MSCPM: MULTI-LEVEL SKILL COMPOSITION FOR PROCEDURAL MEMORY IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Currently, considerable research on LLM-based agents distills procedural memory from the past trajectories of agents to avoid re-exploring recurring tasks and enhance decision-making, yet existing methods remain limited in skill composition. Inspired by human procedural memory, we propose *Multi-level Skill Composition for Procedural Memory* (MSCPM), a framework that organizes memory through asymmetric inter-skill relations and explicitly models multi-level skill composition. Specifically, we extract operation, subgoal, and task-domain skills bottom-up through clustering and chunking, forming a hierarchical skill graph. On this graph, a cascade of coarse-to-fine subgoal selection, typed plan orchestration, and dynamic operation assembly composes skills into execution strategies tailored to each new task. Across AppWorld and BFCL-V3, MSCPM surpasses the strongest baseline by 1.94–2.58/1.99–3.97 points in Avg@4/Pass@4, and improves over the no-memory baseline by 26.1%/18.0% and 14.7%/13.7% with Qwen3-32B, rising to 35.8%/26.4% and 22.5%/23.7% with Qwen3-8B. Smaller agents equipped with MSCPM even surpass larger memory-free ones.

1 INTRODUCTION

Advances in large language models (LLMs) are driving AI systems from passive question answerers toward autonomous executors. In particular, LLM-based agents show promise in personal assistance (Trivedi et al., 2024; Li et al., 2026), web browsing (Yao et al., 2022; Zhou et al., 2024), and tool usage (Yao et al., 2024; Patil et al., 2025), among others. However, they typically rely on from-scratch reasoning, re-exploring even recurring tasks resulting in wasted computation and limited robustness. To overcome this limitation, considerable research distills explicit *procedural memory* (also known as *skill memory*) from agents' past trajectories to enhance decision-making, with the advantages of interpretability and cross-model applicability (Yang et al., 2024; Ouyang et al., 2026; Han et al., 2026). To exploit such memory for new tasks, agents typically need to *compose* existing skills into execution strategies, a process influenced by the organization of memory. Existing studies primarily organize memory into flat, hierarchical, or graph structures: flat methods (Wang et al., 2025; Cao et al., 2026) lack inter-skill relations and often recall superficially similar yet useless skills, which causes incorrect composition; hierarchical methods (Wang et al., 2026a; Tang et al., 2025) model only cross-layer skill relations, which limits within-layer skill composition; and graph-based methods (Liu et al., 2026; Bai et al., 2026) often perform skill retrieval on the full graph with dense inter-skill relations, which increases the uncertainty of skill composition.

Overall, existing methods are limited in skill composition, for which human procedural memory can serve as a reference. First, the formation of human procedural memory follows *hierarchical chunking* (Graybiel, 1998): repeatedly co-executed actions are chunked into larger units, which become new units for higher-level chunking, yielding stable cross-level bindings. Such bindings compress the decision space, alleviating cognitive load. Taking driving as an example, *lane-change overtaking* is chunked from actions such as *checking the mirrors*, *judging inter-vehicle gaps*, and *signaling*, and also underlies the higher-level skill *urban driving*. Second, the use of procedural memory relies on *dynamic co-activation* (Bassett et al., 2011): most inter-skill relations are not pre-stored, only activated on demand and dissolved upon task completion. Such on-demand activation conforms to *cognitive economy*, whereby a capacity-limited system withstands the combinatorial explosion of the task space. For example, when a driver must reverse out of a queue in a narrow

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.

one-way alley, *judging inter-vehicle gaps* (from *lane-change overtaking*) and *steering by reference points* (from *reverse parking*) are co-activated, jointly accomplishing narrow-alley reversing.

Inspired by hierarchical chunking and dynamic co-activation, we propose *Multi-level Skill Composition for Procedural Memory* (MSCPM), a procedural memory framework for LLM agents. MSCPM organizes procedural memory bottom-up into a *hierarchical skill graph* with three layers, i.e., *operation*, *subgoal*, and *task domain*. Lower-level skills are the building blocks of higher-level ones, bound by cross-layer edges that not only link skills vertically but also serve as pathways that transiently associate skills within the same layer. On this graph, we develop a multi-level skill composition method that explicitly models how skills are scoped, reused, and assembled, thereby tailoring skill composition to each task. Given a new task, we first perform coarse matching on the task-domain layer, then curate task-relevant subgoals into a candidate pool. The subgoal orchestrator then determines the reuse type of the candidates (Inherit, Adapt, or Create) to generate a plan. Each plan step is dynamically assembled with operation skills: either the pre-bound set is recalled directly along cross-layer edges, or within-layer co-activation relations are induced online and ranked. Finally, skill iteration keeps the graph updating with use, consolidating recurring composition patterns into new skills. Our contributions are as follows:

- We propose MSCPM, a procedural memory framework that draws inspiration from human memory mechanisms to model inter-skill relations asymmetrically and multi-level skill composition explicitly, thus achieving both hierarchical efficiency and graph flexibility.
- We apply clustering and chunking to construct the hierarchical skill graph bottom-up, and continuously iterate the graph through node creation and absorption, so that its scale tracks the pattern diversity rather than the number of accumulated trajectories.
- We cascade coarse-to-fine subgoal selection, typed plan orchestration, and dynamic operation assembly, thus composing skills vertically along cross-layer edges and horizontally via transient within-layer co-activation.
- We validate MSCPM on two agent benchmarks, i.e., AppWorld and BFCL-V3, where it outperforms representative baselines in task success rate across all settings, and smaller backbones equipped with MSCPM even surpass larger memory-free ones.

2 Related Work

Procedural Memory for Agents. Procedural memory, also known as *skill memory*, is memory of *how to do things* (Tulving, 1985; Graf & Schacter, 1985). Early cognitive architectures such as Soar (Laird, 2012) and ACT-R (Anderson et al., 1997) formalize it as production rules, interpretable yet dependent on expert knowledge. Fine-tuning and reinforcement learning instead internalize it into model parameters, sacrificing interpretability for automation; such training is prohibitively expensive for LLMs, confining studies (Wu et al., 2026; Zhang et al., 2026; Qi et al., 2025; Zhou et al., 2025) to small open-source models. Recent studies therefore distill explicit procedural memory from agent trajectories into workflows (Yang et al., 2024; Wang et al., 2025), execution exemplars (Wang et al., 2023; Zheng et al., 2024), tool usage patterns (Wang et al., 2026b; Shao et al., 2026), and general insights (Fu et al., 2024; Ouyang et al., 2026), which are interpretable and applicable across models. Research on procedural memory for agents is thus shifting back toward explicit modeling.

Organization and Utilization of Procedural Memory. In open-ended task spaces, past experience can rarely be retrieved and applied as-is, so agents must compose existing skills into execution strategies for new tasks. Skill composition relies largely on memory organization, which supplies the necessary structural information and constraints. Early studies such as BoT (Yang et al., 2024), AWM (Wang et al., 2025), and Memp (Fang et al., 2026) store skills as independent, same-level units retrieved by vector similarity or keyword matching (Fang et al., 2026; Zhao et al., 2024; Chen et al., 2024); without inter-skill relations, such flat organization often recalls superficially similar yet useless skills and causes composition failures. Hierarchical systems such as SkillX (Wang et al., 2026a) and Agent KB (Tang et al., 2025) organize procedural memory into layers of different abstraction granularities, where the agent retrieves low-level skills under the constraint of an invoked high-level skill (Liu et al., 2025; Tang et al., 2025; Wang et al., 2026a) or automatically expands it into pre-assembled ones (Wang et al., 2026b; Han et al., 2026). Such hierarchical organization supports explicit cross-layer composition and effectively narrows the decision space, but the neglect

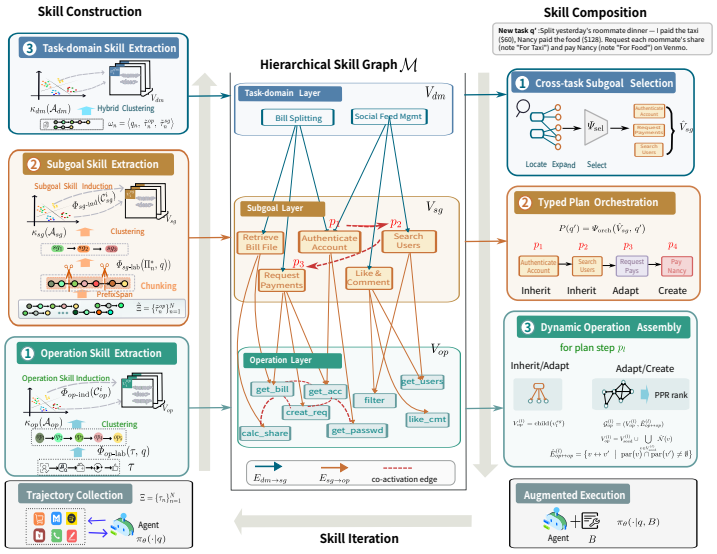


Figure 1: Overview of our framework

of within-layer relations leaves skills in the same layer independent or even isolated, limiting composition. Graph-based approaches such as GoS (Liu et al., 2026), SkillDAG (Bai et al., 2026), and SkillNet (Liang et al., 2026) instead model relations among arbitrary skills, including dependencies, complementarities, substitutions, and similarities, enabling flexible composition; however, graph-wide diffusion is sensitive to seeds and thus destabilizes the composition scope, while storing all inter-skill relations incurs high maintenance and storage costs.

Compared to existing work, MSCPM combines the strengths of both hierarchical and graph-based methods: cross-layer binding edges support vertical composition and alleviate the seed sensitivity of global graph search, while within-layer co-activation relations induced online support horizontal composition and dissolve upon task completion, achieving graph flexibility at near-hierarchical cost.

3 Methodology

3.1 Overview

We consider an agent policy $\pi_\theta(a_t \mid h_t)$ in a partially observable environment, where a_t is the action, h_t the interaction history up to step t, and θ the parameters of the backbone LLM. Given a set of tasks $\mathcal{Q}$ and a pool of raw trajectories $\Xi = \{\tau_n\}_{n=1}^{N}$, with $\tau_n = (o_t, a_t)_{t=1}^{T}$ consisting of T interaction steps, each with an observation o_t and an action a_t. We formulate the memory mechanism into two parts: (i) a *memory construction module* Φ distills a memory bank $\mathcal{M} = \Phi(\Xi)$; (ii) a *skill composition module* Ψ composes skills from $\mathcal{M}$ into an execution strategy $B = \Psi(\mathcal{M}, q)$ tailored to the task $q \in \mathcal{Q}$. The memory-augmented agent generates trajectories $\tau' \sim \pi_\theta(\cdot \mid q, B)$. Our goal is to design Φ and Ψ to maximize the expected cumulative reward $\mathcal{R}(\tau')$:

$$\max_{\Phi,\,\Psi} \; \mathbb{E}_{q\sim\mathcal{Q},\,\tau'\sim\pi_\theta(\cdot|q,\,\Psi(\Phi(\Xi),q))}\big[\,\mathcal{R}(\tau')\,\big], \qquad \mathcal{R}(\tau') \in [0,1]. \tag{1}$$

Our proposed framework MSCPM shown in Figure 1 consists of *hierarchical skill graph construction* (Section 3.2), *multi-level skill composition* (Section 3.3), and *skill iteration* (Section 3.4).

