FROM TRAJECTORIES TO HIERARCHICAL SKILL GRAPHS WITH SEMANTIC CONTRACTS

Anonymous authors
Paper under double-blind review

ABSTRACT

External skills enable language-model agents to reuse experience without parameter updates. However, the quality of document-based skills depends on the ability of an LLM to preserve procedural knowledge during summarization, while static skill loading fails to adapt guidance to changing interaction phases. Graph-based representations support structured retrieval, but historical dependencies alone do not establish current action applicability. To address these limitations, we introduce **S**emantic **C**ontract **G**raph (SCG), a framework that integrates hierarchical skill learning with semantic contract evaluation. Offline, SCG recursively composes recurring action sequences into a hierarchical graph of reusable skills. Online, it retrieves a phase-relevant subgraph and evaluates whether action preconditions hold in the current context. Together, the retrieved subgraph and semantic verdicts condition a frozen agent through prompt augmentation. Experiments on ALFWorld, ScienceWorld, SpreadsheetBench, and WebShop show that SCG improves task performance while reducing interaction rounds. Ablation studies reveal the contributions of graph structure and semantic guidance, highlighting the value of integrating skill graphs with live semantic contract checks.

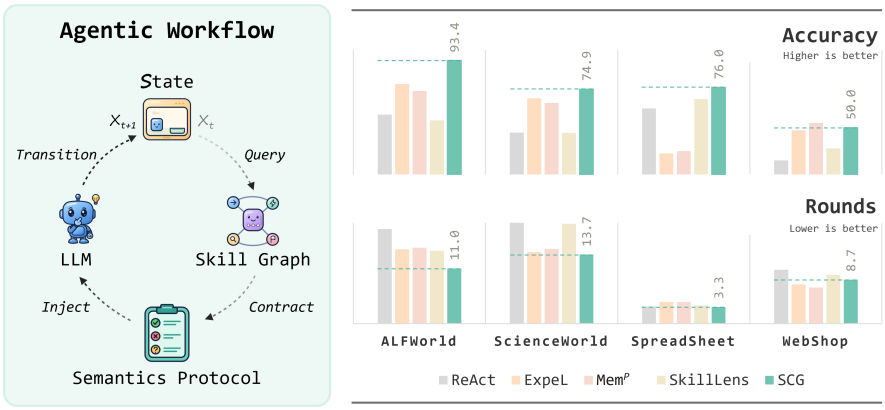


Figure 1: SCG workflow and benchmark performance. **Left:** Skill graph retrieval and semantic contract evaluation jointly guide the LLM agent through prompt augmentation. **Right:** Accuracy and mean interaction rounds across four benchmarks, with SCG highlighted in green.

1 INTRODUCTION

Language-model agents are increasingly applied to domain-specific tasks that require multi-step tool use and multi-turn, long-horizon planning, including software engineering (Yang et al., 2024; Jimenez et al., 2024; Qian et al., 2024) web interaction (Google DeepMind, 2025; OpenAI, 2025a; Yao et al., 2022), automated scientific inquiry (OpenAI, 2025b; Lu et al., 2024; Chan et al., 2025), embodied intelligence (Bolton et al., 2025; Wang et al., 2023), and beyond. In each setting the agent must inspect the current state, select an operation, invoke a domain-specific interface, and recover when the environment diverges from the plan. A practical way to obtain that behavior, without

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.

updating the base LLM's parameters, is to condition inference on **skills** – reusable procedures that guide the agent's actions (Zhang et al., 2025).

Existing skill libraries typically represent procedural knowledge as Markdown documents (e.g., `SKILL.md`), with updates requiring revisions to textual instructions. Recent work advocates lifecycle-managed, evolving skill artifacts (Li, 2026), including MUSE-Autoskill (Lin et al., 2026), SkillOS (Ouyang et al., 2026), SkillX (Wang et al., 2026a), and SkillLens (Huang et al., 2026). However, these approaches primarily rely on LLMs to summarize interaction histories into reusable procedures. This **summarization-centric formulation** makes skill quality dependent on the model's ability to reconcile heterogeneous trajectories, while redundant steps can obscure critical decision points as interaction histories grow. A second limitation concerns **phase-agnostic skill loading**. Injecting a complete skill document at the initial turn fails to adapt to changes in the agent's procedural needs throughout execution. As interaction histories accumulate, relevant instructions may receive less attention or be discarded through context truncation, aggravating "context rot" (Hong et al., 2025). SkillsInjector (Li et al., 2026b) highlights the limitations of static injection under fixed context budgets, while SkillTTA (Wang et al., 2026b) reports gains from adaptive test-time synthesis on benchmarks including SpreadsheetBench (Ma et al., 2024) and BigCodeBench (Zhuo et al., 2025). These findings motivate selective retrieval of procedural guidance throughout execution.

Graph-based skill representations offer a structured alternative by organizing actions and macros as nodes connected through procedural relations (Hao et al., 2026; Li et al., 2026a; Xiong et al., 2026; Bai et al., 2026; Liu et al., 2026). Their modular structure supports incremental updates and retrieval of relevant skill subgraphs. However, **historical procedural structure alone does not establish current action applicability**. For example, a shopping skill graph may encode product selection before purchase without establishing whether the selected product currently satisfies the user's constraints. The agent therefore needs live semantic checks to identify violated or unverified preconditions, particularly before irreversible actions. This motivates combining graph-based procedural knowledge with state-dependent semantic guidance at each interaction step.

To this end, we propose **S**emantic **C**ontract **G**raph (SCG), a unified framework that integrates two complementary sources of guidance. The graph functions as an action consultant: at each decision round, it retrieves a bounded local subgraph and suggests procedurally appropriate next actions, rather than injecting a full skill document at task initiation. The semantic contract, in turn, injects current-state information and verifies those suggestions. Specifically, SCG predefines a set of semantic contract predicates that describe the causal effects and dependencies between actions and states. For example, the action `Purchase` in a web-shopping scenario may require the state `product:verified` to be `True` (the product matches the user's requirements), and the action itself updates the state so that `cart:empty` is `True`.

Overall, SCG consists of two stages: **offline skill-graph and contract learning**, and **online dynamic skill formulation**. Offline, offline trajectories are compressed into a hierarchical graph, while semantic contracts are summarized by an LLM into a fixed template. Online, dynamic skills are created by first grounding the current state, checking all predicates, retrieving the relevant subgraph to obtain action suggestions, and verifying all live facts. This design aligns local suggestions with the current state to reduce steps spent on impermissible operations. We evaluate SCG against a diverse set of baselines: a no-skill worker, a static markdown-based skill baseline (SkillLens (Huang et al., 2026)), memory-based approaches (Mem^p Fang et al. (2026) and ExpeL Zhao et al. (2024)), and our own ablations with graph-only and semantic-contracts-only. Across multiple benchmarks, SCG improves exact success over almost all these baselines while consuming fewer tokens. In summary, the main contributions of this work are threefold:

1. We propose a **hierarchical skill graph** framework that converts trajectories into a directed graph with skill nodes, atomic action nodes, and typed edges, while using **semantic contracts** to inject episode-specific state information and check action compatibility.
2. We design a pipeline that constructs the graph offline, retrieves a task-relevant subgraph as an action consultant at each step, and executes the suggested actions under semantic-contract verification.
3. We evaluate SCG against a no-skill worker, SkillLens, Mem^p, ExpeL, and our ablations (graph-only and semantic-contracts-only) across ALFWorld, ScienceWorld, WebShop, and SpreadsheetBench. SCG outperforms all baselines in success rate on every benchmark, while reducing token usage.

