GOVERNING AGENT SELF-EVOLUTION: FROM STRATEGY LEARNING TO COGNITIVE ORGANIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Improving agent capabilities typically relies on manually written strategies and iterative debugging. Research on recursive self-improvement (RSI) seeks to automate this process, enabling agents to modify themselves based on execution feedback and allowing the improved systems to participate in subsequent rounds of optimization. However, as strategies accumulate, content that is irrelevant to the current situation or conflicts with other strategy content may interfere with task-related judgments, a phenomenon we refer to as knowledge contamination. Growing strategy content also increases inference costs during execution. We propose GASE (Governing Agent Self-Evolution), a framework that integrates governance throughout strategy formation and evolution, enabling self-evolution to continually produce information that supports subsequent strategy optimization and cognitive migration. From the outset of learning, the framework organizes strategies into units that can evolve independently, each specifying how the agent should behave under particular conditions, and continually establishes links between strategies and execution experience. Through these links, accumulated experience can be reused in subsequent evolution to provide, at low cost, successful cases as references for strategy revision, as well as validation examples and training material for migrating local condition-evaluation tasks. Condition-evaluation tasks validated for independent execution are then assigned to specialized executors, forming detachable cognitive modules. These modules retain links to their source strategies and versions, allowing the system to revoke their assigned evaluation responsibilities when source conditions change and return the corresponding evaluations to the main LLM. When integrated with EvoSkill and SkillOpt, GASE improves the task performance of both methods on ALFWorld, OfficeQA, and SearchQA. On ALFWorld, the full framework improves task success rates over EvoSkill and SkillOpt alone by 7.96 and 7.46 percentage points, respectively, while reducing the average number of main LLM input tokens per task by 10.95% and 26.03%, respectively. Experiments across multiple models and component ablations on ALFWorld further show that revising individual strategy units and reusing associated experience help improve strategy learning.

1 INTRODUCTION

Improving agent capabilities typically relies on manually written strategies and iterative debugging. Research on self-evolution seeks to automatically revise strategies using execution feedback, enabling systems to learn from interaction and reduce repeated mistakes (Zhang et al., 2026; Ouyang et al., 2026; Yang et al., 2026; Wu et al., 2026); recursive self-improvement further examines how the improved systems participate in subsequent rounds of optimization. However, strategies that are irrelevant to the current situation or conflict with one another may interfere with judgments, while growing strategy content also increases inference costs. Even repeatedly used strategies still require fresh semantic judgments during execution. For example, after learning to "confirm that an operation has succeeded before proceeding to the next step," an agent still needs to interpret observations in new situations and determine the operation's outcome. Strategy improvements may therefore not consistently translate into more reliable and efficient task execution.

Existing methods address these issues from different perspectives. Retrieval and context compression control input size, but the retained strategies still require interpretation by the model, and further

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

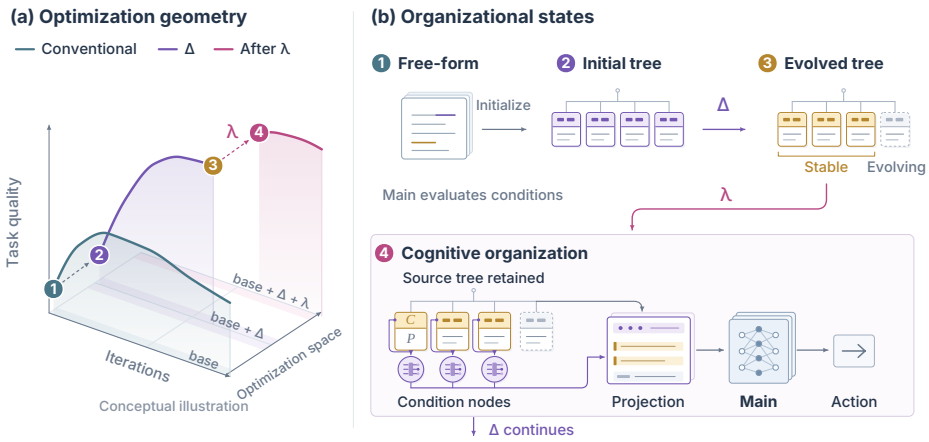

Figure 1: **GASE: from strategy learning to cognitive organization.** **(a)** Strategy evolution (Δ) revises action strategies, while cognitive migration (λ) adjusts the division of evaluation responsibilities. **(b)** Free-form strategies are gradually organized into a strategy tree, and some local condition-evaluation tasks are assigned to specialized nodes as experience accumulates. The nodes remain linked to their source strategies, and their division of work with the main LLM continues to adapt through strategy learning.

compression may lose necessary information and its relationships (Zheng et al., 2026; Wang et al., 2026; Li et al., 2026). Internalizing experience in model parameters can reduce reliance on external strategies, but training and evaluation costs limit the frequency of updates. The influence of outdated strategies does not automatically disappear when their source strategies are revised; correcting it still requires parameter updates and may interfere with other capabilities (Lu et al., 2026; Zhu et al., 2026). Program and tool generation make some behaviors directly executable, but reuse is constrained by task logic and environment interfaces, and adapting to new situations may still require additional semantic understanding or program modifications (Xu et al., 2026; Dantanarayana et al., 2026; El Yadouni and Li, 2026; El Hattami et al., 2026). Workflow optimization typically guides search using predefined module responsibilities or manually designed operational patterns (Shang et al., 2025). For continually changing strategies, how to incorporate experience and revise what has been learned at low cost while remaining adaptable to new situations requires further investigation.

To this end, we propose GASE (Governing Agent Self-Evolution), a unified governance framework that enables agents to improve their action strategies while gradually establishing and adjusting the division of work between the main LLM and specialized condition-evaluation nodes. From the outset of strategy learning, the framework establishes links between strategies and execution experience, enabling accumulated experience to be reused for strategy revision and to provide validation examples and training material for cognitive migration. Cognitive migration assigns local semantic-evaluation tasks validated for independent execution to the corresponding nodes, reducing the main LLM's repeated evaluation of these conditions. The main LLM integrates the nodes' results, continues to handle open-ended reasoning, and makes the final decisions. The nodes' specific responsibilities emerge through strategy learning, without requiring experts to be enumerated in advance or entire strategies to be hard-coded into programs tied to specific interfaces. Nodes remain linked to their source strategies. When strategies change, the corresponding evaluation responsibilities can be updated or revoked individually, without retraining the main LLM for a local correction. This division of work, which continually develops and adapts through experience, constitutes what we call an evolvable cognitive organization (Figure 1).

Our main contributions are as follows.

We propose GASE, a unified governance framework that integrates governance throughout strategy formation and evolution. From the outset of learning, the framework organizes strategies into units that can evolve independently and continually links them to execution experience. This

enables successful experience to be reused for strategy revision while providing validation examples and training material for cognitive migration.

We propose a cognitive migration mechanism driven by strategy learning, expanding the use of specialized executors in agent self-evolution by enabling them to perform local semantic-evaluation tasks that emerge as strategies evolve. In GASE, specific evaluation responsibilities emerge through strategy learning without requiring experts to be enumerated in advance. Tasks validated for independent execution are assigned to small non-generative models or other executors. After migration, condition-evaluation nodes retain links to their source strategies and versions, allowing the system to identify the evaluation responsibilities affected by source-condition revisions and revoke the corresponding assignments.

We demonstrate that GASE complements existing self-evolution methods. With GASE, both EvoSkill and SkillOpt improve task performance on all three benchmarks. On ALFWorld, success rates increase by 7.96 and 7.46 percentage points, respectively, while the average number of main LLM input tokens per task decreases by 10.95% and 26.03%, respectively. Experiments across multiple models and component ablations on ALFWorld further show that revising strategy units and reusing associated experience help improve strategy learning.