3.2 Hierarchical Skill Graph Construction

We generate the node sets of the operation, subgoal, and task-domain layers in a bottom-up manner through clustering and chunking, i.e., V_{op} (Section 3.2.1), V_{sg} (Section 3.2.2), and V_{dm} (Section 3.2.3), as well as edges $E_{dm\to sg} \subseteq V_{dm} \times V_{sg}$ and $E_{sg\to op} \subseteq V_{sg} \times V_{op}$. Each edge (a, b) is directed from a parent a to a child b ($a \in \mathrm{par}(b)$, $b \in \mathrm{child}(a)$), and any node is allowed to have

multiple parents and children. Consequently, these nodes and edges constitute a hierarchical graph that serves as procedural memory.

$$\mathcal{M} = \big(V_{op} \cup V_{sg} \cup V_{dm},\ E_{dm\to sg} \cup E_{sg\to op}\big). \tag{2}$$

3.2.1 Operation Layer

We first design an LLM-based context-aware action annotator $\Phi_{op\text{-lab}}$ that labels every interaction step of a trajectory $\tau_n = (o_{n,1}, a_{n,1}, \ldots, o_{n,T}, a_{n,T})$ as an action:

$$\Phi_{op\text{-lab}}(\tau_n,\, q_n) = \{op_{n,t}\}_{t=1}^{T}, \qquad op_{n,t} = \big\langle \mathrm{name}_{n,t},\, \mathrm{desc}_{n,t},\, a_{n,t} \big\rangle, \tag{3}$$

where $q_n \in \mathcal{Q}$ is the task associated with τ_n, $\mathrm{name}_{n,t}$ and $\mathrm{desc}_{n,t}$ are the name and description of the action, and $a_{n,t}$ the executed action; all annotations form the set $\mathcal{A}_{op} = \big\{op_{n,t} \;\big|\; n \in [N],\, t \in [T]\big\}$. Because these annotations exhibit surface-form variation, we partition $\mathcal{A}_{op}$ via the clustering operator κ_{op}, based on the cosine similarity of name and description embeddings $e : \mathcal{A}_{op} \to \mathbb{R}^d$:

$$\mathcal{C}_{op} = \{\mathcal{C}_{op}^i\}_{i=1}^{|\mathcal{C}_{op}|} = \kappa_{op}(\mathcal{A}_{op}) \quad \text{s.t.} \quad \bigcup_{i=1}^{|\mathcal{C}_{op}|} \mathcal{C}_{op}^i = \mathcal{A}_{op}, \quad \mathcal{C}_{op}^i \cap \mathcal{C}_{op}^{i'} = \emptyset\ (i \neq i'). \tag{4}$$

The operation inducer $\Phi_{op\text{-ind}}$ then induces an operation node from each cluster:

$$v_i^{op} = \Phi_{op\text{-ind}}\big(\mathcal{C}_{op}^i\big), \qquad \mathrm{attr}\big(v_i^{op}\big) = \big\langle \mathrm{name}\big(v_i^{op}\big),\, \mathrm{desc}\big(v_i^{op}\big),\, \mathrm{exmp}\big(v_i^{op}\big) \big\rangle, \tag{5}$$

where the name and description are aggregated over the cluster, i.e., $\mathrm{name}(v_i^{op}) = \mathrm{agg}_{op\in\mathcal{C}_{op}^i}[\mathrm{name}(op)]$ (likewise for desc), and $\mathrm{exmp}(\cdot)$ the example actions sampled from the cluster. This yields the canonical operation node set $V_{op} = \{v_i^{op}\}_{i=1}^{|V_{op}|}$.

3.2.2 Subgoal Layer

We employ a lookup mapping $\sigma_{op} :\ \mathcal{A}_{op} \to V_{op}$ to project the action annotations of each τ_n into an operation sequence $\tilde{\tau}_n^{op} = \big(\sigma_{op}(op_{n,1}),\, \ldots,\, \sigma_{op}(op_{n,T})\big)$, yielding the symbolic trajectory set $\tilde{\Xi} = \{\tilde{\tau}_n^{op}\}_{n=1}^{N}$. We then mine frequent patterns from $\tilde{\Xi}$ with PrefixSpan (Han et al., 2001):

$$\mathcal{F}_h = \big\{ f \;\big|\; \mathrm{supp}(f) \geq s_{\min},\ |f| \in [l_{\min}, l_{\max}] \big\}, \tag{6}$$

where $\mathrm{supp}(f)$ is the number of occurrences of f in $\tilde{\Xi}$, $s_{\min}$ is the threshold for frequent sequences, and $l_{\min}$ and $l_{\max}$ are the minimum and maximum allowed lengths for frequent sequences.

Taking frequent patterns as chunk candidates, we chunk each $\tilde{\tau}_n^{op}$ into non-overlapping contiguous subsequences, i.e., operation chunks, by maximizing the coverage score, which favors *longer* patterns with *higher support*. For a contiguous subsequence u^{op}, the coverage score and the optimal segmentation are

$$g(u^{op}) = \begin{cases} |u^{op}| \cdot \mathrm{supp}(u^{op}), & u^{op} \in \mathcal{F}_h, \\ 0, & \text{otherwise} \end{cases} \qquad \Pi_n^* = \arg\max_{\Pi_n} \sum_{u^{op}\in\Pi_n} g(u^{op}), \tag{7}$$

where $\Pi_n = \big(u_1^{op}, \ldots, u_S^{op}\big)$ ranges over all chunkings of $\tilde{\tau}_n^{op}$ with $|u_s^{op}| \leq l_{\max}$. Chunks with $g(u^{op}) > 0$ are cross-trajectory stable and the rest are trajectory-specific.

As with operation skills, we design a subgoal annotator $\Phi_{sg\text{-lab}}$ that labels each chunk in Π_n^* with the subgoal it accomplishes, i.e., $sg_{n,s} = \big\langle \mathrm{name}_{n,s},\, \mathrm{desc}_{n,s},\, u_s^{op} \big\rangle$. To avoid noisy operation sequences from failed trajectories, we annotate only successful trajectories, indexed by $\mathcal{I}_{succ} \subseteq [N]$, yielding $\mathcal{A}_{sg} = \{sg_{n,s} \mid n \in \mathcal{I}_{succ},\ s \in [S]\}$. Likewise, the clustering operator κ_{sg} partitions $\mathcal{A}_{sg}$ into disjoint clusters $\{\mathcal{C}_{sg}^j\}$, and the subgoal inducer induces a canonical subgoal node from each cluster, i.e., $v_j^{sg} = \Phi_{sg\text{-ind}}(\mathcal{C}_{sg}^j)$, $\mathrm{attr}(v_j^{sg}) = \langle \mathrm{name}(v_j^{sg}),\, \mathrm{desc}(v_j^{sg}) \rangle$. We then take the union of the operation sequences bound to the subgoals within each cluster, establishing the cross-layer edges from the subgoal layer to the operation layer:

$$\mathrm{child}\big(v_j^{sg}\big) = \bigcup_{sg\,\in\,\mathcal{C}_{sg}^j} u^{op}(sg), \qquad E_{sg\to op} = \Big\{ \big(v_j^{sg}, v_i^{op}\big) \;\Big|\; v_i^{op} \in \mathrm{child}\big(v_j^{sg}\big) \Big\}, \tag{8}$$

where $u^{op}(sg)$ is the sequence bound to sg. This yields the subgoal node set $V_{sg} = \{v_j^{sg}\}_{j=1}^{|V_{sg}|}$.

3.2.3 TASK-DOMAIN LAYER

Likewise, a lookup mapping σ_{sg} is devised to project the subgoal annotations of each trajectory into a subgoal sequence $\tilde{\tau}_n^{sg} = \big(\sigma_{sg}(sg_{n,1}), \ldots, \sigma_{sg}(sg_{n,S})\big)$, defining the task instance $\omega_n = \big\langle q_n,\ \tilde{\tau}_n^{op},\ \tilde{\tau}_n^{sg} \big\rangle$. Each task typically yields multiple near-duplicate instances, and clustering directly on $\hat{\Xi} = \{\omega_n\}_{n=1}^{N}$ would bias the result toward frequently attempted tasks rather than the diversity of the tasks themselves. We therefore retain a single representative instance $\bar{\omega}(q)$ for each task q: among its candidate instances $\{\omega_n \mid q_n = q\}$, we prefer successful trajectories and, among these, the one whose deduplicated operation-skill sequence covers the most distinct skills (***Max-Skill***), the former ensuring evidence quality and the latter preventing over-specialization. The representative set $\mathcal{A}_{dm} = \big\{\bar{\omega}(q) \ \big|\ q \in \mathcal{Q}\big\}$ feeds the subsequent task-domain clustering.

We measure the distance between task instances as a weighted combination of the cosine distance between task-instruction embeddings and the edit distance $\mathrm{lev}(\cdot,\cdot)$ between operation sequences, with the weight $\alpha \in [0,1]$ balancing the two distances:

$$d_{\mathrm{mix}}(\omega_n, \omega_{n'}) = \alpha \left(1 - \frac{e(q_n) \cdot e(q_{n'})}{\|e(q_n)\|\, \|e(q_{n'})\|}\right) + (1-\alpha)\, \frac{\mathrm{lev}\big(\tilde{\tau}_n^{op},\, \tilde{\tau}_{n'}^{op}\big)}{\max\big(|\tilde{\tau}_n^{op}|,\ |\tilde{\tau}_{n'}^{op}|\big)}, \tag{9}$$

As in the operation and subgoal layers, the clustering operator κ_{dm} partitions $\mathcal{A}_{dm}$ under the mixed distance d_{mix} into disjoint task domains $\{\mathcal{C}_{dm}^k\}_{k=1}^{K} = \kappa_{dm}\big(\mathcal{A}_{dm}\big)$, each grouping tasks with similar intents and solution structures. Then a task-domain node v_k^{dm} is induced from each cluster, where its description $\mathrm{desc}(v_k^{dm}) = \{q_n \mid \omega_n \in \mathcal{C}_{dm}^k\}$ gives the set of tasks in the domain.

We further take the union of the subgoals of the representative instances within each task domain, establishing the cross-layer edges from the task-domain layer to the subgoal layer:

$$\mathrm{child}\big(v_k^{dm}\big) = \bigcup_{\omega_n \in \mathcal{C}_{dm}^k} \tilde{\tau}_n^{sg}, \qquad E_{dm\to sg} = \Big\{ \big(v_k^{dm}, v_j^{sg}\big) \ \Big|\ v_j^{sg} \in \mathrm{child}\big(v_k^{dm}\big) \Big\}. \tag{10}$$

The above process outputs task-domain nodes $V_{dm} = \{v_k^{dm}\}_{k=1}^{K}$. Since each representative instance is the best solution for its task, we need only retain the subgoal and operation nodes covered by these instances, together with the cross-layer edges among them. The construction of the hierarchical skill graph is summarized in Algorithm 1 in Appendix B.

3.3 MULTI-LEVEL SKILL COMPOSITION

On this graph, we explicitly model skill composition through a cascade of coarse-to-fine subgoal selection (Section 3.3.1), typed plan orchestration (Section 3.3.2), and dynamic operation assembly (Section 3.3.3) to tailor an execution strategy to each task.

3.3.1 COARSE-TO-FINE SUBGOAL SELECTION

Given a task $q' \in \mathcal{Q}$, we first retrieve the r tasks most similar to q' from the existing tasks $\mathcal{Q}$, i.e., $\text{Top-r}_{q\in\mathcal{Q}}\ \cos\big(e(q),\, e(q')\big)$, and locate the task-domain nodes $V_{dm}^* = \{v_k^{dm}\}_{k=1}^{r}$ that r tasks belong to via a lookup on node attributes. Then we recall the subgoals of these domains along the cross-layer edges $E_{dm\to sg}$ and take their union as the candidate set V_{sg}^*, from which the LLM-based subgoal selector Ψ_{sel} selects the subgoals $\hat{V}_{sg}$ required by the task:

$$V_{sg}^* = \bigcup_{v_k^{dm} \in V_{dm}^*} \mathrm{child}\big(v_k^{dm}\big), \qquad \hat{V}_{sg} = \Psi_{\mathrm{sel}}\big(V_{sg}^*,\, q'\big) \subseteq V_{sg}^*. \tag{11}$$

This pipeline follows a locate – expand – select scheme, exploiting the cross-layer edges $E_{dm\to sg}$ and avoiding the granularity mismatch of direct retrieval on the subgoal layer.

