPECIFICATION

In addition to its answer, the student produces two complementary records: `trace` describes the execution path in natural language, while `usage` lists the node and edge IDs that were actually used for subsequent impact analysis.

Student execution contract

```
You are an expert question answering agent.
Answer only from the supplied context.

Before the final answer, write exactly one plain-text graph execution
record inside <reasoning_trace>...</reasoning_trace>. It must expose the
semantic rule path, not merely summarize the task:

- For every applied node, cite its exact graph ID, state the evidence or
  intermediate state entering that node, what the node did, and the new
  intermediate result.
- For every followed edge, cite its exact edge ID, name its endpoints,
  explain why the dependency applied, and state what it enabled.
- State the requested answer type, decisive evidence, relation or
  constraint, rejected endpoint when relevant, and final surface form.
- Do not cite an object that was not actually applied.

Output exactly these blocks in order:
<reasoning_trace>...</reasoning_trace>
<graph_usage>{"used_nodes":["..."],"used_edges":["..."]}</graph_usage>
<answer>...</answer>

graph_usage must be the exact deduplicated set of IDs explicitly cited
as applied in reasoning_trace. Use only IDs visible in the current graph.
Only the graph may evolve; do not memorize case IDs or gold answers.
```

D.4 Abductive Attribution Prompt

This prompt first assigns a verified failure to one root-cause category and then restricts any proposed change to the corresponding field or relation. The retained compatibility field `retrieval_revision_proposals` only revises `when_to_use` for an existing node; it neither invokes nor changes an external retriever.

Case analyzer: root cause and edit routing

You are the GraphOpt Case Analyzer. Read the frozen SkillGraph G_t and one complete case record, then perform semantic node- and edge-level attribution. Return exactly one parseable JSON object.

Evidence:
- Inspect the original task, response, evaluator outcome, semantic trace, graph_usage, and successful siblings together.
- A validated student trace is an execution record bound to exact graph IDs and explicit input--operation--output transitions.
- VERIFIED usage requires exact agreement between cited IDs, the sidecar, and valid graph endpoints.

First locate the earliest observable decision after which the answer is unrecoverable. Then choose exactly one root-cause route:

SUCCESS -> no edit.
MISSING_ACTIVATION_CUE -> clarify only target.when_to_use.
MISSING_OR_INCORRECT_PROCEDURE -> PATCH only target.how_to_use or add the necessary prereq/enhance relation.
HARMFUL_EXISTING_RULE -> replace only the exact harmful span, or atomically replace the incorrect edge.
MISSING_SKILL_FAMILY -> add one complete specialist node and an enhance edge from a trace-used correct parent in the same atomic group.
EXECUTION_LAPSE -> record the divergence and emit no graph proposal.
INSUFFICIENT_EVIDENCE -> emit no graph proposal.

Every proposed edit must implement the smallest reusable contrast with successful cases. Never copy a gold answer, case ID, document name, or hindsight-only condition into a skill. A missed node does not by itself justify broadening its trigger.

Required output fields:
case_id, success, failure_type, badcase_summary,
existing_graph_usage, node_revision_proposals,
retrieval_revision_proposals, edge_correction_proposals,
new_node_proposals, new_edge_proposals.

D.5 EPOCH-LEVEL META-AUDITOR PROMPT

The epoch-level meta-auditor summarizes lessons already supported by Gate evidence for use in later epochs. It may summarize the evidence but cannot directly modify the skill graph or its statistical relations.