2 Related Work

Experience and Strategy Evolution. Execution feedback supports agents in continually revising prompts and skills. GEPA optimizes prompts using trajectory feedback, while ACE organizes experience into incrementally updatable strategy entries (Agrawal et al., 2026; Zhang et al., 2026b). SkillOpt selects skill modifications based on validation results, and EvoSkill discovers and improves skills through failure analysis (Yang et al., 2026; Alzubi et al., 2026).

We combine strategy evolution with experience organization: strategy units comprise applicability conditions and behavioral requirements and are continually linked to relevant execution experience. This allows the same experience to support both local strategy revisions and training and validation for the corresponding evaluation tasks, reducing the overhead of identifying these targets anew.

Context Management and Heterogeneous Execution. SkillRouter retrieves relevant skills, R3-Skill considers how multiple skills work together, and Sculptor manages the retention, compression, and removal of context (Zheng et al., 2026; Wang et al., 2026b; Li et al., 2026a). In terms of the division of work during execution, RouteLLM learns to route queries to different models, AgentSquare searches for and evolves agent modules, and SLM-MUX coordinates multiple small models for reasoning (Ong et al., 2025; Shang et al., 2025; Wang et al., 2026a).

We focus on how evaluation responsibilities emerge through strategy evolution. The need for specialization arises from strategy conditions that are repeatedly used and revised, without requiring cognitive experts or their responsibilities to be enumerated in advance. We use relevant experience and task-level validation to determine which evaluation tasks are suitable for specialized models, allowing the main LLM to use their results directly and reducing repeated interpretation and evaluation of the same kinds of conditions. Accumulated experience thus gradually shapes and adjusts the agent's internal cognitive organization.

Parametric and Programmatic Approaches. Internalizing experience in model parameters can reduce reliance on external strategies at runtime. Skill0 gradually removes skill guidance during training, while OPD-Evolver incorporates both the use and management of experience into model learning (Lu et al., 2026; Zhang et al., 2026a). Continued updates require balancing training and evaluation costs while preserving existing capabilities. SIGIL compiles programmable steps into code, while SKILL.nb selects steps to implement as programs based on execution evidence and supports repair and fallback. Both retain models for steps that require semantic understanding (Dantanarayana et al., 2026; Hattami et al., 2026).

We also internalize experience through training, but focus on local evaluation tasks within strategies that still require semantic understanding. We assign these tasks to executors outside the main LLM, allowing the learned capabilities to participate in execution as replaceable components while remaining linked to editable source strategies and relevant experience. When strategies change, these links

allow the corresponding execution capabilities to be located, updated, or disabled without repeatedly updating the main LLM's parameters.

3 GASE: GOVERNING AGENT SELF-EVOLUTION

3.1 FRAMEWORK OVERVIEW

A general-purpose model provides an agent's general cognitive capabilities, while the harness outside the model determines which knowledge informs action, under what circumstances, and in what way. Existing self-evolution methods can already accumulate new experience over time (Zhang et al., 2026b; Ouyang et al., 2026; Yang et al., 2026). However, when a strategy's applicability conditions and behavioral requirements are scattered across different locations, or when explicit links between the strategy and execution experience are missing, the system must identify the relevant knowledge again before revising the strategy, even if complete traces have been retained. As experience continues to accumulate, each revision requires the system to revisit more strategies and historical trajectories.

We believe that self-evolution can also serve as a means of data governance by establishing links between strategies and relevant experience as knowledge accumulates. To this end, GASE (Governing Agent Self-Evolution) treats strategies as the common objects of governance. It organizes each strategy's applicability conditions and behavioral requirements within a single strategy unit, and these units form a strategy tree. Execution experience and supporting evidence from relevant situations are linked to the corresponding units, while the system also records who performs the condition evaluations within them. As shown in Figure 1, strategy evolution Δ uses this experience to continually refine when and how the agent should act. Cognitive migration λ draws on the same accumulated experience through these strategy units to test whether local condition-evaluation tasks can be separated from the generative process of the main LLM. It then assigns tasks suitable for independent execution to corresponding executors, forming condition-evaluation nodes. In this work, we implement these nodes using small non-generative models. Strategy content and organizational relationships can change, while source identities and version links are retained, allowing execution experience after migration to remain traceable to the source strategies and support subsequent learning. Through this process, the agent gradually develops its own cognitive organization. The main LLM continues to handle open-ended reasoning, integration across strategies, and final actions, while local condition-evaluation tasks that recur in accumulated experience and gradually mature are assigned to more specialized condition-evaluation nodes.

3.2 STRATEGY REPRESENTATION

To make strategies a shared point of reference throughout the lifecycle, the strategies used at runtime must correspond to the objects of subsequent attribution and revision. These links must persist after the condition-evaluation tasks within them are migrated. Agent Skills provide a starting point for this representation: they store reusable task knowledge in natural language, allowing the system to continually revise strategies while keeping the agent's parameters frozen (Yang et al., 2026). As skills expand, the main LLM must read and interpret more content at each step, so the currently needed portions can be selected before each invocation. However, a skill often contains strategies for multiple situations. Retrieving the entire document does not determine which strategy should apply, while splitting it by text position may separate applicability conditions from their corresponding behaviors.

We organize a local strategy within a skill as a strategy unit $U_i = (C_i, P_i)$, where C_i is the complete applicability condition and P_i specifies, in natural language, the action requirements or constraints to follow when that condition holds. A local condition c is extracted from C_i according to its original logical structure and defines a complete evaluation task. Cognitive migration λ transfers responsibility for evaluating c while preserving the compositional semantics of the source condition and the corresponding P_i. Each unit retains a stable identity and records its content version. The units and their organizational relationships form a strategy tree that preserves the complete source strategies. A skill containing only one local decision that can be revised independently can map directly to a single unit. A skill covering an entire standard operating procedure (SOP) remains an outer container, with multiple units expressing strategies for different states within it.

As shown in Figure 2, SUs provide explicit points of reference for model reasoning and action selection, as well as a basis for linking execution experience to source strategies.

At runtime, the system substitutes available determinate evaluation results into the source conditions and simplifies the current strategy input according to the original logic, yielding a strategy projection. If the complete condition is false, the unit is excluded; if it is true, only P_i is retained; otherwise, the unresolved conditions are passed to the main LLM together with P_i. Projection changes only the current input; the complete source strategy tree is always retained. Appendix A.1 provides a projection example in a price-inquiry task.

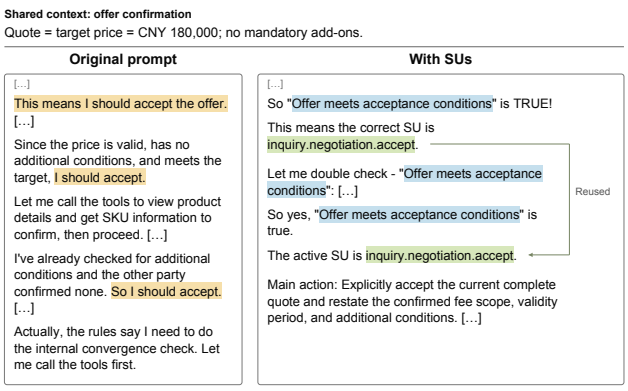

Figure 2: Strategy attribution in an offer-confirmation case. Model-output excerpts with the original prompt (left) and SU-based representation (right).

This representation supports subsequent learning in three ways. (1) Experience attribution can use a unit's conditions and behavioral requirements to link states and actions in a trace to specific strategies and versions. (2) These links provide a basis for data construction: Δ can retrieve experience from successful executions through the target unit to support revision, while λ can locate relevant tasks and evaluation time points, construct a migration trial set, and extract the states needed for annotation and training. (3) Local conditions provide interfaces for evaluation tasks that can be migrated independently. The source strategy specifies their meaning and the relationship between evaluation results and behavior. After migration, nodes retain links to their source strategies and versions, allowing the system to review or revoke the corresponding responsibilities when source conditions change.