3.3.2 TYPED PLAN ORCHESTRATION

The activated subgoals $\hat{V}_{sg}$ provide prior knowledge for planning, yet their temporal order is not pre-stored and past experience is insufficient for the current task. We therefore design a subgoal orchestrator Ψ_{orch} that infers their order and creates new subgoals as needed, forming the plan $P(q')$:

$$P(q') = \Psi_{\text{orch}}\big(\hat{V}_{sg},\, q'\big) = \big(p_1,\, p_2,\, \ldots,\, p_L\big), \tag{12}$$

where the l-th plan step p_l is a four-tuple as follows:

$$p_l = \big\langle \text{name}_l,\, \text{desc}_l,\, \text{type}_l,\, v_l^{sg} \big\rangle, \qquad \text{type}_l \in \{\text{Inherit},\, \text{Adapt},\, \text{Create}\}, \tag{13}$$

Here, name_l and desc_l are the name and description of the step, $v_l^{sg} \in \hat{V}_{sg} \cup \{\varnothing\}$ the subgoal bound to it, and type_l specifies how the step reuses its bound subgoal skill: **Inherit** reuses the skill as a whole for identical goals; **Adapt** reuses the skill with adjustments for partially matching goals; **Create** constructs a new subgoal from scratch when none in $\hat{V}_{sg}$ qualifies ($v_l^{sg} = \varnothing$). This instantiates online a within-layer temporal chain $p_1 \to \cdots \to p_L$ for the current task.

3.3.3 DYNAMIC OPERATION ASSEMBLY

For each plan step p_l, we assemble operation skills according to its type. **Inherit** recalls the skills pre-bound to v_l^{sg} directly along the cross-layer edges, i.e., $\hat{V}_{op}^{(l)} = \text{child}\big(v_l^{sg}\big)$. **Adapt** requires skills beyond those bound to v_l^{sg}. We thus retrieve z operation skills with the joint embedding $e(p_l)$ of the step's name and description, and merge them with the bound skills into the seed set $V_{\text{seed}}^{(l)}$:

$$V_{\text{seed}}^{(l)} = \underset{v \in V_{op}}{\text{Top-z}}\, \cos\big(e(p_l),\, e(v)\big) \;\cup\; \text{child}\big(v_l^{sg}\big). \tag{14}$$

For each seed v, we expand along the cross-layer edges to its sibling skills $\hat{\mathcal{N}}(v) = \big(\bigcup_{v^{sg} \in \text{par}(v)} \text{child}\big(v^{sg}\big)\big) \setminus \{v\}$, and then induce co-occurrence relations online:

$$V_{op}^{(l)} = V_{\text{seed}}^{(l)} \cup \bigcup_{v \in V_{\text{seed}}^{(l)}} \hat{\mathcal{N}}(v), \qquad \hat{E}_{op \leftrightarrow op}^{(l)} = \big\{ v \leftrightarrow v' \;\big|\; \text{par}(v) \cap \text{par}(v') \neq \emptyset \big\}, \tag{15}$$

where each pair of skills sharing a subgoal is connected by a bidirectional edge. The graph $\mathcal{G}_{op}^{(l)} = \big(V_{op}^{(l)},\, \hat{E}_{op \leftrightarrow op}^{(l)}\big)$ holds only transiently for the current task. We then run PageRank on it to rank skills by co-occurrence importance, yielding the operation-skill set $\hat{V}_{op}^{(l)}$:

$$\hat{V}_{op}^{(l)} = \underset{v \in V_{op}^{(l)}}{\text{Top-k}}\; \text{PageRank}\big(v;\, \mathcal{G}_{op}^{(l)}\big). \tag{16}$$

Create has no subgoal to inherit: its seed set contains only the skills retrieved with the step semantics, and the rest proceeds identically to **Adapt**.

Finally, we assemble each plan step of $P(q')$ with its operation-skill set, yielding the hierarchical execution strategy B, under which the agent generates the task trajectory:

$$B = \Big\{ \big(p_l,\, \hat{V}_{op}^{(l)}\big) \Big\}_{l=1}^{L}, \qquad \tau' \sim \pi_\theta\big(\cdot \mid q',\, B\big). \tag{17}$$

3.4 SKILL ITERATION

As interaction trajectories accumulate, we propose *offline batch skill iteration*: each batch of ΔN new trajectories triggers a round that applies the pipeline of Section 3.2 to them, replacing only each layer's clustering operator with the seeded constrained clustering operator κ^{cl}, which clusters the new annotations with existing skill nodes as seeds under cannot-link constraints between seeds (Basu et al., 2002; Wagstaff et al., 2001). Each cluster thus contains at most one seed and is either absorbed or newly created: a seeded cluster merges its new annotations into the seed node; an unseeded cluster yields a new node.The newly added subgraph is then merged into the existing hierarchical skill graph. **Will the hierarchical skill graph grow without bound?** We prove in Appendix C that

the number of skills is determined by the *diversity of patterns* in the experience, not by the number of rounds or trajectories: it is upper-bounded by the total number of patterns $K_{\mathcal{H}}$, and the expected number of new nodes per round decays monotonically. Thus, under stationary task and environment distributions, iteration starts in a creation phase, where most patterns in the new trajectories remain uncovered, and settles into an absorption phase as the number of skills converges. Section 4.4 validates this phenomenon empirically.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Benchmarks and Metrics. We adopt two agent benchmarks: BFCL-V3 (Patil et al., 2025) and AppWorld (Trivedi et al., 2024). For BFCL-V3, which provides no training split by default, we randomly select 50 tasks from the base multi-turn category as the training set; the remaining 150 tasks serve as the evaluation set. For AppWorld, we treat 90 training tasks as the training set and evaluate agents on the test-normal set (168 tasks). We execute each task independently four times and report Avg@4 and Pass@4 task success rates, with all results averaged over three independent runs. Benchmark and metric details are provided in Appendix D.1.

Baselines. We compare MSCPM with five baselines: (1) No-memory, where the agent reasons from scratch; (2) AWM (Wang et al., 2025), which induces abstract workflows from trajectories and injects them upon vector retrieval; (3) ReMe (Cao et al., 2026), which distills multi-faceted experience and rewrites it into task-specific guidance; (4) GoS (Liu et al., 2026), which organizes skills into a directed graph and enables structured retrieval via graph diffusion; (5) SkillX (Wang et al., 2026a), which distills raw trajectories into a three-tier hierarchy of planning, functional, and atomic skills. For fairness, experience retrieval is permitted only once at the beginning of each task. All baselines adopt the same LLM as MSCPM, i.e., the Qwen3 instruct model (Yang et al., 2025).

Implementation Details. In MSCPM, Qwen3-Embedding-8B (Geng et al., 2025) is adopted to obtain the semantic embedding $e(\cdot)$. For efficiency, the number of iteration rounds is capped at 3, with each round running the full training set. Online execution uses the non-thinking mode of Qwen3 (temperature 0.9) for efficiency, while offline skill construction uses the thinking mode (temperature 0.3) for thorough and stable reasoning. Hyperparameters and prompt templates of MSCPM are provided in Appendix D.2. All experiments run on NVIDIA A100 GPUs.

4.2 MAIN RESULTS

Table 1 lists the results on two benchmarks under three backbone sizes (8B/14B/32B). Our main observations are as follows:**(1) MSCPM achieves the best results under all settings, with more pronounced gains on smaller backbones.** AWM injects abstract workflows directly into the context and gains the least. ReMe rewrites multi-faceted insights into task-specific guidance and gains slightly more. GoS and SkillX store structured, concrete tool-execution patterns and thus surpass ReMe under most settings. MSCPM, in contrast, explicitly adjusts the reuse type of skills in the subgoal layer and assembles executable details on demand in the operation layer, thus adapting better across distributions. It consistently surpasses the strongest baseline by 1.94–2.58 points in Avg@4 and 1.99–3.97 points in Pass@4. Besides, its relative gains over No-memory narrow overall as the backbone grows: on AppWorld, Avg@4/Pass@4 drop from 35.8%/26.4% at 8B to 26.1%/18.0% at 32B, and on BFCL-V3 from 22.5%/23.7% to 14.7%/13.7%. Smaller LLM agents equipped with MSCPM even surpass larger memory-free ones.**(2) MSCPM consistently outperforms the hierarchy-only and graph-only baselines by composing skills vertically and horizontally.** MSCPM combines the strengths of both and thus performs better than SkillX and GoS: cross-layer binding edges support vertical composition, narrowing the decision space and thereby alleviating the seed sensitivity of global graph search; the within-layer co-activation relations induced online on demand support horizontal composition beyond hierarchy alone.

4.3 ABLATION STUDIES

We evaluate five ablation variants of MSCPM. **(a) w/o Subgoal Selector** removes the subgoal selector Ψ_{sel} and expands the candidates to all recalled subgoals ($\hat{V}_{sg} = V^*_{sg}$). **(b) w/o Typed Or-**

Table 1: Performance comparison (%) between MSCPM and the baselines on AppWorld and BFCL-V3, averaged over three independent runs; subscripts show standard deviations, and bold marks the best result under each backbone, with underlines marking the second best.

