METADECOMP: ADAPTIVE TASK DECOMPOSITION VIA META-LEARNED PROCEDURAL MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Reusing past experience in long-horizon tasks requires more than storing successful procedures: Language agents must distinguish which procedures apply and what work remains as execution progresses. We introduce MetaDecomp, an in-context meta-learning framework for adaptive task decomposition that jointly learns a task-relative state space and reusable procedures. Its outer loop builds procedural memory from successful training executions, representing task progress as abstract states and procedures between states as directed acyclic graphs of operations and dependencies. As new progress distinctions are learned, MetaDecomp revisits historical execution evidence to reassess state assignments and reorganize the procedures they support. Its inner loop uses this memory to refine an initial decomposition and revise it using environmental feedback, combining applicable procedure fragments with the work required by the current task. Evaluated on ALFWorld and ScienceWorld over three seeds, MetaDecomp achieves the highest mean performance among the compared methods in four of five settings, for example, an 86.3% success rate on ALFWorld with Qwen3-4B. These results are achieved with competitive inference costs across all five settings. Ablation results show that learned cross-task procedural knowledge improves mean performance over within-task revision alone in every setting. Code will be released.

1 INTRODUCTION

Task decomposition helps language agents organize long-horizon tasks into intermediate subgoals and their dependencies (Huang et al., 2022; Khot et al., 2023). Across related task instances, patterns such as acquiring an object, preparing it for use, and placing it at a destination recur even when concrete entities differ. Learning these patterns from past executions can provide reusable guidance for new tasks. Effective reuse, however, requires deciding which procedures apply and which parts remain necessary given the task requirements and the progress already established.

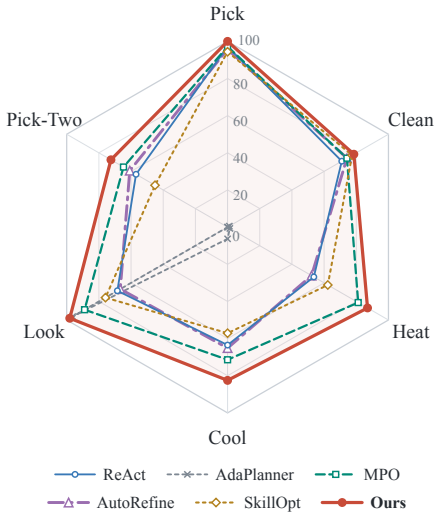

Figure 1: Mean success rates (%) over three seeds on ALFWorld with Qwen3-4B (Table 1).

Prior work learns reusable knowledge in the form of transferable insights, executable skills, and workflows, as in ExpeL, Voyager, and Agent Workflow Memory (Zhao et al., 2024; Wang et al., 2023; 2025). Recent systems organize procedures through hierarchical workflows and skill graphs (Fan et al., 2025; Xu et al., 2026), while HiSkill associates skills with applicability conditions and expected state changes (Hao et al., 2026). These advances motivate a complementary question for decomposition reuse: which aspects of task progress should a shared representation retain to organize procedures from different executions?

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.

Consider cleaning-and-placement tasks in ALFWorld (Shridhar et al., 2021). A mug and a plate can share a procedure for acquisition, cleaning, and placement. An agent already holding a clean object, however, should proceed to placement, whereas one holding a dirty object still needs to clean it. A representation that records possession alone merges histories requiring different continuations. Retaining cleanliness can distinguish these continuations while allowing procedures to transfer across concrete objects. Learning abstractions for planning connects reusable actions to the conditions needed to compose them (Konidaris et al., 2018; Bonet & Geffner, 2020). For decomposition reuse, this connection motivates learning the representation of progress together with the procedures it organizes.

We introduce MetaDecomp, an in-context meta-learning framework that jointly learns a task-relative state space and transition DAGs for task decomposition. States capture abstract progress conditions, separating what has been established from the specific entities involved. Each transition DAG encodes a reusable procedure between two states as a directed acyclic graph of operations and dependencies. Together, the retained state nodes and transition DAGs form a procedural memory, represented as a directed state multigraph whose edges carry transition DAGs. Multiple procedures may connect the same state pair. This representation connects abstract task progress with reusable procedural structure for constructing and revising decompositions.

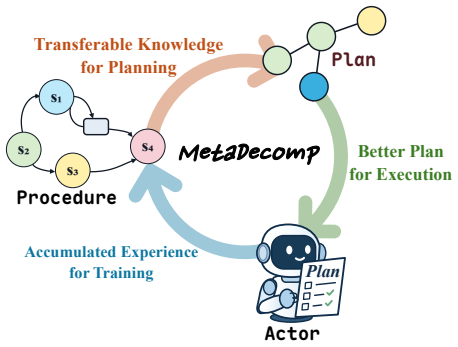

Figure 2: Learning to decompose from experience. Shared procedural knowledge guides task-specific planning and execution. Successful training executions update that knowledge for later tasks.

Learning proceeds on two timescales, separating knowledge accumulated across related tasks from adaptation within each task (Santoro et al., 2016; Hospedales et al., 2022). The outer loop jointly learns the state space and transition DAGs from successful training executions. As new progress distinctions are introduced, it revisits the original execution evidence to reassess historical state assignments and the procedures supported at each transition. Changes to the state space therefore reorganize the procedures in this memory.

The inner loop uses this shared knowledge to guide the initial decomposition and revises it as execution unfolds. Building on feedback-driven planning (Sun et al., 2023; Prasad et al., 2024), it interprets observed progress in the learned state space and recalls procedures that may help resolve unfinished work. The planner can combine parts of stored procedures and add work absent from the procedural memory. Successful training executions return to the outer loop, completing the learning cycle in Figure 2. At test time, the learned state space and transition DAGs remain fixed while the decomposition adapts to each task.

Our contributions are as follows:

- We develop MetaDecomp, an in-context meta-learning framework that jointly learns a shared state space and transition DAGs in that space, forming a procedural memory of state nodes and transition DAGs. It updates the organization of procedures as the state representation evolves and uses the learned knowledge for task-specific decomposition and feedback-driven adaptation.
- We evaluate MetaDecomp on ALFWorld and ScienceWorld, obtaining the highest mean performance among the evaluated methods in most settings. An inference analysis characterizes the performance–cost trade-off, and a loop ablation examines the contribution of cross-task learning beyond within-task revision.

2 Related Work

Meta-Learning. Meta-learning uses experience across tasks to improve adaptation to new tasks (Hospedales et al., 2022). Classical approaches learn parameter initializations or memory-based adaptation mechanisms (Finn et al., 2017; Santoro et al., 2016). For language agents, recent

work learns meta-planners from execution feedback (Xiong et al., 2025), optimizes memory generation and organization (Liang et al., 2026; Zhang et al., 2025), or evolves external playbooks and skill documents (Zhang et al., 2026; Yang et al., 2026). MetaDecomp follows a two-loop formulation in which cross-task learning supplies shared procedural knowledge and within-task in-context adaptation constructs and revises a decomposition.

LLM Agents. LLM agents use language models to interpret observations, choose actions, and invoke tools in pursuit of a goal. ReAct interleaves reasoning with environment actions (Yao et al., 2023), while Reflexion uses verbal feedback to improve decisions across trials (Shinn et al., 2023). AutoGen extends agent interaction to cooperation among multiple agents through conversation (Wu et al., 2024). Agents can also accumulate knowledge over time, as illustrated by Voyager's executable skill library (Wang et al., 2023). A-MEM organizes experience into linked notes and revises historical memory representations as new memories arrive (Xu et al., 2025).

Planning and Decomposition in Agents. Task decomposition organizes goals into subtasks and dependencies, from hierarchical task-network planning (Erol et al., 1994) to language-model plans and modular subproblem solvers (Huang et al., 2022; Khot et al., 2023). AdaPlanner and ADaPT revise plans or recursively decompose subtasks using execution feedback (Sun et al., 2023; Prasad et al., 2024). Reusable procedures also appear in typed skill graphs and hypergraphs (Bai et al., 2026; Xu et al., 2026). Learned planning representations capture preconditions and effects (Konidaris et al., 2018; Bonet & Geffner, 2020), and TheoryCoder-2 synthesizes predicates and operators (Ahmed et al., 2026). MemP studies procedural-memory construction, retrieval, and updating from trajectories (Fang et al., 2026). ReMe combines experience distillation, context-adaptive reuse, and utility-based refinement (Cao et al., 2025). MACLA learns hierarchical procedures with preconditions and postconditions, using Bayesian selection and contrastive refinement (Forouzandeh et al., 2025). MetaDecomp jointly learns a task-relative state space and transition DAGs within it. When the schema changes, it reprojects historical execution boundaries and regroups their procedural evidence under the revised state assignments.

3 METADECOMP

MetaDecomp is an in-context meta-learning framework for task decomposition with two coupled loops. The outer loop jointly learns a state space describing task progress and transition DAGs in that space from executions across related tasks. The retained state nodes and transition DAGs form a procedural memory, which the inner loop uses to construct and adapt a decomposition for the current task. Successful training executions supply evidence for the next outer update. Figure 3 illustrates this cycle, and Appendix B.1 gives the complete algorithm.

3.1 PRELIMINARIES

We consider interactive tasks organized into learning spaces. Each learning space covers a supplied task type or a predefined group of related types within one benchmark and learns its own state space and transition DAGs. For task τ, an actor receives observations and executes environment actions, producing a history $h_t = (o_0, a_1, o_1, \ldots, a_t, o_t)$. Its guidance is an ordered decomposition $D_t = (d_{t,1}, \ldots, d_{t,n_t})$ of task-level instructions. The state space and transition DAGs transfer across instances within the learning space, while D_t is constructed and adapted for the current task.

Consider cleaning and placing an object. A mug and a plate can share a procedure even though their concrete identities differ. Whether the required object has already been cleaned, however, changes the remaining work. MetaDecomp therefore separates task-relative roles, such as the required object, from progress conditions, such as possession and cleanliness. A learned schema Σ specifies the roles and a set of fields $\mathcal{F}$, each with a finite value set V_f. It defines the state space $\mathcal{X}_\Sigma$, on which the procedural memory $\mathcal{H}$ is constructed,

$$\mathcal{X}_\Sigma = \prod_{f \in \mathcal{F}} V_f, \qquad \mathcal{H} = (\mathcal{S}, \mathcal{T}), \qquad \mathcal{S} \subseteq \mathcal{X}_\Sigma. \tag{1}$$

A node $s \in \mathcal{S}$ is a complete field assignment retained from experience, and $\mathcal{T}$ contains transitions between these nodes. Role bindings β associate abstract referents with task entities and do not enter

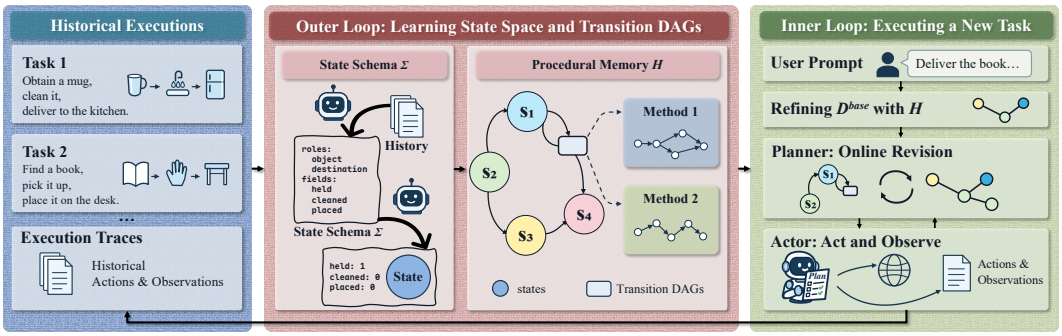

Figure 3: MetaDecomp jointly learns a state space specified by schema Σ and transition DAGs in that space. Retained state nodes and transition DAGs form the procedural memory $\mathcal{H}$. The inner loop refines a cached decomposition D^{base} into D_0 and revises it using actor feedback. Successful training executions update the shared knowledge between episodes; evaluation keeps it fixed. The illustrative binary fields denote established (1) or unestablished (0) progress.

state identity. Two tasks can therefore share a state recording that the required object is held and clean while binding that object to different entities. Field values describe conditions supported by observations, including unestablished knowledge when a condition is not yet known.

Each transition $e \in \mathcal{T}$ connects two distinct states through a transition DAG M_e representing a reusable procedure, which we call a method,

$$e = (s_e^-, s_e^+, M_e), \qquad M_e = (A_e, E_e, x_e), \qquad E_e \subseteq A_e \times A_e. \tag{2}$$

The nodes A_e are parameterized operations, $x_e(a)$ gives the instruction for operation a, and the acyclic edges E_e express inferred prerequisite dependencies. Instructions retain supported applicability, argument-selection, and stopping conditions. Multiple methods can realize the same state change, and the state graph may contain cycles. The state nodes and transition DAGs together form the procedural memory, a directed state multigraph whose transitions carry these DAGs. The learned state space provides a common interface for relating procedures from different task instances.

Let Σ_b define the state space and $\mathcal{H}_b$ be the procedural memory after outer update b. An evidence ledger $\mathcal{P}_b$ retains executions and their state and method assignments. For language-model modules ϕ, the two-loop process is

$$\begin{aligned} \xi_\tau &= \text{Inner}_\phi(\tau; \Sigma_b, \mathcal{H}_b), \\ (\Sigma_{b+1}, \mathcal{H}_{b+1}, \mathcal{P}_{b+1}) &= \text{Outer}_\phi(\Sigma_b, \mathcal{H}_b, \mathcal{P}_b, \mathcal{B}_b), \end{aligned} \tag{3}$$

where $\xi_\tau = (\tau, h_{T_\tau})$ is an executed episode and $\mathcal{B}_b$ is a batch of successful training episodes. The outer loop jointly updates the state space through Σ and the transition DAGs connecting its retained state nodes in $\mathcal{H}$. The inner loop constructs and revises D_t for one task while holding both fixed. This separation of cross-task learning and within-task adaptation defines MetaDecomp's in-context meta-learning process.

3.2 Outer Loop: Jointly Learning the State Space and Transition DAGs

The outer loop jointly learns the state space and transition DAGs from batches of successful training executions. Task descriptions specify requirements, and action–observation traces supply evidence of attained conditions and executed operations. The learner first proposes progress distinctions that change remaining work, method applicability, or goal satisfaction. The schema can acquire new roles, fields, or values while retaining the meanings of existing entries. These additions determine how execution experience is organized in the procedural memory.

Under the updated schema, the learner extracts local realizations from each new execution. Each realization ℓ contains an observed source boundary ω_ℓ^-, a later target boundary ω_ℓ^+, and evidence-supported operations proposed to connect their states. When the schema changes, the learner also returns to the original traces and reprojects retained historical boundaries. It reconsiders only added or expanded fields and new role bindings, preserving the other assignments and execution evidence.

Writing q_{b+1} for the resulting boundary projection, new and reassessed realizations are grouped by their complete endpoint assignments,

$$\mathcal{E}_{uv} = \{\ell \in \mathcal{L}_b : q_{b+1}(\omega_\ell^-) = u,\ q_{b+1}(\omega_\ell^+) = v\}, \qquad u \neq v, \tag{4}$$

where $\mathcal{L}_b$ contains the realizations under review at update b. Equality depends on progress conditions, so different entity bindings can contribute evidence to the same state change.

Changing these groups also changes the support of existing methods. A method retains its association with one state pair only when all its supporting realizations agree on that pair after reprojection. If its support splits, the learner reassesses the procedure separately at the supported pairs. In the cleaning example, adding cleanliness can separate historical occurrences previously pooled into a single "held" state. Replaying their observations determines which procedures start before cleaning and which start after it. The stored links to original evidence make this reassignment possible. This couples learning the state space with learning transition DAGs in that space. A new progress distinction changes which experiences can support a common method and how that method connects states.

