SKILLPYRAMID: A Hierarchical Skill Consolidation Framework for Self-Evolving Agents

Anonymous ACL submission

Abstract

Recent AI agents can flexibly invoke skills to solve complex tasks, but their long-term improvement is fundamentally constrained by a lack of systematic skill construction, accumulation, and transfer. In particular, without a unified framework for skill consolidation, agents tend to redundantly construct similar capabilities across different tasks, are unable to effectively transform experience into reusable assets, and struggle to generalize task-specific skills to novel scenarios. To address this limitation, we propose SKILLPYRAMID, a skill consolidation framework that reuses existing skill experience for broader task generalization. Operating on a hierarchical skill topology, SKILLPYRAMID further introduces a self-evolution mechanism that enables agents to compose, validate, and incorporate new skills during task execution. Experiments on ALFWorld, WebShop, and ScienceWorld across four backbone models show that SKILLPYRAMID substantially increases the average reward by 38.0% and reduces execution steps by 27.7%. Overall, our method transforms a skill collection from a static resource pool into a dynamic evolution system.

1 Introduction

Recent advances in large language models (LLMs) have shifted agent systems from passive response generation to autonomous task completion (Yao et al., 2022b; Shinn et al., 2023; Wang et al., 2023a). Within this shift, *skills*—callable units of capability—have become a key substrate for agents' problem solving, substantially improving performance on complex, long-horizon, and open-ended tasks (Wang et al., 2023a; Liang et al., 2026; Wang et al., 2026; Mi et al., 2026). In open-ended environments, however, manually curated skill sets quickly grow stale, and continual skill evolution becomes essential for sustained competence. Yet a central question remains open: how can skills evolve from isolated, task-specific solutions into reusable, compositional knowledge?

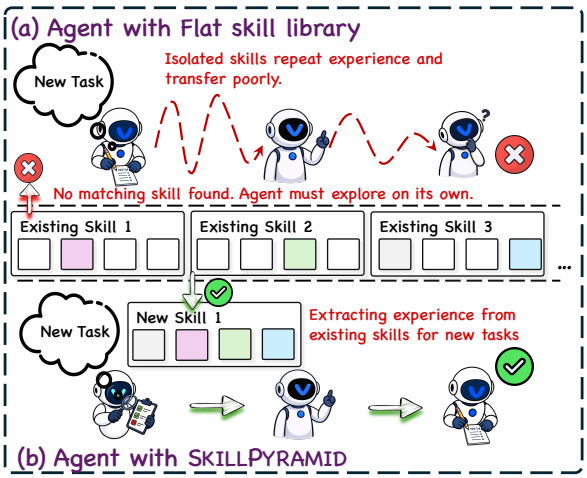

Figure 1: Flat skill library vs. SKILLPYRAMID. (a) With isolated skills, no match is found for an unseen task and the agent must explore from scratch, often failing. (b) SKILLPYRAMID composes new skills by recombining components of existing ones.

Existing approaches tackle parts of this question from different angles. Tool-augmented agents extend LLMs with external APIs to fill capability gaps (Schick et al., 2023; Qin et al., 2024), and modular frameworks coordinate specialized components through task decomposition (Karpas et al., 2022; Shen et al., 2023; Lu et al., 2023). A related line learns reusable skills via program synthesis or experiential accumulation (Liang et al., 2023; Zhao et al., 2024; Fang et al., 2025). More recently, dedicated skill-management frameworks have emerged (Liang et al., 2026; Wang et al., 2026; Mi et al., 2026), demonstrating the value of creating, organizing, retrieving, and refining reusable skill units. Yet two limitations remain: (i) existing skills are typically reused as stored units, leaving their internal transferable experience underexploited; and (ii) new skills are often retrieved, refined, or appended rather than composed from prior

skill experience.

This motivates a different perspective: a growing, structured skill library in which the experience captured by existing skills generalizes effectively to tasks beyond their original scope. As illustrated in Figure 1, related skills often share underlying capabilities, and treating these as atomic components lets agents recombine them to address unseen tasks, turning a flat collection into a structure that supports both reuse and generalization.

Building on this view, we propose SKILLPYRAMID, an evolving skill framework for LLM agents. SKILLPYRAMID organizes skills into a multi-level hierarchy in which lower levels hold fine-grained, reusable atomic capabilities and upper levels abstract recurring problem-solving patterns. A Relation Analyzer and a Relation Builder identify reusable relations and construct this hierarchy through *Downward Atomic Extraction*, which captures minimal reusable capabilities, and *Upward Abstract Induction*, which summarizes common problem-solving patterns. A task-driven self-evolution mechanism then folds each validated new skill back into the hierarchy by linking it to related skills, yielding a dynamic capability system that expands over time, reduces redundancy, and accumulates reusable knowledge.

Our contributions are as follows:

- We propose SKILLPYRAMID, a framework that organizes agent skills into a growing hierarchy and improves transferability through explicit reuse and dependency modeling.
- We design a pyramid organization mechanism with a Relation Analyzer and a Relation Builder, constructing atomic skills via downward extraction and abstract skills via upward induction.
- We introduce a task-driven self-evolution mechanism that retrieves, composes, and generates skills while incrementally updating SKILLPYRAMID. Experiments on ALFWorld, WebShop, and ScienceWorld show 38.0% higher average reward and 27.7% fewer interaction steps across multiple backbone LLMs.

2 Method

Our method improves the generalization of a fixed skill collection by leveraging reusable patterns from existing high-quality skills. To this end, we construct a pyramidal, reusable, and continuously evolving skill system, where successful skills serve as reusable building blocks for related tasks rather than isolated modules. As illustrated in Figure 2, The system is managed by a multi-agent framework that analyzes the existing skill library (§2.2.1), constructs reuse relations among skills (§2.2.2), synthesizes new skills grounded in the resulting hierarchy (§2.3), and incrementally evolves as new skills are added (§2.4).