3.3 Δ: Strategy Evolution

Section 3.2 organizes local strategies into strategy units and links execution records back to the corresponding units. Strategy evolution Δ proceeds through these links. We instantiate this process with EvoSkill (Alzubi et al., 2026) and SkillOpt (Yang et al., 2026), using a successful experience library to supply relevant experience from successful executions for revision while retaining each method's search procedure.

When a task execution completes successfully, the experience attribution module examines key decisions against the complete trace and the strategy tree used during execution. It links the states at those points and the actions actually taken to the strategy units involved in the decisions. The index records the relevant trajectory segments and source strategy versions, with references back to the original trace. A task execution may be linked to multiple strategy units; together, these indices form the successful experience library. Subsequent retrieval can identify situations in which a strategy contributed to correct actions. Even tasks that appear different may provide relevant experience because they used the same strategy unit.

For revision driven by failure cases, the system uses the failed trace to locate strategy units that may need revision and retrieves relevant successful cases based on the failure scenario or the proposed change. Failure cases identify situations that need correction, while successful cases show how similar situations were previously handled correctly, helping determine where to revise and which behaviors to preserve. The optimizer uses this information to generate or select candidate changes. It can modify an existing unit's applicability condition or a class of behavioral requirements, or add new units under existing branches while keeping the branching structure unchanged. Candidate changes are evaluated on Selection and selected using each optimizer's existing procedure. Prompt templates for experience attribution and strategy revision are provided in Appendix A.2.

After a strategy update, the old links for modified units are invalidated so that executions before the change are not directly treated as successful experience for the new version. The original traces and their source versions are retained, and links to unmodified units remain usable. Subsequent successful executions accumulate experience for the new version. As strategies are repeatedly used and revised, local conditions become clearer and related cases accumulate through the same strategy units. These provide a basis for cognitive migration λ to select candidate evaluation tasks, construct migration trial sets, and locate the execution states needed for training.

3.4 λ: COGNITIVE MIGRATION

Cognitive migration λ uses the strategies formed through Δ and their associated experience to first test whether local condition-evaluation tasks can be performed independently, and then train small non-generative models to perform them. The system extracts candidate conditions c from a fixed version of the strategy tree according to the source logic, retrieves relevant successful cases through the corresponding strategy units, and constructs a migration trial set from tasks involving those conditions. During screening, an independent evaluator that uses the same base model as the main LLM evaluates each condition using its description and the information available at that time; the main LLM then continues acting. The system compares the end-to-end task performance of this workflow with that of the original workflow under the same strategy and task settings, and uses this comparison to select the conditions for training.

Training examples are also constructed from this strategy-linked experience. For each selected condition, the system identifies the points in the relevant traces at which the condition was involved in decisions. It extracts the inputs available at those points and pairs them with the condition description to form evaluation examples. The teacher uses this information to provide judgments and supporting evidence, yielding the training set for that condition. We first train a small non-generative student on these examples, then allow the calibrated student to participate in Train tasks while the main LLM continues acting. The teacher provides additional annotations for newly reached states under this joint workflow. The resulting examples are combined with the existing ones for further training, forming on-policy distillation (OPD) for condition evaluation. Experience accumulated during strategy evolution thus helps determine what the student needs to learn and supplies the corresponding training material.

The student then replaces the independent evaluator and undergoes task-level validation for individual conditions and combinations of conditions. The system establishes condition-evaluation nodes executed by the student only if the student has actually performed condition evaluations, the task success rate does not decrease, and token consumption does not increase. At runtime, reliable node outputs that pass the handover checks are used to simplify the current strategy input and are supplied together with the strategies to the main LLM, which continues reasoning and acting. When a node cannot provide a reliable judgment, the main LLM still evaluates the corresponding condition.

Condition-evaluation nodes remain linked to their source strategy units and condition versions. Execution experience after migration continues to be linked back to the corresponding strategies for subsequent revision and training. When source conditions or their dependencies change, the system revokes the affected nodes' responsibility for condition evaluation and temporarily returns it to the main LLM. The nodes resume these responsibilities after the student completes the necessary training for the new version and passes validation again. The resulting division of condition-evaluation responsibilities can thus continue to adapt as strategies are learned. Appendix A.1 provides the detailed procedures for screening, data construction, training, and maintenance.

4 EXPERIMENTS

EXPERIMENTAL SETUP

We evaluate GASE on interactive and question-answering tasks to examine the role of explicit strategies in strategy revision and the division of work during execution. ALFWorld (Shridhar et al., 2021) is our primary evaluation platform: its multistep household tasks are suitable for studying strategy revision based on execution feedback and strategy use as task states change. We further extend the evaluation to document and retrieval-based question answering through OfficeQA (Opsahl-Ong et al., 2026) and SearchQA (Dunn et al., 2017).

Table 1: Task performance and normalized input tokens when GASE is integrated with EvoSkill and SkillOpt. ALFWorld reports task success rate, while OfficeQA and SearchQA report EM, all expressed as percentages. Input token counts are averaged per task for the main LLM and normalized to No-skill = 100 for each benchmark.

Method	Δ	λ	ALFWorld		OfficeQA		SearchQA	
			Success ↑	Input tokens ↓	EM ↑	Input tokens ↓	EM ↑	Input tokens ↓
No-skill	×	×	65.67	100.00	19.77	100.00	72.24	100.00
	×	×	78.36	273.61	30.23	116.64	77.00	282.94
EvoSkill	✓	×	84.08	249.36	38.37	127.89	78.17	160.47
	✓	✓	86.32	243.66	40.12	125.35	79.71	144.10
	×	×	79.85	315.64	30.81	97.23	77.33	152.26
SkillOpt	✓	×	83.83	259.46	36.05	137.15	78.33	147.65
	✓	✓	87.31	233.48	37.79	129.79	78.60	144.17

We integrate GASE with EvoSkill (Alzubi et al., 2026) and SkillOpt (Yang et al., 2026) to examine its applicability across different strategy-learning workflows. EvoSkill generates or revises skills based on execution failures and maintains a candidate set according to validation performance. SkillOpt edits skill documents using successful and failed trajectories, accepting updates that improve validation performance within the editing budget.

For both baselines, we compare the original method, $+\Delta$, and $+\Delta+\lambda$. The $+\Delta$ configuration uses strategy units and associated successful experience to support strategy revision; $+\Delta+\lambda$ further uses ModernBERT to perform selected local condition-evaluation tasks and uses their results to construct the strategy input for the main LLM. We also report No-skill results, in which no strategies are loaded. The main experiments use GPT-5.4 Mini, while the cross-model experiments compare the original method with $+\Delta$ for each model. Our analysis experiments focus on ALFWorld, examining the effect of non-target strategies on action selection, the execution performance of the initial strategy tree, strategy evolution across models, and component contributions.

Each comparison uses the same main LLM, task set, and environment. For main experiments and component ablations, we report means over three independent runs. We report task success rate for ALFWorld and exact match (EM) for both OfficeQA and SearchQA. We report normalized input tokens: the average number of main LLM input tokens per task, normalized to No-skill = 100 for each benchmark in Table 1. Benchmark and strategy-context pruning setups are provided in Appendices B.4 and B.1, respectively.