Epoch meta auditor

```
You are the GraphOpt Epoch Meta Auditor.

Resolve every exact evidence reference through evidence_catalog verbatim
before auditing. The input contains previous_meta, previous and current
graph summaries, longitudinal improved/regressed/persistent-fail/stable-
success pairs, and complete Local- and Big-Gate records.

Use only this input. Read the complete Gate measurements, edit attribution,
original tasks, responses, trajectories, references, and before/after
states together. Independently verify that each lesson is supported by
the original evidence; do not rely on a decision summary when complete
evidence contradicts it.

Preserve every non-contradicted, evidence-grounded lesson in previous_meta,
then add or revise lessons supported by the current epoch. Cover effective
edits, regressions, likely missing skills, and changes to avoid. The
Permanent Protocol is frozen; statistics remain in evolution_cache;
successful cases cannot propose edits; this channel cannot propose
co_occur.

Return exactly:
{"schema_version":"graphopt-meta-v1",
 "bullets":["specific evidence-grounded point", "..."]}

Return 5--12 unique non-empty strings, each at most 500 characters. If
evidence is sparse, state uncertainty and the conservative action instead
of inventing a conclusion.
```

D.6 Patch Synthesis and Gate Audit Prompts

The first template adds one evidence-supported conditional step to an existing `how_to_use` field without rewriting the original field; the other routes use their corresponding field-specific templates. The second template explains the already determined Local-Gate result for audit and has no decision authority.

Targeted node-patch synthesizer

Synthesize the complete epoch evidence for one existing node into one precise conditional lesson. The program appends it to the Original Node; you are not allowed to rewrite the node.

Return exactly {"addition":"..."} or {"addition": null}.

- Inspect original_task, student_output, evaluation, training reference, reasoning evidence, graph_usage, proposed changes, and Successful Uses.
- Treat failures and successes as a paired counterfactual set. Identify the smallest observable state separating them.
- State observable condition -> next action / state update, with a narrow exception when needed. Preserve every byte of the old node.
- Do not create unconditional always/never rules from failure-only patterns. Keep the addition under 80 words.
- Return null if the proposal is already covered, targets another node, conflicts with a Successful Use, or lacks an observable condition.

Local-Gate explanation auditor

Explain one measured local joint graph edit from the supplied paired case summaries and expanded decisive / protected traces. The deterministic Gate decision is already final; do not change it.

Use only the supplied questions, reasoning traces, graph_usage, answers, and hard outcomes. Do not claim that one edit was independently effective when node/edge edits were intervened on jointly. Distinguish observations from hypotheses.

Return strict JSON with joint_summary and one entry per edit containing edit_index, effective_mechanism, ineffective_mechanism, observable_boundary, and evidence_case_ids.

E EXPERIMENTAL PROTOCOL

This appendix describes the evaluation settings behind Tables 2–5 and Figure 3: which examples may update the graph, which model conditions are held fixed, how scores are computed, and which comparisons each experimental block supports. Additional evaluations appear in Appendix C, ablation settings in Appendix F, and residual error analysis in Appendix G.

E.1 DATASET SPLITS AND GROUP CONSTRUCTION

Data splitting first ensures that updating and final evaluation remain independent: evidence from the update pool may change the graph, whereas held-out examples are used only for final scoring. Table 7 lists both protocols.

Protocol / benchmark	Train	Validation	Test
Main / SearchQA	800	0	200
Main / LiveMath	468	0	117
Main / DocVQA	800	0	200
SkillOpt / SearchQA	400	200	1,400
SkillOpt / LiveMath	35	18	124
SkillOpt / DocVQA	107	53	374

Table 7: All benchmark split counts. Main-protocol "Train" is the complete update pool.

For the main protocol, each fixed group contains four distinct update examples. Groups are built only from update-question text and task type: SearchQA uses BGE cosine similarity to form neighborhoods, DocVQA additionally requires compatible question types, and LiveMath first matches fine-grained mathematical categories before applying greedy TF–IDF matching. Test inputs, answers, and run results never enter grouping. In the SkillOpt protocol, train and validation form the update pool, while test remains held out. Every DocVQA setting receives the original image and question; an OCR summary never replaces the input.