To update the transition DAGs, the learner consolidates realizations that support the same parameterized procedure for each state pair under review. It can refine instructions, add evidenced operations, or revise dependencies. Procedures with different operations, applicability, decision semantics, or inferred dependencies remain separate alternatives. New operations require execution support, and proposed DAGs are checked for valid references and acyclicity. These checks establish structural consistency. Whether a procedure applies to a new task is assessed during grounding. The updated state space and transition DAGs are available to later training tasks. Appendix B.2 specifies evidence extraction, historical reassignment, and method updates.

3.3 Inner loop: task adaptation with procedural memory

The inner loop interprets task progress in the learned state space and uses the stored transition DAGs as a procedural prior. The planner binds task roles, selects useful operations, and combines them with any work absent from the memory into an ordered decomposition, which guides the actor in choosing concrete environment commands based on the current observations.

In the main experimental configuration, MetaDecomp starts from MPO's cached task-specific decomposition D^{base}. Initial refinement aligns this decomposition with the learned state space and procedures. The planner receives all methods for the task's learning space, without requiring a stored-state match, and constructs

$$D_0 = \mathrm{Plan}_\phi(\tau, D^{\mathrm{base}}; \Sigma, \mathcal{H}). \tag{5}$$

The planner can replace, insert, or remove instructions to incorporate applicable experience and repair missing or unsuitable work. It grounds method prerequisites in the task and places reused operations after the work needed to establish them. When the procedural memory contains no methods, initialization preserves the cached decomposition in dependency order.

When execution feedback indicates stalled progress, the planner reviews the current decomposition. When a schema is available, it first projects the observed execution history into that schema. Online path recall requires a complete projection with exactly the field values of a stored state. The model selects useful intermediate or completion targets among its reachable states, then recalls a bounded number of shortest simple paths, retaining all alternative methods on their edges. Paths are ranked by state-transition count. The resulting procedures propose ways to continue, whose applicability is judged using the task and current observations.

Let c_t collect the execution evidence, stall information, any complete current state projection, and the cached baseline when available. Let $\mathcal{R}_t$ denote the recalled methods. Plan revision and the next action follow

$$D_t^+ = \mathrm{Refine}_\phi(\tau, D_t, c_t; \Sigma, \mathcal{R}_t), \qquad a_{t+1} \sim \pi_\phi(\cdot \mid \tau, h_t, D_t^+). \tag{6}$$

The reviewer identifies unresolved work from the observations and can replace an instruction, insert work around an existing step, or retain the plan. Partial method reuse is allowed when omitted prerequisites already hold or are established by earlier planned work. For example, evidence that the object is held but still requires cleaning can motivate inserting cleaning before placement. Revision

Method	Pick	Clean	Heat	Cool	Look	Pick-Two	Avg.
			Qwen3-8B				
ReAct	$\underline{98.6}_{\pm 2.4}$	$\mathbf{90.3}_{\pm 0.0}$	$60.9_{\pm 8.7}$	$87.3_{\pm 2.7}$	$88.9_{\pm 0.0}$	$39.2_{\pm 12.2}$	$79.6_{\pm 2.6}$
AdaPlanner	$0.0_{\pm 0.0}$	$3.2_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$85.2_{\pm 3.2}$	$0.0_{\pm 0.0}$	$12.2_{\pm 0.4}$
MPO	$\mathbf{100.0}_{\pm 0.0}$	$\underline{88.2}_{\pm 1.9}$	$\underline{84.1}_{\pm 6.6}$	$\underline{88.9}_{\pm 2.7}$	$\underline{96.3}_{\pm 3.2}$	$\underline{84.3}_{\pm 9.0}$	$\underline{90.3}_{\pm 0.7}$
AutoRefine†	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{90.3}_{\pm 0.0}$	$66.7_{\pm 6.6}$	$87.3_{\pm 7.3}$	$87.0_{\pm 3.2}$	$33.3_{\pm 6.8}$	$79.9_{\pm 2.2}$
SkillOpt†	$\mathbf{100.0}_{\pm 0.0}$	$83.9_{\pm 0.0}$	$\mathbf{92.8}_{\pm 2.5}$	$87.3_{\pm 2.7}$	$85.2_{\pm 3.2}$	$56.9_{\pm 13.6}$	$85.6_{\pm 1.7}$
Ours	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{90.3}_{\pm 3.2}$	$\mathbf{92.8}_{\pm 2.5}$	$\mathbf{95.2}_{\pm 0.0}$	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{90.2}_{\pm 3.4}$	$\mathbf{94.5}_{\pm 0.4}$
			Qwen3-4B				
ReAct	$\underline{97.2}_{\pm 2.4}$	$71.0_{\pm 6.5}$	$53.6_{\pm 6.6}$	$63.5_{\pm 2.7}$	$68.5_{\pm 6.4}$	$56.9_{\pm 3.4}$	$69.4_{\pm 1.3}$
AdaPlanner	$0.0_{\pm 0.0}$	$1.1_{\pm 1.9}$	$1.4_{\pm 2.5}$	$6.3_{\pm 2.7}$	$\underline{96.3}_{\pm 3.2}$	$0.0_{\pm 0.0}$	$14.4_{\pm 0.9}$
MPO	$\underline{97.2}_{\pm 4.8}$	$74.2_{\pm 3.2}$	$\underline{81.2}_{\pm 2.5}$	$\underline{71.4}_{\pm 4.8}$	$88.9_{\pm 0.0}$	$\underline{64.7}_{\pm 0.0}$	$\underline{79.9}_{\pm 2.7}$
AutoRefine†	$95.8_{\pm 0.0}$	$74.2_{\pm 3.2}$	$52.2_{\pm 7.5}$	$65.1_{\pm 9.9}$	$66.7_{\pm 14.7}$	$60.8_{\pm 9.0}$	$70.1_{\pm 2.7}$
SkillOpt†	$94.4_{\pm 6.4}$	$\underline{77.4}_{\pm 0.0}$	$62.3_{\pm 2.5}$	$57.1_{\pm 4.8}$	$75.9_{\pm 3.2}$	$45.1_{\pm 12.2}$	$70.4_{\pm 2.6}$
Ours	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{78.5}_{\pm 4.9}$	$\mathbf{87.0}_{\pm 7.5}$	$\mathbf{82.5}_{\pm 2.7}$	$\mathbf{98.1}_{\pm 3.2}$	$\mathbf{72.5}_{\pm 3.4}$	$\mathbf{86.3}_{\pm 2.6}$

Table 1: ALFWorld success rates (%) on 134 unseen test tasks, reported as the mean with sample standard deviation in subscript across three evaluation runs. Avg. weights task types by their test-task counts within each run. Bold and underlining mark the highest and second-highest distinct displayed means, including ties. †GLM-5.3-Flash is used for distillation only during training.

from execution evidence remains possible when no methods are recalled or applicable. Dependencies guide the model's construction of the sequence, with the actor continuing to ground its instructions as observations arrive.

During training, successful inner-loop executions enter the next outer update, closing the meta-learning cycle in Equation 3. During evaluation, each learning space uses a frozen snapshot of its learned state space and transition DAGs and runs only the inner loop. The agent continues to adapt its decomposition to each test task while keeping Σ and $\mathcal{H}$ fixed.

3.4 Motivation from state abstraction

State abstraction motivates retaining distinctions that affect procedural choices (Li et al., 2006). Consider a fixed method dictionary, continuation loss, and distribution of histories and task context. If a refined representation preserves the old state, an optimal selector can ignore its added information, so the minimum expected loss cannot increase. It decreases strictly when a coarse-state optimal choice is suboptimal under the refined information with positive probability. Appendix B.4 formalizes this observation and relates unresolved choice ambiguity to excess loss under a unique history-optimal method and a uniform positive margin.

This comparison provides a conceptual design rationale. Actual updates can reassign old state values and change procedures, and finer states can reduce exact-match recall coverage. The analysis therefore does not establish improvement across learning updates. Our experiments evaluate the complete pipeline without estimating the idealized ambiguity or selection losses.

4 Experiments

We examine how the state space and transition DAGs learned across tasks support adaptation to new task instances, evaluating task completion and inference cost.

4.1 Experimental Setup

We evaluate on ALFWorld (Shridhar et al., 2021) and ScienceWorld (Wang et al., 2022). ALFWorld tests household manipulation through a text interface. We evaluate on 134 unseen tasks across six types and report success rate. ScienceWorld requires agents to carry out interactive scientific

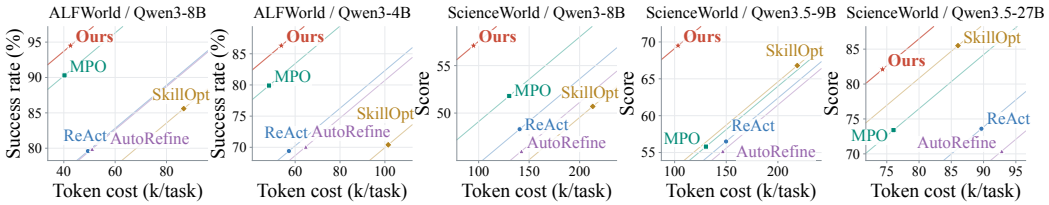

Figure 4: Token cost and task performance across five settings. Cost is evaluation-time input-plus-output tokens per task, in thousands, with the accounting in Table 3. Performance is ALFWorld success rate (%) or ScienceWorld score. Axes are scaled independently. Diagonal lines are visual guides with constant performance minus cost after normalizing each displayed axis range to $[0, 1]$. AdaPlanner is omitted for readability and retained in the tables.

experiments. We use a fixed panel of 211 test tasks spanning 30 native task types, grouped into ten categories, and report their mean score. In both benchmarks, Avg. is the mean over all test tasks, weighting categories by their test-task counts. The main tables report means and sample standard deviations over three evaluation seeds.

ALFWorld uses Qwen3-4B and Qwen3-8B (Qwen Team, 2025). ScienceWorld uses Qwen3-8B, Qwen3.5-9B, and Qwen3.5-27B (Qwen Team, 2026). Given cached MPO decompositions, MetaDecomp uses the reported Qwen model for all subsequent inference calls, including initial plan refinement, state projection, target selection, plan revision, and action generation.

We compare with ReAct (Yao et al., 2023), AdaPlanner (Sun et al., 2023), MPO (Xiong et al., 2025), AutoRefine (Qiu et al., 2026), and SkillOpt (Yang et al., 2026). GLM-5.3-Flash (Zeng et al., 2026) is used only during training, for MetaDecomp's outer-loop learning and for AutoRefine's and SkillOpt's experience distillation. During evaluation, AutoRefine and SkillOpt freeze their learned libraries, while MetaDecomp freezes its state space and transition DAGs. Appendix A.1 provides training and evaluation protocols, category sizes, and implementation settings.

4.2 Main Results

Tables 1 and 2 show that MetaDecomp achieves the highest average performance among the evaluated methods in four of the five benchmark–model settings.

On ALFWorld (Table 1), it reaches 86.3% with Qwen3-4B and 94.5% with Qwen3-8B, improving on MPO by 6.5 and 4.2 percentage points. The ALFWorld results show that the improvement extends across different subgoal requirements. With Qwen3-4B, MetaDecomp leads in all six task types. Relative to MPO, it improves cleaning from 74.2% to 78.5%, heating from 81.2% to 87.0%, and picking and placing two objects from 64.7% to 72.5%. These tasks require maintaining progress through object preparation or repeated acquisition and placement, the kinds of distinctions that motivate our shared state representation. With Qwen3-8B, MetaDecomp leads or ties in all six types, including a tie with SkillOpt on heating. Appendix A.4 examines a learned procedural memory for ALFWorld's Look tasks, illustrating how object possession, lamp state, and agent location distinguish the procedures needed to complete a task. When the object is held but the agent is away from an already lit lamp, the stored continuation approaches the lamp without activating it again.

On ScienceWorld (Table 2), MetaDecomp reaches 57.1 with Qwen3-8B, exceeding the strongest baseline, MPO, by 5.3 points. With Qwen3.5-9B, it reaches 69.5, exceeding SkillOpt by 2.7 points. With Qwen3.5-27B, it reaches 82.1, below SkillOpt's 85.5.

ScienceWorld reveals more variation across task categories. MetaDecomp leads or ties in four, five, and five of the ten categories with Qwen3-8B, Qwen3.5-9B, and Qwen3.5-27B, respectively. At 27B, it outperforms SkillOpt in plant growth (Plants, 63.1 versus 47.7) and genetics (Mende., 76.2 versus 65.3), but has lower scores on lifespan comparison and life cycles (Life-1 and Life-2) and a lower overall average.

Exploratory task-level paired tests against the highest-mean baseline in each setting find significant gains for Qwen3-8B on ALFWorld ($p = 0.0217$) and ScienceWorld ($p = 0.0196$). The other three