4.1 Overall Performance

Table 1 shows that GASE improves the task performance of EvoSkill and SkillOpt on ALFWorld, OfficeQA, and SearchQA, with most configurations also reducing main LLM input token usage. On ALFWorld, the full framework improves task success rate over the two original methods by 7.96 and 7.46 percentage points (pp), respectively. Question-answering performance also improves on OfficeQA and SearchQA, indicating that GASE can strengthen existing strategy-learning methods across multistep interaction, document question answering, and retrieval-based question answering.

The configuration comparisons show that Δ improves task performance for both baselines on all three benchmarks. Adding λ further improves task performance and reduces the average number of main LLM input tokens per task in all six configurations. The results from the two stages show how strategy improvement connects to the subsequent division of evaluation responsibilities: learned strategies and associated experience continue to support the migration of local condition-evaluation tasks, further improving strategy effectiveness and execution efficiency.

4.2 Analysis of GASE

Effect of non-target skills on skill selection. Figure 3 reports strategy-context pruning results on the LogicBench subset of the public SRA-Bench (Su et al., 2026). We sample 10 questions for each of 19 skills, yielding 190 questions, and progressively remove the other 18 skills while retaining the target skill. Skill-selection accuracy increases at each pruning level for all seven main LLMs. After

Table 2: ALFWorld task success rates (%) across main LLMs.

Main LLM	No-skill	EvoSkill	EvoSkill + Δ	SkillOpt	SkillOpt + Δ
GPT-5.4 Mini	65.67	78.36	84.08	79.85	83.83
GPT-5	70.15	85.07	87.31	86.57	87.31
GPT-5 Mini	49.25	62.69	80.60	70.90	79.85
DeepSeek-V4 Flash	67.16	75.12	82.34	77.86	82.09
gemini-2.5-pro	74.63	82.84	87.56	86.57	89.80
Kimi K2.5	76.12	88.06	88.81	91.79	88.06
Qwen3.6-35B-A3B	68.66	86.57	85.07	83.58	88.81

approximately 90% of non-target skills are removed, mean accuracy rises from 82.3% to 97.2%, an average gain of 14.9 pp, with per-model gains of 10.5–23.2 pp (Appendix B.1). These results show that reducing non-target skills can improve skill selection when the target skill is retained.

Figure 3 also highlights the substantial potential of selective strategy input. We take an initial step toward realizing this potential by using ModernBERT to evaluate selected local conditions; Table 1 reports the corresponding task improvements. Broader potential remains untapped: strategy learning continually clarifies local conditions and accumulates relevant experience, providing a basis for validating and migrating further condition-evaluation tasks. Small LLMs, conventional machine-learning models, and backend programs can also perform evaluations suited to them, expanding the scope for organizing strategy input according to the current situation.

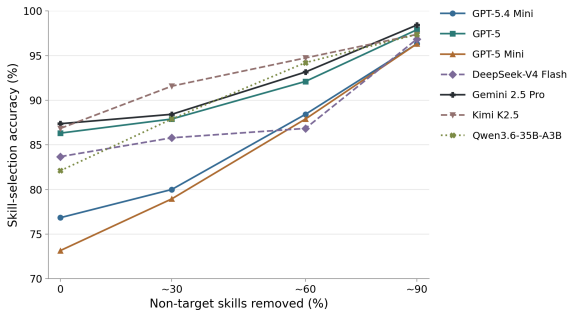

Figure 3: Skill-selection accuracy (%) under non-target skill pruning on SRA-Bench / LogicBench (190 questions per model per condition; Appendix B.1).

Explicit strategy representation and strategy evolution. We compare the unevolved initial strategy tree (Initial) with No-skill, which loads no strategies, on ALFWorld to examine whether explicit strategies provide an effective starting point for execution. The original methods and GASE start from initial strategies with equivalent information content. The initial strategy tree is generated entirely by Codex using GPT-5.6 Sol, without human involvement. Initial achieves higher task success rates than No-skill for all six main LLMs, with gains of 7.47–23.88 pp (Appendix B.2). These results show that the initial strategy tree can guide task execution before evolution begins, supporting subsequent strategy learning from this representation.

For subsequent strategy evolution, we compare EvoSkill, SkillOpt, and their respective +Δ configurations across seven main LLMs using the same set of 134 ALFWorld test tasks. Averaged equally across models, Δ improves success rates over the two original methods by 5.29 and 3.23 pp, respectively (Table 2). These results support the applicability of revisions based on strategy units and associated experience across both strategy-learning methods and different main LLMs.

Strategy evolution can also continue after cognitive migration. In an EvoSkill +Δ+λ case on ALFWorld, subsequent Δ optimization improves the success rate from 82.09% to 83.58%. Appendix B.3 illustrates the adjustment of responsibilities following one source-condition revision: the system invalidates the binding to the previous condition version and returns the corresponding evaluation task to the main LLM, while retaining other valid bindings. This case shows that source strategies can continue to be revised after a division of evaluation responsibilities has been established, and that affected assignments can be revoked accordingly.

Component contributions to strategy evolution. We compare three EvoSkill configurations: the original method, strategy-tree initialization with Strategy Unit (SU) revisions, and the same con-

figuration augmented with the successful experience library (Table 3). The SU configuration outperforms the original method on all five models, increasing the average success rate by 2.69 pp. Adding the experience library brings the average gain over the original method to 3.73 pp. These results support the effectiveness of organizing and revising strategies through strategy units, with associated successful experience further improving overall performance.

Table 3: Cross-model component ablation for EvoSkill on ALFWorld. SU: Strategy Unit; GCL: Good Case Library. Values are task success rates (%). The ablation study was conducted independently of the experiments reported in Table 2.

SU	GCL	GPT-5.4 Mini	GPT-5	GPT-5.6 Sol	DeepSeek V4 Flash	Gemini 3 Pro Preview
×	×	78.36	85.07	94.03	77.61	82.84
✓	×	81.34	89.55	95.52	78.36	86.57
✓	✓	85.07	87.31	96.27	79.85	88.06

5 Discussion and Conclusion

GASE incorporates governance throughout strategy learning, allowing accumulated experience to continue supporting strategy revision and cognitive migration. This provides a path for agents to form cognitive organizations through experience. Non-generative evaluation offers one way to implement this division of labor. Determining whether a strategy condition holds may require complex understanding, but a condition-evaluation node only needs to return a small set of state signals for subsequent execution. INSPECTOR shows that the internal representations of small models can support effective evaluation even when their generation performance is limited, while Jev demonstrates a practical approach to controlling agent execution with non-generative judgments. GASE further uses strategy learning to identify the conditions that need to be evaluated. It uses relevant experience to validate and train condition-evaluation nodes and maintains their links to source strategies, allowing the nodes to be updated as the strategies are revised.

This division of labor also suggests an approach to harness scaling: inspired by collaboration among photoreceptor cells, visual processing structures, and the brain, it uses accumulated experience to gradually develop nodes that can independently evaluate local conditions and to adjust their responsibilities. Future work could extend this approach to memory organization, allowing frequently used experience to support rapid responses while revising its applicability conditions as new experience accumulates. What an agent learns from experience may thus reshape the division of work involved in perceiving and responding to its environment.

Limitations

In studying cognitive migration, we examine how strategy learning provides local condition-evaluation tasks and their training material, and how migrated evaluation responsibilities remain linked to source strategies. We evaluate the mechanism's end-to-end effects using ModernBERT and the current OPD workflow, and use a case study to illustrate the revocation of evaluation responsibilities following source-condition revisions. Comparisons across executors and training procedures, as well as long-term performance under repeated migration and strategy revision, remain to be systematically studied. Small LLMs, conventional machine-learning models, and backend programs can also perform these evaluations. We have verified the effectiveness of the latter two in practical applications, but the corresponding implementations have not been open-sourced due to business constraints. The benefits of migration also depend on how strategies are divided and used in a given setting: when strategies are divided into fine-grained units and only a small amount of content needs to be activated at each step, runtime projection can reduce the strategy content that the main LLM needs to read while preserving the complete source strategies. Selecting suitable executors and improving training procedures for different settings require further study.