2.1 Problem Formalization

We consider an interactive agent setting where an agent receives a natural language task instruction x, interacts with an environment through actions, and receives task feedback after execution. At each step t, the agent observes the environment state o_t, takes an action a_t, and obtains a final reward $r \in [0, 1]$ or a task completion signal after the episode ends. A complete task trajectory is denoted as $\tau = (x, o_0, a_0, o_1, \ldots, a_T, o_{T+1}, r)$.

A skill s is defined as a structured natural language program $s = (n, d, c)$, where n is the skill name, d is a short description, and c is the skill content specifying its applicable conditions, execution procedure, and inputs/outputs.

Unlike methods that directly treat trajectories as training data for parameter updates, SKILLPYRAMID converts task experience and existing skill descriptions into explicit, reusable skills. Given a skill collection $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$, our goal is to construct and maintain a hierarchical SKILLPYRAMID $\mathcal{P} = \{L_0, L_1, \ldots, L_K\}$ (illustrated in Figure 2), where skills become increasingly abstract and compositional from the bottom layer L_0 to the top layer L_K. For a stream of tasks, SKILLPYRAMID aims to improve task performance by reusing, composing, and evolving skills.

2.2 SKILLPYRAMID Construction

To construct the pyramid, we employ two agents: a Relation Analyzer and a Relation Builder. The Analyzer identifies related skill groups, specifies the relation type for each, and dispatches the relation-construction tasks to the Builder. The Builder then extracts atomic skills, induces abstract skills, and rewrites the original skills. The Analyzer–Builder pipeline iterates until no new skill is produced or a maximum of T rounds is reached, with each round's outputs feeding the next.

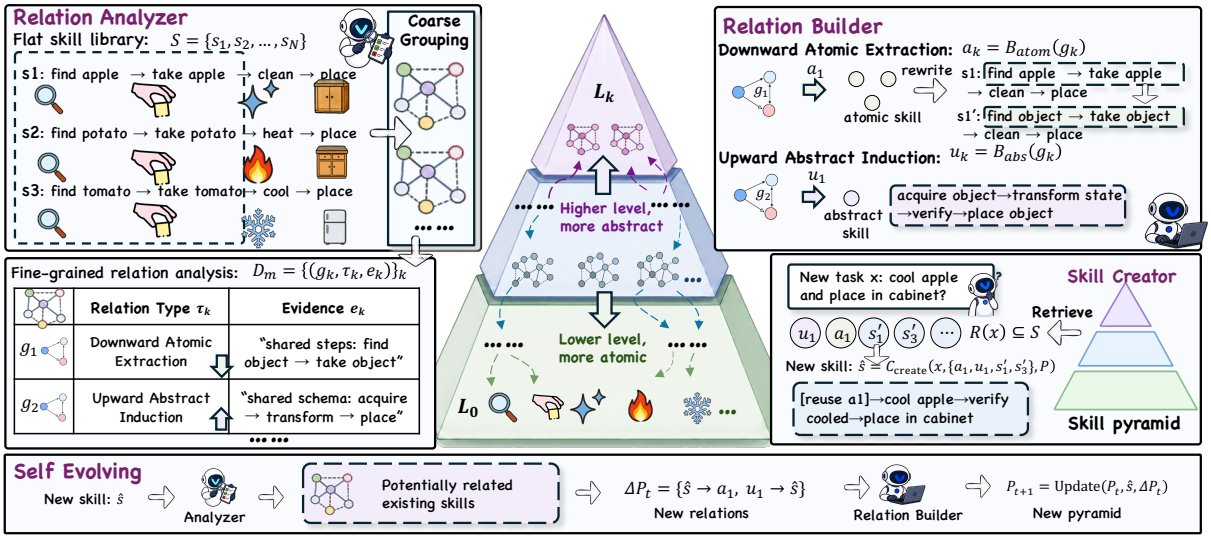

Figure 2: Overview of the proposed SKILLPYRAMID framework. The system analyzes an existing skill library, constructs downward atomic and upward abstract reuse relations, organizes skills into a hierarchical pyramid, creates skills from the pyramid, and incrementally evolves the pyramid as new skills are added.

2.2.1 Relation Analyzer

Given the current skill collection $\mathcal{S} = \{s_i\}_{i=1}^N$, directly analyzing all skills jointly may introduce substantial noise and lead to an excessive analysis workload. We therefore adopt a progressive analysis strategy that proceeds from coarse to fine granularity.

Coarse Grouping. Let n_i and d_i denote the name and description of skill s_i, respectively. The Relation Analyzer first performs a coarse grouping by inspecting only these two fields:

$$\mathcal{G}^0 = \mathcal{A}_c\big(\{(n_i, d_i)\}_{i=1}^N\big)\,, \tag{1}$$

where $\mathcal{A}_c$ denotes the *coarse-grained analyzer* and $\mathcal{G}^0 = \{G_1^0, \ldots, G_M^0\}$ is the resulting set of coarse skill groups. Each group $G_m^0 \subseteq \mathcal{S}$ collects skills whose names and descriptions suggest potential relatedness in terms of functionality, task scope, or reuse potential. We use G_m^0 to refer interchangeably to this subset of skills and to its index set.