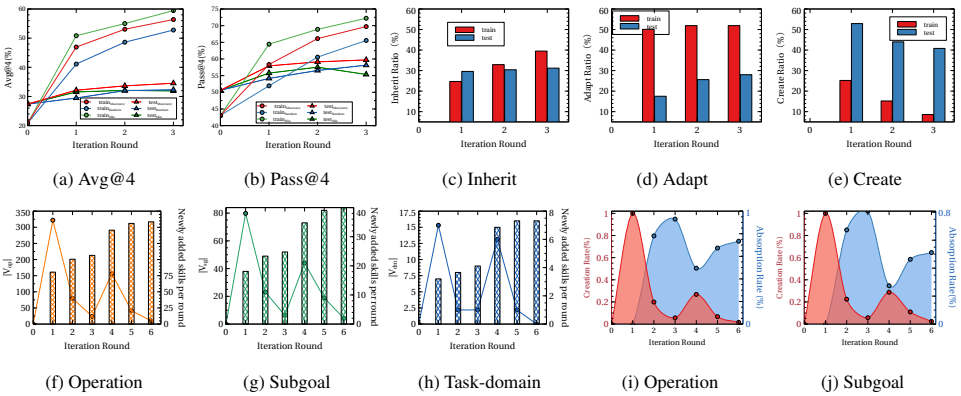
Model	Method	AppWorld		BFCL-V3	
		Avg@4	Pass@4	Avg@4	Pass@4
Qwen3-8B	No-memory	14.29$_{\pm 0.89}$	31.55$_{\pm 1.79}$	40.06$_{\pm 0.92}$	59.11$_{\pm 2.34}$
	AWM	16.37$_{\pm 0.54}$	34.72$_{\pm 2.09}$	42.83$_{\pm 1.20}$	63.33$_{\pm 1.76}$
	ReMe	<u>16.82</u>$_{\pm 0.68}$	<u>35.91</u>$_{\pm 0.90}$	44.72$_{\pm 0.26}$	66.00$_{\pm 1.33}$
	GoS	16.42$_{\pm 0.56}$	34.72$_{\pm 0.69}$	45.61$_{\pm 0.82}$	67.33$_{\pm 1.33}$
	SkillX	16.52$_{\pm 0.68}$	34.92$_{\pm 0.91}$	<u>46.72</u>$_{\pm 0.95}$	<u>69.33</u>$_{\pm 1.33}$
	MSCPM	**19.40**$_{\pm 0.17}$	**39.88**$_{\pm 0.60}$	**49.06**$_{\pm 0.67}$	**73.11**$_{\pm 1.93}$
Qwen3-14B	No-memory	22.62$_{\pm 0.65}$	40.87$_{\pm 1.82}$	48.00$_{\pm 0.67}$	68.00$_{\pm 1.76}$
	AWM	24.90$_{\pm 0.67}$	42.06$_{\pm 1.24}$	50.28$_{\pm 1.27}$	69.78$_{\pm 1.68}$
	ReMe	25.60$_{\pm 1.04}$	<u>46.43</u>$_{\pm 2.15}$	52.44$_{\pm 0.35}$	71.56$_{\pm 1.02}$
	GoS	25.50$_{\pm 0.85}$	45.04$_{\pm 0.91}$	53.94$_{\pm 0.51}$	72.89$_{\pm 0.39}$
	SkillX	<u>25.69</u>$_{\pm 0.60}$	45.83$_{\pm 0.60}$	<u>54.67</u>$_{\pm 0.73}$	<u>75.11</u>$_{\pm 1.02}$
	MSCPM	**28.27**$_{\pm 0.15}$	**49.81**$_{\pm 0.69}$	**56.94**$_{\pm 0.54}$	**78.22**$_{\pm 1.02}$
Qwen3-32B	No-memory	27.38$_{\pm 0.26}$	50.60$_{\pm 1.19}$	54.00$_{\pm 1.09}$	71.56$_{\pm 1.02}$
	AWM	29.81$_{\pm 0.52}$	54.56$_{\pm 0.91}$	55.22$_{\pm 0.59}$	73.78$_{\pm 1.39}$
	ReMe	31.45$_{\pm 0.43}$	57.54$_{\pm 1.45}$	56.11$_{\pm 0.26}$	74.44$_{\pm 0.39}$
	GoS	31.99$_{\pm 0.45}$	57.14$_{\pm 0.60}$	57.00$_{\pm 0.60}$	75.11$_{\pm 1.68}$
	SkillX	<u>32.19</u>$_{\pm 0.52}$	<u>57.73</u>$_{\pm 0.59}$	<u>60.00</u>$_{\pm 0.67}$	<u>79.11</u>$_{\pm 1.02}$
	MSCPM	**34.52**$_{\pm 0.39}$	**59.72**$_{\pm 0.91}$	**61.94**$_{\pm 0.10}$	**81.33**$_{\pm 1.33}$

Table 2: Ablation study (%) of MSCPM with Qwen3-32B on AppWorld and BFCL-V3. Left: the three skill composition mechanisms (✓ = as designed; × = removed or degraded). Values are Avg@4, with the drop relative to the full MSCPM in blue.

Variant	Skill Composition			AppWorld	BFCL-V3
	Selection	Orchestration	Assembly	Avg@4	Avg@4
MSCPM	✓	✓	✓	34.52	61.94
Composition ablations					
(a) w/o Subgoal Selector	×	✓	✓	33.68$_{(-0.84\%)}$	61.89$_{(-0.05\%)}$
(b) w/o Typed Orchestration	✓	×	✓	32.79$_{(-1.73\%)}$	60.22$_{(-1.72\%)}$
(c) w/o Dynamic Assembly	✓	×	×	32.40$_{(-2.12\%)}$	60.11$_{(-1.83\%)}$
Structure ablations					
(d) w/o Task-Domain Layer	×	✓	✓	32.94$_{(-1.58\%)}$	61.56$_{(-0.38\%)}$
(e) w/o Hierarchy	×	×	✓	30.56$_{(-3.96\%)}$	58.94$_{(-3.00\%)}$

chestration still generates the plan from the selected subgoals $\hat{V}_{sg}$, but every plan step is assembled as Create. **(c) w/o Dynamic Assembly** performs flat Top-k semantic search directly on the operation layer for each plan step. **(d) w/o Task-Domain Layer** retrieves the Top-r most similar tasks and takes the union of their subgoals directly as candidates. **(e) w/o Hierarchy** removes the task-domain and subgoal layers, generates the plan directly from the task, and assembles every step as Create. The first three variants ablate the online composition mechanism, and the last two the graph structure. Table 2 summarizes the mechanism configurations and results of all variants.

All variants degrade performance, confirming the necessity of each component. Our observations are mainly as follows:**The hierarchy contributes the most.** The largest drop on both benchmarks comes from (e) w/o Hierarchy (-3.96 points on AppWorld and -3.00 on BFCL-V3). Without the task-domain and subgoal layers, the agent can only plan directly from the task, with neither the cross-task planning priors carried by subgoal skills nor the top-down narrowing of the decision space along cross-layer edges. **Among the composition mechanisms, the dynamic assembly contributes the most,** with variation (c) without dynamic assembly showing the second-largest drop (-2.12 points in AppWorld and -1.83 in BFCL-V3). Without dynamic assembly, flat semantic retrieval recalls similar yet useless operation skills and thus hurts performance. **The gains of the task-domain layer and the subgoal selector grow with task complexity.** Variants (a) and (d) drop slightly on BFCL-V3 (-0.05 and -0.38) but far more on AppWorld (-0.84 and -1.58). BFCL-V3 contains only 129 function tools and exposes 10–30 of them per task, so structured retrieval gains little. AppWorld instead exposes all 457 APIs to every task across diverse scenarios. Variant (d) narrows the skill



(a) Avg@4 (b) Pass@4 (c) Inherit (d) Adapt (e) Create

(f) Operation (g) Subgoal (h) Task-domain (i) Operation (j) Subgoal

Figure 2: Skill iteration on AppWorld with Qwen3-32B across rounds. (a) Avg@4 and (b) Pass@4 (%) on the training and test sets under three representative-instance selection strategies: Max-Skill (ours), Random, and Min-Skill. (c)–(e) Proportions (%) of reuse types (Inherit, Adapt, and Create) on both sets. (f)–(h) Numbers of skill nodes (bars, left axes) and new nodes per round (lines, right axes). (i)–(j) Creation and absorption rates per round in the operation and subgoal layers.

scope to a few nearest-neighbor tasks and thus misses relevant skills, while variant (a) expands the candidates to all subgoals of the located task domains, raising the orchestration burden.

4.4 Skill Iteration Analysis

We analyze skill iteration (Section 3.4) on AppWorld with Qwen3-32B, tracking task performance, reuse-type proportions, and memory bank growth over rounds (Figure 2). Rounds 1–3 iterate on the 90 training tasks, with evaluation on the test-normal set each round; rounds 4–6 iterate instead on 90 tasks sampled from the test-normal set, solely for memory growth analysis. Round 0 denotes memory-free execution. **(1) Task performance.** Figures 2(a)-(b) report Avg@4 and Pass@4 on the training and test sets, comparing three representative-instance selection strategies: (i) *Max-Skill* (ours; Section 3.2.3), retaining the successful instance or the one covering the most operation skills; (ii) *Random*, retaining a random instance per task; and (iii) *Min-Skill*, retaining the successful instance or the one covering the fewest. On the training set, all three rise monotonically, with Min-Skill $>$ Max-Skill $>$ Random. On the test set, Random rises steadily yet lags throughout due to low-quality retained instances; Min-Skill progressively over-specializes and dips slightly at round 3; Max-Skill performs best, as retaining successful instances ensures evidence quality and maximal coverage avoids shortcuts. **(2) Reuse-type proportions.** As iteration proceeds, the proportion of Inherit rises while that of Create falls on both sets (Figures 2(c)-(e)), indicating increasingly complete coverage of required skill patterns and stronger plan-step reuse. The training set favors Adapt whereas the test set favors Create, as in-distribution tasks rely mostly on Inherit and Adapt while tasks under distribution shift allow more Create. **(3) Memory bank growth.** Figures 2(g)-(i) reports the number of skill nodes per layer ($|V_{op}|$, $|V_{sg}|$, and $|V_{dm}|$) after each round: the sizes grow over the first three rounds, yet the number of new nodes per round decays monotonically as the bank converges; once iteration switches to the sampled test tasks in round 4, it rebounds immediately and decays again. Growth is driven by the two cluster outcomes, absorption and creation (Section 3.4); the creation rates of the operation and subgoal layers decay while the absorption rates rise, and the trend reverses after round 4. This empirically confirms the growth analysis in Appendix C.

5 Conclusion

Inspired by human procedural memory, we propose MSCPM, which extracts operation, subgoal, and task-domain skills bottom-up through clustering and chunking into a hierarchical skill graph with asymmetric inter-skill relations and provably convergent growth. Meanwhile, by coarse-to-fine subgoal selection, typed plan orchestration, and dynamic operation assembly, MSCPM composes skills vertically and horizontally into execution strategies tailored to new tasks. On AppWorld and BFCL-V3, MSCPM consistently outperforms representative baselines, with larger relative gains on smaller backbones, and its performance even surpass larger memory-free baselines.

AI USE STATEMENT

In this work, we used generative AI tools for tasks that require disclosure: assisting with translating the manuscript into English; assisting with research execution, including developing and debugging the experimental code and assisting with experiment design and analysis; and assisting in constructing and writing the proofs in Appendix C under the assumptions specified by the authors. We have not used generative AI tools to draft sections of the paper: the technical content was authored by the authors, and the tools assisted with translation and language editing rather than content generation. Tasks such as generating synthetic datasets and qualitative data analysis are not applicable to this work. Additionally, we used generative AI tools to edit the manuscript to improve its readability. We have reviewed all AI-assisted work: the AI-assisted translation and editing were examined and revised sentence by sentence to ensure that they faithfully convey the intended content and terminology; the AI-assisted code was reviewed and tested by the authors, and all reported results were produced and verified by the authors; and the AI-assisted proofs were checked line by line by the authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work develops a procedural memory framework for LLM-based agents and involves no human participants or human subjects, so no institutional review was required. All experiments are conducted on two public benchmarks, AppWorld and BFCL-V3. The user records, emails, and payment activities in AppWorld are synthetic data of about 100 virtual users generated by the benchmark and contain no personally identifiable information of real individuals; BFCL-V3 is a publicly released function-calling benchmark. This work does not collect, process, or release any new personal data, and all content in the procedural memory is distilled from the agents' own interaction trajectories in simulated environments rather than from private human data. As for potential societal impact, LLM-based agents that autonomously execute operations such as sending emails or making payments may cause unintended consequences or be misused when deployed in real environments. All experiments in this paper are performed in the simulated environments provided by the benchmarks and access no real third-party services or user accounts. Moreover, the procedural memory is organized as an explicit and interpretable skill graph, which facilitates manual inspection and auditing. The authors declare no conflicts of interest.