E.2 MODEL ACCESS, INFERENCE, AND OPTIMIZATION SETTINGS

Models. The main protocol uses GPT-5.6-sol, Gemini-3.5-Flash, and Qwen3.8-Flash, with each model serving as both student and teacher. The SkillOpt-protocol comparison uses GPT-5.5, GPT-5.4-mini, and Qwen3.6-35B-A3B under the same arrangement; final held-out evaluation calls only the student. Table C is the sole exception: it holds the student fixed and replaces only the GPT-5.4-mini or GPT-5.6-sol teacher. The release will identify the model versions used in each run.

Inference. All models are called through APIs with `medium` reasoning effort. Student task rollouts use a 16,384-token output limit. Optimizer-side teacher calls use stage-specific limits: 768–4,096 tokens for attribution, audit, and refinement calls and up to 8,192 tokens for synthesis and merging; a generic ablation path may use 16,384. Qwen thinking is enabled for SearchQA and DocVQA and disabled for the LiveMath fast protocol. We record the model version, access method, decoding settings, timestamps, skill graph, and raw outputs so that the executed conditions remain checkable if a provider updates a hosted model.

Optimization. The three independent runs use seeds 42, 43, and 44. Every training intervention in Tables A–C runs for three epochs over the complete update pool; read-only retrieval interventions directly reuse an optimized graph without further updating. Gate scope and acceptance follow Section 3 and Appendix C.7.

Final graph size and computational cost. The final GPT-5.6-sol graphs contain 30/8, 37/40, and 20/19 nodes/edges for SearchQA, LiveMath, and DocVQA. Their rendered prompts contain 5,532, 7,549, and 3,866 tokens under the common offline `o200k_base` estimate. Reusing the current complete reference, each epoch adds $\sum_a |\mathrm{Retest}_a| + N$ student calls, where a ranges over actual Local-Gate attempts; a rejected refinable component can therefore contribute one additional retest before the single merged-graph evaluation on all N examples. The experiment record separately counts teacher calls for attribution, joint synthesis, rejected-patch refinement, Gate explanations,

and meta-audit. Because latency and monetary cost depend on concurrency and provider pricing, they are reported separately as run-level costs.

E.3 Evaluation Protocol

SearchQA and LiveMath use exact-match accuracy; DocVQA counts an answer as correct only when ANLS$= 1$. This binary view matches the $0 \to 1$ and $1 \to 0$ transitions used by the Gates, and no DocVQA column mixes in continuous ANLS. Each optimized run uses seed 42, 43, or 44 to produce a final committed graph and evaluates it with the same scorer, model access method, reasoning effort, and output budget; frozen settings use the same seeds only for repeated evaluation. We report the three-run mean and half-range, $(\max - \min)/2$. Held-out examples are neither run nor inspected during optimization.

E.4 Result Provenance and Comparison Protocol

The experimental blocks answer different questions and therefore require different matched references. Table 8 lists what is optimized, how the teacher is assigned, and which comparison is supported in each block. Table 3 follows the direct-chat setting of SkillOpt Table 1 (Yang et al., 2026): methods use the same released examples, test IDs, original inputs, inference budget, and scorer while retaining their own optimization procedures. GraphSkillAA updates only on train $\cup$ validation and is scored on the held-out test set.

Block	Optimized state	Teacher assignment	Comparison supported
Table A, frozen stages	None; the flat skill or G0 is fixed	No teacher call	Comparison of representation or execution-record requirements under matched test inputs.
Table A, GraphSkillAA	Three-epoch graph optimization	Student is its own teacher	End-to-end gain of the complete method; seeds 42/43/44 produce three committed graphs evaluated on held-out test.
Table B, comparison baselines	Method-specific states for No skill, Trace2Skill, TextGrad, GEPA, and SkillOpt	Settings reported for each method; no teacher call during final evaluation	Held-out accuracy comparison under the SkillOpt direct-chat setting.
Table B, GraphSkillAA	Three-epoch graph optimization on train $\cup$ validation	Student is its own teacher	GraphSkillAA evaluation under the same direct-chat setting and on the same held-out test set.
Table C, training interventions	Three-epoch graph optimization	Self-teacher unless the row names a teacher	Component effects are read against the same-student, same-teacher Full configuration; the named teacher rows form a separate teacher-transfer comparison.
Table C, retrieval interventions	Final graph from three-epoch self-teacher optimization; intervention only at evaluation	Student is its own teacher during optimization; no teacher call at evaluation	The source graph remains fixed and is compared with the self-teacher Full configuration for the same student.
Table C, teacher intervention	Three-epoch graph optimization	GPT-5.4-mini or GPT-5.6-sol as named	Teacher effect with student, data, edit rules, and epoch count held fixed.