Fine-Grained Group Analysis. The Relation Analyzer then analyzes each coarse group G_m^0 in parallel, with tools to inspect the full structured content of every skill in the group. For each G_m^0, a *fine-grained analyzer* $\mathcal{A}_f$ runs for at most t rounds and produces a set of relation-construction assignments:

$$\mathcal{D}_m = \mathcal{A}_f(G_m^0) = \{(g_k, \tau_k, e_k)\}_k, \tag{2}$$

where each $g_k \subseteq G_m^0$ is a fine-grained skill subset, τ_k specifies its relation type , and e_k provides natural-language rationale and construction guidance. A single coarse group may yield multiple such subsets. The assignments from all coarse groups, $\mathcal{D} = \bigcup_m \mathcal{D}_m$, are then dispatched to Relation Builder for actual reuse-relation construction.

2.2.2 Relation Builder

To capture reuse at both the execution level and the task-structure level, the Relation Builder operates in two complementary modes depending on τ_k: *Downward Atomic Extraction*, which surfaces shared low-level capabilities, and *Upward Abstract Induction*, which summarizes shared high-level patterns. It takes the assignments produced by the Relation Analyzer as input and constructs the corresponding reuse relations among existing skills.

Downward Atomic Extraction. An atomic skill captures a minimal reusable capability that appears across multiple skills. Compared with concrete task skills, atomic skills are less tied to a specific task instance and can be reused across different environments or task categories.

For each assignment $(g_k, \tau_k, e_k) \in \mathcal{D}$ with τ_k = *Downward Atomic Extraction*, let $g_k = \{s_1, \ldots, s_m\}$. The Relation Builder extracts the common functional component shared by these skills as an atomic skill:

$$a_k = B_{\text{atom}}(g_k). \tag{3}$$

Following the unified skill representation in §2.1, a_k takes the same form (n_k, d_k, c_k) as any other

skill, where d_k summarizes the common goal of the shared capability. To record explicit reuse relations introduced by the pyramid, we augment each skill with a *dependency field* δ that stores references to other skills it reuses. After a_k is extracted, each original skill $s_i \in g_k$ is rewritten to invoke a_k through an explicit reuse reference:

$$s_i' = \mathrm{Rewrite}(s_i, \rho(a_k)), \quad s_i \in g_k, \tag{4}$$

where the reuse reference $\rho(a_k) = (n_k, q_k, w_k, o_k)$ records the reused skill's name n_k, a unique identifier q_k, the condition w_k under which it should be reused, and the capability o_k it provides. This reference is appended to the dependency field δ_i of s_i' as a structured reuse statement.

If an existing skill is already fully covered by other skills, the Relation Builder directly treats this covered skill as the reusable module and updates the covering skills to reference it. To further reduce redundancy, highly similar extracted atomic skills are merged into a single canonical skill.

The extracted atomic skill a_k is placed one layer below the source skills in the pyramid:

$$\ell(a_k) = \min_{s \in g_k} \ell(s) - 1, \tag{5}$$

where $\ell(\cdot)$ denotes the layer index of a skill.

Upward Abstract Induction. While atomic skills capture reusable low-level capabilities, a group of skills may also share similar high-level functional structures. For each assignment $(g_k, \tau_k, e_k) \in \mathcal{D}$ with $\tau_k =$ *Upward Abstract Induction*, the Relation Builder induces an abstract skill u_k that summarizes the common solving pattern of g_k:

$$u_k = B_{\mathrm{abs}}(g_k). \tag{6}$$

The induced u_k takes the same form (n_k, d_k, c_k) as in §2.1, with its dependency field δ_k recording references to the lower-level skills in g_k that it composes. Unlike atomic skills, which focus on reusable operations, abstract skills focus on generalizable task structures: they describe how lower-level skills should be organized to solve a class of related tasks, rather than specifying only concrete execution steps.

The induced abstract skill u_k is placed one layer above the source skills in the pyramid:

$$\ell(u_k) = \max_{s \in g_k} \ell(s) + 1. \tag{7}$$

In this way, u_k serves as higher-level guidance, while the referenced concrete or atomic skills provide executable support for its realization.

2.3 Skill Creation

While the pyramid built so far organizes existing skills into reusable structures, a new task often does not match any single skill exactly and instead requires composing several. To turn the pyramid from a passive repository into an active problem solver, we introduce a task-driven Skill Creator agent that generates new skills by composing relevant components from the existing hierarchy rather than synthesizing them from scratch. Given a task instruction x, the Skill Creator retrieves relevant skills from the current SKILLPYRAMID, analyzes how they can support the task, and synthesizes a new skill grounded in the existing hierarchy. Specifically, the Skill Creator retrieves a candidate set $\mathcal{R}(x) \subseteq \mathcal{S}$ by jointly considering task semantics, applicable conditions, and the constraints encoded in the pyramid.

Based on $\mathcal{R}(x)$, the Skill Creator generates a new skill in two stages: (i) *framework construction*, where higher-level (abstract) skills are prioritized to outline the task structure and high-level solving logic; (ii) *detail instantiation*, where lower-level (atomic) skills are prioritized to fill in executable operations, inputs, checks, and outputs. If an existing skill fully covers the task, the Skill Creator directly reuses it. Otherwise, it composes a new task-specific skill:

$$\hat{s} = C_{\mathrm{create}}(x, \mathcal{R}(x), \mathcal{P}), \tag{8}$$

where $\mathcal{P}$ denotes the current pyramid, which encodes both the layer structure and the reuse relations among skills. The generated skill follows the same structured format as existing skills.

2.4 Self-Evolution

A static skill library cannot keep pace with the open-ended stream of tasks that an agent encounters during deployment. To make SKILLPYRAMID continually adaptive, we introduce an incremental self-evolution mechanism that absorbs each newly created skill into the existing hierarchy without rebuilding it from scratch.