REPRODUCIBILITY STATEMENT

To support reproducibility, this statement points to the materials provided in the paper and the supplementary code. The complete implementation of MSCPM is released at `https://anonymous.4open.science/r/iclr2027_MSCPM-D2CF/`. For the theoretical results, Appendix C proves the growth of the memory bank (Proposition 1, with Lemmas 1 and 2) under the explicitly stated ϵ-separability assumption (Assumption 1) and further discusses extensions under imperfect clustering and annotation noise. For the experiments, Section 4 specifies the backbone models, the embedding model, the inference temperatures, the training and evaluation splits of the two benchmarks, and the evaluation metrics Avg@4 and Pass@4, which are defined in Appendix D.1; all results are averaged over three independent runs with standard deviations reported, and all experiments run on NVIDIA A100 GPUs. Appendix D.2 provides all hyperparameters and implementation details, including the per-layer clustering thresholds, the mixed-distance weight α, the PrefixSpan parameters, the retrieval sizes r, z, and k, and the damping factor and convergence threshold of PageRank. Appendix D.3 presents the complete prompt templates of all six LLM modules, and Appendix A summarizes the notation used throughout the paper.

REFERENCES

John R. Anderson, Michael Matessa, and Christian Lebiere. ACT-R: A theory of higher level cognition and its relation to visual attention. *Human–Computer Interaction*, 12(4):439–462, 1997.

T. Bai, Z. Wan, P. Zhou, et al. SkillDAG: Self-evolving typed skill graphs for LLM skill selection at scale. *arXiv preprint arXiv:2606.03056*, 2026.

Danielle S Bassett, Nicholas F Wymbs, Mason A Porter, Peter J Mucha, Jean M Carlson, and Scott T Grafton. Dynamic reconfiguration of human brain networks during learning. *Proceedings of the National Academy of Sciences*, 108:7641–7646, 2011.

Sugato Basu, Arindam Banerjee, and Raymond J. Mooney. Semi-supervised clustering by seeding. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML)*, pp. 27–34, 2002.

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. In *Findings of the Association for Computational Linguistics: ACL*, pp. 16803–16822, 2026.

M. Chen, Y. Li, Y. Yang, et al. AutoManual: Constructing instruction manuals by LLM agents via interactive environmental learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 589–631, 2024.

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*, pp. 17490–17502, 2026.

Y. Fu, D. K. Kim, J. Kim, et al. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 119919–119948, 2024.

Shijie Geng et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

Peter Graf and Daniel L Schacter. Implicit and explicit memory for new associations in normal and amnesic subjects. *Journal of Experimental Psychology: Learning, memory, and cognition*, 11(3): 501, 1985.

Ann M Graybiel. The basal ganglia and chunking of action repertoires. *Neurobiology of learning and memory*, 70(1-2):119–136, 1998.

D. Han, C. Couturier, D. M. Diaz, et al. LEGOMem: Modular procedural memory for multi-agent LLM systems for workflow automation. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 3116–3118, 2026.

Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 17th International Conference on Data Engineering (ICDE)*, pp. 215–224, 2001.

Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 2009.

John E Laird. *The Soar cognitive architecture*. The MIT Press, 2012.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, et al. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. In *International Conference on Learning Representations (ICLR)*, volume 2026, pp. 53359–53394, 2026.

Y. Liang, R. Zhong, H. Xu, et al. SkillNet: Create, evaluate, and connect AI skills. *arXiv preprint arXiv:2603.04448*, 2026.

Dawei Liu, Zongxia Li, Hongyang Du, Xiyang Wu, Shihang Gui, Yongbei Kuang, and Lichao Sun. Graph-of-skills: Dependency-aware structural retrieval for massive agent skills. *arXiv preprint arXiv:2604.05333*, 2026. Accepted to EMNLP 2026.

Y. Liu, C. Si, K. R. Narasimhan, et al. Contextual experience replay for self-improvement of language agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 14179–14198, 2025.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations (ICLR)*, volume 2026, pp. 94327–94354, 2026.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 2025.

Z. Qi, X. Liu, I. L. Iong, et al. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations (ICLR)*, pp. 79791–79821, 2025.

J. J. Shao, H. Yin, Y. Lyu, et al. Lifting traces to logic: Programmatic skill induction with neuro-symbolic learning for long-horizon agentic tasks. *arXiv preprint arXiv:2605.01293*, 2026.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, He Zhu, Ge Zhang, Jiaheng Liu, et al. AGENT KB: A hierarchical memory framework for cross-domain agentic problem solving. In *ICML 2025 Workshop on Collaborative and Federated Agentic Workflows*, 2025.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL), Volume 1: Long Papers*, pp. 16022–16076, 2024.

Endel Tulving. Memory and consciousness. *Canadian Psychology*, 26(1):1–12, 1985.

Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, pp. 577–584, 2001.

Chenxi Wang, Peng Zhang, and Shumin Deng. SkillX: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026a.

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.

Zhaoyang Wang, Qianhui Wu, Xuchao Zhang, Chaoyun Zhang, Wenlin Yao, Fazle Elahi Faisal, Baolin Peng, Si Qin, Suman Nath, Qingwei Lin, Chetan Bansal, Dongmei Zhang, Saravan Rajmohan, Jianfeng Gao, and Huaxiu Yao. WebXSkill: Skill learning for autonomous web agents. *arXiv preprint arXiv:2604.13318*, 2026b.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267, pp. 63897–63911, 2025.

Zijun Wu, Yongchang Hao, and Lili Mou. TokMem: One-token procedural memory for large language models. In *International Conference on Learning Representations (ICLR)*, volume 2026, pp. 32460–32480, 2026.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

L. Yang, Z. Yu, T. Zhang, et al. Buffer of thoughts: Thought-augmented reasoning with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 113519–113544, 2024.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. MemGen: Weaving generative latent memory for self-evolving agents. In *International Conference on Learning Representations (ICLR)*, volume 2026, pp. 22555–22588, 2026.

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

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *International Conference on Learning Representations (ICLR)*, volume 2024, pp. 19036–19066, 2024.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, volume 2024, pp. 15585–15606, 2024.

Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Erran Li. Proposer-agent-evaluator (PAE): Autonomous skill discovery for foundation model internet agents. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 79490–79528. PMLR, 2025.

A NOTATION TABLE

Table 3 summarizes the notation used throughout this paper. Entries are grouped following the structure of Sections 3.2–3.4, and within each group ordered by first appearance in the main text.

Table 3: Summary of notation.

Symbol	Meaning
General conventions	
$(\cdot)$, $\langle\cdot\rangle$, $\{\cdot\}$, $[\cdot]$	Parentheses denote ordered tuples and sequences, angle brackets structured records, braces sets, and brackets integer intervals (e.g., $[N] = \{1, \ldots, N\}$).
MSCPM overview	
h_t, θ	Interaction history up to time step t; parameters of the backbone model.
$\tau = (o_1, a_1, \ldots, o_T, a_T)$	A complete trajectory, with o_t and a_t the observation and action of the t-th step and T the trajectory length; the n-th trajectory in the pool is denoted τ_n.
$\Xi = \{\tau_n\}_{n=1}^{N}$	Pool of raw interaction trajectories (N trajectories in total).
$\mathcal{Q}$, q, q'	Task set, a training (historical) task instruction, and the current new task instruction.
$\pi_\theta(a_t \mid h_t)$	Agent policy parameterized by θ.
Φ, Ψ, B	Memory construction module ($\mathcal{M} = \Phi(\Xi)$), skill composition module ($B = \Psi(\mathcal{M}, q)$), and the execution strategy the latter composes for a task.
τ', $\mathcal{R}(\tau')$	Trajectory generated by the memory-augmented agent ($\tau' \sim \pi_\theta(\cdot \mid q, B)$) and its cumulative reward, with $\mathcal{R}(\tau') \in [0, 1]$.
Hierarchical skill graph: overall structure	
$\mathcal{M} = \big(V_{op} \cup V_{sg} \cup V_{dm},\ E_{dm\to sg} \cup E_{sg\to op}\big)$	Hierarchical skill graph; V_{op}, V_{sg}, and V_{dm} are the skill-node sets of the operation, subgoal, and task-domain layers; only the downward cross-layer edges $E_{dm\to sg} \subseteq V_{dm} \times V_{sg}$ and $E_{sg\to op} \subseteq V_{sg} \times V_{op}$ are stored.
$\mathrm{child}(v)$, $\mathrm{par}(v)$	Children and parents of node v along the cross-layer edges.
Operation layer	
$\Phi_{op\text{-lab}}$, $\Phi_{op\text{-ind}}$	LLM-based context-aware action annotator (applied as $\Phi_{op\text{-lab}}(\tau_n, q)$) and operation inducer.
$op_{n,t} = \langle \mathrm{name}_{n,t}, \mathrm{desc}_{n,t}, a_{n,t} \rangle$	Action annotation of the t-th step of trajectory τ_n (action name, description, and executed action); all annotations form the set $\mathcal{A}_{op} = \{op_{n,t} \mid n \in [N],\ t \in [T]\}$.
$e : \mathcal{A}_{op} \to \mathbb{R}^d$	Semantic embedding function over names and descriptions (d: embedding dimension), with cosine similarity as the metric; reused throughout for task instructions, plan steps, and skill nodes (e.g., $e(q)$, $e(p_l)$).
κ_{op}, $\mathcal{C}_{op} = \{\mathcal{C}_{op}^i\}$	Clustering operator on action annotations and the disjoint clusters it induces (κ_{sg} and κ_{dm} are defined likewise).
$v_i^{op} = \Phi_{op\text{-ind}}(\mathcal{C}_{op}^i)$, V_{op}	i-th operation node induced from cluster $\mathcal{C}_{op}^i$ and the operation node set.
$\mathrm{attr}(v)$, $\mathrm{exmp}(\cdot)$, $\mathrm{agg}[\cdot]$	Node attribute record $\langle$name, description, examples$\rangle$; example actions sampled from the cluster (operation nodes only); and the aggregation of synonymous attributes over the cluster, e.g., $\mathrm{name}(v_i^{op}) = \mathrm{agg}_{op \in \mathcal{C}_{op}^i}[\mathrm{name}(op)]$ (likewise for desc).
Subgoal layer	
$\sigma_{op} : \mathcal{A}_{op} \to V_{op}$	Lookup mapping from each action annotation to the skill node induced from its cluster (σ_{sg} likewise).
$\tilde{\tau}_n^{op}$, $\tilde{\Xi}$	Symbolic trajectory—the ordered operation sequence obtained by projecting τ_n through σ_{op}—and the set of all symbolic trajectories; heterogeneous raw interactions are thereby unified in a symbolic space.

Continued on next page

Table 3: Summary of notation (continued).