Method	Matter	Meas.	Elec.	Class.	Plants	Chem.	Life-1	Life-2	Force	Mende.	Avg.
Qwen3-8B (Qwen Team, 2025)											
ReAct	$24.8_{\pm 7.7}$	$66.3_{\pm 3.8}$	$57.6_{\pm 1.6}$	$79.5_{\pm 3.4}$	$20.4_{\pm 6.2}$	$36.5_{\pm 2.3}$	$65.6_{\pm 8.1}$	$27.2_{\pm 11.2}$	$37.5_{\pm 6.0}$	$\mathbf{40.3}_{\pm 13.2}$	$48.3_{\pm 1.3}$
AdaPlanner	$2.4_{\pm 1.2}$	$17.1_{\pm 3.2}$	$12.5_{\pm 1.3}$	$22.9_{\pm 2.0}$	$3.9_{\pm 0.8}$	$10.7_{\pm 2.4}$	$50.0_{\pm 2.4}$	$6.7_{\pm 0.1}$	$8.0_{\pm 0.7}$	$17.5_{\pm 6.3}$	$15.6_{\pm 0.4}$
MPO	$32.4_{\pm 6.6}$	$66.2_{\pm 5.2}$	$57.2_{\pm 1.3}$	$\mathbf{94.0}_{\pm 2.6}$	$11.5_{\pm 1.4}$	$\mathbf{48.2}_{\pm 3.5}$	$63.5_{\pm 2.7}$	$\underline{42.0}_{\pm 12.2}$	$39.9_{\pm 8.6}$	$28.6_{\pm 10.4}$	$\underline{51.8}_{\pm 1.4}$
AutoRefine†	$17.5_{\pm 3.3}$	$64.2_{\pm 9.6}$	$56.7_{\pm 1.5}$	$76.8_{\pm 0.7}$	$21.0_{\pm 4.6}$	$32.7_{\pm 5.3}$	$65.6_{\pm 3.0}$	$20.7_{\pm 10.6}$	$41.1_{\pm 2.4}$	$\underline{36.1}_{\pm 13.1}$	$46.0_{\pm 2.3}$
SkillOpt†	$\underline{41.2}_{\pm 3.3}$	$\mathbf{71.3}_{\pm 0.4}$	$\underline{58.1}_{\pm 7.4}$	$82.9_{\pm 1.1}$	$\underline{25.3}_{\pm 5.4}$	$26.6_{\pm 8.1}$	$\mathbf{73.0}_{\pm 2.7}$	$33.4_{\pm 11.4}$	$\mathbf{46.6}_{\pm 6.5}$	$7.6_{\pm 6.9}$	$50.7_{\pm 1.2}$
Ours	$\mathbf{58.2}_{\pm 4.4}$	$\underline{69.2}_{\pm 4.1}$	$\mathbf{59.5}_{\pm 2.0}$	$\underline{91.9}_{\pm 4.7}$	$\mathbf{28.8}_{\pm 4.6}$	$\underline{43.1}_{\pm 7.3}$	$\underline{68.8}_{\pm 3.7}$	$\mathbf{44.8}_{\pm 12.5}$	$\underline{45.1}_{\pm 5.1}$	$20.7_{\pm 8.5}$	$\mathbf{57.1}_{\pm 4.0}$
Qwen3.5-9B (Qwen Team, 2026)											
ReAct	$31.1_{\pm 1.4}$	$54.2_{\pm 8.3}$	$\underline{64.1}_{\pm 5.4}$	$90.3_{\pm 1.2}$	$\underline{40.6}_{\pm 5.3}$	$57.0_{\pm 6.4}$	$69.8_{\pm 2.7}$	$20.0_{\pm 4.6}$	$52.3_{\pm 12.4}$	$\mathbf{57.0}_{\pm 10.2}$	$56.5_{\pm 2.1}$
AdaPlanner	$34.5_{\pm 8.8}$	$24.4_{\pm 6.1}$	$46.3_{\pm 0.8}$	$44.2_{\pm 2.9}$	$18.1_{\pm 5.6}$	$19.4_{\pm 5.2}$	$35.9_{\pm 2.4}$	$4.4_{\pm 2.0}$	$28.8_{\pm 6.9}$	$48.0_{\pm 7.5}$	$32.7_{\pm 2.2}$
MPO	$25.1_{\pm 1.8}$	$56.4_{\pm 1.5}$	$\underline{64.1}_{\pm 5.8}$	$91.6_{\pm 0.9}$	$37.0_{\pm 1.9}$	$59.7_{\pm 4.8}$	$71.4_{\pm 8.2}$	$23.4_{\pm 13.5}$	$49.4_{\pm 7.5}$	$52.0_{\pm 6.9}$	$55.8_{\pm 1.5}$
AutoRefine†	$28.0_{\pm 3.0}$	$50.5_{\pm 2.9}$	$59.7_{\pm 5.4}$	$89.1_{\pm 2.3}$	$36.2_{\pm 2.5}$	$58.3_{\pm 5.0}$	$68.3_{\pm 14.5}$	$19.7_{\pm 8.0}$	$58.0_{\pm 16.8}$	$\underline{55.5}_{\pm 7.8}$	$55.2_{\pm 1.2}$
SkillOpt†	$\mathbf{54.0}_{\pm 7.7}$	$\underline{63.0}_{\pm 4.2}$	$63.4_{\pm 3.0}$	$\underline{91.7}_{\pm 6.1}$	$27.1_{\pm 0.6}$	$\mathbf{64.5}_{\pm 7.5}$	$\underline{77.8}_{\pm 2.7}$	$\mathbf{56.4}_{\pm 6.9}$	$\mathbf{94.2}_{\pm 2.5}$	$45.4_{\pm 8.0}$	$\underline{66.8}_{\pm 2.1}$
Ours	$\underline{50.7}_{\pm 3.3}$	$\mathbf{70.1}_{\pm 4.9}$	$\mathbf{82.2}_{\pm 4.6}$	$\mathbf{96.8}_{\pm 2.1}$	$\mathbf{42.0}_{\pm 6.8}$	$\underline{63.1}_{\pm 4.0}$	$\mathbf{79.4}_{\pm 5.5}$	$\underline{44.1}_{\pm 15.7}$	$\underline{82.6}_{\pm 7.5}$	$48.9_{\pm 10.2}$	$\mathbf{69.5}_{\pm 2.3}$
Qwen3.5-27B (Qwen Team, 2026)											
ReAct	$77.4_{\pm 1.9}$	$68.2_{\pm 2.4}$	$75.4_{\pm 3.0}$	$\underline{96.7}_{\pm 2.3}$	$41.8_{\pm 13.3}$	$89.4_{\pm 6.9}$	$63.5_{\pm 2.7}$	$20.3_{\pm 10.6}$	$79.8_{\pm 1.5}$	$\underline{70.3}_{\pm 15.3}$	$73.6_{\pm 1.9}$
AdaPlanner	$56.8_{\pm 9.6}$	$37.2_{\pm 3.4}$	$63.1_{\pm 3.6}$	$78.3_{\pm 8.8}$	$31.5_{\pm 5.9}$	$37.0_{\pm 1.5}$	$48.7_{\pm 14.1}$	$12.4_{\pm 1.5}$	$43.0_{\pm 6.4}$	$60.3_{\pm 3.1}$	$50.7_{\pm 3.6}$
MPO	$70.6_{\pm 2.3}$	$62.3_{\pm 5.0}$	$75.1_{\pm 1.5}$	$94.7_{\pm 1.2}$	$42.7_{\pm 1.1}$	$\underline{98.5}_{\pm 2.6}$	$63.5_{\pm 2.7}$	$46.1_{\pm 2.9}$	$\underline{80.0}_{\pm 2.3}$	$64.4_{\pm 3.8}$	$73.4_{\pm 0.8}$
AutoRefine†	$74.4_{\pm 5.3}$	$67.4_{\pm 5.6}$	$76.5_{\pm 3.9}$	$96.6_{\pm 1.7}$	$37.5_{\pm 3.5}$	$87.0_{\pm 1.1}$	$57.1_{\pm 4.8}$	$20.0_{\pm 13.5}$	$69.9_{\pm 13.5}$	$62.5_{\pm 2.6}$	$70.4_{\pm 0.4}$
SkillOpt†	$\mathbf{87.5}_{\pm 1.9}$	$\mathbf{78.3}_{\pm 4.3}$	$\underline{77.1}_{\pm 0.9}$	$\mathbf{100.0}_{\pm 0.0}$	$\underline{47.7}_{\pm 1.6}$	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{88.9}_{\pm 2.7}$	$\mathbf{86.9}_{\pm 6.1}$	$\mathbf{100.0}_{\pm 0.0}$	$65.3_{\pm 5.5}$	$\mathbf{85.5}_{\pm 0.6}$
Ours	$\underline{84.4}_{\pm 3.6}$	$\underline{69.6}_{\pm 8.7}$	$\mathbf{78.1}_{\pm 2.7}$	$94.9_{\pm 4.1}$	$\mathbf{63.1}_{\pm 6.1}$	$\mathbf{100.0}_{\pm 0.0}$	$\underline{68.3}_{\pm 5.5}$	$\underline{59.3}_{\pm 6.4}$	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{76.2}_{\pm 1.5}$	$\underline{82.1}_{\pm 2.9}$

Table 2: ScienceWorld final scores (0–100) on 211 test tasks, reported as the mean with sample standard deviation in subscript across three evaluation runs. Avg. weights categories by their task counts within each run. Formatting and † follow Table 1.

comparisons do not reach the pairwise $p < 0.05$ threshold. Appendix A.2 reports all pairwise results and confidence intervals.

4.3 Inference Efficiency

Figure 4 compares mean task performance with mean evaluation-time inference tokens per task across the five settings. MetaDecomp achieves higher mean performance than ReAct and AutoRefine with fewer tokens in every setting. Compared with MPO, it improves performance with lower cost on ScienceWorld and higher cost on ALFWorld. It also improves on SkillOpt in both performance and cost in four settings. On ScienceWorld with Qwen3.5-27B, SkillOpt achieves a higher score at a higher cost.

Relative to MPO, MetaDecomp reduces mean token cost by 27.1%, 20.7%, and 2.2% on ScienceWorld with Qwen3-8B, Qwen3.5-9B, and Qwen3.5-27B, respectively. On ALFWorld, its success gains of 4.2 and 6.5 percentage points with Qwen3-8B and Qwen3-4B come with 5.8% and 11.3% more tokens, respectively. Table 3 separates input and output consumption over the same three evaluations used in the main performance tables. All methods include both successful and failed episodes. For MPO and MetaDecomp, accounting starts from the cached initial decomposition. MetaDecomp's total includes initial refinement and all subsequent actor and controller calls.

The aggregate differences vary substantially across task types (Table 7). On ScienceWorld with Qwen3-8B, Electricity accounts for 11.8% of test tasks but 41.3% of the mean token savings relative to MPO. On ALFWorld with Qwen3-4B, cleaning and heating together account for 40.3% of tasks and 82.9% of the additional tokens. Both savings and additional expenditure can therefore be concentrated in particular task types. The category breakdown describes where these differences arise without separating the effects of planning, retries, and execution length.

	ScienceWorld						ALFWorld			
	Qwen3-8B		Qwen3.5-9B		Qwen3.5-27B		Qwen3-8B		Qwen3-4B	
Method	Input	Output	Input	Output	Input	Output	Input	Output	Input	Output
ReAct	$138.7_{\pm 6.0}$	$1.8_{\pm 0.1}$	$143.9_{\pm 2.0}$	$6.0_{\pm 0.2}$	$86.4_{\pm 3.6}$	$3.3_{\pm 0.2}$	$47.8_{\pm 0.6}$	$1.6_{\pm 0.0}$	$56.5_{\pm 1.0}$	$0.7_{\pm 0.0}$
AdaPlanner	$35.0_{\pm 0.8}$	$5.1_{\pm 0.3}$	$35.4_{\pm 0.7}$	$7.5_{\pm 0.1}$	$25.3_{\pm 0.4}$	$5.5_{\pm 0.1}$	$20.3_{\pm 0.2}$	$8.1_{\pm 0.1}$	$23.6_{\pm 0.4}$	$8.7_{\pm 0.1}$
MPO	$128.6_{\pm 5.0}$	$1.7_{\pm 0.1}$	$125.4_{\pm 11.8}$	$4.9_{\pm 0.4}$	$73.3_{\pm 1.2}$	$2.7_{\pm 0.0}$	$39.1_{\pm 1.3}$	$1.2_{\pm 0.1}$	$47.8_{\pm 2.3}$	$0.6_{\pm 0.0}$
AutoRefine	$140.6_{\pm 4.4}$	$1.8_{\pm 0.1}$	$141.0_{\pm 2.0}$	$5.7_{\pm 0.1}$	$89.4_{\pm 17.4}$	$3.4_{\pm 0.7}$	$49.4_{\pm 1.2}$	$1.6_{\pm 0.0}$	$64.2_{\pm 3.4}$	$0.6_{\pm 0.0}$
SkillOpt	$211.5_{\pm 9.3}$	$1.5_{\pm 0.1}$	$216.1_{\pm 6.8}$	$2.7_{\pm 0.1}$	$84.7_{\pm 1.4}$	$1.3_{\pm 0.0}$	$85.4_{\pm 1.4}$	$1.3_{\pm 0.0}$	$101.0_{\pm 2.8}$	$0.5_{\pm 0.0}$
Ours	$92.8_{\pm 1.3}$	$2.1_{\pm 0.0}$	$98.6_{\pm 3.8}$	$4.7_{\pm 0.7}$	$71.1_{\pm 5.2}$	$3.2_{\pm 0.2}$	$41.2_{\pm 0.4}$	$1.4_{\pm 0.0}$	$52.7_{\pm 2.1}$	$1.1_{\pm 0.0}$

Table 3: Evaluation-time input and output tokens per task, in thousands, reported as the mean with sample standard deviation in subscript across the three evaluations used in the main performance tables. Appendix A.3 gives the accounting and aggregation protocol.

MetaDecomp Components		ScienceWorld			ALFWorld	
Inner loop	Outer loop	Qwen3-8B	Qwen3.5-9B	Qwen3.5-27B	Qwen3-4B	Qwen3-8B
×	×	$51.8_{\pm 1.4}$	$55.8_{\pm 1.5}$	$73.4_{\pm 0.8}$	$79.9_{\pm 2.7}$	$90.3_{\pm 0.7}$
✓	×	$55.7_{\pm 2.9}$	$54.6_{\pm 1.5}$	$76.8_{\pm 1.3}$	$76.6_{\pm 0.9}$	$90.8_{\pm 1.6}$
✓	✓	$\mathbf{57.1}_{\pm 4.0}$	$\mathbf{69.5}_{\pm 2.3}$	$\mathbf{82.1}_{\pm 2.9}$	$\mathbf{86.3}_{\pm 2.6}$	$\mathbf{94.5}_{\pm 0.4}$

Table 4: Ablation of MetaDecomp's inner and outer loops, reporting means with sample standard deviations in subscript across three evaluation runs. The first row reports MPO. The second uses an empty procedural memory, and the third enables both loops. The two MetaDecomp configurations share the cached MPO decomposition and Qwen actor/controller model. Bold marks the highest mean.

4.4 Ablation Study: Contribution of Cross-task Learning

Table 4 compares MPO with the inner-only and full MetaDecomp configurations over three evaluation seeds. Both MetaDecomp configurations start from the same cached MPO decomposition and use the same Qwen model for acting and online control. The inner-only configuration preserves the initial decomposition and revises it using current-task feedback. The full configuration uses the learned state space and transition DAGs to align the initial decomposition with stored procedures and guide subsequent revision.

Compared with MPO, the inner-only configuration has mixed results. On ScienceWorld, the respective mean scores are 51.8 and 55.7 with Qwen3-8B, 55.8 and 54.6 with Qwen3.5-9B, and 73.4 and 76.8 with Qwen3.5-27B. On ALFWorld, the respective mean success rates are 79.9% and 76.6% with Qwen3-4B and 90.3% and 90.8% with Qwen3-8B. The full configuration improves mean performance over the inner-only configuration in all five settings. With a shared MPO starting point and the same inference model, these gains support the contribution of learned cross-task knowledge through initial refinement and subsequent revision.

5 Conclusion

We introduced MetaDecomp, an in-context meta-learning framework that jointly learns a state space and transition DAGs in that space for task decomposition. Its outer loop revisits past executions to reorganize procedures as progress distinctions evolve, while its inner loop uses the resulting procedural memory to adapt decompositions through environmental feedback. On ScienceWorld and ALFWorld, MetaDecomp achieves the highest mean performance among the evaluated methods in four of five benchmark–model settings. The full configuration achieves higher mean performance than within-task revision alone in all five settings. Relative to MPO, MetaDecomp achieves higher performance with fewer inference tokens on ScienceWorld and more on ALFWorld. These results support using learned procedural memory for initial decomposition and feedback-driven adaptation.

Ethics Statement

Our experiments use the simulated ALFWorld and ScienceWorld benchmarks and do not involve human participants or deployment in physical environments. Procedural memory can propagate incorrect or inappropriate behaviors across tasks. Practical deployment should therefore validate stored procedures, restrict permitted actions, and provide appropriate human oversight.

Reproducibility Statement

Section 3 and Appendix B describe the procedural-memory representation and the outer- and inner-loop algorithms. Appendix A.1 reports model configurations, generation and execution limits, evaluation task counts, the three evaluation seeds, and representative prompt excerpts. Appendices A.2 and A.3 specify the paired statistical tests and inference-token accounting. The theoretical assumptions and proofs are given in Appendix B.4. Code will be released upon acceptance.

AI Use Statement

We used generative AI tools to improve the language and readability of the manuscript and to assist with code implementation and debugging. The authors take full responsibility for the final text, code, and reported results, including all AI-assisted contributions.

References

Zergham Ahmed, Kazuki Irie, Joshua B. Tenenbaum, Christopher J. Bates, and Samuel J. Gershman. Learning abstractions for hierarchical planning in program-synthesis agents. *arXiv preprint arXiv:2602.00929*, 2026. URL https://arxiv.org/abs/2602.00929.

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.