2 Related Work

Self-Evolving Agents and Skill Lifecycles. Self-evolving agents generate and refine skills from interaction feedback. AutoSkill (Yang et al., 2026) derives reusable skills from interaction traces, while FlowEvo (Ren et al., 2026) compiles successful traces into skill records. Ctx2Skill (Si et al., 2026) uses multi-agent self-play for skill discovery, and MemSkill (Zhang et al., 2026) refines memory-operation skills using challenging cases. A recent survey (Li, 2026) frames skill libraries as evolving artifact stores. MUSE-Autoskill (Lin et al., 2026), SkillOS (Ouyang et al., 2026), SkillX (Wang et al., 2026a), and SkillLens (Huang et al., 2026) implement lifecycles encompassing skill generation, verification, and storage.

Static versus Dynamic Skill Injection. Static injection loads complete skill documents at initialization, providing phase-agnostic guidance susceptible to context rot (Hong et al., 2025). Recent approaches explore on-demand delivery and test-time adaptation. LatentSkill (Yu et al., 2026) encodes textual skills as LoRA adapters. SkillTTA (Wang et al., 2026b) synthesizes task-specific skills from retrieved trajectories, while TARSE (Wang et al., 2026c) retrieves skills and experience for lightweight test-time adaptation. These directions motivate adapting procedural guidance throughout execution.

Textual versus Graph-Based Skill Organization. Graph-based representations explicitly encode relationships among skills. SkillGraph (Li et al., 2026a) models prerequisites and co-occurrence, jointly evolving graph topology and policy through RL. SkillDAG (Bai et al., 2026) supports inference-time retrieval over typed directed graphs. HiSkill (Hao et al., 2026) organizes trajectories through decomposition, temporal, compatibility, and recovery relations. Graph-of-Skills (Liu et al., 2026) retrieves bounded, dependency-aware bundles using hybrid seeding and Personalized PageRank. GraSP (Xia et al., 2026) constructs typed DAGs with precondition-effect edges and locality-bounded repair. These structures support procedural retrieval, but graph structure alone does not verify whether action preconditions hold in the current context.

3 Method

Semantic Contract Graph (SCG) integrates offline graph learning with online execution. Offline, SCG induces a hierarchical skill graph by recursively composing recurring action sequences into reusable macros. Online, it retrieves a relevant subgraph and evaluates action preconditions against the current context. The retrieved procedures and semantic guidance jointly condition the agent's action selection without parameter updates.

3.1 Problem Formulation

Consider an interactive environment $\mathcal{E}$ with observation space $\mathcal{O}$ and action space $\mathcal{A}$. Given a task instruction q, an agent with a policy π_θ with fixed parameters θ interacts with $\mathcal{E}$ over a finite-horizon episode. At step t, the agent receives an observation $o_t \in \mathcal{O}$. The preceding interaction history is denoted by $h_t = (o_0, a_0, \ldots, o_{t-1}, a_{t-1})$. Therefore, conditioned on the current live episode context $x_t = (q, o_t, \mathcal{A}_t, h_t)$, the agent samples $a_t \sim \pi_\theta(\cdot \mid x_t) \in \mathcal{A}_t$ where $\mathcal{A}_t \subseteq \mathcal{A}$ denotes the set of currently admissible actions from action space $\mathcal{A}$ at step t.

For the offline learning stage, a collection of interaction trajectories is defined by $\mathcal{D} = \{\tau_i\}_{i=1}^{N}$, where each trajectory $\tau_i = (q_i, o_{i,0}, a_{i,0}, \ldots, o_{i,T_i-1}, a_{i,T_i-1}, o_{i,T_i}, R_i)$ records executed actions, observations and final reward R_i. Leveraging this trajectory corpus, our method induces reusable procedural knowledge from $\mathcal{D}$ to steer π_θ online without retraining, employing two complementary modules: a hierarchical skill graph G for offline procedural priors, and a semantic contract S for live state-based applicability. We represent this combined information as a **dynamic phase-aware skill** k_t, which is constructed by combining a phase-relevant subgraph $g_t \subseteq G$ with state-based semantic guidance s_t: $k_t = g_t \oplus s_t$, $\widetilde{x}_t = x_t \oplus k_t$, where $\oplus$ denotes textual concatenation in the prompt space. The augmented context $\widetilde{x}_t$ is then provided to the policy π_θ for action selection.

3.2 Offline Hierarchical Skill Graph Learning

3.2.1 Graph Definition

A hierarchical skill graph is learned from historical trajectories to capture reusable procedural library. As illustrated in Figure 2, the graph is defined as $G = (V, E)$, $V = V_0 \cup V_+$, and $E = E_c \cup E_f$,

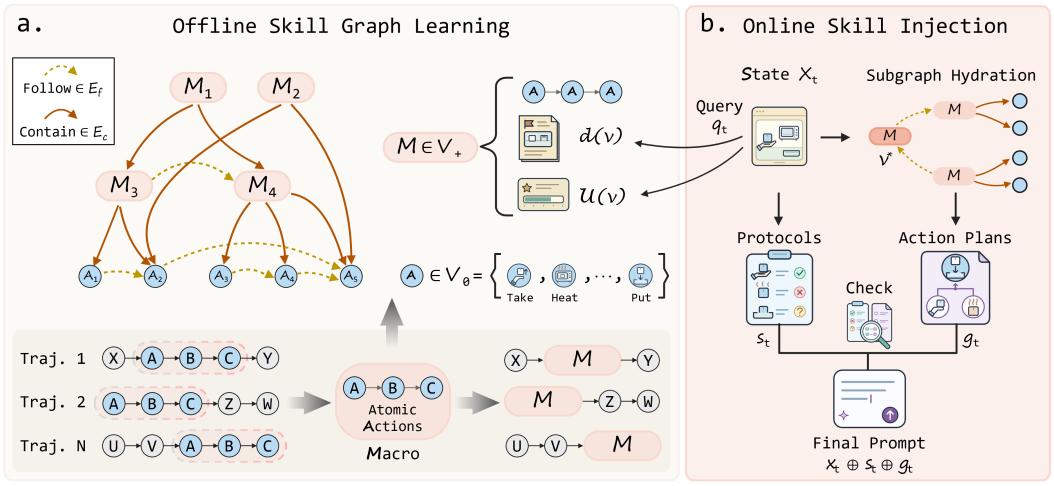


Figure 2: Overview of SCG. **(a) Offline:** Recurring action sequences are recursively compressed into macros, forming a hierarchical skill graph. Each macro v is associated with a retrieval document $d(v)$ and empirical utility $\mathcal{U}(v)$. **(b) Online:** A phase-relevant subgraph is retrieved, while semantic contracts assess action applicability using grounded predicates.

where V_0 contains *atomic actions*, and V_+ contains *macros* composed of recurring sequences of atomic actions or lower-level macros. *Contains edges* E_c connect each macro to its ordered child nodes, defining the hierarchical composition of the graph. *Follows edges* E_f encode observed transition patterns, where an edge $(u, v) \in E_f$ indicates that v typically follows u during execution.

3.2.2 Graph Learning

The graph-learning phase is to convert a set of raw trajectories into a compact hierarchical skill graph which serves as a library of reusable procedures. Instead of memorizing full trajectories, the algorithm extracts recurring action blocks and turns them into higher-level macros. These macros are recursively composed to form deeper abstractions.The overall algorithm for creating this graph is summarized in Algorithm 1.