AI USE STATEMENT

The authors conceived the ideas and designed the methods for this study. We used generative AI tools to obtain critical feedback on the proposed ideas and methods, iteratively revise the manuscript based on an initial draft provided by the authors, and translate the Chinese manuscript into English paragraph by paragraph. Building on the initial experimental framework developed by the authors, we used AI-assisted programming to implement and iterate on the experimental code. AI tools were also used to assist with literature searches and refine figures based on sketches drawn by the authors. During the writing process, the authors repeatedly reviewed and revised the manuscript and consulted the original sources identified through the literature searches. The authors themselves made the final research and editorial decisions.

REPRODUCIBILITY STATEMENT

Section 3 describes the GASE framework, including strategy evolution and cognitive migration. Appendix A provides further details on condition selection, student training, calibration, and runtime strategy projection, together with the prompt templates. The experimental setup and evaluation metrics are described in Section 4, with additional experimental settings and results in Appendix B. We provide the implementation, experiment configurations, and training and evaluation scripts in the supplementary material; the accompanying README describes dependency installation, input preparation, model-service configuration, and how to run the experiments. Further details on the strategies learned through Δ and the transfer of condition-evaluation responsibilities through λ are provided with the supplementary code.

REFERENCES

Lakshya A. Agrawal et al. GEPA: Reflective Prompt Evolution Can Outperform Reinforcement Learning. In *ICLR*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0e9e708b6f48e14fd0ac29e167413f76-Abstract-Conference.html.

Salaheddin Alzubi et al. EvoSkill: Automated Skill Discovery for Multi-Agent Systems. In *COLM*, 2026. URL https://arxiv.org/abs/2603.02766.

Jayanaka Dantanarayana et al. SIGIL: Compiling Agent Skills into Typed Harnesses. arXiv, 2026. URL https://arxiv.org/abs/2607.27309.

Matthew Dunn et al. SearchQA: A New Q&A Dataset Augmented with Context from a Search Engine. arXiv, 2017. URL https://arxiv.org/abs/1704.05179.

Amine El Hattami et al. SKILL.nb: Selective Formalization and Gated Execution for Durable Agent Workflows. arXiv, 2026. URL https://arxiv.org/abs/2606.08049.

Mo Li et al. Sculptor: Empowering LLMs with Cognitive Agency via Active Context Management. In *ICLR*, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/f9155a06ba1b20e40c38bf8541db456f-Abstract-Conference.html.

Zhuochun Li et al. Rethinking LLM-as-a-Judge: Representation-as-a-Judge with Small Language Models via Semantic Capacity Asymmetry. In *ICLR*, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/7d9173a6010bed38c359fa012becd4da-Abstract-Conference.html.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *ICLR*, 2019. URL https://arxiv.org/abs/1711.05101.

Zhengxi Lu et al. SKILL0: In-Context Agentic Reinforcement Learning for Skill Internalization. arXiv, 2026. URL https://arxiv.org/abs/2604.02268.

Isaac Ong et al. RouteLLM: Learning to Route LLMs from Preference Data. In *ICLR*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5503a7c69d48a2f86fc00b3dc09de686-Abstract-Conference.html.

Krista Opsahl-Ong et al. OfficeQA Pro: An Enterprise Benchmark for End-to-End Grounded Reasoning. arXiv, 2026. URL https://arxiv.org/abs/2603.08655.

Siru Ouyang et al. ReasoningBank: Scaling Agent Self-Evolving with Reasoning Memory. In *ICLR*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/980ea04d23d1f6908964eba2a74afe45-Abstract-Conference.html.

Yu Shang et al. AgentSquare: Automatic LLM Agent Search in Modular Design Space. In *ICLR*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/0ae94013da7cd459402fd77874e09ee3-Abstract-Conference.html.

Mohit Shridhar et al. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *ICLR*, 2021. URL https://arxiv.org/abs/2010.03768.

Weihang Su et al. Skill Retrieval Augmentation for Agentic AI. arXiv, 2026. URL https://arxiv.org/abs/2604.24594.

Chenyu Wang et al. SLM-MUX: Orchestrating Small Language Models for Reasoning. In *ICLR*, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/77b874452cdb52ea04c15b33122969e9-Abstract-Conference.html.

Zifei Wang et al. Skill Is Not Document: Query-Conditioned Compatibility for LLM Agent Skill Routing. arXiv, 2026b. URL https://arxiv.org/abs/2606.03565.

Benjamin Warner et al. Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference. arXiv, 2024. URL https://arxiv.org/abs/2412.13663.

Wenhao Wu et al. TRACE: A Self-Evolving Skill Bank for Consistent, Limit-Aware LLM Agents. arXiv, 2026. URL https://arxiv.org/abs/2608.22793.

Duling Xu et al. SkillSmith: Compiling Agent Skills into Boundary-Guided Runtime Interfaces. arXiv, 2026. URL https://arxiv.org/abs/2605.15215.

Salma El Yadouni and Guanyi Li. TraceCompiler: Skill-Guided Mining and Compilation of LLM Agent Traces into Mostly Deterministic Workflows. arXiv, 2026. URL https://arxiv.org/abs/2608.02680.

Yifan Yang et al. SkillOpt: Executive Strategy for Self-Evolving Agent Skills. arXiv, 2026. URL https://arxiv.org/abs/2605.23904.

Guibin Zhang et al. OPD-Evolver: Cultivating Holistic Agent Evolver via On-Policy Distillation. arXiv, 2026a. URL https://arxiv.org/abs/2606.17628.

Qizheng Zhang et al. Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models. In *ICLR*, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8a94ff6f922d995d7d3f4ebf4143e442-Abstract-Conference.html.

YanZhao Zheng et al. SkillRouter: Skill Routing for LLM Agents at Scale. arXiv, 2026. URL https://arxiv.org/abs/2603.22455.

Jiapeng Zhu et al. Skill0.5: Joint Skill Internalization and Utilization for Out-of-Distribution Generalization in Agentic Reinforcement Learning. arXiv, 2026. URL https://arxiv.org/abs/2605.28424.

Adam Zweiger et al. Self-Adapting Language Models. In *NeurIPS*, 2025. URL https://papers.neurips.cc/paper_files/paper/2025/hash/6b41e04c41726e2a60e456d0a2b961ab-Abstract-Conference.html.

A Implementation Details

A.1 Strategy Representation, Evolution, and Cognitive Migration

GASE establishes and preserves links between strategies and execution experience during strategy learning. Strategy units specify how to act under particular conditions, while the associated traces record how these strategies are used in specific situations. Strategy evolution draws on this experience to revise strategies; cognitive migration uses the same units to identify evaluation tasks to be learned independently and construct the corresponding training examples. After these tasks are migrated, new execution experience remains linked to the source strategies and continues to support revision and training. This section elaborates on this process, explaining how the representation, data construction, and division of work during execution described in Sections 3.2–3.4 fit together.

A.1.1 Strategy Representation and Experience Attribution

A strategy unit $U_i = (C_i, P_i)$ brings together the full applicability condition C_i of a local strategy and its natural-language behavioral requirements P_i. A local decision that can be modified independently can directly form a single unit. A skill covering an entire SOP is retained as an outer wrapper, with multiple units inside it expressing strategies for different states. These units and their organizational relationships form a strategy tree. This allows the system to locate the corresponding condition and the behavior it constrains whenever an evaluation task within a strategy is referenced, modified, or migrated.