Blai Bonet and Hector Geffner. Learning first-order symbolic representations for planning from the structure of the state space. In *Proc. ECAI*, pp. 2322–2329. IOS Press, 2020. doi: 10.3233/FAIA200361. URL https://bonetblai.github.io/reports/ECAI20-learning-strips.pdf.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. *arXiv preprint arXiv:2512.10696*, 2025. URL https://arxiv.org/abs/2512.10696.

Kutluhan Erol, James Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *Proc. AAAI*, pp. 1123–1128, 1994. URL https://www.cs.umd.edu/~nau/papers/erol1994htn.pdf.

Shengda Fan, Xin Cong, Zhong Zhang, Yuepeng Fu, Yesai Wu, Hao Wang, Xinyu Zhang, Enrui Hu, and Yankai Lin. Generalizing experience for language agents with hierarchical MetaFlows. In *Proc. NeurIPS*, volume 38. Curran Associates, Inc., 2025. doi: 10.52202/085713-2149. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/5c882988ce5fac487974ee4f415b96a9-Abstract-Conference.html.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. MemP: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502. Association for Computational Linguistics, 2026. URL https://aclanthology.org/2026.findings-acl.866.pdf.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proc. ICML*, volume 70 of *Proceedings of Machine Learning Research*, pp.

1126–1135. PMLR, 2017. URL https://proceedings.mlr.press/v70/finn17a.html.

Saman Forouzandeh, Wei Peng, Parham Moradi, Xinghuo Yu, and Mahdi Jalili. Learning hierarchical procedural memory for LLM agents through Bayesian selection and contrastive refinement. *arXiv preprint arXiv:2512.18950*, 2025. URL https://arxiv.org/abs/2512.18950.

Yu Hao, Jinxuan Cai, Qi Zhang, Yawen Li, Zhiqiang Zhang, Chuan Shi, and Cheng Yang. HiSkill: Empowering LLM agents with hierarchical skill graphs. *arXiv preprint arXiv:2607.25853*, 2026. URL https://arxiv.org/abs/2607.25853.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9): 5149–5169, 2022. doi: 10.1109/TPAMI.2021.3079209.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *Proc. ICML*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9118–9147. PMLR, 2022. URL https://proceedings.mlr.press/v162/huang22a.html.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. In *Proc. ICLR*, 2023. URL https://arxiv.org/abs/2210.02406.

George Konidaris, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018. doi: 10.1613/jair.5575. URL https://jair.org/index.php/jair/article/view/11175.

Lihong Li, Thomas J. Walsh, and Michael L. Littman. Towards a unified theory of state abstraction for MDPs. In *Proc. ISAIM*, 2006. URL https://thomasjwalsh.net/pub/aima06Towards.pdf.

Sirui Liang, Pengfei Cao, Jian Zhao, Wenhao Teng, Xiangwen Liao, Jun Zhao, and Kang Liu. Learning how to remember: A meta-cognitive management method for structured and transferable agent memory. *arXiv preprint arXiv:2601.07470*, 2026. URL https://arxiv.org/abs/2601.07470.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. ADaPT: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4226–4252, Mexico City, Mexico, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-naacl.264.

Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Liangyu Li, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. AutoRefine: Compiling trajectories into validated typed agent artifacts, 2026. URL https://arxiv.org/abs/2601.22758.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *Proc. ICML*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1842–1850. PMLR, 2016. URL https://proceedings.mlr.press/v48/santoro16.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Proc. NeurIPS*, volume 36, 2023. doi: 10.52202/075280-0377.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *Proc. ICLR*, 2021. URL https://arxiv.org/abs/2010.03768.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. AdaPlanner: Adaptive planning from feedback with language models. In *Proc. NeurIPS*, volume 36. Curran Associates, Inc., 2023. doi: 10.52202/075280-2537.

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

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Proc. EMNLP*, pp. 11279–11298, Abu Dhabi, United Arab Emirates, 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.775.

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

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *Proc. COLM*, 2024. URL https://www.microsoft.com/en-us/research/wp-content/uploads/2023/08/LLM_agent.pdf.

Weimin Xiong, Yifan Song, Qingxiu Dong, Bingchan Zhao, Feifan Song, Xun Wang, and Sujian Li. MPO: Boosting LLM agents with meta plan optimization. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 3914–3935, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-emnlp.210.

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.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. *arXiv preprint arXiv:2502.12110*, 2025. URL https://arxiv.org/abs/2502.12110.

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.

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

GLM-5-Team: Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui,

Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. GLM-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. MemEvolve: Meta-evolution of agent memory systems. *arXiv preprint arXiv:2512.18746*, 2025. URL https://arxiv.org/abs/2512.18746.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Y. Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *Proc. ICLR*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8a94ff6f922d995d7d3f4ebf4143e442-Abstract-Conference.html.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

A Experimental Details

A.1 Implementation Details

ScienceWorld uses 211 test instances from 30 native task types, grouped into ten reporting categories. ALFWorld uses 134 unseen instances across six task types. Table 5 lists their sizes. Evaluation measures transfer to new instances within these task types. Each learning space maintains its own state space and transition DAGs and may cover a native task type or a group of types. For example, ScienceWorld Qwen3.5-9B shares one learning space across all four Matter subtypes.

Benchmark	Category	Test tasks
ScienceWorld	Changes of State (Matter)	30
	Measurement Instrument (Meas.)	23
	Electrical Circuits (Elec.)	25
	Classification (Class.)	29
	Growing Plants (Plants)	14
	Chemistry (Chem.)	22
	Life Spans (Life-1)	21
	Life Stages (Life-2)	9
	Forces (Force)	23
	Mendelian Genetics (Mende.)	15
ALFWorld	Pick & Place (Pick)	24
	Clean & Place (Clean)	31
	Heat & Place (Heat)	23
	Cool & Place (Cool)	21
	Examine in Light (Look)	18
	Pick Two & Place (Pick-Two)	17

Table 5: Category sizes for the fixed evaluation panels. Categories use the original benchmark names and follow the main-table order, with the corresponding main-table abbreviations in parentheses. ScienceWorld contains 211 tasks and ALFWorld contains 134 tasks. These counts determine the weights in Avg.

For ScienceWorld, let $r_i = \max(0, \text{finalScore}_i/100)$ be the normalized score at episode termination, including partial credit. For ALFWorld, r_i is a binary success indicator. With n_c tasks in category c, its reported score is $s_c = 100 n_c^{-1} \sum_{i \in c} r_i$, and the overall score is

$$\text{Avg.} = \frac{\sum_c n_c s_c}{\sum_c n_c} = \frac{100}{N} \sum_{i=1}^{N} r_i. \tag{7}$$

Missing final scores contribute zero, and episodes with errors remain in the denominator. For three run-level scores x_1, x_2, x_3, we report the mean $\bar{x} = \frac{1}{3}\sum_{j=1}^{3} x_j$ and sample standard deviation $s = \sqrt{\frac{1}{2}\sum_{j=1}^{3}(x_j - \bar{x})^2}$. Avg. is computed within each run using Equation 7, then summarized across runs. Scores are displayed to one decimal place; differences are computed before rounding.

The actor models are Qwen3-8B, Qwen3.5-9B, and Qwen3.5-27B on ScienceWorld, and Qwen3-4B and Qwen3-8B on ALFWorld. In each setting, initial plan refinement and the online controller use the same Qwen model as the actor. The controller performs state projection, target selection, and plan revision. In MetaDecomp, GLM-5.3-Flash is used exclusively for outer-loop learning during training but never called during inference. We follow the original algorithms for all baselines, including AutoRefine's and SkillOpt's learning and validation procedures and their test-time use of learned artifacts. MPO uses the official planner checkpoint to generate the cached baseline decompositions. AdaPlanner uses the official ALFWorld notebook implementation, adapted to the ScienceWorld interface. AutoRefine and SkillOpt use GLM-5.3-Flash for experience distillation during training and freeze the resulting libraries before testing.

We evaluate with three execution seeds, 1234 (the default in our Qwen evaluation setup), 1235, and 1236.

MetaDecomp trains on an ordered stream of task instances. Each pair of successful executions from the same learning space, with final normalized reward equal to one, triggers a joint update of the state space and transition DAGs. The state space and transition DAGs remain fixed within each episode and throughout testing, while the task-specific decomposition can be revised using observations. Appendix B specifies the update and adaptation procedures.

The inner-only and full configurations in Table 4 share MPO's cached decomposition D^{base} and the same Qwen actor and controller models. The inner-only configuration uses an empty procedural memory, preserves the decomposition in dependency order without an initial refinement call, and enables revision from execution feedback. The full configuration uses learned knowledge in both initial refinement and online revision, as specified in Appendix B.3. The row with neither loop reports MPO. All configurations are evaluated on the complete test panels with the same three seeds and report means and sample standard deviations.

Each online review recalls at most 3 target states and 3 shortest simple state paths per target. The default per-call output-token limits are 65,536 for the outer learner, 2,048 for the actor, 8,192 for initial plan refinement and the online controller. Actor and controller temperatures are zero. Episodes allow at most 50 interaction steps on ALFWorld or 100 on ScienceWorld, and 12 plan reviews. The execution budget is 1,000,000 input-plus-output tokens per episode for actor and online-review calls. Outer-loop learning and initial plan refinement are outside this execution budget.

The plan reviewer receives the full visible episode on ALFWorld and recent events together with retrieved earlier context on ScienceWorld. State projection and target selection use the full execution graph.

A.1.1 PROMPT EXCERPTS

The following excerpts illustrate state-space learning, transition-DAG aggregation, and online revision, with omissions marked by `[...]`. D0 denotes the cached decomposition D^{base}, D1 the current decomposition, and a stored method is represented by a transition DAG.

State-space learning.

> Stage 1 — Propose necessary additions to the reusable state schema.
>
> Use the supplied complete task executions jointly. Task descriptions define requirements; public observations define facts. Do not use hidden state, D0/D1, intended effects, or future outcomes as intermediate-state evidence.
>
> [...]
>
> Add a field or value only if all are true:
>
> 1. The distinction is supported at identifiable execution boundaries.
> 2. It changes remaining task-relevant work, applicability of an evidenced reusable method, or task satisfaction.
> 3. The current schema cannot already represent or derive it.
> 4. It is reusable across concrete instances and is not merely an action parameter or method-internal condition.

Transition-DAG aggregation.

> Stage 4 — Aggregate evidence into methods for one fixed directed state pair.
>
> [...]
>
> Use each group's own supporting/context evidence to establish its executed actions, effects, and dependencies. Cross-group evidence or existing method support may be used only to support reusable decision reasoning, not to supply missing executed actions or effects.
>
> [...]
>
> Require:
>
> - the complete action DAG to support the fixed source-to-target transition;
> - every retained action to remain atomic;
> - prerequisites of the first action to hold at the source boundary;
> - later prerequisites to be established by earlier retained actions when needed.
>
> [...]
>
> Dependencies reference only actions in that method and encode necessary acyclic precedence.

Online plan revision.

> Decide whether the current D1 needs repair after the reported stall. Use `task_description`, actual execution evidence, and current D1. Experience from the procedural memory is optional supporting knowledge, not a goal by itself.
>
> [...]
>
> Only execution evidence establishes that a prerequisite currently holds. If it does not hold, retain suitable earlier D1 steps that establish it, or add the missing work.

A.2 Statistical Comparisons

We pair outcomes by test task and average each task's score over the three evaluation seeds. For each baseline, d_i is the difference between MetaDecomp's and the baseline's seed-averaged scores on task i, and $\Delta = N^{-1} \sum_i d_i$. We use exact two-sided paired randomization tests with $|\Delta|$ as the statistic, swapping method labels independently across tasks and jointly across all three seeds within each task. Exact two-sided tail probabilities are computed by dynamic programming over integer-valued task-wise score differences summed across the three seeds. The test assumes independent task pairs and, under the null, exchangeable method labels within each pair. The analysis uses 134 task pairs on ALFWorld and 211 on ScienceWorld and compares MetaDecomp with all five baselines in each setting. We report these as exploratory pairwise comparisons, with significance assessed separately for each comparison at $p < 0.05$.

Table 6 also reports pointwise 95% percentile bootstrap confidence intervals from 100,000 paired task resamples, stratified by the categories in Table 5. Each resample preserves category sizes and keeps a task's methods and seeds together. These intervals quantify task-level uncertainty conditional on the three evaluation runs.

Table 6: Exploratory task-level paired comparisons of MetaDecomp with every main-table baseline. Positive Δ favors MetaDecomp, in success-rate percentage points for ALFWorld and score points for ScienceWorld. Confidence intervals are pointwise. The p-values are from exact two-sided paired randomization tests, with significance assessed per comparison at $p < 0.05$.

Baseline	Δ	95% CI	p
ALFWorld / Qwen3-4B			
ReAct	$+16.92$	$[10.45, 23.38]$	< 0.0001
AdaPlanner	$+71.89$	$[66.67, 76.62]$	< 0.0001
MPO	$+6.47$	$[0.25, 12.94]$	0.0543
AutoRefine	$+16.17$	$[9.45, 22.89]$	< 0.0001
SkillOpt	$+15.92$	$[9.70, 22.39]$	< 0.0001
ALFWorld / Qwen3-8B			
ReAct	$+14.93$	$[9.95, 19.90]$	< 0.0001
AdaPlanner	$+82.34$	$[78.11, 86.32]$	< 0.0001
MPO	$+4.23$	$[1.00, 7.71]$	0.0217
AutoRefine	$+14.68$	$[9.45, 19.65]$	< 0.0001
SkillOpt	$+8.96$	$[4.48, 13.68]$	0.0005
ScienceWorld / Qwen3-8B			
ReAct	$+8.77$	$[3.80, 13.67]$	0.0013
AdaPlanner	$+41.47$	$[36.79, 46.11]$	< 0.0001
MPO	$+5.27$	$[1.13, 9.46]$	0.0196
AutoRefine	$+11.04$	$[6.17, 15.89]$	< 0.0001
SkillOpt	$+6.41$	$[1.60, 11.24]$	0.0121
ScienceWorld / Qwen3.5-9B			
ReAct	$+12.99$	$[8.13, 17.82]$	< 0.0001
AdaPlanner	$+36.84$	$[32.03, 41.60]$	< 0.0001
MPO	$+13.74$	$[8.99, 18.55]$	< 0.0001
AutoRefine	$+14.32$	$[9.48, 19.12]$	< 0.0001
SkillOpt	$+2.68$	$[-2.03, 7.35]$	0.2825

Continued on next page

Table 6 continued

Baseline	Δ	95% CI	p
	ScienceWorld / Qwen3.5-27B		
ReAct	$+8.49$	$[4.13, 12.90]$	0.0003
AdaPlanner	$+31.48$	$[26.30, 36.64]$	< 0.0001
MPO	$+8.69$	$[5.01, 12.48]$	< 0.0001
AutoRefine	$+11.73$	$[7.29, 16.20]$	< 0.0001
SkillOpt	-3.39	$[-7.33, 0.58]$	0.1146

A.3 Inference Cost Analysis

Tables 3 and 7 report inference-time input and output tokens in thousands per test task for all methods, including successful and failed episodes. For both MPO and MetaDecomp, accounting starts from the cached initial decomposition. MetaDecomp's costs include initial plan refinement, actor calls, and online reviews. Costs use the same three evaluation runs as the main performance tables. We compute task-count-weighted averages within each run, then report their mean and sample standard deviation across the three runs. Total cost is calculated by summing input and output consumption within each run before aggregation. Percentage comparisons in Section 4.3 use mean total costs. A category's contribution to the panel-level mean cost difference is its fraction of the test tasks multiplied by the difference in mean total cost per task.