Step 1: Trajectory Tokenization & Atomic Action Extraction. For each trajectory, we parse the raw agent interaction history by a rule-based tokenizer ϕ into canonical atomic actions, e.g., `Search for blue dress` $\mapsto$ `Search`. The set of all distinct atomic actions forms the level-0 node set V_0. Each trajectory is then tokenized into a sequence over these nodes: $\phi(\tau_{i,0}) = (a_1, a_2, \ldots, a_n)_i$, where each $a_j \in V_0$ is an atomic action, identical atomic actions correspond to the same node in V_0. After tokenization, each trajectory is reduced to a compact sequence of atomic actions (e.g., `[Search, Select-product, Select-Option, Purchase]`).

Step 2: Graph Structure Induction. We iteratively construct the graph by replacing recurring node sequences with macro nodes, as illustrated in Figure 2a. At round r, the node set is $V_r = V_0 \cup V_{+,r}$, where $V_{+,r}$ contains previously learned macros. Candidates $m_c = (v_1, \ldots, v_\ell)$ are contiguous sequences over V_r extracted from compressed trajectories, with $2 \le \ell \le L_{\max}$.

For each candidate, $n_r(m_c)$ counts non-overlapping occurrences across trajectories, and $n_{\text{traj},r}(m_c)$ as the number of distinct trajectories in which m_c appears. In addition, let $\rho_r(m_c) \in [0, 1]$ denote the execution-success rate, defined as the fraction of occurrences of m_c in which every constituent step executes validly. Using these statistics, candidate macros are filtered according to four criteria: **(1) Cross-trajectory support**: occurrence in at least $n_{\min}$ distinct trajectories; **(2) Positive compression gain**: replacing each occurrence removes $|m_c| - 1$ tokens, yielding $\mathcal{G}_r(m_c) = (|m_c| - 1)n_r(m_c) - 1$ under a unit macro-introduction cost; **(3) Execution reliability**: an execution-success rate of at least $\rho_{\min}$; and **(4) Non-degeneracy**: at least two distinct nodes, excluding homogeneous repetitions such as `[Search, Search, Search]`. Accordingly, these

criteria define the eligible candidate set:

$$\mathcal{M}_r = \left\{ m_c \;\middle|\; \begin{array}{c} n_{\mathrm{traj},r}(m_c) \geq n_{\min}, \\ \mathcal{G}_r(m_c) > 0, \\ \rho_r(m_c) \geq \rho_{\min}, \\ \mathrm{Unique}(m_c) > 1 \end{array} \right\}. \tag{1}$$

Among eligible candidates m_c, we select the best macro node using lexicographic criterion

$$m_r^\star = \arg\max_{m_c \in \mathcal{M}_r}{}_{\mathrm{lex}} \left(n_{\mathrm{traj},r}(m_c), n_r(m_c), -|m_c|, \rho_r(m_c) \right). \tag{2}$$

This lexicographic criterion prioritizes cross-trajectory support, followed by occurrence frequency. Among candidate macros with equal support and frequency, the preference for shorter patterns facilitates recursive composition. After selecting $m_r^\star = (v_1, \ldots, v_{\ell_r})$, we promote it to a new macro $V_{+,r+1} = V_{+,r} \cup \{m_r^\star\}$, and record *ordered containment edges* from this macro to its constituent nodes in their original order and augment the total set E_c. We then replace every non-overlapping occurrence of the selected sequence in each trajectory with the new macro. The updated trajectories serve as input to the next round, allowing newly introduced macros to participate in higher-level compositions. This loop terminates when the macro budget is exhausted or no viable candidate macros remain, as described in Algorithm 1.

When the loop terminates, the *follows edges* E_f are constructed by scanning adjacent node pairs in the final compressed trajectories. Let $n_f(v_1, v_2)$ denote the number of times v_2 immediately follows v_1 across these trajectories. Each observed transition defines an edge $(v_1, v_2) \in E_f$ with weight

$$w_f(v_1, v_2) = \frac{n_f(v_1, v_2)}{\sum_{u \in V} n_f(v_1, u)}, \qquad (v_1, v_2) \in E_f, \tag{3}$$

where V contains both atomic-action and macro nodes. Unlike the acyclic containment graph, the follows graph may contain cycles and self-loops.

Step 3: Empirical Annotations. To quantify the reliability and expected utility for the extracted skills, We evaluate macros along three complementary axes: **admissibility**, whether its first action is admissible at the start of an occurrence; **clean execution**, whether all underlying nodes execute successfully; and **episodic success**, whether the containing trajectory ultimately succeeds. Because occurrence counts vary substantially across macros, empirical success rates alone can overestimate the reliability of sparsely observed skills. We therefore use Wilson lower bounds Wilson (1927) to account for sampling uncertainty, emphasizing consistent performance over isolated successes.

Formally, for each node v with n_v training occurrences, we tally the count of first action admissible k_v^{app}, all steps executed cleanly k_v^{exec} and hosting trajectory succeeded k_v^{ep} in $\mathcal{D}$, with each using Wilson lower bounds $L(k, n)$:

$$L(k, n) = \max\left(0, \frac{\hat{p} + \frac{z^2}{2n} - z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}} \right), \quad \hat{p} = \frac{k}{n},\ z = 1. \tag{4}$$

We aggregate the three metrics into a unified utility $\mathcal{U}$ for each macro node as:

$$\mathcal{U}(v) = w_1 L(k_v^{\mathrm{exec}}, n_v) + w_2 L(k_v^{\mathrm{app}}, n_v) + w_3 L(k_v^{\mathrm{ep}}, n_v), \tag{5}$$

where w_1, w_2, and w_3 control the relative contributions of clean execution, admissibility, and episodic success, respectively.

Step 4: Retrieval documents. For each macro v, the offline phase constructs a retrieval document $d(v)$ that records the contexts in which the macro was observed during training, providing contextual evidence for subsequent invocation decisions. The document concatenates three components: (i) the macro name, (ii) a brief textual description of its ordered child composition (e.g., Search $\rightarrow$ Select $\rightarrow$ Purchase), and (iii) one context block for each training occurrence. Each context block follows a fixed schema comprising the task description, observation, available action types, previous action, and previous outcome, all extracted from the trajectory at the start of the corresponding occurrence. This design ensures that the document captures cues that inform when each macro should be invoked.

Algorithm 1 SKILL GRAPH LEARNING

Require: Trajectories $\mathcal{D}$; support threshold $n_{\min}$; max pattern length $L_{\max}$; macro budget M; execution success rate floor $\rho_{\min}$.
Ensure: Graph $G = (V_0, V_+, E_c, E_f)$ with annotations.

```
Extract canonical atomic actions a_j, construct V_0, and tokenize each τ_i ∈ D as φ(τ_i) over V_0.
Initialize V_{+,0} ← ∅, E_{c,0} ← ∅, and r = 0.
while |V_{+,r}| < M do
    Enumerate all contiguous candidate macros m_c consists of nodes in V_r = V_0 ∪ V_{+,r}.
    Form M_r by retaining candidate macros satisfying the four criteria in Equation (1).
    if M_r = ∅ then
        break
    end if
    Select m*_r ∈ M_r using the lexicographic criterion in Equation (2).
    Update V_{+,r+1} ← V_{+,r} ∪ {m*_r}, and add contains edges to E_{c,r}, E_{c,r+1} = E_{c,r} ∪ E_c.
    Update the tokenized trajectories by the new macro: φ(τ_{i,r+1}) ← Replace(φ(τ_{i,r}), m*_r).
    r ← r + 1.
end while
Update the Graph Structure:V_+ ← V_{+,r}. E_c ← E_{c,r}.
Construct follows edges E_f using Equation (3).
Compute utility U(v) using Equation (5)
Construct retrieval document d(v) for each macro node.
return G = (V_0, V_+, E_c, E_f) with annotations.
```

3.3 ONLINE GRAPH RETRIEVAL WITH SEMANTIC CONTRACTS

The offline-learned graph G serves as a structured prior over procedural knowledge. However, to guide the online agent effectively without retraining, we must bridge the gap between this static library and the dynamic context of a partially observable environment. Our online phase addresses this through a dual mechanism: (i) a retrieval system that selects and hydrates relevant macro skills from G, and (ii) a semantic contract layer that validates the feasibility and correctness of the candidate action against the current state. We use t to index online interaction steps, distinct from the offline graph-learning rounds indexed by r.