At iteration t, the system maintains a skill collection $\mathcal{S}_t$ and a pyramid structure $\mathcal{P}_t$, which encodes both the layer assignment and the reuse relations among skills in $\mathcal{S}_t$. Given a newly created skill $\hat{s}$,

the goal of self-evolution is to update $\mathcal{P}_t$ by connecting $\hat{s}$ with relevant prior skills.

Unlike the initial construction stage, the Relation Analyzer does not re-analyze the whole skill library. Instead, it treats $\hat{s}$ as a query skill, retrieves only potentially related existing skills, and dispatches the corresponding relation-construction tasks to the Relation Builder.

The Relation Builder then follows the same procedure as in §2.2.2 to construct new reuse relations involving $\hat{s}$. The resulting updates, denoted $\Delta\mathcal{P}_t$, are incrementally inserted into the pyramid:

$$\mathcal{P}_{t+1} = \mathrm{Update}(\mathcal{P}_t,\ \hat{s},\ \Delta\mathcal{P}_t)\,. \tag{9}$$

This procedure allows SKILLPYRAMID to absorb new skills incrementally while preserving its existing hierarchical organization.

3 Experiment

3.1 Setup

Benchmarks. We evaluate our method on three challenging benchmarks: ALFWorld (Shridhar et al., 2020), ScienceWorld (Wang et al., 2022), and WebShop (Yao et al., 2022a). ALFWorld is a text-based environment that aligns embodied household tasks with their abstract textual counterparts. ScienceWorld is an interactive text environment in which agents must complete elementary-school-level science experiments. WebShop simulates a real-world e-commerce website in which agents are required to navigate, search, and purchase products. Further details on each benchmark are provided in Appendix A.1.

Baselines. We compare our approach against four baselines: ReAct (Yao et al., 2022b), Reflexion (Shinn et al., 2023), ExpeL (Zhao et al., 2024), and ReAct+Skills. ReAct interleaves chain-of-thought reasoning with environment actions. Reflexion and ExpeL are experience-based methods that reflect on past trajectories to guide future decisions. The ReAct+Skills baseline and our proposed SKILLPYRAMID method have access to the same initial skill library, ensuring a fair comparison. Further details on each baseline are provided in Appendix A.2.

Models. We evaluate all methods on a set of representative large language models, including DeepSeek-V3.2 (Liu et al., 2025), GPT-4.1 (OpenAI, 2025), Gemini 2.5 Pro (Comanici et al., 2025), and Qwen3-235B (Yang et al., 2025). To ensure deterministic and reproducible outputs, the sampling temperature is set to 0 for all models.

3.2 Main Results

We conduct comprehensive experiments on ALFWorld, WebShop, and ScienceWorld with four representative backbone models. The overall results are summarized in Table 1. We report both task reward / success rate (R) and the average number of interaction steps (S), where higher reward and fewer steps indicate better agent performance.

SKILLPYRAMID improves performance and generalization. Across different environments and backbone models, SKILLPYRAMID achieves the best overall results among all evaluated methods. Averaged over all reward metrics in Table 1, SKILLPYRAMID obtains an average reward of 73.7, outperforming ReAct (53.4), Reflexion (59.6), ExpeL (61.3), and ReAct+Skills (65.8). Since ReAct+Skills and SKILLPYRAMID are provided with the same initial skill library, this improvement shows that the gain does not merely come from access to additional skills, but from organizing skills into a reusable pyramidal structure. This advantage is especially clear on unseen tasks, where SKILLPYRAMID achieves an average unseen-task reward of 84.8, substantially higher than ReAct+Skills (73.9), ExpeL (68.2), Reflexion (64.8), and ReAct (58.1). These results suggest that the atomic and abstract skills induced by SKILLPYRAMID capture transferable task-solving knowledge and enable stronger generalization to novel task instances.

SKILLPYRAMID improves interaction efficiency. In addition to improving task reward, SKILLPYRAMID also reduces the average number of interaction steps. Across all step metrics, SKILLPYRAMID uses only 14.6 steps on average, compared with 20.2 for ReAct, 18.5 for Reflexion, 17.4 for ExpeL, and 17.7 for ReAct+Skills. This efficiency gain is particularly evident on ALFWorld: for example, on DeepSeek-V3.2, SKILLPYRAMID reduces the number of steps from 15.6 to 10.7 on seen tasks and from 14.7 to 10.6 on unseen tasks compared with ReAct+Skills. On Qwen3-235B, it further reduces the steps from 17.3 to 13.7 on ALFWorld seen tasks and from 18.3 to 11.0 on unseen tasks. These results indicate that hierarchical skill reuse helps agents reach successful states through shorter and more targeted action sequences.