Table 7: Category-level evaluation-time inference tokens per task, in thousands, using the evaluations and accounting in Table 3. Each entry reports the mean with sample standard deviation in subscript. Input and output are listed on separate rows. N is the number of tasks per run. Avg. weights categories by N within each run before computing the mean and standard deviation across runs.

Category	N	Tokens	ReAct	AdaPlanner	MPO	AutoRefine	SkillOpt	Ours
			ScienceWorld, Qwen3-8B					
Matter	30	Input	$122.15_{\pm 29.85}$	$36.40_{\pm 1.65}$	$154.23_{\pm 16.12}$	$140.25_{\pm 6.30}$	$215.08_{\pm 31.78}$	$107.43_{\pm 16.72}$
		Output	$1.39_{\pm 0.34}$	$4.45_{\pm 0.46}$	$1.94_{\pm 0.15}$	$1.60_{\pm 0.12}$	$1.50_{\pm 0.33}$	$2.32_{\pm 0.33}$
Meas.	23	Input	$131.57_{\pm 16.68}$	$30.98_{\pm 1.85}$	$194.58_{\pm 12.13}$	$148.19_{\pm 6.67}$	$227.37_{\pm 26.99}$	$117.25_{\pm 20.01}$
		Output	$1.57_{\pm 0.25}$	$4.05_{\pm 0.55}$	$2.18_{\pm 0.39}$	$1.64_{\pm 0.18}$	$0.83_{\pm 0.07}$	$2.47_{\pm 0.40}$
Elec.	25	Input	$222.49_{\pm 5.31}$	$32.28_{\pm 1.63}$	$249.50_{\pm 3.54}$	$220.70_{\pm 4.04}$	$451.65_{\pm 10.48}$	$126.44_{\pm 18.41}$
		Output	$2.55_{\pm 0.07}$	$3.62_{\pm 0.70}$	$2.61_{\pm 0.10}$	$2.51_{\pm 0.18}$	$3.55_{\pm 0.21}$	$2.51_{\pm 0.43}$
Class.	29	Input	$139.19_{\pm 1.93}$	$31.72_{\pm 1.47}$	$51.95_{\pm 5.64}$	$123.41_{\pm 20.53}$	$221.26_{\pm 5.64}$	$52.36_{\pm 11.25}$
		Output	$1.42_{\pm 0.17}$	$3.76_{\pm 0.59}$	$0.77_{\pm 0.07}$	$1.13_{\pm 0.25}$	$1.18_{\pm 0.07}$	$1.54_{\pm 0.17}$
Plants	14	Input	$213.52_{\pm 36.14}$	$51.49_{\pm 1.15}$	$202.66_{\pm 18.74}$	$212.30_{\pm 19.67}$	$314.62_{\pm 5.77}$	$222.74_{\pm 16.94}$
		Output	$2.78_{\pm 0.64}$	$11.69_{\pm 0.90}$	$1.83_{\pm 0.14}$	$2.90_{\pm 0.20}$	$2.75_{\pm 0.23}$	$4.04_{\pm 0.42}$
Chem.	22	Input	$147.03_{\pm 23.01}$	$47.08_{\pm 0.91}$	$116.13_{\pm 10.65}$	$161.12_{\pm 28.57}$	$248.42_{\pm 68.77}$	$79.28_{\pm 4.86}$
		Output	$1.62_{\pm 0.34}$	$9.57_{\pm 0.70}$	$1.35_{\pm 0.15}$	$1.81_{\pm 0.48}$	$1.56_{\pm 0.35}$	$1.68_{\pm 0.31}$
Life-1	21	Input	$40.26_{\pm 13.50}$	$26.46_{\pm 3.39}$	$21.05_{\pm 4.75}$	$40.51_{\pm 10.50}$	$12.60_{\pm 0.11}$	$15.04_{\pm 4.12}$
		Output	$0.89_{\pm 0.24}$	$2.99_{\pm 1.49}$	$0.60_{\pm 0.09}$	$0.86_{\pm 0.19}$	$0.17_{\pm 0.00}$	$0.44_{\pm 0.07}$
Life-2	9	Input	$136.98_{\pm 20.16}$	$33.75_{\pm 1.28}$	$89.89_{\pm 25.36}$	$147.94_{\pm 38.29}$	$285.01_{\pm 120.34}$	$65.26_{\pm 16.88}$
		Output	$2.29_{\pm 0.46}$	$3.32_{\pm 0.71}$	$1.70_{\pm 0.37}$	$2.35_{\pm 0.30}$	$2.43_{\pm 0.97}$	$2.30_{\pm 0.31}$
Force	23	Input	$124.18_{\pm 32.71}$	$37.03_{\pm 0.94}$	$99.11_{\pm 28.74}$	$111.34_{\pm 15.60}$	$69.14_{\pm 10.68}$	$85.81_{\pm 40.67}$
		Output	$2.59_{\pm 0.50}$	$6.22_{\pm 0.38}$	$1.97_{\pm 0.54}$	$2.19_{\pm 0.44}$	$0.94_{\pm 0.05}$	$2.47_{\pm 0.73}$
Mende.	15	Input	$121.90_{\pm 14.15}$	$25.11_{\pm 3.90}$	$91.47_{\pm 1.89}$	$113.16_{\pm 11.58}$	$63.13_{\pm 33.69}$	$83.24_{\pm 40.97}$
		Output	$2.03_{\pm 0.05}$	$2.83_{\pm 0.78}$	$1.54_{\pm 0.10}$	$1.77_{\pm 0.15}$	$0.76_{\pm 0.28}$	$1.76_{\pm 0.46}$
Avg.	211	Input	$138.75_{\pm 6.04}$	$34.95_{\pm 0.78}$	$128.63_{\pm 4.97}$	$140.61_{\pm 4.40}$	$211.50_{\pm 9.35}$	$92.83_{\pm 1.30}$
		Output	$1.83_{\pm 0.09}$	$5.11_{\pm 0.27}$	$1.65_{\pm 0.13}$	$1.79_{\pm 0.12}$	$1.51_{\pm 0.13}$	$2.09_{\pm 0.01}$
			ScienceWorld, Qwen3.5-9B					
Matter	30	Input	$197.62_{\pm 41.92}$	$50.44_{\pm 2.31}$	$125.68_{\pm 22.62}$	$197.20_{\pm 17.86}$	$357.77_{\pm 36.91}$	$174.25_{\pm 60.54}$
		Output	$7.61_{\pm 1.82}$	$9.99_{\pm 0.23}$	$4.50_{\pm 1.20}$	$7.19_{\pm 0.54}$	$3.68_{\pm 1.12}$	$6.99_{\pm 2.32}$
Meas.	23	Input	$126.23_{\pm 13.17}$	$16.86_{\pm 1.22}$	$207.84_{\pm 53.65}$	$106.31_{\pm 32.96}$	$79.47_{\pm 15.50}$	$87.21_{\pm 33.34}$
		Output	$6.17_{\pm 0.69}$	$4.88_{\pm 0.19}$	$8.30_{\pm 2.06}$	$4.80_{\pm 1.23}$	$1.49_{\pm 0.52}$	$3.96_{\pm 1.05}$
Elec.	25	Input	$205.54_{\pm 22.91}$	$35.44_{\pm 2.18}$	$373.70_{\pm 38.60}$	$187.17_{\pm 45.60}$	$537.54_{\pm 43.45}$	$193.87_{\pm 51.23}$
		Output	$7.68_{\pm 1.43}$	$6.78_{\pm 0.89}$	$14.65_{\pm 1.60}$	$7.28_{\pm 1.84}$	$6.90_{\pm 0.69}$	$8.99_{\pm 1.12}$

Continued on next page

Table 7 continued

Category	N	Tokens	ReAct	AdaPlanner	MPO	AutoRefine	SkillOpt	Ours
Class.	29	Input	$82.46_{\pm 25.81}$	$39.58_{\pm 2.43}$	$30.73_{\pm 6.51}$	$54.33_{\pm 12.07}$	$125.24_{\pm 38.69}$	$28.72_{\pm 3.88}$
		Output	$3.82_{\pm 1.25}$	$7.79_{\pm 0.73}$	$1.12_{\pm 0.27}$	$2.31_{\pm 0.45}$	$1.64_{\pm 0.74}$	$1.79_{\pm 0.73}$
Plants	14	Input	$317.63_{\pm 53.56}$	$20.33_{\pm 0.96}$	$96.92_{\pm 14.51}$	$328.16_{\pm 40.97}$	$356.11_{\pm 29.53}$	$219.47_{\pm 55.70}$
		Output	$11.93_{\pm 1.95}$	$4.58_{\pm 0.42}$	$2.67_{\pm 0.61}$	$12.64_{\pm 1.38}$	$3.48_{\pm 0.60}$	$11.22_{\pm 5.05}$
Chem.	22	Input	$169.45_{\pm 38.32}$	$38.23_{\pm 2.01}$	$146.68_{\pm 25.60}$	$153.69_{\pm 53.59}$	$238.82_{\pm 18.00}$	$108.28_{\pm 39.23}$
		Output	$6.60_{\pm 1.98}$	$6.54_{\pm 0.39}$	$5.62_{\pm 1.44}$	$5.92_{\pm 1.85}$	$3.59_{\pm 0.45}$	$5.19_{\pm 1.96}$
Life-1	21	Input	$16.84_{\pm 6.24}$	$38.71_{\pm 3.38}$	$10.69_{\pm 0.60}$	$18.49_{\pm 6.23}$	$26.79_{\pm 0.13}$	$18.44_{\pm 2.41}$
		Output	$0.87_{\pm 0.21}$	$9.23_{\pm 0.80}$	$0.61_{\pm 0.03}$	$1.04_{\pm 0.25}$	$0.29_{\pm 0.02}$	$1.01_{\pm 0.18}$
Life-2	9	Input	$169.93_{\pm 37.20}$	$31.88_{\pm 1.14}$	$78.52_{\pm 37.33}$	$212.54_{\pm 82.91}$	$241.11_{\pm 38.92}$	$94.65_{\pm 9.72}$
		Output	$7.16_{\pm 2.04}$	$7.69_{\pm 0.44}$	$2.75_{\pm 1.37}$	$7.59_{\pm 1.99}$	$3.38_{\pm 0.52}$	$4.96_{\pm 1.57}$
Force	23	Input	$140.38_{\pm 67.20}$	$38.36_{\pm 1.47}$	$74.92_{\pm 26.53}$	$163.11_{\pm 17.97}$	$38.52_{\pm 0.12}$	$32.13_{\pm 1.68}$
		Output	$6.72_{\pm 3.36}$	$8.38_{\pm 0.44}$	$3.79_{\pm 1.60}$	$7.77_{\pm 0.61}$	$0.37_{\pm 0.00}$	$1.78_{\pm 0.22}$
Mende.	15	Input	$47.04_{\pm 16.58}$	$28.35_{\pm 2.09}$	$28.90_{\pm 5.80}$	$73.53_{\pm 69.98}$	$140.13_{\pm 25.90}$	$30.87_{\pm 5.08}$
		Output	$2.31_{\pm 0.76}$	$7.16_{\pm 0.64}$	$1.98_{\pm 0.32}$	$3.74_{\pm 2.54}$	$2.08_{\pm 1.43}$	$2.23_{\pm 0.30}$
Avg.	211	Input	$143.86_{\pm 1.98}$	$35.39_{\pm 0.71}$	$125.39_{\pm 11.84}$	$140.98_{\pm 2.02}$	$216.07_{\pm 6.84}$	$98.62_{\pm 3.77}$
		Output	$5.96_{\pm 0.19}$	$7.48_{\pm 0.06}$	$4.93_{\pm 0.43}$	$5.72_{\pm 0.06}$	$2.69_{\pm 0.10}$	$4.69_{\pm 0.70}$
ScienceWorld, Qwen3.5-27B								
Matter	30	Input	$102.48_{\pm 9.66}$	$29.01_{\pm 2.65}$	$85.83_{\pm 12.79}$	$100.91_{\pm 25.54}$	$93.66_{\pm 8.25}$	$124.66_{\pm 18.53}$
		Output	$3.74_{\pm 0.40}$	$5.93_{\pm 0.36}$	$2.92_{\pm 0.39}$	$4.02_{\pm 1.09}$	$1.70_{\pm 0.17}$	$5.58_{\pm 0.93}$
Meas.	23	Input	$36.37_{\pm 2.05}$	$11.67_{\pm 1.19}$	$66.63_{\pm 9.36}$	$42.06_{\pm 2.28}$	$46.95_{\pm 0.96}$	$66.79_{\pm 6.90}$
		Output	$1.81_{\pm 0.06}$	$3.55_{\pm 0.15}$	$2.34_{\pm 0.34}$	$2.09_{\pm 0.17}$	$0.87_{\pm 0.03}$	$2.96_{\pm 0.23}$
Elec.	25	Input	$216.56_{\pm 33.13}$	$33.51_{\pm 1.53}$	$218.06_{\pm 3.06}$	$215.32_{\pm 20.85}$	$286.85_{\pm 3.85}$	$144.14_{\pm 26.86}$
		Output	$7.47_{\pm 1.07}$	$6.84_{\pm 0.13}$	$7.36_{\pm 0.06}$	$7.35_{\pm 0.73}$	$3.77_{\pm 0.06}$	$5.94_{\pm 1.03}$
Class.	29	Input	$18.08_{\pm 1.83}$	$29.98_{\pm 0.78}$	$36.51_{\pm 22.55}$	$32.85_{\pm 23.69}$	$25.36_{\pm 0.09}$	$35.28_{\pm 5.62}$
		Output	$0.77_{\pm 0.06}$	$5.55_{\pm 0.04}$	$1.44_{\pm 0.97}$	$1.43_{\pm 1.24}$	$0.42_{\pm 0.01}$	$2.13_{\pm 0.52}$
Plants	14	Input	$201.48_{\pm 39.74}$	$24.34_{\pm 1.86}$	$145.54_{\pm 15.56}$	$191.64_{\pm 67.97}$	$115.78_{\pm 12.32}$	$87.22_{\pm 8.96}$
		Output	$7.62_{\pm 1.70}$	$5.92_{\pm 0.50}$	$5.05_{\pm 1.19}$	$7.36_{\pm 2.91}$	$1.37_{\pm 0.16}$	$3.31_{\pm 0.45}$
Chem.	22	Input	$44.87_{\pm 2.79}$	$27.86_{\pm 1.68}$	$54.61_{\pm 3.41}$	$50.71_{\pm 17.90}$	$49.32_{\pm 3.57}$	$46.35_{\pm 2.32}$
		Output	$1.73_{\pm 0.17}$	$5.12_{\pm 0.15}$	$2.03_{\pm 0.21}$	$2.09_{\pm 0.84}$	$0.80_{\pm 0.08}$	$1.56_{\pm 0.09}$
Life-1	21	Input	$7.54_{\pm 0.87}$	$18.43_{\pm 2.62}$	$7.16_{\pm 0.17}$	$7.75_{\pm 0.35}$	$11.54_{\pm 0.01}$	$14.64_{\pm 0.22}$
		Output	$0.67_{\pm 0.03}$	$4.71_{\pm 0.26}$	$0.61_{\pm 0.04}$	$0.63_{\pm 0.02}$	$0.35_{\pm 0.01}$	$1.24_{\pm 0.10}$
Life-2	9	Input	$208.47_{\pm 27.75}$	$25.02_{\pm 6.60}$	$42.27_{\pm 4.46}$	$210.78_{\pm 48.25}$	$53.79_{\pm 3.91}$	$92.62_{\pm 10.24}$
		Output	$6.85_{\pm 1.21}$	$4.82_{\pm 1.18}$	$1.65_{\pm 0.18}$	$6.58_{\pm 1.63}$	$0.89_{\pm 0.13}$	$3.49_{\pm 0.25}$
Force	23	Input	$38.83_{\pm 5.72}$	$29.56_{\pm 2.00}$	$29.72_{\pm 4.50}$	$44.70_{\pm 0.81}$	$43.63_{\pm 0.23}$	$28.80_{\pm 0.50}$
		Output	$1.83_{\pm 0.16}$	$6.86_{\pm 0.58}$	$1.81_{\pm 0.22}$	$2.03_{\pm 0.08}$	$0.83_{\pm 0.00}$	$1.63_{\pm 0.04}$
Mende.	15	Input	$109.68_{\pm 8.51}$	$17.00_{\pm 3.40}$	$26.69_{\pm 7.88}$	$109.45_{\pm 7.17}$	$109.30_{\pm 7.51}$	$70.74_{\pm 4.83}$
		Output	$4.32_{\pm 0.32}$	$4.55_{\pm 0.66}$	$1.51_{\pm 0.21}$	$4.29_{\pm 0.19}$	$2.28_{\pm 0.12}$	$3.84_{\pm 0.29}$
Avg.	211	Input	$86.40_{\pm 3.63}$	$25.34_{\pm 0.41}$	$73.33_{\pm 1.24}$	$89.38_{\pm 17.37}$	$84.70_{\pm 1.41}$	$71.13_{\pm 5.22}$
		Output	$3.27_{\pm 0.15}$	$5.48_{\pm 0.07}$	$2.72_{\pm 0.05}$	$3.45_{\pm 0.74}$	$1.34_{\pm 0.04}$	$3.22_{\pm 0.23}$
ALFWorld, Qwen3-8B								
Pick	24	Input	$24.01_{\pm 0.72}$	$20.77_{\pm 0.50}$	$22.27_{\pm 1.91}$	$22.59_{\pm 1.06}$	$46.06_{\pm 0.89}$	$34.82_{\pm 0.99}$
		Output	$0.72_{\pm 0.03}$	$8.05_{\pm 0.13}$	$0.64_{\pm 0.07}$	$0.68_{\pm 0.05}$	$0.50_{\pm 0.02}$	$1.31_{\pm 0.04}$
Clean	31	Input	$36.64_{\pm 1.69}$	$21.47_{\pm 0.26}$	$46.59_{\pm 2.94}$	$36.93_{\pm 0.27}$	$96.42_{\pm 4.03}$	$47.83_{\pm 0.94}$
		Output	$1.23_{\pm 0.08}$	$8.38_{\pm 0.11}$	$1.42_{\pm 0.09}$	$1.22_{\pm 0.02}$	$1.47_{\pm 0.04}$	$1.69_{\pm 0.03}$
Heat	23	Input	$64.62_{\pm 6.44}$	$18.85_{\pm 0.10}$	$44.01_{\pm 9.24}$	$59.55_{\pm 4.74}$	$72.35_{\pm 1.57}$	$44.74_{\pm 3.06}$
		Output	$2.27_{\pm 0.17}$	$7.38_{\pm 0.16}$	$1.31_{\pm 0.27}$	$2.12_{\pm 0.18}$	$1.05_{\pm 0.03}$	$1.54_{\pm 0.13}$
Cool	21	Input	$42.98_{\pm 2.06}$	$18.97_{\pm 0.20}$	$35.12_{\pm 3.32}$	$55.11_{\pm 2.37}$	$69.19_{\pm 5.43}$	$32.05_{\pm 1.34}$
		Output	$1.54_{\pm 0.09}$	$7.59_{\pm 0.15}$	$1.11_{\pm 0.10}$	$1.64_{\pm 0.05}$	$1.07_{\pm 0.10}$	$1.34_{\pm 0.03}$
Look	18	Input	$45.62_{\pm 3.04}$	$10.26_{\pm 1.01}$	$30.38_{\pm 5.63}$	$48.65_{\pm 4.09}$	$75.51_{\pm 5.38}$	$24.25_{\pm 1.09}$
		Output	$1.39_{\pm 0.11}$	$4.74_{\pm 0.41}$	$0.90_{\pm 0.19}$	$1.53_{\pm 0.16}$	$1.17_{\pm 0.12}$	$0.67_{\pm 0.04}$
Pick-Two	17	Input	$87.25_{\pm 6.64}$	$32.06_{\pm 0.21}$	$56.73_{\pm 6.41}$	$90.27_{\pm 2.07}$	$169.35_{\pm 15.05}$	$62.57_{\pm 3.31}$
		Output	$2.92_{\pm 0.29}$	$12.53_{\pm 0.15}$	$1.58_{\pm 0.32}$	$2.96_{\pm 0.11}$	$2.59_{\pm 0.20}$	$1.70_{\pm 0.07}$
Avg.	134	Input	$47.80_{\pm 0.57}$	$20.34_{\pm 0.16}$	$39.10_{\pm 1.33}$	$49.44_{\pm 1.17}$	$85.44_{\pm 1.40}$	$41.20_{\pm 0.38}$
		Output	$1.60_{\pm 0.03}$	$8.06_{\pm 0.08}$	$1.17_{\pm 0.05}$	$1.61_{\pm 0.05}$	$1.27_{\pm 0.02}$	$1.41_{\pm 0.01}$