Symbol	Meaning
$\mathcal{F}_h$, $\mathrm{supp}(f)$, $s_{\min}$, $l_{\min}$, $l_{\max}$	Set of frequent sequence patterns mined from $\tilde{\Xi}$ (by PrefixSpan), $\mathcal{F}_h = \{f \mid \mathrm{supp}(f) \geq s_{\min},\ \lvert f\rvert \in [l_{\min}, l_{\max}]\}$, with $\mathrm{supp}(f)$ the number of occurrences of f in $\tilde{\Xi}$, $s_{\min}$ the minimum support threshold, and $l_{\min}$/$l_{\max}$ the minimum/maximum pattern lengths.
u^{op}, $g(u^{op})$, Π_n^*, S	Contiguous subsequence of a symbolic trajectory and its coverage score ($\lvert u^{op}\rvert \cdot \mathrm{supp}(u^{op})$ for a frequent pattern, 0 otherwise), the optimal segmentation of a trajectory maximizing the total coverage, and its number of segments ($\Pi_n = (u_1^{op}, \ldots, u_S^{op})$).
$\Phi_{sg\text{-lab}}$, $\Phi_{sg\text{-ind}}$, κ_{sg}, $\mathcal{C}_{sg} = \{\mathcal{C}_{sg}^j\}$	LLM-based subgoal annotator, subgoal inducer, and the associated clustering operator and cluster set.
$sg_{n,s} = \langle \mathrm{name}_{n,s}, \mathrm{desc}_{n,s}, u_s^{op}\rangle$	Annotation of the subgoal accomplished by the s-th segment of trajectory τ_n; all subgoals form $\mathcal{A}_{sg} = \{sg_{n,s} \mid n \in \mathcal{I}_{succ},\ s \in [S]\}$, where $\mathcal{I}_{succ} \subseteq [N]$ is the index set of successful trajectories.
v_j^{sg}, V_{sg}, $u^{op}(sg)$	j-th subgoal node, subgoal node set, and the operation sequence bound to subgoal sg; cross-layer edges are established by $\mathrm{child}(v_j^{sg}) = \bigcup_{sg \in \mathcal{C}_{sg}^j} u^{op}(sg)$.
Task-domain layer	
$\tilde{\tau}_n^{sg}$	Ordered subgoal sequence obtained by projecting the subgoal annotations of τ_n through the lookup mapping σ_{sg}.
$\omega_n = \langle q_n, \tilde{\tau}_n^{op}, \tilde{\tau}_n^{sg}\rangle$, $\hat{\Xi} = \{\omega_n\}_{n=1}^N$	Task instance (task instruction with its two symbolic trajectories) and the set of all instances.
$\bar{\omega}(q)$, $\mathcal{A}_{dm}$	Representative instance of task q and the representative set $\mathcal{A}_{dm} = \{\bar{\omega}(q) \mid q \in \mathcal{Q}\}$, the input to task-domain clustering; among the candidate instances $\{\omega_n \mid q_n = q\}$, successful trajectories are preferred and, among them, the one whose deduplicated operation-skill sequence covers more distinct skills.
$d_{\mathrm{mix}}(\cdot,\cdot)$, α, $\mathrm{lev}(\cdot,\cdot)$	Hybrid semantic–structural distance between task instances (a weighted combination of the cosine distance between task-instruction embeddings and the length-normalized edit distance between operation sequences), the balancing weight $\alpha \in [0,1]$, and the edit distance.
κ_{dm}, $\mathcal{C}_{dm} = \{\mathcal{C}_{dm}^k\}_{k=1}^K$, v_k^{dm}, V_{dm}	Clustering operator under d_{mix}, its disjoint task-domain clusters (K: number of task domains), the task-domain node induced from each cluster, with $\mathrm{desc}(v_k^{dm})$ the set of task instructions in the domain, and the task-domain node set.
Multi-level skill composition	
Top-r/Top-z/Top-k, $\cos(\cdot,\cdot)$	Selection operators keeping the top r/z/k items by score, and the cosine similarity function.
V_{dm}^*, V_{sg}^*	Task-domain nodes located via a lookup on node attributes, covering the Top-r existing tasks most similar to q', and the candidate subgoal set recalled along $E_{dm\to sg}$, $V_{sg}^* = \bigcup_{v_k^{dm} \in V_{dm}^*} \mathrm{child}(v_k^{dm})$.
Ψ_{sel}, $\hat{V}_{sg}$	LLM-based subgoal selector and the subgoal skills it selects for task q', $\hat{V}_{sg} = \Psi_{\mathrm{sel}}(V_{sg}^*, q') \subseteq V_{sg}^*$.
Ψ_{orch}, $P(q') = (p_1, \ldots, p_L)$	LLM-based subgoal orchestrator and the high-level plan it generates for q' (L: number of plan steps).
$p_l = \langle \mathrm{name}_l, \mathrm{desc}_l, \mathrm{type}_l, v_l^{sg}\rangle$	The l-th plan step: name, description, the way it reuses the bound subgoal skill—$\mathrm{type}_l \in \{\mathrm{Inherit}, \mathrm{Adapt}, \mathrm{Create}\}$, i.e., reuse as a whole, reuse with adjustments, or construct from scratch (Create has $v_l^{sg} = \varnothing$)—and the bound subgoal skill $v_l^{sg} \in \hat{V}_{sg} \cup \{\varnothing\}$.
$V_{\mathrm{seed}}^{(l)}$, $\hat{\mathcal{N}}(v)$	Seed operation-skill set of plan step l—for Adapt, the union of the Top-z semantic retrieval with the bound skills $\mathrm{child}(v_l^{sg})$; for Create, the retrieval part only—and the sibling skills of seed v, $\big(\bigcup_{v^{sg} \in \mathrm{par}(v)} \mathrm{child}(v^{sg})\big) \setminus \{v\}$, induced online by tracing the cross-layer edges upward and then downward.

Continued on next page

Table 3: Summary of notation (continued).

Symbol	Meaning
$\mathcal{G}_{op}^{(l)} = \left(V_{op}^{(l)}, \hat{E}_{op\leftrightarrow op}^{(l)}\right)$	Operation-skill co-activation graph of plan step l, which holds only transiently for the current task: the nodes are the seeds united with their sibling skills, and bidirectional edges connect skill pairs sharing a subgoal ($\mathrm{par}(v) \cap \mathrm{par}(v') \neq \emptyset$).
$\hat{V}_{op}^{(l)}$, $B = \{(p_l, \hat{V}_{op}^{(l)})\}_{l=1}^{L}$	Operation-skill set assembled for plan step l—Inherit directly takes $\hat{V}_{op}^{(l)} = \mathrm{child}(v_l^{sg})$, whereas Adapt/Create take the Top-k skills ranked by co-occurrence importance via PageRank on $\mathcal{G}_{op}^{(l)}$—and the hierarchical execution strategy finally composed for the task, under which $\tau' \sim \pi_\theta(\cdot \mid q', B)$.
Skill iteration	
ΔN, m, κ^{cl}	Iteration batch size (each batch of ΔN new trajectories triggers one round of iteration), the iteration-round index (superscript (m), as distinct from the plan-step superscript (l)), and the seeded constrained clustering operator that replaces the clustering operators of the construction pipeline—clustering new annotations with existing skill nodes as seeds under cannot-link constraints, so that each cluster contains at most one seed; a seeded cluster is absorbed into its seed node (attributes unchanged), and an unseeded one yields a new node.
$K_{\mathcal{H}}$	Total number of patterns in the experience (the pattern complexity of tasks and environments); it upper-bounds the number of skills in the memory bank, independently of the number of iteration rounds and trajectories.

B Pseudocode of Hierarchical Skill Graph Construction

Algorithm 1 summarizes the construction procedure of the hierarchical skill graph $\mathcal{M}$ described in Section 3.2. Given the trajectory pool Ξ, it builds the node sets of the three layers bottom-up through annotation, clustering, and induction, establishes the cross-layer edges, and prunes the nodes uncovered by the representative instances.

C Growth Analysis of the Memory Bank: Formalization and Proofs

To make the analysis self-contained, we first give a formal instantiation of the seeded constrained clustering operator κ^{cl} that realizes the absorption or creation mechanism described in the main text.

Index embedding. The index embedding $e(v)$ of a node v is defined as the embedding of its representative annotation at the moment the node is first created and is frozen thereafter. Under the definitions in the main text, absorption merges only new annotations into the node, leaving its name, description, and index embedding unchanged.

Absorption. An annotation a is absorbed by a seed $v \in V^{(m)}$ if and only if $\|e(a) - e(v)\| < \epsilon$; if several seeds qualify, a is assigned to the nearest one (by Lemma 2, this case never arises).

Creation. Annotations absorbed by no seed are grouped by single-linkage clustering with threshold ϵ, and the inducer Φ_{ind} creates one new node from each connected cluster.

This construction immediately yields the following fact, cited directly wherever node identity must remain stable: node sets only grow,

$$V^{(m)} \subseteq V^{(m+1)}, \tag{P1}$$

and every existing node keeps its name and index embedding across rounds. Note that the iteration in Section 3.4 applies graph pruning to the subgraph newly derived from the incoming trajectories and merges it into the existing hierarchical skill graph only afterwards; pruning removes nodes only and creates none, so it cannot enlarge the node set, and the size bounds below remain valid. Moreover,

Algorithm 1 Hierarchical Skill Graph Construction (Section 3.2)

Require: Trajectory pool $\Xi = \{\tau_n\}_{n=1}^{N}$ with associated tasks $\{q_n\}$; annotators and inducers $\Phi_{op\text{-lab}}, \Phi_{op\text{-ind}}, \Phi_{sg\text{-lab}}, \Phi_{sg\text{-ind}}$; clustering operators $\kappa_{op}, \kappa_{sg}, \kappa_{dm}$; mining thresholds $s_{\min}, l_{\min}, l_{\max}$

Ensure: Hierarchical skill graph $\mathcal{M} = \big(V_{op} \cup V_{sg} \cup V_{dm},\ E_{sg\to op} \cup E_{dm\to sg}\big)$

% *Operation layer (Section 3.2.1)*

$\mathcal{A}_{op} \leftarrow \big\{\Phi_{op\text{-lab}}(\tau_n, q_n)\big\}_{n=1}^{N}$ ▷ context-aware action annotation

$\{\mathcal{C}_{op}^{i}\}_i \leftarrow \kappa_{op}(\mathcal{A}_{op});\quad V_{op} \leftarrow \big\{\Phi_{op\text{-ind}}(\mathcal{C}_{op}^{i})\big\}_i$ ▷ clustering and induction

$\tilde{\Xi} \leftarrow \big\{\big(\sigma_{op}(op_{n,1}), \ldots, \sigma_{op}(op_{n,T})\big)\big\}_{n=1}^{N}$ ▷ symbolic trajectories

% *Subgoal layer (Section 3.2.2)*

$\mathcal{F}_h \leftarrow$ frequent patterns mined from $\tilde{\Xi}$ with PrefixSpan ▷ $\mathrm{supp}(f) \geq s_{\min}$, $|f| \in [l_{\min}, l_{\max}]$

for $n \in \mathcal{I}_{succ}$ **do**

  $\Pi_n^{*} \leftarrow \arg\max_{\Pi_n} \sum_{u^{op}\in\Pi_n} g\big(u^{op}\big)$ ▷ chunking by coverage score, Eq. equation 7

end for

$\mathcal{A}_{sg} \leftarrow \big\{\Phi_{sg\text{-lab}}\big(u_s^{op}, q_n\big)\big\}_{n\in\mathcal{I}_{succ},\, s\in[S]}$ ▷ subgoal annotation of chunks

$\{\mathcal{C}_{sg}^{j}\}_j \leftarrow \kappa_{sg}(\mathcal{A}_{sg});\quad V_{sg} \leftarrow \big\{\Phi_{sg\text{-ind}}(\mathcal{C}_{sg}^{j})\big\}_j$ ▷ clustering and induction

$E_{sg\to op} \leftarrow \Big\{\big(v_j^{sg}, v_i^{op}\big) \;\Big|\; v_i^{op} \in \mathrm{child}\big(v_j^{sg}\big)\Big\},\quad \mathrm{child}\big(v_j^{sg}\big) = \bigcup_{sg\in\mathcal{C}_{sg}^{j}} u^{op}(sg)$

% *Task-domain layer (Section 3.2.3)*

$\tilde{\tau}_n^{sg} \leftarrow \big(\sigma_{sg}(sg_{n,1}), \ldots, \sigma_{sg}(sg_{n,S})\big);\quad \omega_n \leftarrow \big\langle q_n, \tilde{\tau}_n^{op}, \tilde{\tau}_n^{sg}\big\rangle$ for all n

$\mathcal{A}_{dm} \leftarrow \big\{\bar{\omega}(q) \;\big|\; q \in \mathcal{Q}\big\}$ ▷ prefer success, then most distinct skills covered

$\{\mathcal{C}_{dm}^{k}\}_{k=1}^{K} \leftarrow \kappa_{dm}(\mathcal{A}_{dm});\quad V_{dm} \leftarrow \{v_k^{dm}\}_{k=1}^{K}$ ▷ mixed-distance clustering and induction

$E_{dm\to sg} \leftarrow \Big\{\big(v_k^{dm}, v_j^{sg}\big) \;\Big|\; v_j^{sg} \in \mathrm{child}\big(v_k^{dm}\big)\Big\},\quad \mathrm{child}\big(v_k^{dm}\big) = \bigcup_{\omega_n\in\mathcal{C}_{dm}^{k}} \tilde{\tau}_n^{sg}$

% *Pruning*

prune the subgoal and operation nodes uncovered by $\mathcal{A}_{dm}$, together with the cross-layer edges incident to them

return $\mathcal{M} = \big(V_{op} \cup V_{sg} \cup V_{dm},\ E_{sg\to op} \cup E_{dm\to sg}\big)$

the proofs rely only on the pair of inequalities that same-pattern distances are smaller than ϵ and cross-pattern distances are at least ϵ, never on the triangle inequality, so $\|\cdot\|$ can be taken directly as the cosine distance.