Model	Method	ALFWorld				WebShop		ScienceWorld			
		Seen		Unseen		Seen		Seen		Unseen	
		R↑	S↓	R↑	S↓	R↑	S↓	R↑	S↓	R↑	S↓
DeepSeek-V3.2	ReAct	66.4	19.5	69.4	19.3	31.6	24.1	69.9	17.6	64.7	19.3
	Reflexion	70.1	18.2	73.6	18.0	**35.3**	**22.4**	72.4	16.3	67.8	17.9
	ExpeL	67.9	18.9	76.1	17.4	29.2	24.0	74.9	16.0	74.1	17.5
	ReAct + Skills	75.7	15.6	80.6	14.7	27.0	25.1	71.6	**15.9**	68.5	**16.6**
	SKILLPYRAMID (Ours)	**81.4**	**10.7**	**84.8**	**10.6**	32.0	23.9	**76.3**	16.7	**79.1**	17.0
GPT-4.1	ReAct	58.6	20.1	56.1	20.4	30.4	21.8	57.4	19.1	52.3	20.2
	Reflexion	64.2	18.5	62.8	18.6	34.6	20.1	66.8	17.0	61.6	18.2
	ExpeL	65.7	17.6	64.8	17.4	32.6	18.7	72.1	**14.4**	66.8	15.6
	ReAct + Skills	70.7	16.4	75.8	16.2	38.5	22.1	78.6	17.6	71.8	18.6
	SKILLPYRAMID (Ours)	**77.1**	**15.1**	**85.1**	**13.1**	**42.5**	**15.3**	**86.6**	15.2	**88.6**	**14.2**
Gemini 2.5 Pro	ReAct	60.0	18.7	61.9	19.2	31.7	22.1	58.2	18.4	56.1	19.1
	Reflexion	67.2	17.3	69.4	17.7	**38.2**	19.9	66.8	16.9	63.2	17.5
	ExpeL	68.6	17.9	70.2	17.0	33.1	19.3	72.8	15.0	67.4	14.9
	ReAct + Skills	**75.0**	17.0	73.9	16.7	33.0	18.7	77.3	16.7	79.1	17.0
	SKILLPYRAMID (Ours)	73.6	**16.0**	**80.6**	**11.9**	35.0	**18.2**	**89.7**	**9.0**	**82.9**	**14.3**
Qwen3-235B	ReAct	55.4	20.5	53.8	21.0	28.6	22.4	54.8	19.6	50.6	20.8
	Reflexion	61.8	19.0	60.2	19.5	32.4	20.6	63.6	17.7	59.4	18.9
	ExpeL	63.2	18.2	62.4	17.8	30.8	19.2	69.4	15.3	63.8	16.2
	ReAct + Skills	71.4	17.3	68.6	18.3	31.0	22.2	74.2	15.5	73.0	15.9
	SKILLPYRAMID (Ours)	**77.1**	**13.7**	**91.0**	**11.0**	**38.0**	**18.4**	**85.6**	**14.0**	**86.3**	**14.3**

Table 1: Main results across ALFWorld, WebShop, and ScienceWorld. R denotes reward / success rate (↑ higher is better); S denotes the average number of interaction steps (↓ lower is better). The best result for each metric within each model is shown in **bold**, and the second-best is underlined.

4 Analysis

In this section, we further analyze the contribution of each component in SKILLPYRAMID. We focus on two questions: whether downward atomic extraction, upward abstract induction, and task-driven self-evolution are necessary for the final performance gain, and how self-evolution affects agent performance as the skill system is gradually updated.

Component Contribution. To evaluate the contribution of each component in SKILLPYRAMID, we conduct ablation experiments on ALFWorld and ScienceWorld with DeepSeek-V3.2 as the backbone (Table 2). We consider four ablated variants: (i) *w/o atomic skills*, in which Downward Atomic Extraction is disabled so that no atomic skills are produced and source skills retain their original form; (ii) *w/o abstract skills*, in which Upward Abstract Induction is disabled so that no higher-level guidance is induced; (iii) *w/o self-evolution*, in which the pyramid is frozen after initial construction and newly created skills are not folded back; and (iv) *w/ scratch-generated skills*, in which the Skill Creator generates new skills without retrieving or reusing skills from the pyramid. As reference baselines, we also report vanilla REACT without external skills and REACT W/ FLAT SKILLS, which uses the same initial skill collection but without pyramidal organization. Removing atomic skills drops the average reward from $81.4 \rightarrow 78.6$ on ALFWorld and $76.3 \rightarrow 75.3$ on ScienceWorld, while removing abstract skills yields $81.4 \rightarrow 80.0$ and $76.3 \rightarrow 71.1$, respectively. This asymmetry suggests that atomic skills mainly help on benchmarks rich in reusable low-level operations, whereas abstract skills are more critical for multi-step procedural composition—a point we revisit in §5. Freezing the pyramid further reduces performance to $76.4/73.7$, confirming that incremental incorporation of new skills is needed for sustained adaptation. Most strikingly, scratch-generation collapses the

Method	ALFWorld	ScienceWorld
Baselines		
REACT	66.4	69.9
REACT w/ flat skills	75.7	71.6
Ablations		
w/o Atomic Skills	78.6	75.3
w/o Abstract Skills	80.0	71.1
w/o Self-evolution	76.4	73.7
w/ Scratch-generated Skills	64.3	70.1
Full method		
SKILLPYRAMID	**81.4**	**76.3**

Table 2: Ablation results on ALFWorld and ScienceWorld using DeepSeek-V3.2 as the backbone model.

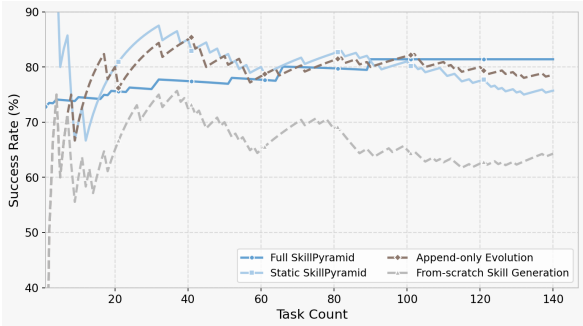

Figure 3: Self-evolution learning curves over incoming tasks on the ALFWorld seen split, obtained using the DeepSeek-V3.2. The full SKILLPYRAMID maintains stronger and more stable late-stage performance than static, append-only, and from-scratch variants.

reward to 64.3/70.1, falling below the flat-skill baseline (75.7 on ALFWorld) and indicating that ungrounded synthesis injects more noise than benefit. Overall, every component contributes, and grounding new skills in the existing hierarchy is essential for reliable skill synthesis.