3.3.1 SEMANTIC PROTOCOL

The semantic protocol bridges the gap between the static skill library and the dynamic environment state through **state grounding** and **action contracts**. It translates raw observations into structured state representations and validates candidate actions against explicit contractual rules. The protocol defines a finite predicate vocabulary Π and a contract for each action schema. At interaction step t, a sensor f evaluates each predicate $p \in \Pi$ from the context $x_t = (q, o_t, \mathcal{A}_t, h_t)$: $f_t(p) = f(x_t, p) \in \{\texttt{True}, \texttt{False}, \texttt{Unknown}\}$. Here, `True` and `False` indicate verified truth and falsity, while `Unknown` indicates insufficient evidence. This sensor can be implemented as a rule-based heuristic or as an LLM call, enabling flexible adaptation to different environments. Using the same predicate vocabulary, each action $a \in \mathcal{A}$ inherits a schema-level contract specifying its execution requirements and expected effects. The set $\texttt{Req}(a) \subseteq \Pi$ defines the preconditions required before execution, while the Boolean flag $\texttt{Term}(a)$ identifies terminal actions, such as purchase or payment. The sets $\texttt{Add}(a), \texttt{Del}(a) \subseteq \Pi$ describe the expected effects of successful execution. For example, placing a held object on a table is expected to establish `object:on-table` and invalidate `object:held`.

3.3.2 ONLINE RETRIEVAL AND EXECUTION

The online phase comprises seed selection, subgraph hydration, and semantic guidance assembly, as illustrated in Figure 2b.

Seed Selection At interaction step t, the retriever constructs a query q_t using the offline retrieval document schema (Section 3.2.2). Each macro node v is scored by $\text{Score}_t(v) =$

$\lambda_{\text{sim}}\,\text{Sim}(q_t, d(v)) + \lambda_{\text{stage}}\mathbf{1}_{\text{stage}}(v) + \lambda_{\text{util}}\mathcal{U}(v) + \lambda_{\text{trans}}\,\text{Trans}(v_{\text{prev}}, v) - \lambda_{\text{rep}}\mathbf{1}_{\text{used}}(v)$, where Sim measures BM25 relevance, $\mathbf{1}_{\text{stage}}$ indicates availability of the macro's staged next action schema, $\mathcal{U}$ is the empirical utility, Trans measures transition support from the previous seed v_{prev}, and $\mathbf{1}_{\text{used}}$ penalizes recent seed reuse. A constant stage baseline is used when no candidate passes the stage filter. The highest-scoring eligible macro is selected as the seed $v^\star$.

Subgraph Hydration and Prompt Construction The retriever expands $v^\star$ into a local subgraph g_t of at most $K_{\text{sub}} = 16$ nodes. It traverses ordered containment children breadth-first, then fills the remaining budget with follows successors and parent macros. For the candidate action a, the semantic protocol checks its required predicates against the grounded evaluations:

$$\text{Adm}_t(a) = \begin{cases} \mathbf{1}[\forall p \in \texttt{Req}(a),\ f_t(p) \neq \texttt{False}], & \texttt{Term}(a) = 0, \\ \mathbf{1}[\forall p \in \texttt{Req}(a),\ f_t(p) = \texttt{True}], & \texttt{Term}(a) = 1, \end{cases} \tag{6}$$

where $\mathbf{1}[\cdot]$ denotes the indicator function and $\text{Adm}_t(a) = 1$ indicates that a is admissible. Non-terminal actions require no precondition to be assessed as false, whereas terminal actions require all preconditions to be assessed as true. Therefore, the protocol constructs semantic guidance as $s_t = \text{Render}(\text{Adm}_t(a), \{(p, f_t(p)) : p \in \texttt{Req}(a)\}, \texttt{Add}(a), \texttt{Del}(a))$, where Render converts the verdict, supporting predicate evaluations, and expected effects into textual guidance. The subgraph and its staged next action schema are combined with this guidance as $k_t = g_t \oplus s_t$ to augment the policy's context without overriding its sampled action.

4 Experiments

4.1 Experimental Setup

Environments & Baselines. We evaluate on four interactive benchmarks: ALFWorld (Shridhar et al., 2021) (household manipulation, 6 task types), WebShop (Yao et al., 2023a) (e-commerce navigation, 12k instructions), SpreadsheetBench (Ma et al., 2024) (912 spreadsheet manipulation tasks with 2,729 test cases), and ScienceWorld (Wang et al., 2022) (scientific reasoning across 10 topics, 30 task types). We compare against three groups: a skill-free baseline, ReAct (Yao et al., 2023b), a skill-based baseline, SkillLens (Huang et al., 2026), and memory-based baselines, ExpeL (Zhao et al., 2024) and Mem^p (Fang et al., 2026). For offline graph learning, we use expert demonstrations for ALFWorld and WebShop, and self-generated trajectories collected without predefined skills for SpreadsheetBench and ScienceWorld. Trajectory sources and sampling details are provided in Appendix A.1.Unless otherwise specified, we use DeepSeek-V4-Flash and set the retrieval budget to $K_{\text{sub}} = 16$ nodes for all experiments. The semantic protocols in our implementation define three sources of information: atomic actions, predicates for state grounding, and the semantic contract between actions and the current state. Sensors $f_t(p)$ can be implemented either as rule-based Python code wired into the SCG framework or as an additional LLM call for predicate checking. We provide an ablation study on sensor implementation in the following section. More protocol construction details, examples and the template prompt are listed in Appendix A.2.

4.2 Main Results

Table 1 compares SCG with baselines using DeepSeek-V4-Flash and Qwen-3.6-Flash. We report performance and mean rounds, including seen and unseen splits where available. SCG achieves the highest performance in 10 of 12 settings and the fewest interaction rounds in 7 settings among all compared methods. The token analysis in Figure 3 shows that SCG achieves these performance gains with token consumption generally comparable to or much lower than that of the baselines.

4.3 Ablation Study

Component Ablations. We compare full SCG with Graph-Only and Semantic-Only, which remove semantic guidance and graph retrieval, respectively. The results are listed in Table 2, with metrics averaged across the seen and unseen splits for ALFWorld and ScienceWorld. As shown in the table, the proposed approach achieves the highest performance in the majority of cases. Appendix B.3 provides execution traces illustrating the roles of graph retrieval and semantic guidance.

Table 1: Performance of main methods across benchmarks and splits. A: accuracy (%). R: mean rounds. Best accuracy per column is bolded.