Assumption 1 (ϵ-separable semantic patterns)**.** *There exist a finite set of semantic patterns $\mathcal{H} = \{H_1, \ldots, H_{K_\mathcal{H}}\}$ and a distribution μ over $\mathcal{H}$ such that each annotation belongs to a pattern sampled i.i.d. from μ, the embeddings of same-pattern annotations are less than ϵ apart, and those of different-pattern annotations are at least ϵ apart, where ϵ coincides with the threshold used by κ^{cl}.*

Here $K_\mathcal{H}$ characterizes the pattern complexity of the tasks and the environment. For the operation and subgoal layers, whose iteration is governed by κ^{cl},[1] let $V^{(m)}$ be the node set of the layer at the beginning of iteration round m, and $\mathcal{O}_m \subseteq \mathcal{H}$ the set of patterns covered within the first m rounds, i.e., those with a corresponding node.

Proposition 1 (Coverage-driven growth)**.** *Under Assumption 1: (a) $|V^{(m)}| = |\mathcal{O}_m| \leq K_\mathcal{H}$, i.e., the number of skills equals the number of covered patterns and is independent of the number of iteration rounds and trajectories. (b) Let the missing mass $M_m := \sum_{k:H_k\notin\mathcal{O}_m} \mu_k$ be the probability of the patterns uncovered at the beginning of round m. If round $m+1$ contains b new annotations, then*

$$\mathbb{E}\big[|V^{(m+1)}| - |V^{(m)}|\big] \;\leq\; b\, M_m, \tag{18}$$

and M_m is monotonically non-increasing over rounds and converges to zero.

Lemma 1 (No cross-pattern merging)**.** *Under Assumption 1, the annotations within any cluster belong to the same pattern.*

[1] Iteration on the task-domain layer builds on the mixed distance and representative-instance selection and thus differs in form from κ^{cl}.

Proof. Consider a seeded cluster. If an annotation a is absorbed by a seed v, then $\|e(a)-e(v)\| < \epsilon$; since $e(v)$ is the embedding of an annotation from some pattern H_k, and annotations from different patterns are at least ϵ apart, a also belongs to H_k.

Consider an unseeded cluster. Two annotations directly linked by single linkage at threshold ϵ are less than ϵ apart and hence belong to the same pattern. Since the patterns partition the annotation set, sharing a pattern is an equivalence relation, and transitivity along connected paths extends it to the entire cluster. □

Lemma 2 (Pattern-to-node injection)**.** *Under Assumption 1, for every m, each pattern corresponds to at most one node, and the index embeddings of distinct nodes are pairwise at least ϵ apart.*

Proof. We argue by induction on the round m. For $m = 0$, $V^{(0)} = \varnothing$ and the claim holds trivially. Suppose the claim holds at round m.

If a pattern $H_k \in \mathcal{O}_m$ has its corresponding node v, then any new annotation a of H_k satisfies $\|e(a) - e(v)\| < \epsilon$ by the same-pattern diameter, while by the induction hypothesis all other seeds belong to different patterns, so $\|e(a) - e(v')\| \geq \epsilon$ for every other seed v'. Hence a is absorbed exactly by v and creates no node, and v is never deleted or renamed afterwards by equation P1. New nodes in this round can therefore arise only from patterns outside $\mathcal{O}_m$.

By Lemma 1, the annotations within each new cluster share one pattern; conversely, any two annotations of the same uncovered pattern are less than ϵ apart and are thus placed in one connected cluster by single linkage. Each uncovered pattern therefore yields at most one new node in this round. The new nodes take their index embeddings from mutually distinct patterns, each different from those of the existing nodes, and are thus pairwise at least ϵ apart. The induction is complete. □

By this lemma, distinct seeds belong to distinct patterns, so each annotation falls within the ϵ-neighborhood of at most one seed; the absorption assignment is thus unique, and the cannot-link constraints between seeds are satisfied automatically.

Throughout the proof, let n_m denote the cumulative number of annotations within the first m rounds, and $\mathfrak{F}_m$ the σ-algebra generated by the annotations of the first m rounds.

Proof of Proposition 1. **(a)** By Lemmas 1 and 2, the map from each node to its pattern is well defined and injective, so $|V^{(m)}| \leq |\mathcal{O}_m|$. Conversely, let $H_k \in \mathcal{O}_m$, and let its first appearance be an annotation a in round $m' \leq m$. By Lemma 1, a node is created only by annotations of its own pattern, so no node of H_k exists at the beginning of round m'; the annotation a is thus at least ϵ away from every seed, is absorbed by none, enters a new cluster, and creates the node of H_k, which persists thereafter by equation P1. The map is therefore a bijection from $V^{(m)}$ onto $\mathcal{O}_m$, so $|V^{(m)}| = |\mathcal{O}_m| \leq K_{\mathcal{H}}$, independent of both m and n_m.

(b) By the argument in (a), the nodes added in round $m + 1$ are exactly those of the uncovered patterns appearing among its b annotations. Conditioned on $\mathfrak{F}_m$, a pattern H_k appears at least once among the b independent annotations with probability $1-(1-\mu_k)^b$, so linearity of expectation gives

$$\mathbb{E}\big[\,|V^{(m+1)}| - |V^{(m)}| \;\big|\; \mathfrak{F}_m\,\big] = \sum_{k:H_k \notin \mathcal{O}_m} \big(1-(1-\mu_k)^b\big) \;\leq\; b\,M_m, \tag{19}$$

where the inequality follows from Bernoulli's inequality $(1-\mu_k)^b \geq 1 - b\mu_k$, with equality when $b = 1$. Taking a further expectation (tower property) yields the unconditional form stated in the main text. Since $\mathcal{O}_m$ grows monotonically with m, M_m is non-increasing along every sample path. Finally, fix the per-round batch sizes so that n_m is deterministic; we show $\mathbb{E}[M_m] \to 0$. By (a), $H_k \notin \mathcal{O}_m$ if and only if H_k is absent from the first n_m annotations, so independent sampling gives

$$\begin{aligned} \mathbb{E}[M_m] &= \sum_k \mu_k \Pr[H_k \notin \mathcal{O}_m] = \sum_k \mu_k (1-\mu_k)^{n_m} \leq \sum_k \mu_k e^{-n_m \mu_k} \\ &\leq K_{\mathcal{H}} \max_{t>0} t e^{-n_m t} = \frac{K_{\mathcal{H}}}{e\, n_m} \to 0. \quad \square \end{aligned} \tag{20}$$

Extensions. The qualitative conclusions survive two relaxations. First, if the clustering operator only guarantees that same-pattern annotations are assigned to at most c clusters, Lemma 2 weakens to at most c nodes per pattern, and both upper bounds in Proposition 1 scale by a factor of c. Second, if at most an η fraction of the annotations are assigned to wrong patterns by the annotator, each round additionally creates a number of nodes proportional to η at most; this is one motivation for the context-aware prompts and the aggregation of synonymous attributes in our operation and subgoal annotation (Sections 3.2.1 and 3.2.2).

D Detailed Experimental Setup

D.1 Benchmark Details

BFCL-V3. The Berkeley Function Calling Leaderboard V3 (BFCL-V3) (Patil et al., 2025) benchmarks the function calling and tool usage capabilities of LLMs, emphasizing multi-turn interaction and multi-step reasoning. It contains more than 1,800 test instances across multiple programming languages, including Python, Java, and JavaScript, and requires models to generate valid API calls and handle nontrivial interaction patterns. Evaluation considers both structural validity and functional correctness: we first check the syntactic validity of the generated code via abstract syntax tree (AST) analysis and then execute the code to verify that its output matches the expected result. A task is deemed successful only when the agent produces every required function call with valid syntax and returns the correct result.

AppWorld. AppWorld (Trivedi et al., 2024) is a benchmark suite that evaluates function-calling agents and interactive coding systems in realistic application environments. It simulates an ecosystem of nine everyday applications, such as email services, music streaming platforms, and payment systems, offering 457 API endpoints together with the activity data of about 100 virtual users. Tasks in AppWorld are typically long-horizon, requiring long sequences of interdependent actions; many require exploring and discovering suitable APIs rather than directly reusing familiar patterns, placing higher demands on exploration and planning. Moreover, the benchmark exhibits a pronounced train–test distribution shift: the API usage patterns and task structures of the test set differ from those observed during training. Task execution is also tightly coupled with a dynamically evolving environment state: intermediate actions modify the system state and affect subsequent decisions, increasing the sensitivity to planning errors and making robust multi-step reasoning more difficult. Evaluation relies on state-driven unit tests that assess task completion along multiple dimensions.

Metrics. Each task is executed independently four times. Avg@4 denotes the average success rate over the four executions of each task, and Pass@4 denotes the proportion of tasks that succeed in at least one of the four executions. All reported results are averaged over three independent runs.

D.2 Implementation Details

Model serving. The Qwen3 backbones and the Qwen3-Embedding-8B embedding model are served locally with vLLM, with the context length set to the maximum supported by each model.

Skill graph construction. The clustering operator of each layer is implemented as agglomerative hierarchical clustering with average linkage (Kaufman & Rousseeuw, 2009): it merges the closest pair of clusters bottom-up and stops once their distance exceeds the layer threshold, requiring no pre-specified number of clusters. The operation and subgoal layers take the cosine distance between elements and define the distance between clusters C and C' as the average distance between their elements:

$$d(C, C') = \frac{1}{|C|\,|C'|} \sum_{a \in C} \sum_{a' \in C'} \Big(1 - \frac{e(a) \cdot e(a')}{\|e(a)\|\,\|e(a')\|}\Big). \tag{21}$$

The thresholds are set to $\epsilon_{op} = \epsilon_{sg} = 0.23$ on BFCL-V3 and $\epsilon_{op} = \epsilon_{sg} = 0.28$ on AppWorld. The task-domain layer replaces the element-wise distance with the mixed distance d_{mix} (Section 3.2.3), taking $\alpha = 0.6$ and $\epsilon_{dm} = 0.40$ on BFCL-V3 and $\alpha = 0.4$ and $\epsilon_{dm} = 0.35$ on AppWorld. In skill iteration, the seeded constrained clustering operator κ^{cl} reuses the distance measure and threshold of the corresponding layer, adding only the constraints that existing nodes serve as seeds and that no two seeds are merged. Frequent-pattern mining adopts PrefixSpan (Han et al., 2001) on both benchmarks, with minimum support $s_{\min} = 5$ and pattern lengths within $[l_{\min}, l_{\max}] = [2, 4]$.