Effect of Self-Evolution. We further analyze how self-evolution affects performance as the skill system is updated over incoming tasks. Figure 3 compares the full SKILLPYRAMID with three variants: a static version without skill updates, an append-only version without consolidation, and a from-scratch version that generates new skills without reusing the pyramid. The full SKILLPYRAMID maintains the most stable late-stage performance and gradually converges to a high success rate. In contrast, the static variant performs well at early stages but degrades over time, suggesting that a fixed skill system cannot continuously adapt to new tasks. Append-only evolution also achieves competitive performance in the middle stage, but its performance becomes less stable as redundant and weakly organized skills accumulate. The from-scratch variant remains substantially worse, showing that grounding new skills in the existing pyramid is important for reliable skill generation. Overall, these results indicate that self-evolution is most effective when newly generated skills are reused through the hierarchical pyramid rather than simply appended or generated independently.

5 Discussion

The results suggest that the effectiveness of SKILLPYRAMID comes not only from exposing agents to external skills, but from reorganizing these skills into reusable low-level components and high-level procedural guidance. We further discuss two questions: how atomic and abstract skills contribute differently, and whether hierarchical consolidation remains useful when scaling to noisy web-mined skills.

Why Do Atomic and Abstract Skills Help? To avoid post-hoc grouping, we partition tasks *before* the ablations using action statistics from oracle trajectories. For each task x, define the low-level action mass

$$m_{\text{low}}(x) = \frac{1}{T_x} \sum_{t=1}^{T_x} \mathbb{I}[a_t \in \mathcal{A}_{\text{low}}],$$

where $\mathcal{A}_{\text{low}}$ collects reusable primitives (*navigation*, *search*, *pick*, *put*, *open*, *use*, *observe*, *measure*), and let $\ell_{\text{dep}}(x)$ be the longest ordered dependency chain over state-changing or diagnostic motifs (e.g., *clean-then-place*, *heat-then-place*, *test-then-observe*, *measure-then-compare*). Using the empirical quartiles Q_1, Q_3 of each statistic over the evaluation set, we label x as *atomic-heavy* if $m_{\text{low}}(x) \geq Q_3(m_{\text{low}})$ and $\ell_{\text{dep}}(x) \leq Q_1(\ell_{\text{dep}})$, and as *abstract-heavy* if $m_{\text{low}}(x) \leq Q_1(m_{\text{low}})$ and $\ell_{\text{dep}}(x) \geq Q_3(\ell_{\text{dep}})$; tasks in neither extreme are discarded as ambiguous. Table 3 shows an asymmetric pattern: removing atomic skills degrades atomic-heavy tasks more, whereas removing abstract skills disproportionately hurts abstract-heavy tasks. This supports our hypothesis that atomic skills primarily provide reusable execution, while abstract skills drive high-level decomposition and the composition of lower-level skills.

Pyramidal Consolidation Improves Web-Mined Skills. To examine whether SKILLPYRAMID remains useful beyond benchmark-specific skill libraries, we evaluate web-mined skills on GAIA-Lite, a lightweight 75-task subset sampled from

Task Group	#Tasks	m_{low}	ℓ_{dep}	Full	w/o Atomic	w/o Abstract
Atomic-heavy	184	0.93	1.96	**82.6**	76.1 (−6.5)	80.2 (−2.4)
Abstract-heavy	147	0.43	5.14	**78.9**	76.8 (−2.1)	70.9 (−8.0)

Table 3: Action-based task grouping and ablation results. m_{low} and ℓ_{dep} denote atomicity and dependency depth. Parentheses indicate drops from full SKILLPYRAMID.

Setting	Level 1 Acc.↑ / Step↓	Level 2 Acc.↑ / Step↓	Overall Acc.↑ / Step↓
No Web Skills	57.8 / 5.6	43.3 / **4.7**	52.0 / 5.2
Flat Web Skills	62.2 / 5.3	46.7 / 6.8	56.0 / 5.9
SKILLPYRAMID Web Skills	**68.9 / 4.5**	**53.3** / 5.3	**62.7 / 4.8**

Table 4: Performance of web-mined skills on GAIA-Lite, a 75-task subset of GAIA covering Level 1 and Level 2 tasks. Acc. denotes exact-match accuracy, and Step denotes the average number of steps.

the public validation split of GAIA (Mialon et al., 2024). Starting from 150K web-mined candidate skills, we retain 2,271 high-quality skills and compare three settings: no web skills, flat web skills, and the same web skills organized by SKILLPYRAMID. As shown in Table 4, flat web skills already improve over the no-skill baseline, while SKILLPYRAMID Web Skills further achieve higher accuracy and fewer execution steps on both Level 1 and Level 2 tasks. This indicates that the gain comes not only from acquiring more skills, but from consolidating noisy web-mined skills into a pyramidal structure with atomic operations, abstract task schemas, and explicit reuse relations.

6 Related Work

LLM agents and experience reuse. LLM agents have made substantial progress on long-horizon interactive tasks by interleaving reasoning, action, and experience reuse (Yao et al., 2022b; Shinn et al., 2023; Zhao et al., 2024; Schick et al., 2023; Qin et al., 2024). Existing methods can be broadly grouped into reasoning–action coupling, experience-driven self-improvement, and tool-augmented orchestration. ReAct interleaves verbal reasoning with environment actions to support sequential decision making (Yao et al., 2022b). Building on this paradigm, Reflexion accumulates verbal self-reflection across episodes to refine future trials (Shinn et al., 2023), while ExpeL consolidates cross-task experiential insights to bootstrap unseen tasks (Zhao et al., 2024). Beyond pure reasoning, another line augments agents with external tools and expert modules: MRKL connects language models with symbolic reasoners through modular routing (Karpas et al., 2022), and Toolformer learns API invocation from self-supervised demonstrations (Schick et al., 2023). Subsequent work scales this paradigm through expert-model orchestration (Shen et al., 2023), plug-and-play tool composition (Lu et al., 2023), and large-scale API use (Qin et al., 2024).