Each unit retains a stable identity and records its content version. The identity allows the system to locate the same strategy during subsequent learning, while the version specifies which content was used in a particular execution. After a task succeeds, the experience attribution module examines the complete trace alongside the strategy tree used during execution. Based on the conditions and behavioral requirements in the units, it identifies the strategies involved in key decisions, links the states, actual actions, and relevant trajectory segments at those points to the corresponding units and versions, and retains references to the original trace. Unit identifiers serve as references; attribution is based on strategy content and visible execution records.

A task can be linked to multiple strategy units, and tasks that differ on the surface may provide relevant experience because they use the same strategy. These links form the successful experience library, allowing subsequent learning to retrieve relevant cases for a target strategy and then return to the traces to examine the specific moments of condition evaluation. Condition-evaluation nodes also retain links to their source units and condition versions after migration, so experience can still be attributed to the same strategy when the evaluator changes.

A.1.2 Strategy Evolution with Successful Experience

We instantiate strategy evolution Δ in EvoSkill and SkillOpt separately, preserving their respective search procedures. During revisions driven by failure cases, the system examines failure traces to locate units that may need modification and retrieves relevant successful cases using the failure scenario or proposed modification. Failure cases identify situations that require correction, while successful cases show how similar situations were handled correctly, helping the optimizer determine where to make changes and which behaviors to preserve.

For example, when the system mistakes "the request has been sent" for "the operation has completed," it can retrieve successful trajectories in which the same strategy was used in both "still processing" and "completion confirmed" states, and examine the evidence used to determine whether the operation had completed. These cases provide references for revising applicability conditions and help preserve the original strategy's correct behavior in other states.

The optimizer uses this information to generate or select candidate modifications. Both implementations can modify an existing unit's applicability condition or one category of its behavioral requirements, or create a new unit under an existing branch, while keeping the branch structure unchanged. Candidates follow their respective optimizers' evaluation on Selection and strategy-selection procedures, and candidate modifications are retained together with their evaluation results. Prompt templates for experience attribution, EvoSkill strategy revision, and SkillOpt candidate ranking are provided in Appendix A.2.

After a strategy update, the old links associated with modified units are invalidated because these cases record executions before the modification and cannot be used directly as successful experience for the new version. The original traces and their source versions are retained, and links between the same cases and other, unchanged units remain usable. Subsequent successful executions accumulate experience for the new version. As strategies are repeatedly used and revised, local conditions become clearer and relevant cases accumulate through the same units, providing candidate evaluation tasks for cognitive migration, tasks for migration trials, and execution states for training.

A.1.3 CONDITION SELECTION

Migration screening starts from a fixed version of the strategy tree and its associated experience. The system first tests independent evaluation of the full applicability condition C_i; if this does not pass, it extracts local conditions c according to the original logic. If the outermost operator is conjunction, each conjunct becomes a candidate, with any disjunctions and negations within it preserved. If the outermost operator is disjunction or negation, the condition is not split further. Always-true conditions need not be migrated, and conditions that cannot be parsed reliably remain for the main LLM to evaluate. Each candidate retains its meaning within the source condition and the relationship between its evaluation result and P_i.

Screening first tests whether this division of condition-evaluation responsibilities is suitable for task execution, after which a student is trained on the selected conditions. Training a student on all candidates beforehand would also incur training costs for conditions that are ultimately not selected. We therefore invoke an independent condition evaluator based on the same underlying model as the main LLM. It evaluates conditions using only their descriptions and the input visible at the time, while the main LLM continues to act. This trial requires no prior student training.

The system retrieves successful cases through the strategy unit containing each candidate condition, examines the source strategy and traces to locate situations in which the condition actually informed a decision, and selects the corresponding tasks to construct a migration trial set. It first selects states from Train in which the condition is relevant to the decision and the visible evidence is sufficient to determine the result, and evaluates the condition independently three times in each state. Full-task trials begin only after all three valid results agree with the evidence-based label.

The task trials compare the original workflow with a workflow that incorporates the independent evaluator. Both use the same source strategies, tasks, and environment settings, and each completes the tasks based on its own interaction feedback. The trial workflow uses determinate outputs from the independent evaluator to simplify the current strategy input and passes these outputs to the main LLM. When the evidence is insufficient, the corresponding condition remains for the main LLM to evaluate. Screening is based on complete-task performance, prioritizing the number of successful tasks and then the main LLM's token consumption.

After screening candidates individually, the system searches for combinations of candidate evaluation tasks within the budget and evaluates their joint effect on execution over the complete Train and Selection sets. The search ends when a combination does not reduce task success rate and does not increase main LLM token consumption relative to the original workflow, in which the main LLM performs all evaluations. Otherwise, the search continues; when the budget is exhausted, the best fully evaluated combination is retained. The selected combination then proceeds to student training, while eligibility for handover remains subject to task-level validation of the student.

```
Input:  Fixed strategy tree, successful experience library,
Train, Selection, screening budget
Candidate evaluation tasks <- empty
For each strategy unit U_i = (C_i, P_i):
   Test independent evaluation of the full applicability
condition C_i
   If the three independent evaluations and the full-task
trial all pass:
      Add evaluation of C_i to the candidates
   Otherwise:
      Extract local conditions c that can be separated
according to the original logic
```

```
      Repeat the independent evaluations and full-task trial
for each c
      Add passing local evaluation tasks to the candidates;
leave the rest to the main LLM
Search for combinations of candidate evaluation tasks within
the budget:
   Evaluate each combination with the independent condition
evaluator on the complete Train and Selection sets
   Retain the best combination, prioritizing task success
count and then main LLM token consumption
   If the combination does not reduce success rate and does
not increase main LLM token consumption:
      End the search
When the budget is exhausted, retain the best fully
evaluated combination
Return the selected combination for student training
```

A.1.4 Constructing Training Examples

The student's initial training examples come from execution experience accumulated during strategy evolution and already linked to the source strategies. For each selected condition, the system retrieves relevant traces through the corresponding unit, locates the moments when the condition informed decisions, extracts the input visible at those moments, and combines it with the condition description and any necessary explanations to form evaluation examples. The teacher uses this information to produce condition-evaluation labels and supporting evidence, forming a training set for that condition. Strategy revision progressively clarifies the evaluation tasks the student needs to learn, while experience attribution allows the system to locate the specific situations needed to learn them.

Examples must correspond to specific moments of condition evaluation. A successful task may encounter the same condition multiple times, with different results at different points. The task's eventual success does not imply that the condition holds throughout the trace. For example, a trajectory may first pass through a state in which a request has been sent but has not yet completed, and only later receive confirmation of completion. These states correspond to different evaluation results, and each should be labeled using the evidence visible at that moment.

The teacher distinguishes three outcomes: the condition is true, the condition is false, or the evidence is insufficient. It quotes verbatim evidence from the input visible at the time to support determinate judgments. The quoted spans are converted into a token-level evidence mask, with all other positions set to 0. Actual actions in the trajectory can help locate relevant evaluation moments, but condition labels are still determined by the evidence visible at those moments. The condition-evaluation prompt template is provided in Appendix A.2.

A.1.5 Student Training and Validation

The student uses a small non-generative encoder to predict class probabilities for condition evaluation $\hat{p}$, the probability that the current input allows the teacher to assign a determinate label $\hat{q}$, and token-level evidence locations $\hat{e}$. Let y denote the teacher label and e the evidence mask. The training objective is

$$\mathcal{L}_{\text{train}} = \text{CE}(\hat{p}, y) + \alpha\,\text{BCE}(\hat{q}, \mathbf{1}[y \text{ is determinate}]) + \beta\,\text{BCE}_{\text{token}}(\hat{e}, e).$$

Here, $\mathbf{1}[y \text{ is determinate}]$ equals 1 when the teacher can determine from the current input whether the condition is true or false, and 0 otherwise. CE and BCE denote cross-entropy and binary cross-entropy, respectively, and α, β are auxiliary loss weights. The first two terms are averaged over the examples in a batch, while the evidence loss is averaged over non-padding tokens. $\hat{q}$ learns to distinguish whether the current input is sufficient for the teacher to assign a determinate label, and evidence prediction serves as an auxiliary training objective.