Skill composition. The skill composition settings are identical on the two benchmarks. Subgoal selection retrieves the $r = 6$ most similar task instructions; operation assembly retrieves $z = 6$ operation skills for the seed set and assembles $k = 5$ operation skills per plan step. PageRank is implemented as its personalized variant with restart, with damping factor 0.85 and convergence threshold 10^{-6}. Transition weights take the inverse-frequency form $w(v) = \log \frac{1+|V_{sg}|}{1+|\mathrm{par}(v)|}$, where $|\mathrm{par}(v)|$ counts the subgoals bound to v, to keep high-frequency skills from dominating the ranking; the restart preference $\phi_l(v) \propto \cos\big(e(p_l),\, e(v)\big)$, normalized over $V_{op}^{(l)}$, ties the ranking to the semantics of plan step p_l.

D.3 Prompt Template Details

All LLM modules in hierarchical skill graph construction (Section 3.2) and multi-level skill composition (Section 3.3) are driven by structured prompts. This section presents the complete templates: Prompts D.1–D.4 are used for offline skill annotation and induction, and Prompts D.5 and D.6 for online skill composition.

Action Annotator $\Phi_{op\text{-lab}}$ (Prompt D.1). Applied in operation-layer construction (Section 3.2.1), it abstracts the code of each interaction step of a trajectory into a pseudo action signature with generic parameter placeholders, free of implementation details.

Prompt D.1 Action Annotator $\Phi_{op\text{-lab}}$

```
You are a senior software architect. In this task, you are given ONE
    task instruction together with the agent's execution trajectory
    collected under it. Annotate each interaction step of the
    trajectory: abstract each step into exactly one general pseudo
    action signature (name + parameter placeholders only; do NOT
    write any implementation). The task instruction provides the
    context for understanding what each step intends to accomplish.

========== 1. Annotation Rules ==========

1. Action Naming
   - Use snake_case with generic parameter placeholders; reflect
    what the step "intends to do", not a literal translation of the
    code.
   - Hard-coded concrete values are strictly forbidden: abstract
    away names, dates, place names, numeric thresholds and string
    literals.

2. Parameter Placeholders
   - Parameters hold only the inputs the caller must logically
    supply; never pass a value the action itself fetches or has
    already fetched as a parameter.

3. Description
   - Use ONE sentence following the pattern: "<verb-object> +
    <object> + [condition], (optional: to <goal>)". Keep a moderate
    granularity: neither too vague nor pinned to specific names /
    dates / numbers.

========== 2. Output Format ==========

Return strictly a JSON array. No extra explanation or comments. The
    array length must equal the number of steps in the input
    trajectory. Format:

[
  {
    "idx": <idx>,
    "action_name": "action_name(params)",
```

```
    "description": "the specific intent of this step"
  }
]
```

Operation Inducer $\Phi_{op\text{-ind}}$ (Prompt D.2). Applied in operation-layer construction (Section 3.2.1), it induces the abstract signature and description of the operation node from each cluster obtained by semantic clustering; the example attribute $\mathrm{exmp}(\cdot)$ is sampled programmatically from the executed actions within the cluster and requires no prompt.

Prompt D.2 Operation Inducer $\Phi_{op\text{-ind}}$

```
You are a strict operation normalization expert. The input consists
    of operation skill clusters obtained by semantic clustering. For
    each cluster, generate an abstract signature of the operation
    skill together with the corresponding description.

========== 1. Induction Rules ==========

1. Operation Naming
   - Use snake_case with generic parameter placeholders; reflect
    what the operation "intends to do", not a literal translation of
    the code.
   - Hard-coded concrete values are strictly forbidden: abstract
    away names, dates, place names, numeric thresholds and string
    literals.

2. Parameter Placeholders
   - Parameters hold only the inputs the caller must logically
    supply; never pass a value the operation itself fetches or has
    already fetched as a parameter.

3. Description
   - Use ONE sentence following the pattern: "<verb-object> +
    <object> + [condition], (optional: to <goal>)". Keep a moderate
    granularity: neither too vague nor pinned to specific names /
    dates / numbers.

========== 2. Output Format ==========

Return strictly a JSON array. No extra explanation or comments. The
    array length must equal the number of operation clusters in the
    input. Format:

[
  {
    "idx": <idx>,
    "operation_name": "operation_name(params)",
    "description": "the specific intent of this operation"
  }
]
```

Subgoal Annotator $\Phi_{sg\text{-lab}}$ (Prompt D.3). Applied in subgoal-layer construction (Section 3.2.2), it generates the abstract signature and description of the subgoal for each operation chunk produced by the chunking that maximizes the coverage score (Eq. equation 7); in the prompt, chunk boundaries are delimited by `ANCHOR` markers.

Prompt D.3 Subgoal Annotator $\Phi_{sg\text{-lab}}$

```
You are an expert in task operation sequence analysis. You are given
    ONE task instruction together with the operation skill sequence
    of a trajectory; the sequence has already been partitioned into
    multiple subsequences, each delimited by the following markers.
    The task instruction provides the context for understanding the
    subsequences:

=== ANCHOR START: steps {start}-{end} ===
    <the operation skill subsequence covered by this anchor>
=== ANCHOR END: steps {start}-{end} ===

`{start}-{end}` is the index range of interaction steps covered by
    the anchor, which suggests the span of the subgoal.

For the operation skill subsequence within each anchor span,
    generate an abstract signature of the subgoal together with the
    corresponding description.

========== 1. Annotation Rules ==========

1. Subgoal Naming
   - Format: English words in underscore style `Verb_Noun_Modifier`
    -- each word capitalized and joined by underscores, as a
    high-level abstraction of the operational essence.
   - Concrete data instances (IDs, usernames, e-mail addresses, song
    titles, amounts) or technical implementation details are strictly
    forbidden.

2. Description
   - Use ONE sentence following the pattern: "<verb-object> +
    <object> + [condition], (optional: to <goal>)". Keep a moderate
    granularity: neither too vague nor pinned to specific names /
    dates / numbers.

========== 2. Output Format ==========

Return strictly a JSON array. No extra explanation or comments. The
    array length must equal the number of subgoal anchors in the
    input. Format:

[
  {
    "idx": "<start_idx>-<end_idx>",
    "subgoal_name": "Subgoal_Name_Modifier",
    "description": "the specific intent of this subgoal"
  }
]
```

Subgoal Inducer $\Phi_{sg\text{-ind}}$ (Prompt D.4). Applied in subgoal-layer construction (Section 3.2.2), it induces the abstract signature and description of the subgoal node from each subgoal cluster.

Prompt D.4 Subgoal Inducer $\Phi_{sg\text{-ind}}$

```
You are an expert in subgoal skill induction. The input consists of
    subgoal skill clusters obtained by semantic clustering. For each
    cluster, generate an abstract signature of the subgoal skill
    together with the corresponding description.

========== 1. Induction Rules ==========
```

```
1. Subgoal Skill Naming
   - Format: English words in underscore style 'Verb_Noun_Modifier'
    -- each word capitalized and joined by underscores, as a
    high-level abstraction of the operational essence.
   - Concrete data instances (IDs, usernames, e-mail addresses, song
    titles, amounts) or technical implementation details are strictly
    forbidden.

2. Description
   - Use ONE sentence following the pattern: "<verb-object> +
    <object> + [condition], (optional: to <goal>)". Keep a moderate
    granularity: neither too vague nor pinned to specific names /
    dates / numbers.

========== 2. Output Format ==========

Return strictly a JSON array. No extra explanation or comments. The
    array length must equal the number of subgoal skill clusters in
    the input. Format:

[
  {
    "idx": <idx>,
    "subgoal_skill_name": "Subgoal_Skill_Name_Modifier",
    "description": "the specific intent of this subgoal skill"
  }
]
```

Subgoal Selector Ψ_{sel} (Prompt D.5). Applied in online skill composition (Section 3.3.1), it selects the subgoals necessary for completing the current task among the candidate subgoals recalled from the located task domains.

Prompt D.5 Subgoal Selector Ψ_{sel}

```
You are the Planning Controller of an agent. Given the current user
    task instruction, your task is to select, from the candidate
    subgoals recalled from the memory bank, the subgoals that are
    necessary and tightly coupled for completing the user task.

========== 1. Selection Rules ==========

1. This is NOT semantic similarity retrieval. A subgoal deserves to
    be selected if and only if the task truly requires the
    corresponding subgoal to be completed.
   - Good selection: task "clean up my Spotify library" -> subgoal
    "Filter_And_Remove_Music_Items" (this subgoal is the core of the
    task).
   - Bad selection: task "how many playlists do I have" -> subgoal
    "Filter_And_Remove_Music_Items" (semantically related, but the
    task does not need to delete anything at all).

2. Prefer completeness over omission: the recalled subgoals are the
    primary material for writing the subsequent plan; omitting a
    necessary subgoal leads to an incomplete plan and faulty
    execution. But do NOT select everything blindly -- irrelevant
    subgoals inject noise into the plan and mislead the execution.

3. The operation skills bound under each candidate subgoal can help
    you judge whether the subgoal is necessary, but you do not need
    to care about their implementation details.
```

```
4. Output ONLY the selected subgoals: for each selected subgoal,
    state its name and give a brief reason for why this subgoal is
    necessary. Subgoals that are not selected are simply omitted from
    the output.

========== 2. Output Format ==========

Return strictly a JSON array containing ONLY the selected subgoals.
    No extra explanation or comments. The array length must equal the
    number of selected subgoals. Format:

[
  {
    "subgoal_name": "Subgoal_Name_Modifier",
    "reason": "the specific reason why this subgoal is necessary"
  }
]
```

Subgoal Orchestrator Ψ_{orch} (Prompt D.6). Applied in online skill composition (Section 3.3.2), it integrates the subgoals selected by the subgoal selector into an ordered and fully covering high-level execution plan, and labels each plan step with its reuse type (Inherit / Adapt / Create).

Prompt D.6 Subgoal Orchestrator Ψ_{orch}

```
You are the Planning Controller of an agent. Given the current user
    task instruction and the reference subgoals selected by the
    subgoal selector, your task is to integrate them into an ordered,
    fully covering, and well-grounded execution plan.

========== 1. Planning Rules ==========

1. Reference subgoals are raw MATERIAL: they narrow the search space
    and provide grounded starting points, rather than confining the
    plan itself. Do NOT merely rearrange them verbatim into a plan.

2. The plan you write must truly fit the CURRENT task: on the basis
    of the reference subgoals, generate an ordered and fully covering
    execution plan according to the task instruction.

3. For each plan step, generate an abstract signature and a
    description: the signature is a verb-object phrase in the format
    Verb_Noun_Modifier -- each word capitalized and joined by
    underscores; the description is a one-sentence docstring of the
    subgoal achieved by this step.

4. Each plan step either directly inherits a reference subgoal
    (Inherit), or reuses one with adaptation (Adapt); if no usable
    reference subgoal exists, you may also create a new plan step
    yourself (Create). Annotate each plan step with its subgoal reuse
    type (Inherit / Adapt / Create), and give a brief reason for
    choosing this source type.

5. For plan steps of type Inherit / Adapt, state in the output the
    name of the reference subgoal they inherit or adapt; for plan
    steps of type Create, leave the reference subgoal name empty.

========== 2. Output Format ==========

Return strictly a JSON array. No extra explanation or comments. The
    array length must equal the number of plan steps. Format:
```

```
[
  {
    "plan_step_name": "Plan_Step_Name_Modifier",
    "description": "the specific intent of this plan step",
    "subgoal_reuse_type": "Inherit / Adapt / Create",
    "reference_subgoal_name": "name of the reference subgoal if
    Inherit/Adapt, else empty",
    "reason": "brief reason for choosing this reuse type"
  }
]
```