Continued on next page

Table 7 continued

Category	N	Tokens	ReAct	AdaPlanner	MPO	AutoRefine	SkillOpt	Ours
					ALFWorld, Qwen3-4B			
Pick	24	Input	$28.56_{\pm 1.93}$	$19.84_{\pm 0.17}$	$26.53_{\pm 3.27}$	$30.76_{\pm 1.57}$	$68.42_{\pm 5.95}$	$33.81_{\pm 0.51}$
		Output	$0.40_{\pm 0.00}$	$7.34_{\pm 0.11}$	$0.38_{\pm 0.03}$	$0.42_{\pm 0.02}$	$0.46_{\pm 0.01}$	$0.82_{\pm 0.01}$
Clean	31	Input	$54.49_{\pm 5.25}$	$27.55_{\pm 0.57}$	$52.12_{\pm 2.61}$	$63.83_{\pm 3.73}$	$87.12_{\pm 0.55}$	$61.56_{\pm 5.46}$
		Output	$0.75_{\pm 0.07}$	$9.95_{\pm 0.18}$	$0.55_{\pm 0.02}$	$0.53_{\pm 0.03}$	$0.45_{\pm 0.01}$	$1.15_{\pm 0.06}$
Heat	23	Input	$70.37_{\pm 3.53}$	$24.72_{\pm 0.77}$	$45.63_{\pm 1.90}$	$71.72_{\pm 5.65}$	$107.82_{\pm 3.06}$	$57.90_{\pm 6.32}$
		Output	$0.81_{\pm 0.17}$	$9.16_{\pm 0.21}$	$0.51_{\pm 0.03}$	$0.82_{\pm 0.08}$	$0.50_{\pm 0.02}$	$1.14_{\pm 0.13}$
Cool	21	Input	$58.15_{\pm 2.20}$	$24.27_{\pm 0.96}$	$50.56_{\pm 3.70}$	$56.32_{\pm 9.94}$	$127.16_{\pm 3.90}$	$56.72_{\pm 3.03}$
		Output	$0.65_{\pm 0.03}$	$8.93_{\pm 0.30}$	$0.62_{\pm 0.04}$	$0.63_{\pm 0.09}$	$0.62_{\pm 0.07}$	$1.58_{\pm 0.04}$
Look	18	Input	$50.87_{\pm 4.47}$	$6.27_{\pm 1.26}$	$35.85_{\pm 1.84}$	$72.08_{\pm 15.00}$	$77.65_{\pm 3.30}$	$30.49_{\pm 2.14}$
		Output	$0.77_{\pm 0.07}$	$3.88_{\pm 0.39}$	$0.50_{\pm 0.02}$	$0.48_{\pm 0.05}$	$0.54_{\pm 0.01}$	$0.83_{\pm 0.04}$
Pick-Two	17	Input	$84.80_{\pm 1.89}$	$37.47_{\pm 0.23}$	$81.98_{\pm 1.78}$	$103.06_{\pm 7.04}$	$155.19_{\pm 10.27}$	$75.03_{\pm 1.15}$
		Output	$1.01_{\pm 0.03}$	$12.76_{\pm 0.12}$	$0.88_{\pm 0.03}$	$0.65_{\pm 0.02}$	$0.74_{\pm 0.02}$	$0.82_{\pm 0.11}$
Avg.	134	Input	$56.50_{\pm 0.97}$	$23.57_{\pm 0.44}$	$47.78_{\pm 2.35}$	$64.17_{\pm 3.43}$	$100.97_{\pm 2.80}$	$52.74_{\pm 2.13}$
		Output	$0.72_{\pm 0.02}$	$8.73_{\pm 0.11}$	$0.56_{\pm 0.02}$	$0.58_{\pm 0.04}$	$0.54_{\pm 0.01}$	$1.07_{\pm 0.02}$

A.4 Qualitative example of learned procedural memory

We examine a learned procedural memory for ALFWorld's Look task type, which requires examining a target object under a desklamp. The snapshot contains 12 retained state nodes and 18 transition DAGs. Its two roles bind the target object and desklamp to task entities. Six binary fields describe observed progress, recording whether the target object has been located, whether it is carried, whether the lamp has been located, whether it is on, whether the agent is at the lamp, and whether the task is satisfied. Location knowledge is distinct from physical presence, and task satisfaction is recorded separately from possession and illumination.

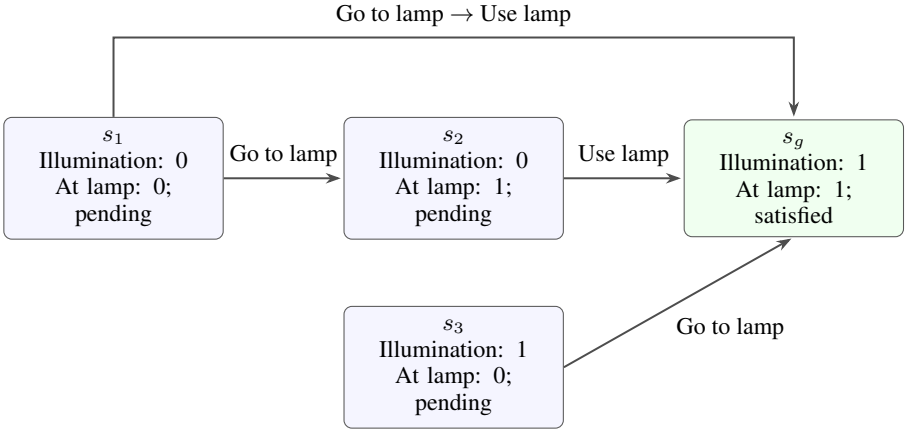

Figure 5: A subgraph of the learned procedural memory for ALFWorld Look tasks. In all four states, the target object and lamp have been located and the object is carried. Nodes show the remaining three fields. For illumination and presence at the lamp, 1 means currently established by observation and 0 means not established. Each edge is a stored transition DAG, with shortened operation names. The internal arrow in the upper edge's label denotes the dependency between its two operations. State assignments, transition endpoints, and DAG dependencies are preserved from the snapshot.

Figure 5 illustrates how these distinctions organize the remaining work. In state s_1, neither illumination nor presence at the lamp is established by the current evidence. The memory stores both a two-operation DAG that first goes to the lamp's location and then uses it, and separate transitions through s_2 for these operations. In s_3, illumination is established but the task remains pending. The stored transition reaches s_g by going to the lamp while carrying the object, without an additional use operation. The procedure instructions ground navigation in the receptacle or surface where the lamp was observed and require the agent to be there before using it.

The same snapshot also retains alternative procedures between one state pair. From a state where the lamp is located but the target object has not been found, two transition DAGs lead to a state where the object is located but not yet carried. One contains a single operation that repeatedly visits uninspected receptacles until the target is observed. The other orders three operations: visit a receptacle, open it if closed, and visit another receptacle if the object remains unfound. Their shared endpoints preserve a common progress change while their instructions retain different search behavior.

B Method Details

B.1 Representation and overall algorithm

B.1.1 State space, transition DAGs, and procedural memory

Each learning space covers a supplied task type or a predefined group of related types within one benchmark. Its learned schema is

$$\Sigma = \big(\mathcal{R}, \mathcal{F}, \{V_f, \iota_f, \sigma_f\}_{f\in\mathcal{F}}\big), \tag{8}$$

where $\mathcal{R}$ specifies task-relative roles and $\mathcal{F}$ is a finite set of progress fields. Each field f has finite values V_f, an initial value $\iota_f \in V_f$, and an observable interpretation σ_f. The schema defines $\mathcal{X}_\Sigma = \prod_{f\in\mathcal{F}} V_f$. Only retained assignments become state nodes in the memory, so the Cartesian product can include unsupported or unreachable combinations.

A state $s \in \mathcal{X}_\Sigma$ assigns one declared value to every field. The initial assignment $s_{\text{init}} = (\iota_f)_f$ marks progress not yet established. An unestablished condition need not be physically false. Current-condition fields track observed changes, while retained-knowledge fields follow their declared persistence conditions.

Role bindings β associate the referents in $\mathcal{R}$ with task entities and may remain unresolved. Complete field assignments determine state identity independently of these bindings.

The procedural memory $\mathcal{H} = (\mathcal{S}, \mathcal{T})$ consists of retained state nodes $\mathcal{S} \subseteq \mathcal{X}_\Sigma$ and transition DAGs. Each transition in $\mathcal{T}$ records a source state, a target state, and a transition DAG M_e,

$$e = (s_e^-, s_e^+, M_e), \qquad M_e = (A_e, E_e, x_e), \qquad s_e^- \neq s_e^+.$$

Here A_e contains parameterized operations, E_e encodes acyclic prerequisite dependencies inferred from execution evidence, and $x_e(a)$ gives the operation's instruction, including supported applicability, argument-selection, and stopping conditions. Alternative procedures may connect the same state pair. Thus $\mathcal{H}$ is a directed state multigraph whose state graph can contain cycles, although each transition DAG is acyclic. The dependencies guide planning rather than define an environment transition model.

B.1.2 Execution evidence and the two-loop cycle

The actor produces history $h_t = (o_0, a_1, o_1, \ldots, a_t, o_t)$, retained as an execution DAG of actual actions, observations, and execution order, together with o_0. This graph supplies evidence for state projection and learning. The actor is guided by an ordered decomposition $D_t = (d_{t,1}, \ldots, d_{t,n_t})$, initialized from a cached baseline DAG D^{base}. Planned instructions do not establish attained progress.

A boundary occurrence ω identifies the entry observation or the observation after an executed action. A local realization ℓ records boundaries $\omega_\ell^-, \omega_\ell^+$, their state assignments and role bindings, and evidence-supported operations proposed to connect their states. Occurrences remain distinct even when their assignments coincide. The evidence ledger $\mathcal{P}_b$ retains original executions, realizations, state assignments, and mappings from local operations to methods for subsequent reassessment.

Algorithm 1 holds $(\Sigma_b, \mathcal{H}_b)$ fixed within each episode while adapting D_t. Successful training executions enter a buffer for their learning space. At size B, the outer loop jointly updates the state space and transition DAGs using this buffer and the ledger. Before the first update, the schema is unavailable and the procedural memory is empty, so initialization uses the cached baseline while online revision uses observations. Evaluation freezes one snapshot $(\Sigma^*, \mathcal{H}^*)$ per learning space and excludes test executions from the training buffer.