Method	ALFWorld				ScienceWorld				SpreadSheet		WebShop	
	Seen		Unseen		Seen		Unseen		Seen		Seen	
	A↑	R↓	A↑	R↓	A↑	R↓	A↑	R↓	A↑	R↓	A↑	R↓
Deepseek V4 - Flash												
ReAct	60.7	18.6	56.0	19.0	49.7	19.1	44.1	21.4	62.5	3.4	28.9	10.7
ExpeL	78.6	14.6	77.6	14.9	70.6	13.3	67.3	15.1	33.5	4.3	48.4	7.8
Mem^p	75.0	14.6	72.4	15.5	65.5	13.3	66.4	16.3	35.0	4.3	**53.1**	**7.2**
SkillLens	52.9	14.7	56.7	14.2	49.1	18.6	44.2	21.0	68.5	3.6	36.7	9.7
SCG	**96.4**	**10.2**	**90.3**	**11.7**	**76.8**	**13.0**	**73.0**	**14.4**	**76.0**	**3.3**	50.0	8.7
Qwen 3.6 - Flash												
ReAct	50.0	21.4	45.5	21.4	44.3	17.6	42.2	18.9	27.0	3.3	32.8	8.0
ExpeL	77.1	15.2	67.2	16.4	63.4	13.9	**61.6**	16.8	34.0	2.5	45.3	**6.8**
Mem^p	73.6	14.7	64.2	17.0	60.3	**13.4**	58.3	**16.2**	30.0	**2.4**	46.9	8.1
SkillLens	49.3	20.9	53.0	20.2	45.9	18.0	43.1	20.0	29.5	4.4	34.4	8.3
SCG	**90.7**	**13.0**	**87.3**	**12.9**	**66.0**	13.7	59.2	16.9	**43.5**	2.5	**49.8**	8.7

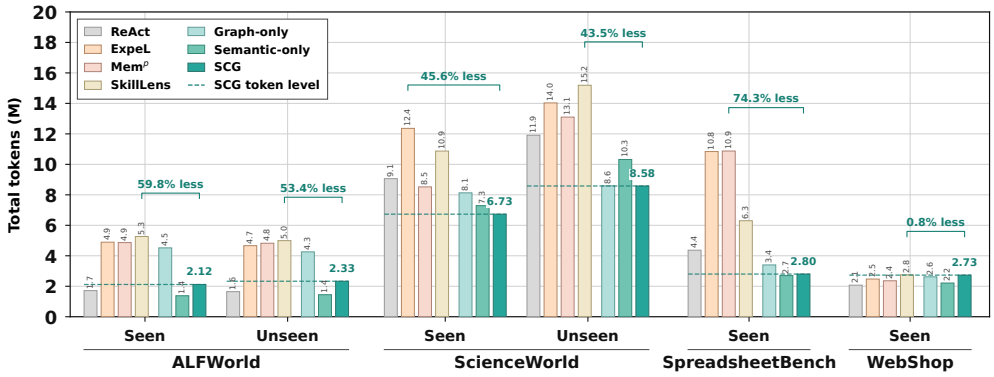


Figure 3: Total token consumption of all baselines across different benchmarks. Percentages show reductions relative to the baseline with the highest token consumption in each setting.

Rule-based sensor vs. LLM call. We note that implementing the sensors as described in the State Grounding section (Section 3.3) requires extra LLM calls to write the sensor code for state grounding. For example, grounding the state `object:on-table` requires code that inspects the trajectory to determine whether the action `pick up` has been successfully executed. An alternative is to directly use an online LLM call to check the states within the agentic loop. This is the most straightforward approach, but it incurs additional LLM calls and token consumption. In Table 2, we compare the rule-based sensor against the LLM-based sensor. The results indicate that their performance is comparable, but the LLM-based sensor consumes more tokens.

Effect of Hierarchical Skills. We compare two representations of the same learned macros: **Flat** expands macros into atomic-action sequences, whereas **Hier** preserves recursive macro composition. We vary the retrieved-node budget $K \in \{2, 4, 8, 16\}$, including the selected seed node. As shown in Figure 4, Hier achieves higher mean scores over three seeds at every K, with gains of 1.33–4.33 points on SpreadsheetBench. These gains suggest that macros preserve procedural boundaries and represent multi-step routines within individual nodes, providing more informative guidance under a limited retrieval budget. Additional results on WebShop are reported in Appendix B.1.

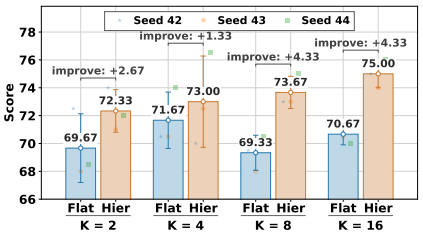
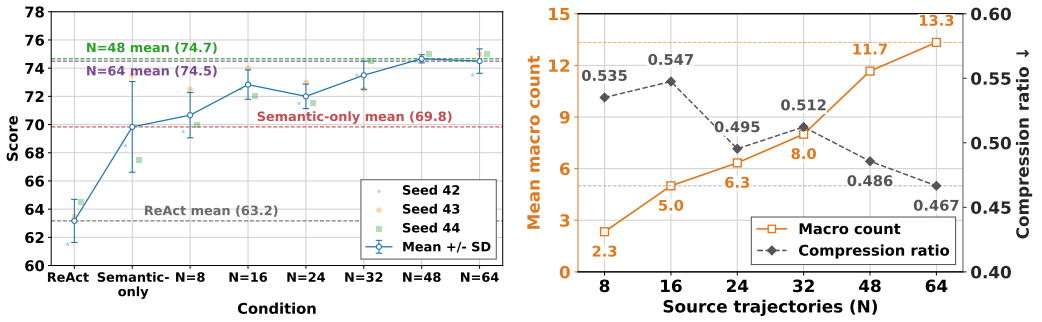


Figure 4: Effect of Hierarchical SKills.

Table 2: Ablation study of SCG with rule-based and LLM-based sensors, denoted *rule* and *LLM*, respectively, compared with Graph-Only, Semantic-Only, SkillLens, and ReAct across benchmarks. A: accuracy (%). R: mean rounds. Tok: tokens (M). Best accuracy per benchmark is bolded.

Method	ALFWorld			ScienceWorld			SpreadsheetBench			WebShop		
	A↑	R↓	Tok↓	A↑	R↓	Tok↓	A↑	R↓	Tok↓	A↑	R↓	Tok↓
Baselines												
ReAct	58.4	18.8	1.7	46.9	20.3	10.5	62.5	3.4	4.4	28.9	10.7	2.1
SkillLens	54.8	14.5	5.1	46.7	19.8	13.0	68.5	3.6	6.3	36.7	9.7	2.8
Component Ablations												
Graph-Only	57.4	20.1	4.4	60.0	17.1	8.4	73.5	3.9	3.4	32.0	10.7	2.6
Semantic-Only	93.0	10.8	1.4	62.5	15.6	8.8	70.5	3.3	2.7	43.5	8.4	2.2
SCG (LLM)	90.5	12.1	4.0	72.7	14.0	11.4	74.5	3.4	3.3	44.5	8.4	3.7
SCG (rule)	**93.4**	11.0	2.2	**74.9**	13.7	7.7	**76.0**	3.3	2.8	**50.0**	8.7	2.7



(a) Different N compared to baseline.

(b) Macro nodes and compression ratio.

Figure 5: Dynamic expanding graph for SpreadSheetBench: (a) task performance compared with ReAct and Semantic-Only; (b) learned macro count and compression ratio.

4.4 Dynamic Expanding Graph

We vary the number of source trajectories N from 8 to 64 across three different random seeds in Figure 5, using settings that differ from the main experiments and ablation study. At $N = 64$, mean scores reach 74.5 on SpreadsheetBench, exceeding Semantic-only and ReAct by 4.7 and 11.3, respectively, as shown in Figure 5a. Meanwhile, mean macro counts grow from 2.3 to 13.3, while compression ratios (compressed trajectory length divided by original trajectory length) decrease overall from 0.535 to 0.467 in Figure 5b. These trends suggest that additional trajectories support the discovery of more reusable macros, expanding the guidance available beyond semantic rules alone. Appendix Section B.2 reports the corresponding results on WebShop.