Table 8: Configuration and comparison ledger for the reported experimental blocks.

Half-range cells summarize seeds 42, 43, and 44: graph-optimization conditions use three independent optimization-and-evaluation runs, whereas frozen conditions repeat evaluation only. Table 3 retains the point estimates reported for the baselines, while GraphSkillAA additionally reports the half-range over three held-out evaluations. We will release the data splits and fixed update groups so that every comparison can be reproduced without exposing held-out information to optimization.

F Ablation Protocols and Retrieval Definitions

Table 4 examines four aspects of the method: graph structure, retrieval, updating, and teacher assignment. Each intervention changes only one aspect while holding benchmark examples, original inputs, scorer, and student inference settings fixed. The following sections define the variants and their matched references.

F.1 Shared Settings and Graph-Structure Ablation

Baseline. Flat skill uses the original SkillOpt prompt without graph conversion or optimization and serves as the common non-graph reference in Table C. It is distinct from Structured skill G0 in Table A. The Table-C results come from new executions under the same split, inputs, scorer, inference settings, and three-run reporting rule; they are not copied from Table A.

Nodes only. Nodes only deletes every edge from the matched optimized graph without regenerating node fields. Comparison with Full isolates explicit relations; comparison with Flat skill includes both graph representation and optimized-node content.

F.2 Retrieval-Mechanism Ablations

Retrieval interventions change only what the student sees from the final graph at evaluation time; they perform no further attribution, patch synthesis, Gate validation, or teacher calls. Each student uses the graph obtained when that same model served as its teacher. Test examples, generation settings, and original inputs remain fixed, and DocVQA always receives the complete image and question.

No activation boundaries. The renderer omits only `when_to_use` and `avoid`. Node IDs, titles, `how_to_use`, and all edge fields remain unchanged and in the same order.

No edge rationale. The renderer omits only each edge's `rationale`. Endpoints, type, direction, strength, order, and all node fields remain unchanged.

S–Q retrieval. This variant follows the embedding-retrieval core of SkillRL (Xia et al., 2026). The fixed encoder `Qwen/Qwen3-Embedding-0.6B` embeds the current question q and node text s_j, formed by concatenating title, `how_to_use`, and `when_to_use`. Normalized inner product ranks nodes:

$$r_j(q) = f(q)^\top f(s_j), \qquad R_{\mathrm{SQ}}(q) = \mathrm{TopK}_j\, r_j(q), \quad K = 6.$$

The `avoid` field does not affect ranking but is rendered for selected nodes. Nodes outside the top six are omitted, and an edge is retained only when both endpoints are selected.

Nearest-Q reuse. For an update query $q^{(u)}_{m,j}$ in a fixed update quadruple $\mathrm{Group}_m = \{q^{(u)}_{m,1}, \ldots, q^{(u)}_{m,4}\}$, the variant selects the most similar one from the other three update questions:

$$i^* = \arg \max_{i \in \{1,2,3,4\} \setminus \{j\}} \mathrm{sim}_d(q^{(u)}_{m,j}, q^{(u)}_{m,i}).$$

It then renders the active terminal nodes whose accepted-patch provenance cites that neighboring update question:

$$R_{\mathrm{NQ}}(q^{(u)}_{m,j}) = A(q^{(u)}_{m,i^*}; G^*).$$

It neither computes question-to-skill similarity nor adds initial nodes or nodes from outside the quadruple. The selected set can therefore be empty or contain multiple nodes. This intervention measures within-quadruple case-to-skill reuse among update examples rather than graph semantic activation.