Algorithm 1 MetaDecomp outer-loop learning and inner-loop adaptation

Require: Training tasks and cached plans for one learning space, modules ϕ, batch size B

Initialize Σ as unavailable, with empty procedural memory $\mathcal{H}$, ledger $\mathcal{P}$, and buffer $\mathcal{B}$
for each training task τ **do**
  $D \leftarrow \mathrm{Plan}_\phi(\tau, D^{\mathrm{base}}; \Sigma, \mathcal{H})$, preserving the cached plan if no methods exist
  Initialize $h \leftarrow (o_0)$ and hold $(\Sigma, \mathcal{H})$ fixed throughout the episode
  while execution budget remains and the episode is active **do**
    Execute an actor action guided by D and visible history
    Append the action and its observation to h
    if execution feedback triggers review **then**
      Project h into Σ when a schema is available
      Recall methods on paths from an exact stored-state match, if one exists
      Revise or retain D using execution evidence and any grounded methods
    end if
  end while
  if the training task succeeds **then**
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(\tau, h)\}$
  end if
  if $|\mathcal{B}| = B$ **then**
    Learn or extend schema Σ from $\mathcal{B}$ to define the state space
    If the schema changes, reproject historical boundaries using $\mathcal{P}$
    Extract observed boundaries, state changes, and supported operations from $\mathcal{B}$
    Group new and reassessed realizations by complete source–target assignments
    Migrate unambiguous old method associations and reassess split support
    Consolidate supported transition DAGs in $\mathcal{H}$, retaining alternative methods
    Publish updated $(\Sigma, \mathcal{H}, \mathcal{P})$ and clear $\mathcal{B}$
  end if
end for
For evaluation, freeze $(\Sigma^*, \mathcal{H}^*)$ and run only within-task planning and execution

B.2 Outer-loop learning of the state space and transition DAGs

An outer update maps the current snapshot, ledger, and batch $\mathcal{B}_b$ of successful training executions to $(\Sigma_{b+1}, \mathcal{H}_{b+1}, \mathcal{P}_{b+1})$. Separate model calls propose schema additions, extract local realizations, reproject retained historical boundaries, and aggregate methods. The program matches states, reconciles method support, and materializes the procedural memory between these calls.

B.2.1 Learning the state space and extracting evidence

The schema learner receives the current schema, task descriptions, and execution graphs in $\mathcal{B}_b$. It proposes observable distinctions that change remaining work, procedure applicability, or task satisfaction, transfer across instances, and are not already expressible. Independent dimensions occupy separate fields, while values of one field answer the same question. Task constants, concrete identities, temporary search locations, and unsupported values are excluded.

Schema updates may add roles, fields, or field values, preserving existing definitions, meanings, and initial values. Additive vocabulary changes can still revise historical assignments and need not satisfy the refinement assumption in Appendix B.4.

Under the updated schema, extraction assigns the entry boundary and selects later boundaries at observed state changes used by subsequent operations or marking task completion. Each retained boundary receives a complete field assignment and role bindings. Task descriptions identify requirements; observations establish attained conditions. The extractor proposes transitions between adjacent retained boundaries on the execution path.

Each realization ℓ contains operation intents and inferred dependencies. An intent describes one executed atomic operation or its repetition with supported argument-selection and stopping conditions. Distinct operations remain separate intents, parameterized by task roles or relations supported by the

task, state, and observations. Failed actions may provide context; invalid attempts and unexecuted plan steps cannot supply reusable operation evidence.

A nonentry boundary ω is the observation immediately after action $j_\tau(\omega)$. Let $\preceq_\tau$ denote equality or reachability in the execution DAG. Each retained local operation a cites a nonempty set of supporting actions satisfying

$$\emptyset \neq C_\ell(a) \subseteq \{j : j_\tau(\omega_\ell^-) \preceq_\tau j \preceq_\tau j_\tau(\omega_\ell^+)\}. \tag{9}$$

For an entry source, the lower bound is omitted. Equation 9 bounds admissible evidence citations and includes the source action. Extraction and aggregation are instructed to retain operations needed from the already attained source state, distinguishing them from actions that established it. The interval check does not itself enforce this semantic distinction. Context references identify observations but do not count as execution support. Realizations lacking a supported complete transition may be discarded.

B.2.2 Reprojecting historical boundaries

When Σ changes, the learner reprojects retained historical boundaries from their original execution graphs. Let $\mathcal{F}_\Delta$ contain new fields and fields with expanded value sets. Remapping has the form

$$q_{b+1}(\omega)_f = \begin{cases} \widehat{q}_{b+1}(\omega)_f, & f \in \mathcal{F}_\Delta, \\ q_b(\omega)_f, & f \in \mathcal{F}_b \setminus \mathcal{F}_\Delta, \end{cases} \tag{10}$$

where $\widehat{q}_{b+1}$ is the model's reassessment. Only new role bindings are added. Existing bindings, unaffected field values, boundary locations, and action evidence remain fixed.

Remapping is instructed to use only observations available through each boundary, accounting for intervening changes and declared persistence semantics.

New and reassessed realizations are grouped by exact equality of their complete source and target assignments, excluding role bindings, as in Equation 4.

B.2.3 Reconciling support and updating transition DAGs

Let $\psi(\omega)$ be a boundary's reassigned state identity, or $\perp$ for an excluded occurrence. For the complete support set $\mathrm{supp}(e)$ of an old transition e, its projected endpoint pairs are

$$\Pi_b(e) = \{(\psi(\omega_\ell^-), \psi(\omega_\ell^+)) : \ell \in \mathrm{supp}(e)\}. \tag{11}$$

A method migrates directly only if $\Pi_b(e) = \{(u, v)\}$ with $u, v \neq \perp$ and $u \neq v$. Split support supplies the old procedure as a candidate for reassessment at valid supported pairs, rather than copying to all endpoint combinations. Empty support, excluded endpoints, or coincident endpoints prevent direct migration. Changed pairs without a migrated method also enter review.

For each pair (u, v) under review, aggregation receives local realizations, supporting observations, and admissible existing methods. It groups procedures by operations, applicability, argument-selection and stopping semantics, and inferred dependencies, ignoring wording, concrete bindings, and ordering of independent operations. Evidence from another realization can inform reusable decision rules but cannot supply missing actions or effects for the current realization.

Let I_{uv} be the local operation intents at this pair, indexed by their realizations, and let A_m^+ be the resulting operations of a retained or proposed method m. Aggregation supplies the mapping

$$\mu : I_{uv} \longrightarrow \bigcup_m \big(\{m\} \times A_m^+\big) \cup \{\perp\}, \tag{12}$$

where $\perp$ denotes a discarded intent. Every intent is accounted for exactly once, all retained intents from one realization select the same method, and every new operation receives mapped execution evidence. Multiple intents may support one operation. Mappings and observations are retained in the ledger.

Aggregation refines supported operations and dependencies while leaving unselected methods unchanged. Dependencies remain acyclic and are transitively reduced. Distinct alternatives remain separate transition DAGs.

Newly extracted states are admitted only when used by a materialized method. Previously retained historical occurrences remain represented under their reassigned states. The update produces the revised schema, procedural memory, and evidence ledger. Representative prompts appear in Appendix A.1.1.

B.3 Inner-loop adaptation with procedural memory

The inner loop constructs and revises an ordered decomposition while holding Σ and $\mathcal{H} = (\mathcal{S}, \mathcal{T})$ fixed. State projection uses actor observations as evidence of attained progress; planned instructions and retrieved states do not establish it.

B.3.1 Initial decomposition from a cached baseline

Initial refinement aligns MPO's cached decomposition D^{base} with the learned state space and procedures. The baseline DAG is first ordered by dependency. The refinement model receives this sequence, the task, the schema, and all methods in the learning space, including their endpoints, instructions, and precedence edges. Initialization requires neither a stored-state match nor path selection.

Refinement can replace an instruction with one or more ordered steps, insert steps before or after it, or remove it. The resulting nonempty decomposition preserves the baseline order of unchanged instructions and the proposed order of inserted steps.

Grounding binds roles to the current task's entities, destination, and requirements. The model may combine partial methods, repeat operations, or add unstored work. Source states specify prerequisites rather than attained facts, and earlier planned steps must establish required progress. Reused operations must respect their dependencies, and repetition requires an observable stopping condition.

If the procedural memory contains no methods, initialization retains the ordered baseline.

B.3.2 Online state projection and method recall

When feedback triggers review, the state projector receives task requirements, the full execution graph, and the previous confirmed projection. Its prompt processes evidence in execution order, separates requirements from attained facts, and gives observations precedence over the previous projection. Ineffective actions alone do not revoke established progress.

A projection is complete when every field has a supported declared value, including any declared value for unknown or unestablished progress. If any field cannot be assigned such a value, the projection is incomplete and is omitted from review. Role bindings may remain unresolved.

A complete projection matches $\mathcal{S}$ by exact field equality, independently of role bindings. Without a match, the reviewer receives the projection but no recalled methods. Given a match $\widehat{s}_t$, target selection chooses from reachable stored states using the task, decomposition, execution graph, and stall information. It may select no targets. Retrieved methods are subsequently grounded in the current task and observations.

Let $\mathcal{G}_t$ denote the selected targets, with $|\mathcal{G}_t| \le G_{\max}$, and let $\mathcal{P}_K(\widehat{s}_t, g)$ contain up to K shortest simple state paths from $\widehat{s}_t$ to g. For the set $\mathcal{M}(u, v)$ of alternative methods stored on a state pair, recall returns

$$\mathcal{R}_t = \bigcup_{g \in \mathcal{G}_t} \bigcup_{p \in \mathcal{P}_K(\widehat{s}_t, g)} \bigcup_{(u,v) \in \text{edges}(p)} \{(u, v, M) : M \in \mathcal{M}(u, v)\}. \tag{13}$$

Paths are ranked by transition count, not operation count or execution cost. Each is retained in full with every alternative transition DAG on its state pairs, without a maximum path-length cutoff. Appendix A.1 specifies $G_{\max}$ and K. Review can proceed without recalled methods.

B.3.3 GROUNDED REVISION AND CONTINUED EXECUTION

The reviewer receives D_t, Σ, $\mathcal{R}_t$, and the context c_t, which includes execution evidence, stall information, any complete current projection, and the cached baseline when available. It infers completed and remaining work from observations and may retain an adequate plan.

Recalled target states are outcomes to attain, while source states specify prerequisites. Partial reuse requires omitted prerequisites to hold in the observed execution or be established by earlier instructions. Otherwise the reviewer adds or preserves the necessary work. Revision may also proceed without reusing a method.

Online revision uses only existing instructions as anchors. Each edit inserts one step before or after an anchor, or replaces that anchor with one step. For the lists $\mathbf{u}_i$ and $\mathbf{v}_i$ inserted before and after $d_{t,i}$, and its replacement or unchanged instruction $\widetilde{d}_{t,i}$, materialization is

$$D_t^+ = \Big\Vert_{i=1}^{n_t} \big(\mathbf{u}_i \| [\widetilde{d}_{t,i}] \| \mathbf{v}_i\big), \tag{14}$$

where $\|$ denotes concatenation. Each anchor permits at most one replacement, insertions at the same position preserve proposal order, and unmentioned instructions remain. Online edits do not delete, explicitly reorder, or replace an anchor by multiple steps.

The actor resumes with the current decomposition, as in Equation 6. Later observations may trigger further reviews while Σ and $\mathcal{H}$ remain fixed.

B.4 STATE-ABSTRACTION ANALYSIS AND PROOFS

The schema Σ defines a state space whose retained nodes and transition DAGs form the procedural memory $\mathcal{H} = (\mathcal{S}, \mathcal{T})$. We study two simplified questions to motivate this design. The first concerns procedure selection with fixed methods, continuation losses, and a history distribution. The second concerns connections between fixed historical occurrences when their endpoint assignments are refined. These comparisons describe the value and structural effects of retained information under their assumptions, rather than the performance of the complete learning and adaptation procedure.

B.4.1 DECISION SETTING AND VALUE OF REFINEMENT

Fix one learning space and a nonempty finite dictionary $\mathcal{M}$ of reusable methods. We compare selectors choosing among the same methods under the same continuation procedure. Let (H, C) have a fixed joint distribution, where H is the observed action–observation history and C contains the task information and role bindings available to every selector. The random history H is distinct from the procedural memory $\mathcal{H}$. All random variables are defined on a common probability space, and the history and context spaces are assumed to be standard Borel spaces so that the conditional quantities below admit measurable versions. All equalities between conditional quantities are understood almost surely.

For each $m \in \mathcal{M}$, let $\ell(m, h, c) \in [0, 1]$ be a measurable conditional expected continuation loss after grounding and choosing m at (h, c). The continuation procedure is fixed across the comparison. Subsequent environment and execution randomness is integrated into this loss, which can reflect failure, normalized execution cost, or their fixed combination. Any learned objects needed to specify the dictionary or continuation procedure are held fixed. The analysis therefore compares what can be achieved with different information about the same procedural decision.

A measurable projection q maps the history and common context to a complete schema state $S = q(H, C) \in \mathcal{X}_\Sigma$. A state-limited selector observes (S, C), whereas a history selector observes (H, C). Both receive the same C, so the comparison concerns the history information retained by q rather than a difference in task requirements or bindings. Define

$$\mathfrak{R}(q) = \mathbb{E}\left[\min_{m \in \mathcal{M}} \mathbb{E}[\ell(m, H, C) \mid S, C]\right], \qquad \mathfrak{R}_H = \mathbb{E}\left[\min_{m \in \mathcal{M}} \ell(m, H, C)\right]. \tag{15}$$

Boundedness of the losses makes both risks finite. We next verify that they are the optimal risks for their respective information sets, including when selection may be randomized.

Write

$$r_m(S,C) = \mathbb{E}[\ell(m,H,C) \mid S,C], \qquad m_q^*(S,C) \in \arg\min_{m\in\mathcal{M}} r_m(S,C).$$

Fix an ordering of the finite dictionary and choose the first minimizer whenever there is a tie. This gives a measurable deterministic selector because it requires only finitely many comparisons of measurable functions. A randomized state-limited selector is a probability vector $a(\cdot \mid S,C)$ measurable with respect to (S,C). Its internal randomization is conditionally independent of H given (S,C), so it cannot convey additional history information. Its risk is

$$\begin{aligned}\mathfrak{R}(a;q) &= \mathbb{E}\left[\sum_{m\in\mathcal{M}} a(m \mid S,C)\ell(m,H,C)\right] \\ &= \mathbb{E}\left[\sum_{m\in\mathcal{M}} a(m \mid S,C) r_m(S,C)\right] \\ &\geq \mathbb{E}\left[\min_{m\in\mathcal{M}} r_m(S,C)\right] = \mathfrak{R}(q).\end{aligned}$$

The second equality conditions on (S,C) and uses the measurability of the selection probabilities. The inequality holds pointwise because the conditional risk is a convex combination of the r_m. Choosing m_q^* attains equality, establishing $\inf_a \mathfrak{R}(a;q) = \mathfrak{R}(q)$. Thus randomization cannot lower this optimal risk. The same finite-minimum argument applied to $\ell(m,H,C)$ gives a measurable history selector attaining $\mathfrak{R}_H$. The history selector has access only to observations already obtained, so partial observability or unavoidable continuation loss can still leave $\mathfrak{R}_H > 0$.

Let q' be another measurable projection and write $S' = q'(H,C)$. We say that q' refines q when there is a measurable map π such that $q = \pi \circ q'$. In particular, the old state can be recovered from the new state without observing anything beyond it. Adding fields while preserving every old coordinate satisfies this condition by taking π to discard the new coordinates. This is a condition on the assignments induced by the projections, not merely on the names or number of schema entries.