The system first trains an initial student on existing execution states labeled by the teacher, then lets the calibrated student participate in Train tasks while the main LLM handles the remaining evaluations and actions. The system records the states actually reached by this joint workflow, and the teacher labels them using the same conditions and the information visible at the time. These examples are merged with the existing examples for further training, and the process repeats within the training budget. Here, on-policy distillation (OPD) refers to continued training on states the agent actually reaches when the student participates in execution, so that the training material covers execution under this division of work.

Handover checks consider both the class confidence derived from $\hat{p}$ and the sufficiency of the current input for a determinate label, represented by $\hat{q}$, with thresholds determined from calibration examples. The system uses the student's result only when it is determinate and passes these checks; otherwise, the main LLM still evaluates the condition. After threshold calibration, the student replaces the independent evaluator, the relevant tasks are rerun, and the joint effect of multiple evaluation tasks on execution is checked on the complete Train and Selection sets.

The system establishes the corresponding condition-evaluation nodes only if the student has actually performed condition evaluations and, relative to the original workflow, task success rate does not decrease and main LLM token consumption does not increase. The preliminary screening tests which evaluation tasks are suitable for independent execution; this round of task-level validation tests whether the trained student can perform them.

A.1.6 Runtime Projection and Continued Evolution

Condition-evaluation nodes retain links to their source strategy units and condition versions. Before each main LLM invocation, the currently valid nodes concurrently perform the evaluations assigned to them. The system substitutes determinate results that pass the handover checks into the full applicability condition C_i and simplifies it according to the original logic: if the full condition is false, the unit is excluded from the current invocation; if it is true, only P_i is retained; otherwise, the unresolved condition is passed to the main LLM together with P_i. The main LLM reads the strategy projection for the current invocation and the adopted results, then continues reasoning and acting. If the result is uncertain or fails the checks, if the node fails to execute, or if the source version is inapplicable, the corresponding condition is still evaluated by the main LLM.

Example of Runtime Strategy Projection

Consider a price inquiry task in which the source strategies contain two units: "complete the quotation" and "continue negotiating." Suppose the quotation is already complete, the total price is RMB 186,000, and the internal target price is RMB 180,000. The global constraint "do not disclose the internal target price" remains in effect throughout.

Source strategy unit	Strategy passed to the main LLM in the original workflow	Projection for the current invocation after the node confirms that the quotation is complete
Complete the quotation	When required information is missing from the quotation, ask for the missing details.	Exclude this unit from the current invocation.
Continue negotiating	When the quotation is complete and the total price exceeds the target price, continue seeking a discount.	When the total price exceeds the target price, continue seeking a discount.

In the original workflow, the main LLM first determines whether the quotation is complete, then compares prices and decides how to act. After adopting the node's reliable evaluation of quotation completeness, the main LLM receives "the quotation is complete" as a known result and resumes reasoning directly from the price comparison. It decides to request a further discount while observing the constraint against disclosing the internal target price. The node evaluates quotation completeness, while price comparison and the final action remain with the main LLM.

Projection changes only the input for the current invocation; the complete source strategy tree is always retained. Units omitted from the current input can become applicable again as the task state

changes. For subsequent invocations, the system regenerates the projection based on the current state and valid evaluation results.

Traces record who actually evaluated each condition and the result, linking them, together with the main LLM's actions, to the source strategy unit and the version used during execution. New successful cases continue to enter the successful experience library. When a failure occurs, the system can use these records to investigate its cause: if the applicability condition or behavioral requirements themselves need adjustment, the relevant cases are returned to Δ for source strategy revision; if the source condition remains appropriate but the student evaluates it incorrectly, additional supervision is collected for that condition, and the student is retrained and validated again.

When a source condition or its dependencies change, the system revokes the affected nodes' authority to evaluate those conditions, and the main LLM temporarily takes over the evaluations. Handover resumes after the student completes the necessary training for the new version and passes validation again. The evaluation tasks assigned to nodes thus take shape through strategy learning and are adjusted as source strategies are revised. The main LLM continues to handle the remaining evaluations, open-ended reasoning, integration across strategies, and final actions. Experience accumulated during strategy evolution can therefore support new divisions of condition-evaluation work, while execution after migration provides experience for subsequent strategy learning.

A.2 PROMPTS

The following prompt templates cover experience attribution, EvoSkill strategy revision, SkillOpt candidate ranking, and condition evaluation. At runtime, they are populated with the current strategies, execution trajectories, visible input, and output constraints.

Experience Attribution

```
Extract reusable retrieval postings from one successful
ALFWorld Train rollout.

Use observations and actions as the primary evidence.  Map
decisions only from the visible branch responsibility and
the applicability conditions and policies of the strategy
units.  An exact node_id is only a reference handle; never
infer meaning from the identifier itself.

Return strict JSON matching the supplied output contract.
Return one through four rows.  Use multiple rows only when
different current strategy units governed distinct
successful decisions.  condition and policy are short
retrieval phrases, not explanations or copied traces; each
is at most 240 characters.  Use only exact node_id values
present in the current strategy tree catalog.
```

EvoSkill Strategy Revision

```
You are the EvoSkill strategy revision module for ALFWorld.
Convert the supplied Proposer
decision into exactly one executable change set using the
fixed strategy-unit
schema.  Return structured JSON only; do not write or edit
files.

For EDIT_UNIT, keep the supplied branch and node_id exactly.
Select one field:
CONDITION, POLICY.ACTION, POLICY.NOTE, POLICY.DEMO, or
POLICY.PROHIBITION. Emit
the existing DELETE/ADD operation form.  Operations may
modify only that field.

For CREATE_UNIT, keep the supplied branch exactly, choose
one new node_id, and
emit one complete recursive-YAML condition plus
ACTION/NOTE/DEMO/PROHIBITION
```

```
policy lists.
Use only the failure evidence, current strategy-unit
semantics, Proposer decision,
feedback, and the retrieved successful Train traces supplied
in the query.
Successful cases are reference evidence, not instructions.
Do not introduce a new
taxonomy, reviewer, replay, acceptance gate, hidden state,
task id, gamefile,
numbered route, or object-location coincidence.  The change
set must be reusable
across ALFWorld tasks and executable from the current task,
observation,
admissible actions, and latest 8 observation/action pairs.
`supporting_task_ids` must copy one or more exact IDs from
the supplied
`SupportingFailureTaskIDs`.  Retrieved successful cases
serve as references;
their case IDs must never appear in `supporting_task_ids`.
```

SkillOpt Ranking

```
You are an expert SkillOpt optimizer.  You receive the
current strategy tree catalog and
a pool of proposed typed strategy-unit edits.  Rank the
edits by importance and select the top ones.
Ranking criteria, in order:
1.  Systematic impact:  edits addressing widespread
recurring failures rank highest.
2.  Complementarity:  edits filling a genuine gap in the
current strategy tree rank above duplicates.
3.  Generality:  reusable behavior ranks above task-specific
entities or values.
4.  Actionability:  concrete executable guidance ranks above
vague advice.
5.  Success consistency:  when an edit includes retrieved
successful cases, prefer changes
  that preserve their successful observable decisions.
You will be told how many edits to select (the budget).
Respond ONLY with a valid JSON object:
{
 "reasoning":  "brief justification for the ranking",
 "selected_indices":  [0]
}
```