Skill organization and evolution. Reusable skills have emerged as an essential substrate for long-horizon agent learning, allowing agents to consolidate interaction experience into transferable capabilities (Liang et al., 2023; Wang et al., 2023a, 2026; Mi et al., 2026; Fang et al., 2025). Existing work can be broadly categorized into executable skill libraries, procedural memory accumulation, and structured skill knowledge bases. Code as Policies synthesizes executable policy code by composing low-level control primitives for embodied control (Liang et al., 2023). DEPS plans in open worlds through description, explanation, planning, and selection (Wang et al., 2023b), and Voyager builds an ever-growing executable skill library for lifelong Minecraft exploration (Wang et al., 2023a). From a procedural memory perspective, Memp explores reusable procedural traces accumulated from agent interaction (Fang et al., 2025), while Skill-Pro learns reusable skills via non-parametric PPO (Mi et al., 2026). More recently, structured skill knowledge bases have emerged: SkillNet creates, evaluates, and connects skills under a unified interface (Liang et al., 2026), and SkillX automatically constructs multi-level skill knowledge bases from agent trajectories (Wang et al., 2026).

7 Conclusion

We presented SKILLPYRAMID, a hierarchical skill consolidation framework for improving the reuse, transfer, and evolution of agent skills. By analyzing an existing skill collection, SKILLPYRAMID extracts atomic reusable skills, induces abstract guidance skills, and maintains explicit reuse and dependency relations among them. Its task-driven self-evolution mechanism further enables agents to compose, validate, and incorporate new skills during execution. Experiments on ALFWorld, WebShop, and ScienceWorld show that SKILLPYRAMID consistently improves task reward and interaction efficiency across multiple backbone LLMs, suggesting that structured skill consolidation is a promising direction for building agents that accumulate reusable capabilities over time.

Limitations

This paper has several limitations. First, the initial pyramid construction requires a one-time analysis pass over the entire skill collection, and we have not investigated lighter-weight construction strategies, which would be necessary for scaling to ultra-large skill repositories. Second, both the Relation Analyzer and the Relation Builder rely on a capable backbone LLM to identify reuse relations and synthesize new skills; we have not systematically studied how the quality of the constructed pyramid degrades under weaker backbones, which limits our understanding of when the framework remains effective. Third, our evaluation is restricted to text-based interactive benchmarks, and we do not validate the framework in multimodal or embodied settings, leaving its applicability to such scenarios open. Fourth, all results are reported from a single deterministic run with sampling temperature 0, and we do not measure run-to-run variance, which leaves the stability of skill construction and selection uncharacterized. Finally, the pyramid uses only two reuse axes and does not model richer relations such as temporal or causal dependencies needed for more complex compositional reasoning.

Ethical Considerations

Our experiments use synthetic interactive benchmarks (ALFWorld, ScienceWorld, WebShop, and GAIA-Lite) that contain no personally identifying or offensive content, and involve no human subjects. As a general framework for skill accumulation and self-evolution, SKILLPYRAMID nevertheless inherits the dual-use concerns of capable LLM agents: a continually evolving skill library could, in principle, accumulate skills aligned with harmful objectives if deployed without task-level safeguards. We recommend that downstream users couple SKILLPYRAMID with task-level access control and skill-level auditing, and apply dedicated content review to web-mined skill libraries, before deployment in user-facing applications.

Use of AI Assistants. AI assistants were used to aid coding and to polish the writing of this paper. All scientific contributions, including the framework design, experimental setup, analyses, and conclusions, are the authors' own.

References

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2025. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, and 1 others. 2022. Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pages 9493–9500. IEEE.

Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, and 1 others. 2026. Skillnet: Create, evaluate, and connect ai skills. *arXiv preprint arXiv:2603.04448*.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36:43447–43478.

Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. 2026. Skill-pro: Learning reusable skills from experience via non-parametric ppo for llm agents. *arXiv preprint arXiv:2602.01869*.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. Gaia: a benchmark for general ai assistants. In *International Conference on Learning Representations*, volume 2024, pages 9025–9049.

OpenAI. 2025. Introducing GPT-4.1 in the API. OpenAI blog. Accessed 2026-05-19.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pages 9695–9717.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36:38154–38180.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and 1 others. 2026. Skillx: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*.

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

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. 2022. Scienceworld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11279–11298.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023b. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022b. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

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

A Experimental Setup and Implementation Details

A.1 Benchmarks Implementation

ALFWorld. ALFWorld (Shridhar et al., 2020) is an interactive household benchmark that aligns abstract TextWorld games with embodied ALFRED tasks, requiring agents to complete long-horizon household goals by navigating rooms, observing state changes, and manipulating objects through textual actions. Following prior language-agent evaluations (Yao et al., 2022b; Shinn et al., 2023) and the SkillNet protocol (Liang et al., 2026), we use `eval_in_distribution` as the seen split and `eval_out_of_distribution` as the unseen split, which contain 140 and 134 tasks, respectively.

ScienceWorld. ScienceWorld (Wang et al., 2022) is an interactive text-based benchmark for evaluating grounded scientific reasoning, where agents must perform experiments, operate instruments, and reason about changing world states in environments inspired by elementary science curricula. Following SkillNet (Liang et al., 2026), we use the dev split defined by `src/scienceworld/data/valid_indices.json` as the seen split and `src/scienceworld/data/test_indices.json` as the unseen split; these splits contain 194 and 211 task variations, respectively, covering 24 ScienceWorld task types.