Retrieval records. To keep the comparison checkable, we record the nodes selected for each question, their similarity scores, patch provenance, prompt length, and test order. A run is excluded if node selection uses the observed prediction, provenance cannot be checked, the graph or generation settings change, or the original multimodal input is replaced.

F.3 Update-Mechanism Ablations

The four training interventions use the same data, student, self-teacher setting, and three-epoch schedule. *No structured attribution* replaces ordered localization and routing with a general failure summary. *No correct-case protection* removes successful examples as contrasts. *No Local Gate* keeps candidate edits but omits affected-case retesting. *No Big Gate* commits locally accepted edits without complete-update-pool validation. Every variant matches Full in epoch count and update-pool exposure, so these are one-factor-at-a-time ablations rather than a full factorial design.

F.4 Teacher-Transfer Evaluation

Teacher transfer changes only the teacher. The student, data, edit rules, three-epoch schedule, and evaluation remain fixed, and each final graph is evaluated three times. Assigning GPT-5.4-mini and GPT-5.6-sol as teachers for the two fixed students yields two self-teacher pairings, a smaller teacher guiding the stronger student, and a stronger teacher guiding the smaller student.

F.5 Comparison Protocol for Table C

Table C uses colored superscripts to report total gains over Flat skill, but component effects and teacher effects require different references. Table 9 explains how to read the two comparisons.

Rows interpreted	Student	Teacher	Matched reference
Graph, retrieval, and update variants	Either listed student	Self-teacher	The self-teacher Full row for the same student; retrieval variants are read against the same frozen source graph before the intervention.
Teacher-transfer evaluation	Either listed student	GPT-5.4-mini vs. GPT-5.6-sol	The two Full rows for that fixed student; every non-teacher setting is identical.

Table 9: References for interpreting Table C.

Component effects compare absolute scores with the self-teacher Full row for the same student; teacher effects compare the two Full rows after fixing the student. On SearchQA/LiveMath/DocVQA, the teacher differences are +3.3/+2.0/+1.2 for the GPT-5.6-sol student and +2.5/+5.2/+2.7 for the GPT-5.4-mini student. All training interventions run for three epochs and therefore use the update pool the same number of times.

G Residual Error Analysis

To analyze the remaining errors, we select two held-out failures with different causes from each benchmark. The purpose is not to estimate the prevalence of each category, but to determine the appropriate type of repair from the question, prediction, reference answer, and execution record.

SearchQA: scoring mismatch versus missing ordinal evidence

Correct entity, rejected surface form. For the clue about the leader convicted by Panamanian courts in 1993, the model answers *Manuel Noriega*, while the reference is *Noriega*. The execution record shows that the correct person and relation were selected, but strict exact match rejects the fuller name. This is an evaluation problem rather than a skill-graph problem; editing the skill would fit the scorer instead of improving reasoning. Alias-aware normalization or a separate semantic score is the appropriate remedy.

Missing ordinal evidence leads to the wrong person. Asked for the first U.S. Secretary of State, the model answers *James Madison* rather than *Thomas Jefferson*. The retrieved passages state that Madison served as Secretary of State under Jefferson but never identify who was first; Jefferson appears only as the president associated with Madison. The execution record therefore follows the clearest visible role statement and selects the wrong endpoint. This is primarily an evidence-coverage problem: retrieval or an answerability check must supply the missing ordinal fact, rather than a graph patch memorizing Jefferson from the label.