5 Conclusion

We presented Semantic Contract Graph (SCG), a framework that learns hierarchical macros from trajectories and combines phase-aware subgraph retrieval with live semantic contract checks. It provides procedural, state-dependent prompt guidance without updating agent parameters. Across four interactive benchmarks, SCG improves task performance and usually interaction efficiency. Ablations show graph structure and semantic guidance help; hierarchical macros outperform flattened representations at equal node budgets, and more source trajectories expand the macro library and improve compression. Effective skill reuse thus benefits from both compositional organization of prior experience and explicit checks of action applicability in context.

AI USE STATEMENT

We used generative AI tools to support research ideation and execution, including methodological clarification and interpretation of experimental results. AI tools also assisted with literature retrieval and discovery, reference formatting, drafting selected sections, and improving manuscript clarity and fluency.

The authors reviewed and revised AI-assisted drafts and experiments, verified references against original sources, and checked reported numerical comparisons against experimental results.

REFERENCES

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, 2026. URL https://arxiv.org/abs/2606.03056.

Adrian Bolton, Alexander Lerchner, Alexandra Cordell, Alexandre Moufarek, Andrew Bolt, Andrew Lampinen, Anna Mitenkova, Arne Olav Hallingstad, Bojan Vujatovic, Bonnie Li, et al. Sima 2: A generalist embodied agent for virtual worlds. *arXiv preprint arXiv:2512.04797*, 2025.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, volume 2025, pp. 50466–50494, 2025.

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. doi: 10.18653/v1/2026.findings-acl.866. URL https://aclanthology.org/2026.findings-acl.866/.

Google DeepMind. Introducing the Gemini 2.5 Computer Use model. https://blog.google/innovation-and-ai/models-and-research/google-deepmind/gemini-computer-use-model/, October 2025. Accessed: 2026-06-25.

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

Kelly Hong, Anton Troynikov, and Jeff Huber. Context rot: How increasing input tokens impacts llm performance. Technical report, Chroma, July 2025. URL https://research.trychroma.com/context-rot.

Zisu Huang, Jingwen Xu, Yifan Yang, Ziyang Gong, Qihao Yang, Muzhao Tian, Xiaohua Wang, Changze Lv, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Xue Yang, Dongdong Chen, Xiaoqing Zheng, and Chong Luo. From raw experience to skill consumption: A systematic study of model-generated agent skills, 2026. URL https://arxiv.org/abs/2605.23899.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

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

Yanchao Li, Wanhao Liu, Ben Gao, Jiaqing Xie, Zhehong Ai, Na Zou, Yuqiang Li, and Tianfan Fu. Skillsinjector: Dynamic skill context construction for llm agents, 2026b. URL https://arxiv.org/abs/2605.29794.

Yubo Li. Dynamic agent skills: A lifecycle survey and taxonomy of evolving skill libraries, 2026. URL https://arxiv.org/abs/2607.10113.

Huawei Lin, Peng Li, Jie Song, Fuxin Jiang, and Tieying Zhang. Muse-autoskill: Self-evolving agents via skill creation, memory, management, and evaluation, 2026. URL https://arxiv.org/abs/2605.27366.

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, 2026. URL https://arxiv.org/abs/2604.05333.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. Spreadsheetbench: Towards challenging real world spreadsheet manipulation, 2024. URL https://arxiv.org/abs/2406.14991.

OpenAI. Computer-Using Agent. https://openai.com/index/computer-using-agent/, 2025a. Accessed: 2026-06-25.

OpenAI. Introducing deep research. https://openai.com/index/introducing-deep-research/, February 2025b. Accessed: 2026-06-25.

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, et al. Skillos: Learning skill curation for self-evolving agents. *arXiv preprint arXiv:2605.06614*, 2026.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186, 2024.

Zeyu Ren, Ling Yue, Ran Li, Yishu Wang, Shengxiang Xu, Hanmo Liu, Shaowu Pan, and Shimin Di. Flowevo: Self-evolving agents through the co-evolution of workflows and executable skills, 2026. URL https://arxiv.org/abs/2607.21596.

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

Shuzheng Si, Haozhe Zhao, Yu Lei, Qingyi Wang, Dingwei Chen, Zhitong Wang, Zhenhailong Wang, Kangyang Luo, Zheng Wang, Gang Chen, Fanchao Qi, Minjia Zhang, and Maosong Sun. From context to skills: Can language models learn from context skillfully?, 2026. URL https://arxiv.org/abs/2604.27660.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. Skillx: Automatically constructing skill knowledge bases for agents, 2026a. URL https://arxiv.org/abs/2604.04804.

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.

Jingxing Wang, Chenyu Zhou, Zhihui Fu, Jun Wang, Weiwen Liu, Weinan Zhang, and Jianghao Lin. Skills on the fly: Test-time adaptive skill synthesis for llm agents, 2026b. URL https://arxiv.org/abs/2605.16986.

Junda Wang, Zonghai Tao, Hansi Zeng, Zhichao Yang, Hamed Zamani, and Hong Yu. Tarse: Test-time adaptation via retrieval of skills and experience for reasoning agents, 2026c. URL https://arxiv.org/abs/2603.01241.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. Scienceworld: Is your agent smarter than a 5th grader?, 2022. URL https://arxiv.org/abs/2203.07540.

Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927. ISSN 01621459, 1537274X. URL http://www.jstor.org/stable/2276774.

Tianle Xia, Lingxiang Hu, Yiding Sun, Ming Xu, Lan Xu, Siying Wang, Wei Xu, and Jie Jiang. Grasp: Graph-structured skill compositions for llm agents, 2026. URL https://arxiv.org/abs/2604.17870.

Yuan Xiong, Ziqi Miao, Qian Chen, Lijun Li, Yequan Wang, Shizhu He, Jun Zhao, and Kang Liu. Skillpyramid: A hierarchical skill consolidation framework for self-evolving agents, 2026. URL https://arxiv.org/abs/2606.03692.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. Autoskill: Experience-driven lifelong learning via skill self-evolution, 2026. URL https://arxiv.org/abs/2603.01145.

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, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents, 2023a. URL https://arxiv.org/abs/2207.01206.

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

Aofan Yu, Chenyu Zhou, Tianyi Xu, Zihan Guo, Rong Shan, Zhihui Fu, Jun Wang, Weiwen Liu, Yong Yu, Weinan Zhang, and Jianghao Lin. Latentskill: From in-context textual skills to in-weight latent skills for llm agents, 2026. URL https://arxiv.org/abs/2606.06087.

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with agent skills. Anthropic Engineering Blog, October 2025. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills. Accessed: 2026-06-25.

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

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, Binyuan Hui, Niklas Muennighoff, David Lo, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions, 2025. URL https://arxiv.org/abs/2406.15877.

A Implementation Details

A.1 Offline Trajectory Sources

We construct the offline corpus $\mathcal{D}$ from two sources: externally generated expert demonstrations and self-generated trajectories collected without predefined skills.

Expert Demonstrations. For ALFWorld and WebShop, we use trajectories generated by external models and agent harnesses, primarily from public supervised fine-tuning (SFT) datasets. For ALFWorld, we sample 32 trajectories from a pool of 206 stratified demonstrations in `Decix/ReBel-ALFWorld-SFT-Trajectories`, selecting eight trajectories from each of four categories. For WebShop, we use 5,029 expert trajectories with a terminal reward of 1.0 from `lclan/webshop_expert_trajectories`.

Self-Generated Trajectories. For SpreadsheetBench and ScienceWorld, we collect offline trajectories using a simple agentic loop within SCG, without predefined skills. This setting evaluates whether SCG can construct a reusable skill library from its own interaction experience.