Proposition 1 (Decision value of a state distinction)**.** *Suppose $q = \pi \circ q'$ and keep the dictionary $\mathcal{M}$, continuation losses, and joint distribution of (H,C) fixed. Let m_q^* be an optimal selector using (S,C) and let $r'_m(S',C) = \mathbb{E}[\ell(m,H,C) \mid S',C]$. Then*

$$\begin{aligned}&\mathfrak{R}_H \leq \mathfrak{R}(q') \leq \mathfrak{R}(q), \\ &\mathfrak{R}(q) - \mathfrak{R}(q') = \mathbb{E}\left[r'_{m_q^*(S,C)}(S',C) - \min_{m\in\mathcal{M}} r'_m(S',C)\right].\end{aligned} \tag{16}$$

The refinement has strictly positive value if and only if the coarse optimal selector is suboptimal under the refined information with positive probability.

Proof. Since $S = \pi(S')$, the information in (S,C) is contained in that in (S',C). For each fixed m, the tower property gives

$$r_m(S,C) = \mathbb{E}[r'_m(S',C) \mid S,C].$$

The coarse selector $m_q^*(S,C)$ is also measurable with respect to (S',C). Expanding its loss over the finite dictionary and conditioning on the refined information therefore yields

$$\begin{aligned}\mathfrak{R}(q) &= \mathbb{E}[\ell(m_q^*(S,C),H,C)] \\ &= \mathbb{E}\left[\sum_{m\in\mathcal{M}} \mathbf{1}\{m_q^*(S,C)=m\}\ell(m,H,C)\right] \\ &= \mathbb{E}\left[\sum_{m\in\mathcal{M}} \mathbf{1}\{m_q^*(S,C)=m\} r'_m(S',C)\right] \\ &= \mathbb{E}[r'_{m_q^*(S,C)}(S',C)].\end{aligned}$$

Subtracting $\mathfrak{R}(q') = \mathbb{E}[\min_m r'_m(S',C)]$ proves the exact improvement identity in Equation 16. Its integrand is nonnegative, proving $\mathfrak{R}(q') \leq \mathfrak{R}(q)$.

For the other inequality, any selector based on (S', C) can be implemented from (H, C) by first applying q'. The class of history selectors therefore includes the class of refined-state selectors, giving $\mathfrak{R}_H \leq \mathfrak{R}(q')$. Equivalently, the pointwise minimum over history-conditioned losses is no larger than the loss of the refined-state optimal choice at any history.

Finally, the integrand in Equation 16 is a bounded nonnegative random variable. Its expectation is zero exactly when it is zero almost surely, and positive exactly when it is positive on an event of positive probability. Positivity is precisely the event that $m_q^*(S, C)$ fails to minimize the refined conditional losses, proving the strictness statement. □

The equality condition allows tied optimal methods. More explicitly, equality holds if and only if almost every coarse state–context pair admits a method that remains optimal with conditional probability one over its refined subclasses. In terms of conditional probabilities, this condition is

$$\max_{m \in \mathcal{M}} \Pr\left(r'_m(S', C) = \min_{k \in \mathcal{M}} r'_k(S', C) \,\middle|\, S, C \right) = 1 \quad \text{almost surely.}$$

To see necessity, equality in Equation 16 makes the coarse optimal choice a refined minimizer almost surely. Conditioning that event on (S, C) gives a method with conditional probability one of remaining optimal. For sufficiency, choose the first method with that conditional probability equal to one. The finite dictionary makes this choice measurable. It is a coarse selector attaining the refined optimal risk, which forces equality by the proposition. Consequently, equality does not require identical refined loss vectors. It requires a common optimal choice within each coarse information class. If equality holds, every coarse optimal selector has zero refined excess loss by the same identity.

Comparison with the full observed history gives a related exact decomposition,

$$\mathfrak{R}(q) - \mathfrak{R}_H = \mathbb{E}\left[\ell(m_q^*(S, C), H, C) - \min_{m \in \mathcal{M}} \ell(m, H, C) \right]. \tag{17}$$

Thus $\mathfrak{R}(q) = \mathfrak{R}_H$ exactly when an optimal state-limited choice is history-optimal almost surely. This is sufficiency for the specified decision under the fixed loss and distribution, not a requirement to predict future observations or identify a latent environment state.

B.4.2 Unresolved procedural ambiguity

Equation 17 identifies the loss associated with retaining too little history information. To quantify this loss through disagreement about the preferred method, assume that the history-optimal method is unique and separated from the alternatives by a positive loss margin.

Corollary 1 (Loss forced by aliased procedural choices)**.** *Suppose the history-optimal method $J = \arg\min_{m \in \mathcal{M}} \ell(m, H, C)$ is unique almost surely. Assume there is $\gamma > 0$ such that, almost surely, every $m \neq J$ satisfies $\ell(m, H, C) - \ell(J, H, C) \geq \gamma$. Define*

$$A(q) = \mathbb{E}\left[1 - \max_{m \in \mathcal{M}} \Pr(J = m \mid S, C) \right]. \tag{18}$$

Then

$$\gamma A(q) \leq \mathfrak{R}(q) - \mathfrak{R}_H \leq A(q). \tag{19}$$

Proof. Let $p_m(S, C) = \Pr(J = m \mid S, C)$. For any deterministic state-limited choice $\widehat{J}$,

$$\Pr(\widehat{J} \neq J \mid S, C) = 1 - p_{\widehat{J}}(S, C) \geq 1 - \max_{m \in \mathcal{M}} p_m(S, C).$$

The same bound holds for a randomized choice with probabilities $a(m \mid S, C)$ because its conditional probability of matching J is $\sum_m a(m \mid S, C) p_m(S, C) \leq \max_m p_m(S, C)$. A deterministic choice $\widetilde{J}$ of a conditionally most probable method attains equality. Ties are resolved by the fixed ordering of $\mathcal{M}$, so $\widetilde{J}$ is measurable. Taking expectations proves that $A(q)$ is the smallest probability of selecting a method different from J using (S, C).

By the margin assumption, every choice satisfies the pointwise inequality

$$\ell(\widehat{J}, H, C) - \ell(J, H, C) \geq \gamma \mathbf{1}\{\widehat{J} \neq J\}.$$

In particular, applying it to m_q^* and using Equation 17 gives

$$\mathfrak{R}(q) - \mathfrak{R}_H \geq \gamma \Pr(m_q^*(S, C) \neq J) \geq \gamma A(q),$$

which proves the lower bound.

For the upper bound, use the conditionally most probable method $\widetilde{J}$. Its excess loss is zero when $\widetilde{J} = J$, and at most one otherwise because all losses lie in $[0, 1]$. Since m_q^* minimizes expected continuation loss among state-limited selectors,

$$\begin{aligned}\mathfrak{R}(q) - \mathfrak{R}_H &\leq \mathbb{E}[\ell(\widetilde{J}, H, C) - \ell(J, H, C)] \\ &\leq \Pr(\widetilde{J} \neq J) = A(q).\end{aligned}$$

The selector minimizing misclassification need not minimize continuation loss. Its role in this argument is to provide a feasible state-limited choice with controlled loss. □

Under the margin assumption, $A(q) > 0$ forces positive excess risk even for the optimal state-limited selector. Conversely, $A(q) = 0$ means that J is determined almost surely by (S, C) and gives zero excess risk. Improving the selector while holding its information fixed cannot remove this ambiguity.

The bounds are sharp. Fix a common context and two histories that q maps to the same state, with probabilities α and $1 - \alpha$, where $0 < \alpha < 1$. For two methods with respective loss vectors $(0, \gamma)$ and $(\gamma, 0)$ across the two histories, where $0 < \gamma \leq 1$, the full-history risk is zero and

$$A(q) = \min(\alpha, 1 - \alpha), \qquad \mathfrak{R}(q) - \mathfrak{R}_H = \gamma \min(\alpha, 1 - \alpha) = \gamma A(q).$$

Thus the lower bound is attained for every such margin. Taking $\gamma = 1$ attains the upper bound as well, so both constants can be attained under the stated assumptions.

For a preparation example, fix a task and role context in which the required object is held and its preparation status is observed. The coarse state records only possession, with probability $\alpha \in (0, 1)$ of an unprepared object. Two methods have normalized continuation losses

	unprepared	prepared
prepare-then-deliver	ε	ε
direct delivery	1	0

$0 < \varepsilon < 1.$

Preparation incurs a positive execution cost even when unnecessary. These illustrative losses do not assume that repeated preparation fails.

The coarse conditional expected losses are ε and α, respectively. Hence its optimal risk is $\mathfrak{R}(q) = \min(\varepsilon, \alpha)$. A refinement recording the observed preparation status selects prepare-then-deliver only for unprepared histories and direct delivery otherwise. It attains

$$\mathfrak{R}(q') = \mathfrak{R}_H = \alpha\varepsilon,$$

and the improvement is

$$\mathfrak{R}(q) - \mathfrak{R}(q') = \begin{cases} \alpha(1 - \varepsilon), & \alpha \leq \varepsilon, \\ \varepsilon(1 - \alpha), & \alpha > \varepsilon. \end{cases}$$

The improvement is strictly positive, including at the coarse-choice tie $\alpha = \varepsilon$. The uniquely optimal methods differ across histories, with margin $\min(\varepsilon, 1 - \varepsilon)$ and unresolved choice probability $\min(\alpha, 1 - \alpha)$. If preparation always succeeded at zero relevant cost, prepare-then-deliver would be optimal in both histories, so the same physical distinction would have zero decision value.

B.4.3 Historical reassignment and procedure connections

We now fix a finite set L of historical transition occurrences whose endpoints support the procedural memory. Each $\ell \in L$ has fixed source and target boundary occurrences ω_ℓ^- and ω_ℓ^+ in a boundary set Ω. A boundary occurrence identifies a particular boundary in its original execution, rather than merely the text of an observation. The projections $q : \Omega \to \mathcal{X}_\Sigma$ and $q' : \Omega \to \mathcal{X}_{\Sigma'}$ assign complete states consistently whenever the same boundary is referenced. In this part of the analysis, q and q' act on the fixed boundary records, and no boundary is excluded.

Define candidate occurrence connections and continuations sharing an identical boundary by

$$\begin{aligned} J_q &= \{(\ell, k) \in L^2 : q(\omega_\ell^+) = q(\omega_k^-)\}, \\ W &= \{(\ell, k) \in L^2 : \omega_\ell^+ = \omega_k^-\}. \end{aligned} \tag{20}$$

The ordered pairs in J_q share an intermediate abstract state and may come from different task instances. Those in W share the same recorded boundary.

Assume that $q = \pi \circ q'$ on Ω for a map $\pi : q'(\Omega) \to q(\Omega)$. For each coarse state s, let $\mathcal{J}_s = \{j \in q'(\Omega) : \pi(j) = s\}$ denote its refined subclasses. Only subclasses used by an endpoint contribute to the following finite counts,

$$a_{sj} = \big|\{\ell \in L : q'(\omega_\ell^+) = j\}\big|, \qquad b_{sj} = \big|\{\ell \in L : q'(\omega_\ell^-) = j\}\big|, \quad j \in \mathcal{J}_s.$$

Thus a_{sj} counts incoming occurrences and b_{sj} counts outgoing occurrences at refined state j within coarse state s. All sums below can be restricted to endpoint assignments appearing in the finite pool.

Proposition 2 (Refining historical state assignments)**.** *Under the fixed-occurrence and refinement assumptions above, $W \subseteq J_{q'} \subseteq J_q$. The number of candidate connections removed at an old state s is*

$$\begin{aligned} \Delta_s &= \Big(\sum_{j \in \mathcal{J}_s} a_{sj}\Big)\Big(\sum_{j \in \mathcal{J}_s} b_{sj}\Big) - \sum_{j \in \mathcal{J}_s} a_{sj} b_{sj} \\ &= \sum_{\substack{i,j \in \mathcal{J}_s \\ i \neq j}} a_{si} b_{sj}. \end{aligned} \tag{21}$$

Moreover, $|J_q \setminus J_{q'}| = \sum_s \Delta_s$.

Proof. If $(\ell, k) \in J_{q'}$, then $q'(\omega_\ell^+) = q'(\omega_k^-)$. Applying π to the two equal assignments gives $q(\omega_\ell^+) = q(\omega_k^-)$, hence $(\ell, k) \in J_q$. This proves $J_{q'} \subseteq J_q$. If $(\ell, k) \in W$, both endpoints name the same boundary occurrence. Consistency of q' assigns them the same refined state, so $(\ell, k) \in J_{q'}$. This proves $W \subseteq J_{q'}$.

At coarse state s, every occurrence ending at s can be paired with every occurrence starting at s. There are $\sum_j a_{sj}$ incoming occurrences and $\sum_j b_{sj}$ outgoing occurrences, so their Cartesian product has size $(\sum_j a_{sj})(\sum_j b_{sj})$. Under q', equality of the intermediate states permits only pairs within the same subclass. The product for subclass j has size $a_{sj} b_{sj}$, and these products are disjoint. Subtracting their total size gives the first expression for Δ_s. Expanding the product and removing its diagonal terms gives the second expression.

Each removed pair has a unique coarse intermediate state. The removed pairs are therefore partitioned by s, and summing their counts gives $|J_q \setminus J_{q'}| = \sum_s \Delta_s$. □

For any fixed sequence $(\ell_1, \ldots, \ell_n)$ with $n \geq 2$ and

$$\omega_{\ell_i}^+ = \omega_{\ell_{i+1}}^- \quad \text{for every } i = 1, \ldots, n-1,$$

every adjacent pair lies in W and hence in $J_{q'}$. Such a fixed occurrence chain remains connected after refinement. Conversely, every chain connected by $J_{q'}$ is also connected by J_q, by applying the other inclusion to each adjacent pair. This conclusion concerns the fixed occurrence sequence and does not assert that arbitrary abstract chains are executable in a new task.

Equation 21 counts cross-subclass candidate joins removed from the fixed occurrence pool. These may include useful sharing. The count concerns occurrence pairs, not canonical methods, runtime failures, or computational savings.

Historical reassignment also determines where a previously learned method can retain its evidence. For an old method-bearing transition e, let $\mathrm{supp}(e)$ be its finite set of supporting realizations. After a schema revision, let $\psi(\omega)$ be the retained state identity assigned to boundary ω, or $\perp$ if the occurrence is excluded. This reassignment need not be a refinement. Recall the complete projected support set from Equation 11,

$$\Pi_b(e) = \{(\psi(\omega_\ell^-), \psi(\omega_\ell^+)) : \ell \in \mathrm{supp}(e)\}.$$

The method can retain its entire nonempty support association at a single valid state pair if and only if

$$\Pi_b(e) = \{(u, v)\}, \qquad u, v \neq \perp, \quad u \neq v.$$

For necessity, retaining the entire nonempty association at one pair means that every supporting realization has the same projected source u and target v. The complete set of projected pairs must therefore be the displayed singleton. The pair must have nonexcluded, distinct endpoints because these are required for a retained transition. For sufficiency, a singleton with valid endpoints assigns every supporting realization to the common pair, so attaching the method to that pair preserves the entire support association. Empty support, any excluded endpoint, or a coincident-endpoint singleton cannot satisfy this criterion. In particular, removing invalid pairs before forming the set would no longer test preservation of the entire support.

If support separates into multiple pairs, direct migration cannot preserve the entire association at one pair. The learner must reassess the evidence at its supported pairs. Support at (u_0, v_0) and (u_1, v_1), with all four endpoints distinct, supplies no association at the cross-pairs (u_0, v_1) and (u_1, v_0). This criterion concerns evidence placement, not applicability or target attainment on new histories.