LiveMath: theorem identification versus realizability

Related theorem mistaken for the requested equivalence. The model chooses a co-t-structure characterization (E) instead of restriction of the induced t-structure to $\mathcal{T}_c^-$ (B). The execution record shows that the model checked the hypotheses and options but treated a related consequence as the equivalence itself. The procedure was followed; the theorem relation used by that procedure was wrong. This case calls for source-grounded theorem lookup, not another generic comparison rule.

Necessary conditions mistaken for realizability. The model selects C, which includes polygon type $\{6, 10\}$; the reference A permits only $\{4, 12\}$ and $\{8, 8\}$. The execution record shows that the model correctly derived that both side counts are even, sum to 16, and cannot equal 2, but then assumed that every remaining even partition is realizable. The missing step is a topological construction or obstruction, not another pass over the option wording. A source-backed classification or an explicit realizability check is therefore more appropriate than a generic comparison rule.

DocVQA: character recognition versus role selection

Correct region, wrong handwritten date. The model locates the date in the memo and reads `7/18/82`; the reference is `11/18/82`. All later extraction and formatting steps preserve the first reading, so the execution record is internally consistent but cannot detect that the initial visual observation was wrong. The appropriate response is a crop-level reread or OCR cross-check, not a graph patch.

Correct signature block, wrong endpoint. For the question *Who need to sign at bottom of page?*, the model returns the handwritten name *Lillian B Dalton*; the reference expects the role *Chief, Employment Branch*. The execution record shows that the model saw both strings but interpreted the question as asking who signed rather than which office should sign. Unlike the date error, this may reflect a defect in the role-selection boundary; however, the wording is ambiguous, so one example does not justify a persistent rule. A graph edit is warranted only if the same name-versus-role confusion recurs on clearer cases.

These cases point to four repair directions: scorer normalization, evidence retrieval, domain knowledge or visual checking, and skill-graph editing for recurring semantic-boundary errors. GRAPHSKILLAA already uses `NO_PATCH` to avoid writing the first three types into the graph. Future work can also record which external component should receive each non-graph error.

H DISCUSSION, LIMITATIONS, AND FUTURE WORK

H.1 INTERPRETATION OF EXPERIMENTAL FINDINGS

Taken together, the results suggest that the benefit of the skill graph does not come from the graph form alone. Its main value is that it organizes editable content into state that can be located precisely. Node fields separately describe applicability, execution, and exclusion boundaries, while typed edges coordinate related procedures. This explains why optimized nodes retain substantial value without edges, whereas hiding activation boundaries or using fixed retrieval causes larger losses.

The same node and edge identifiers connect failure analysis with behavioral validation: attribution first identifies a field or relation, the Local Gate measures the effect of that edit on related examples, and the Big Gate then evaluates the combined edits as a whole. The update ablations show that structured attribution and local retesting account for most of the repair benefit. The residual analysis also shows why theorem knowledge, visual perception, retrieval, and scoring failures should not be written into persistent graph edits.

ALFWorld demonstrates the preservation side of the method. When the initial graph already expresses the needed procedure, maintaining it through `NO_PATCH` is preferable to treating exploration or execution inefficiency as a reusable defect. GraphSkillAA's operative principle is therefore selective change: edit the smallest supported graph object, and otherwise preserve the validated state.

Claim–evidence alignment. Table 10 connects each principal conclusion to the experiment that directly tests it.

Claim	Direct evidence	Supported scope
End-to-end efficacy	Tables A and B across models and tasks	Highest observed mean in every reported main setting and all nine existing-method comparisons.
Structured node semantics	Flat-to-G0 progression, Nodes only, and activation-boundary intervention	Applicability, procedure, and exclusion fields jointly provide an effective skill representation.
Explicit relations	Full graph versus Nodes only, plus edge-rationale intervention	Typed edges add value when already-relevant procedures must be coordinated in the evaluated moderate-size graphs.
Attribution-guided routing	Generic failure-summary intervention, field-level routing checks, and paired Gate outcomes	Object-level routing improves repair effectiveness and confines each candidate to one clearly identified edit location.
Scoped validation	Local-/Big-Gate interventions and epoch commit trajectories	The Local Gate protects affected cases; the Big Gate enforces merged-graph commitment and exact restoration.
Semantic activation	Frozen-graph S–Q, Nearest-Q, field-hiding, and rationale-hiding interventions	Full-graph semantic activation is effective at the reported graph sizes; massive-library retrieval remains a separate scaling problem.