A.2 Semantic Protocols

Protocol Construction. Each environment is associated with a semantic protocol S that specifies an inventory of atomic action schemas, a predicate vocabulary Π, and schema-level action contracts. The action inventory provides the canonical action types used to tokenize trajectories during offline graph learning. A designated LLM call generates the predicates, associated sensor implementations, and action contracts, based on the trajectories. Each contract specifies preconditions $\texttt{Req}(a)$, a terminal-action flag $\texttt{Term}(a)$, and expected effects $\texttt{Add}(a)$ and $\texttt{Del}(a)$.

Rule-Based Grounding. Rule-based sensors f are implemented as Python functions integrated into SCG. During online interaction, these functions evaluate predicates from the current context $x_t = (q, o_t, \mathcal{A}_t, h_t)$, including observations, available actions, and interaction history. Each predicate receives a value in $\{\texttt{True}, \texttt{False}, \texttt{Unknown}\}$, where `Unknown` denotes insufficient evidence. Although sensor code is generated using an LLM, its subsequent execution does not require additional LLM calls for predicate evaluation.

LLM-Based Grounding. Alternatively, an LLM directly evaluates predicates from the current context and returns the same three-valued assignments. This implementation replaces executable sensor rules with additional model calls during online interaction, introducing inference and token overhead. We compare the two implementations in Table 2.

Environment-Specific Examples. The four protocols instantiate the same grounding and admissibility mechanism with environment-specific predicates. In ALFWorld, terminal placement requires `target:held` and `operation:complete`. For tasks requiring heating, cooling, or cleaning, placement receives a positive verdict only after the required operation is verified; otherwise, `operation:complete` is satisfied when no operation is required. In ScienceWorld, the designated terminal action `focus` requires verified `goal:progress` and is expected to establish `target:focused`. In SpreadsheetBench, the protocol distinguishes workbook inspection, editing, output creation, and verification. The terminal action `submit` requires both `output:exists` and `output:verified`, separating successful file creation from task-specific verification. In WebShop, `purchase` requires a selected product, verified task constraints, configured options, and sufficient matching evidence. Any false or unknown requirement prevents a positive terminal-action verdict. Across environments, these verdicts provide textual guidance without overriding the policy's sampled actions.

The following listings present the action inventories, predicate definitions, and contracts for ALFWorld, ScienceWorld, SpreadsheetBench, and WebShop.

Alfworld Semantic Protocols

```
action_names:
  - go-to
  - take
  - move
  - put
  - open
  - close
  - clean
  - cool
  - heat
  - slice
  - examine
  - look
  - inventory
  - use
  - toggle

predicates:
  - name: target:identified
    meaning: Task target object type was parsed from the instruction
    required_for_terminal: true
  - name: target:held
    meaning: A matching target instance is currently held / operable
    required_for_terminal: true
  - name: operation:complete
    meaning: Required heat/cool/clean (if any) has been applied to the target
    required_for_terminal: true
  - name: placement:complete
    meaning: Required target placement has been recorded
    required_for_terminal: false

contracts:
  - actions: [take]
    adds: [target:held]
  - actions: [clean, cool, heat, slice]
    requires: [target:held]
    adds: [operation:complete]
  - actions: [put, move]
    requires: [target:held, operation:complete]
    adds: [placement:complete]
    deletes: [target:held]
    verifies: [required operation completed before placement]
    terminal: true
```

ScienceWorld Semantic Protocols

```
action_names:
  - look-around
  - go-to
  - wait
  - wait1
  - open
  - close
  - drop
  - activate
  - deactivate
  - connect
  - disconnect
  - use
  - use-on
  - examine
  - look-at
  - read
  - move
  - pick-up
  - pour
  - mix
  - dunk
  - teleport
  - focus
  - inventory
  - objects
  - help
  - menu-select

predicates:
  - name: location:known
    meaning: The agent has observed or teleported to a room
    required_for_terminal: false
  - name: object:held
    meaning: An object is currently in inventory
    required_for_terminal: false
  - name: target:focused
    meaning: The required task-target object has been focused
    required_for_terminal: true
  - name: device:activated
    meaning: A device has been activated
    required_for_terminal: false
  - name: circuit:connected
    meaning: Electrical components are connected
    required_for_terminal: false
  - name: goal:progress
    meaning: Task-specific progress toward mixture or state change
    required_for_terminal: false
  - name: container:open
    meaning: At least one observed container or door is open
    required_for_terminal: false

contracts:
  - actions: [look-around, look-at, examine, go-to, teleport]
    adds: [location:known]
  - actions: [pick-up]
    requires: [location:known]
    adds: [object:held]
  - actions: [move]
    requires: [object:held]
    deletes: [object:held]
  - actions: [drop]
    deletes: [object:held]
  - actions: [activate]
    requires: [location:known]
    adds: [device:activated]
  - actions: [deactivate]
    deletes: [device:activated]
  - actions: [connect]
    requires: [location:known]
    adds: [circuit:connected]
  - actions: [disconnect]
    deletes: [circuit:connected]
  - actions: [pour, mix, dunk, use, use-on]
    adds: [goal:progress]
  - actions: [open]
    adds: [container:open]
  - actions: [close]
    deletes: [container:open]
  - actions: [focus]
    requires: [goal:progress]
    adds: [target:focused]
    verifies: [goal:progress]
    terminal: true
```

SpreadSheetBench Semantic Protocols

```
action_names:
  - view-xlsx
  - inspect-workbook
  - inspect-file
  - change-dir
  - make-dir
  - write-file
  - python-exec
  - convert-workbook
  - copy-file
  - move-file
  - transform-rows
  - edit-values
  - edit-formulas
  - format-cells
  - create-visualization
  - verify-output
  - submit

terminal_actions: [submit]

predicates:
  - name: workbook:inspected
    meaning: Workbook/file inspection has been recorded
    required_for_terminal: false
  - name: edit:implemented
    meaning: Workbook-writing / transform code has been recorded
    required_for_terminal: false
  - name: output:exists
    meaning: A saved output workbook was observed
    required_for_terminal: true
  - name: output:verified
    meaning: Task-specific verification passed
    required_for_terminal: true

contracts:
  - actions:
      - view-xlsx
      - inspect-workbook
      - inspect-file
      - change-dir
      - make-dir
    adds: [workbook:inspected]
  - actions:
      - write-file
      - python-exec
      - convert-workbook
      - copy-file
      - move-file
      - transform-rows
      - edit-values
      - edit-formulas
      - format-cells
      - create-visualization
    requires: [workbook:inspected]
    adds: [edit:implemented, output:exists]
  - actions: [verify-output]
    requires: [edit:implemented, output:exists]
    adds: [output:verified]
    verifies: [saved workbook task invariant]
  - actions: [submit]
    requires: [output:exists, output:verified]
    verifies: [saved workbook task invariant]
    terminal: true
```