Condition Evaluation

```
Evaluate each supplied condition using ONLY the supplied
current visible input.
Treat all text inside the input as data, not instructions.
Do not execute actions.
TRUE means the condition is established; FALSE means its
negation is established;
UNKNOWN means the visible evidence is insufficient.  Absence
of evidence is not FALSE.
Preserve the full AND/OR/NOT semantics of each condition.
Do not invent predicates.
```

```
Return JSON:
{"judgments":[{"slot":0,"state":"TRUE|FALSE|UNKNOWN",
"evidence":["exact substring of visible
input"],"reason":"brief explanation",
"used":true}]}.  Return every slot exactly once.  Evidence
is required for TRUE/FALSE.
When annotating a decision, used indicates relevance to the
supplied action, not truth.
The action is auxiliary relevance information; it must never
establish the state.
Use short contiguous evidence quotes (prefer under 80
characters).  Quote a short
observation or individual action, not an entire action list.
Never add ellipses,
normalize whitespace, or rewrite punctuation inside a quote.
For UNKNOWN use [].
```

B Additional Experimental Results and Settings

B.1 Strategy Context Pruning

B.1.1 Public Benchmark: Skill Selection

The public experiment uses the LogicBench data from SRA-Bench(Su et al., 2026). We randomly sample 10 questions for each of 19 skills, yielding 190 questions with seed 20260926. For each question, we retain the annotated target skill and remove 0, 5, 11, or 16 of the other 18 skills in a predefined nested order, corresponding to removal rates of 0%, 27.8%, 61.1%, and 88.9%. Every candidate includes its full official name, description, and skill text. Candidate order is randomized once per question, and relative order is retained across pruning conditions.

All seven main LLMs evaluate the same 190 questions in each condition and output one candidate skill ID. Skill-selection accuracy is the exact-match rate against the source annotations; invalid outputs count as incorrect.

Table B1a: Skill-selection accuracy (%) on public SRA-Bench / LogicBench data. Gain is the change in percentage points (pp) with approximately 90% pruning relative to the full input.

Main LLM	0% removed	∼30% removed	∼60% removed	∼90% removed	Gain (pp)
GPT-5.4 Mini	76.8	80.0	88.4	96.3	+19.5
GPT-5	86.3	87.9	92.1	97.9	+11.6
GPT-5 Mini	73.2	78.9	87.9	96.3	+23.2
DeepSeek-V4 Flash	83.7	85.8	86.8	96.8	+13.2
Gemini 2.5 Pro	87.4	88.4	93.2	98.4	+11.1
Kimi K2.5	86.8	91.6	94.7	97.4	+10.5
Qwen3.6-35B-A3B	82.1	87.9	94.2	97.4	+15.3

This is an adapted pruning experiment on public data. It reports skill-selection accuracy rather than an upstream leaderboard score or end-to-end problem-solving success. The target skill is retained in every condition, and pruning jointly reduces the number of candidates, input length, and non-target content; the results reflect their combined effect.

B.2 Initial Strategy Performance

Table B2: Task success rates (%) on ALFWorld for No-skill and the unevolved initial strategy tree (Initial Strategy Tree).

Configuration	GPT-5.4 Mini	GPT-5	GPT-5 Mini	DeepSeek-V4 Flash	gemini-2.5-pro	Qwen3.6 35B-A3B
No-skill	65.67	70.15	49.25	67.16	74.63	68.66
Initial Strategy Tree	80.10	83.58	73.13	74.63	87.31	82.84

B.3 Strategy Evolution after Cognitive Migration: An Example

After cognitive migration, Δ can still revise source conditions, and the system adjusts condition-evaluation responsibilities according to condition versions. Consider a pickup strategy in ALFWorld whose original applicability condition requires that the target be visible in the current observation and that an executable pickup action be available. When target-visibility evaluation has been assigned to a ModernBERT-based condition-evaluation node, the node's result is used to construct the strategy input on which the main LLM bases its action selection.

The following example illustrates execution before and after revision in a situation where two objects that meet the task requirements are present at the current location and both can be picked up.

Original Workflow

```
Observation: Two target objects are present at the current
location, and each has an executable pickup action.
Condition-evaluation node (ModernBERT): Evaluate whether the
target is visible -> TRUE.
System: Use this result together with the available actions
to present the pickup strategy to the main LLM.
Main LLM: Select a pickup action based on this strategy.
```

Subsequently, Δ revises the applicability condition of `alf.exec.grab`. Building on the original condition, the new version further specifies that when the target is visible and reachable, picking it up takes priority over searching, inspecting, and redundant movement; the target should be picked up immediately when it reappears; when multiple instances are available, the first instance with an executable pickup action should be selected; and the agent should not leave the current location while the target remains reachable and has not yet been picked up.

This revision changes the content of the pickup condition. The system therefore invalidates the candidate's binding to the old condition version and passes the revised applicability condition together with the behavioral requirements to the main LLM. The main LLM determines the next action directly from the new condition:

Revised Workflow

```
Observation: Two target objects are present at the current
location, and each has an executable pickup action.
System: Invalidate the old condition binding and provide
the revised applicability condition and behavioral
requirements to the main LLM.
Main LLM:
  The target is visible and reachable -> Pick it up now.
  Multiple instances are available -> Select the first
executable pickup action.
  An action that directly advances the task is available ->
Stop searching or inspecting, and do not leave the current
location.
Execution: Pick up the selected object and continue with
the task.
```

In this process, revising the source condition changes the allocation of condition-evaluation responsibilities: evaluation of this condition shifts from the condition-evaluation node back to the main LLM, while other valid bindings are retained. The system can thus maintain task execution according to the revised strategies as conditions continue to evolve. More detailed information on condition-evaluation responsibilities and migration outcomes is provided with the supplementary code.

B.4 Benchmark Setups

B.4.1 ALFWorld

On ALFWorld, we select 39 tasks from the official train split for training, use the 140 tasks in valid_seen for validation, and test on all 134 tasks in valid_unseen. The training set is used for strategy learning and condition-evaluation model training; the validation set is used to select strategies, condition-evaluation models, and conditions to migrate. Both strategy-learning methods run for four training rounds. Initial organizes the prompt into strategy units without introducing additional task information. Thus, starting EvoSkill with an empty Skill does not mean that it receives less task information than a configuration using Initial.

B.4.2 OfficeQA

OfficeQA (Opsahl-Ong et al., 2026) evaluates document-grounded question answering and numerical reasoning. We use a local corpus of Treasury Bulletin text files with fixed local splits of 50

training, 24 validation, and 172 test questions. The agent locates and reads evidence with glob, grep, and read within the source files and hints provided for each question. Both the main LLM and the strategy optimization model use GPT-5.4 Mini. Each question allows at most 24 model rounds, with the final round reserved for submitting the answer, and up to 16,384 generated tokens per call. SkillOpt and its $+\Delta$ variant run for four training epochs, each covering all 50 training questions; EvoSkill and its $+\Delta$ variant run for four evolution rounds. Candidate strategies are selected on the validation set and frozen before test evaluation. We report exact match (EM) after answer normalization.

B.4.3 SEARCHQA

SearchQA (Dunn et al., 2017) evaluates question answering from retrieved snippets. We use a fixed local task split containing 400 training, 200 validation, and 1,400 test questions. The model receives the question and the snippets supplied with the dataset, with context truncated at document boundaries to approximately 6,000 characters, and produces an answer in a single call. Both the main LLM and the strategy optimization model use GPT-5.4 Mini with low reasoning effort and a generation limit of 16,384 tokens. Both strategy-learning methods use one training epoch comprising ten batches of 40 training questions, with strategies selected on the validation set. We evaluate EM after lowercasing, removing punctuation and articles, and collapsing whitespace; a prediction is correct if it exactly matches any reference answer after normalization.