Table 10: Claim–evidence map for the principal conclusions.

H.2 Limitations

Statistical and evaluation scope. Table 3 fixes the released test IDs, inputs, inference budget, and scorer while retaining the optimization procedure of each method. The conclusions therefore apply to the protocol and benchmark splits used in this paper. Results from three seeds are summarized by a half-range, not a confidence interval, and strict-match scorers may reject aliases, formatting variants, or partial matches.

Scale and horizon. The experiments use moderate-size skill graphs whose contents can mostly be rendered in full, and the tasks are primarily single-turn. Libraries with thousands of skills or longer-horizon decisions will require explicit subgraph selection and better treatment of delayed credit. The Big Gate optimizes hard accuracy over the complete update pool; safety-critical settings should specify group constraints or risk weights before evaluation.

H.3 Near-Term Improvements

First, the robustness of the conclusions should be tested more thoroughly. Future work should use larger test sets, include distribution shifts, and add more independent runs. It should also report token consumption, concurrent requests, and response time under a common accounting scheme. Alias-aware or semantic-equivalence scoring could reduce errors caused by strict string matching.

Second, edit granularity can be made finer. The current system mainly changes a complete node field or a single edge; future versions could target one rule sentence, theorem statement, table cell, or image region. Whatever the granularity, before–after retesting and rollback should remain: the same examples are run around an edit, and the edit is not committed unless overall behavior improves.

Finally, the system should identify the type of failure before choosing a repair. Missing facts call for better retrieval, unreadable characters for OCR or a closer crop, persistent execution failures under a correct rule for model training, and inconsistent evaluation criteria for scorer revision. The skill graph should change only for recurring errors that can be clearly attributed to a skill rule or relation.

H.4 Long-Term Research Directions

Beyond GraphSkillAA, long-lived skill systems must solve four connected problems.

Scalable selection with semantic guarantees. Large libraries cannot be rendered in full, but similarity-only Top-K retrieval can omit prerequisites or retrieve a procedure outside its valid boundary. Graph-of-Skills, SkillDAG, HyperSkill, and CaSKG point toward dependency-aware and higher-order retrieval (Liu et al., 2026; Bai et al., 2026; Xu et al., 2026; Li et al., 2026c). The next step is to optimize selection, dependency completion, and context cost jointly, while measuring whether the retrieved subgraph contains every required step and excludes conflicting skills.

Interventional credit assignment. Counterfactual removal, substitution, and reordering can test whether a node or edge caused an outcome and separate individual effects from interactions (Li et al., 2026c; Jiang et al., 2026). Extending these interventions to long-horizon tasks would strengthen delayed credit assignment and rollback.

Lifecycle management rather than unchecked accumulation. Growing repositories must decide when to merge, split, specialize, retire, or restore skills. SkillOS, SkillsBench, and boundary-aware memory motivate versioned evidence, conflict tests, usage-aware pruning, and trust policies for imported skills rather than monotonic accumulation (Ouyang et al., 2026a; Li et al., 2026a; Lin et al., 2026; Xu & Yan, 2026).

Joint learning of models and external skills. External skills offer inspectability and rapid revision; model parameters offer compact execution and broad generalization. A promising closed loop keeps uncertain or changing procedures external, internalizes repeatedly validated behavior, and uses the improved model to generate and test subsequent skills without amplifying errors (Ni et al., 2026; Ouyang et al., 2026b; Zhang et al., 2026b;a).