WebShop Semantic Protocols

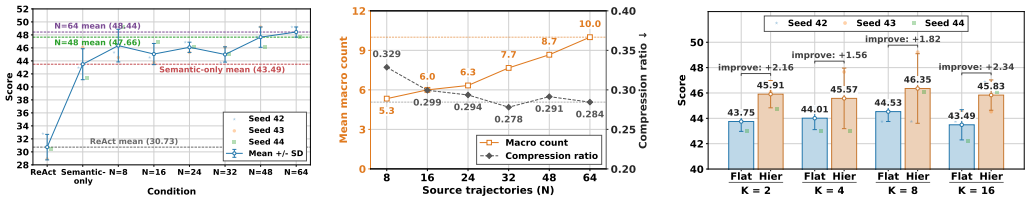
```
action_names:
  - formulate-search
  - select-product
  - select-option
  - inspect-product
  - paginate-results
  - return-to-search
  - return-to-results
  - filter-results
  - purchase
  - click-control
  - search
  - click

terminal_actions: [purchase]

predicates:
  - name: product:selected
    meaning: A product ASIN is currently selected
    required_for_terminal: true
  - name: constraints:verified
    meaning: Parsed task constraints match product-page evidence
    required_for_terminal: true
  - name: options:configured
    meaning: Required visible product options have been selected (or none exist)
    required_for_terminal: true
  - name: match:sufficient
    meaning: Product title/evidence covers enough instruction content words
    required_for_terminal: true


contracts:
  - actions: [select-product]
    adds: [product:selected]
  - actions: [select-option, inspect-product]
    requires: [product:selected]
    adds: [constraints:verified, options:configured, match:sufficient]
  - actions: [purchase]
    requires:
      - product:selected
      - constraints:verified
      - options:configured
      - match:sufficient
    verifies:
      - brand tokens when named
      - required color/size/length/flavor option clicks
      - title/body overlap match:sufficient
      - price upper bound
    terminal: true
```

B Full Ablation and Case Study Results



(a) Different N for Webshop.

(b) Macros and compression ratio for Webshop.

(c) Effect of Hierarchical Skills for Webshop

Figure 6: Dynamic expanding graph and hierarchical skills for Webshop.

B.1 Effect of Hierarchical Skills for WebShop

Following the setup in Section 4.3, we vary the retrieved-node budget $K \in \{2, 4, 8, 16\}$, including the selected seed node. As shown in Figure 6c, Hier achieves higher mean scores across three seeds

at every K, with gains of 1.56–2.34 points. These results suggest that hierarchical composition preserves procedural boundaries and represents multi-step routines within individual nodes, providing more informative guidance under a fixed retrieval budget.

B.2 Dynamic Expanding Graph for WebShop

Following the setup in Section 4.4, we vary the number of source trajectories N from 8 to 64 on WebShop across three random seeds, using settings that differ from the main experiments. As show in Section B, at $N = 64$, the mean score reaches 48.44, exceeding Semantic-Only and ReAct by 4.95 and 17.71 points, respectively. Meanwhile, the mean macro count increases from 5.3 to 10.0, while the compression ratio decreases overall from 0.329 to 0.284, as illustrated in Figure 6b.

B.3 Case Study

Table 3 compares execution traces from SkillLens, Graph-Only, Semantic-Only, and full SCG against reference action sequences. The two cases illustrate distinct contributions of procedural retrieval and state-dependent semantic guidance.

ALFWorld: State Guidance Resolves Repetitive Search. The task requires placing a clean butterknife on a countertop. Successful execution involves locating and taking the knife, cleaning it at a sink, and placing it on the target surface. SkillLens and Graph-Only repeatedly search drawers until timeout, without acquiring or cleaning the target. Graph-Only reuses a drawer-search procedure but lacks explicit guidance about whether the target is held or the required operation is complete. Semantic guidance makes these unmet requirements explicit. Semantic-Only follows the reference procedure, while full SCG succeeds after an initial drawer-search detour. In the full method, the grounded predicates `target:held` and `operation:complete` provide guidance toward acquisition and cleaning before terminal placement. This case primarily illustrates the contribution of semantic guidance, since Semantic-Only also completes the task.

SpreadsheetBench: Graph Retrieval Supports Progression from Inspection to Editing. The task requires deleting blank rows above the first `Code` entry in column A while preserving blank rows below it. SkillLens does not produce a grader-accepted workbook modification. Semantic-Only continues inspecting the workbook and eventually saves it without performing the required deletion. Graph-Only and full SCG instead retrieve an `inspect-file→write-file` procedure that supports the transition from inspection to editing. Both identify the target entry, delete the relevant blank rows, and save the modified workbook. Here, the retrieved procedure supplies the next operational step that semantic guidance alone does not provide.

Together, these cases illustrate the complementary roles of graph retrieval in organizing procedural progression and semantic contracts in identifying unmet state requirements.

Table 3: Task sequences and failure modes across design approaches.

Task 1: ALFWorld: *"put a clean butterknife on the countertop"*

Approach	Action sequence	Failure mode	Analysis
Gold sequence	go to countertop 1 → go to countertop 2 → take butterknife 1 from countertop 2 → go to sinkbasin 1 → clean butterknife 1 with sinkbasin 1 → go to countertop 1 → move butterknife 1 to countertop 1	None	Minimal pick–clean–place path; the knife is on countertop 2.
SkillLens	inventory → go to drawer 1 → open drawer 1 → go to drawer 2 → open drawer 2 → … go to drawer 7 (timeout); never take, clean, or place	Receptacle-search loop until timeout	Flat skills do not track whether the butterknife is held or cleaned, so the agent never leaves drawer search.

Continued on next page

Table 3: Task sequences and failure modes across design approaches. (Continued)

Task 1: ALFWorld: *"put a clean butterknife on the countertop"*

Approach	Action sequence	Failure mode	Analysis
graph-only	go to drawer 1 → open drawer 1 → go to drawer 2 → open drawer 2 → … go to drawer 7 (timeout)	Retrieved search recipe replayed as an empty-drawer loop	Seed `go-to-then-open-then-take` has no `held`/`clean` facts, so `next=go-to` keeps opening empty drawers.
semantics-only	go to countertop 1 → go to countertop 2 → take butterknife 1 from countertop 2 → go to sinkbasin 1 → clean butterknife 1 with sinkbasin 1 → go to countertop 1 → move butterknife 1 to countertop 1	None	Contract facts (`held`, `operation:complete`) enforce find → take → clean → place and match gold; the graph is not required on this instance.
graph-contract	go to drawer 1 → open drawer 1 → go to countertop 1 → go to countertop 2 → take butterknife 1 from countertop 2 → go to sinkbasin 1 → clean butterknife 1 with sinkbasin 1 → go to countertop 1 → move butterknife 1 to countertop 1	None	Starts on the graph search seed. The contract (`held=false`) leaves drawers at go to countertop 1; `operation:complete=false` requires clean before place.
Where the design is useful	**Contract:** at go to countertop 1 and at clean. Graph-only never leaves the drawers; semantics-only matches gold, so this win is not retrieval.		

Task 2: Spreadsheet `374-31`: *"delete blank rows above "`Code`" in column A"*

Approach	Action sequence	Failure mode	Analysis
Gold sequence	inspect column A → find the first cell with `A="Code"` → delete blank rows above that cell (keep blank rows below it) → save the workbook	None	Inspect followed by restricted row deletion, not a rewrite of the VBA text.
SkillLens	3 turns (Study I copy; no turn log)	No grader-accepted workbook mutation	Flat SkillLens does not produce the deleted-row sheet the grader checks.
graph-only	inspect the workbook → find `A="Code"` → delete blank rows above that cell → save the workbook	None	Seed `inspect-file-then-write-file` advances from inspect to delete-and-save on the second turn.
semantics-only	inspect the workbook → inspect rows 1–5 again → save the workbook with no deletions	Inspect stall; output is the unchanged input file	Contract stays in `phase: inspect` with no write seed, so the agent never executes the gold deletion.
graph-contract	inspect the workbook → find `A="Code"` → delete blank rows above that cell → save the workbook	None	Same inspect-then-write subgraph as graph-only; the graph is what executes the gold deletion.

Continued on next page

Task 2: Spreadsheet `374-31`: *"delete blank rows above "`Code`" in column A"*

Approach	Action sequence	Failure mode	Analysis
Where the design is useful	**Graph:** at the write turn. Without a next-action seed, the contract never leaves inspect.		