WebShop. WebShop (Yao et al., 2022a) is a simulated e-commerce web environment with real-world products and crowd-sourced shopping instructions, where agents search, compare, customize, and purchase products according to natural-language user goals. Following SkillNet (Liang et al., 2026), we evaluate on a fixed set of 200 randomly sampled WebShop tasks specified by `src/webshop/data/test_indices.json`.

A.2 Baselines Implementation

ReAct. ReAct (Yao et al., 2022b) interleaves language reasoning with executable actions. Follow-

ing the configuration of Liang et al. (2026), we implement a ReAct-style loop in which the agent observes the current state, produces a `Thought` and a valid `Action`, and updates its context with environment feedback. ReAct uses no external skills, reflections, or experience memory, and shares the same prompts as our evaluation setting.

Reflexion. Reflexion (Shinn et al., 2023) extends ReAct with verbal self-reflection. After each failed or low-reward trial, the model summarizes the error and stores a concise reflection for future attempts. These reflections are used only as textual guidance and are not transformed into structured skills or reuse relations.

ExpeL. ExpeL (Zhao et al., 2024) distills past trajectories into reusable natural-language lessons. We extract general strategies and failure patterns from previous tasks and provide the retrieved lessons as additional context for a ReAct-style agent. Unlike SkillPyramid, ExpeL keeps a flat experience memory without atomic extraction, abstract induction, or hierarchical updating.

ReAct+Skills. ReAct+Skills equips ReAct with the same initial skill library used by SkillPyramid. For each task, relevant skills are retrieved and inserted into the system prompt. This baseline isolates the effect of skill access, as it uses a flat skill library without hierarchical consolidation, relation construction, or self-evolution.

A.3 Additional Implementation Details

During SkillPyramid construction, we use Qwen3.5-35B-A3B as the backbone model for all construction agents, and Qwen3-Embedding-4B to compute embedding similarities among skills for candidate retrieval, coarse grouping, and relation screening before full-content analysis. The initial skill library is constructed from the training splits of ALFWorld, ScienceWorld, and WebShop. The outer Analyzer–Builder pipeline is run for at most 3 rounds, and the fine-grained analyzer within each Relation Analyzer call is run for at most 5 rounds.

A.4 Compute and Model Budget

All experiments use API calls only and no model parameters were fine-tuned, updated, or trained. The main evaluation spans 17,580 episodes (879 instances × 5 methods × 4 backbones). Total API usage: 8.4K calls (21.6M input / 3.1M output tokens) for the one-time SkillPyramid construction, 315K calls (0.92B / 46.8M tokens) for the main evaluation, and 24K calls (71.5M / 5.4M tokens) for ablations and analysis. Local computation (environment simulation, retrieval, indexing, logging) ran on CPU-only nodes. All results come from a single deterministic run at temperature 0, with no hyperparameter tuning on test splits.

A.5 Prompt Templates

Prompt Template for Relation Analyzer The Relation Analyzer identifies reusable relations in the skill library. Its prompt follows a two-stage process: SCREEN filters candidate groups using names, descriptions, and similarity signals, while DECIDE inspects full `SKILL.md` files when needed and outputs relation-construction assignments for the Relation Builder.

Relation Analyzer Prompt

```
You are the RELATION ANALYZER in SKILLPYRAMID.
 You may be called multiple times.
Each time you are called, you will be clearly
informed that you are currently in one of the
following two stages.

STAGES
- SCREEN: Inspect skill names, descriptions
and similarity signals to decide which
candidate pairs or groups deserve DECIDE-stage
 analysis.
- DECIDE: Produce the final relation
assignment for the RELATION BUILDER. Use the
available tools to inspect full `SKILL.md`
contents when necessary.

Your job is to identify only relations that
are useful for building a hierarchical
reusable skill pyramid. Be conservative and
evidence-grounded.

INPUT FOR SCREEN
{
  "stage": "SCREEN",
  "candidates": [
    {
      "skills": [
        {"name": "<skill name>", "description":
         "<skill description>"}
      ],
      "similarity": "<optional score or signal
      >"
    }
  ]
}

INPUT FOR DECIDE
{
  "stage": "DECIDE",
  "candidate": {
    "skills": [...],
    "screen_reason": "<optional reason>",
    "similarity": "<optional score or signal>"
  }
```

```
}

TOP-LEVEL ASSIGNMENTS

DOWNWARD_ATOMIC_EXTRACTION: Resolve the
relation by creating, reusing, or
canonicalizing a lower-level atomic capability.
 It includes three relation types:
- SHARED_PART: Multiple skills share a
concrete reusable sub-capability, while each
still has substantial task-specific logic. Do
not use for vague thematic similarity.
- SUBSET: One skill is a strict reusable
subskill of another, and the broader skill
adds extra workflow, constraints, or context.
Do not use for merely related skills in the
same domain.
- MERGE: Two or more skills are near-
equivalent in triggers, inputs, workflow, and
outputs, and should become one canonical skill.
 Do not use when one skill should delegate to
another.

UPWARD_ABSTRACT_INDUCTION: Create an abstract
parent skill when peer skills share a higher-
level solving pattern or task schema. The
parent summarizes composition rather than
becoming an executable merged skill.

DECISION RULES
[...DECISION_RULES_PLACEHOLDER...]

OUTPUT FOR SCREEN
{
  "stage": "SCREEN",
  "selected": [...],
  "rejected": [...]
}

OUTPUT FOR DECIDE
ASSIGNMENT: <...>
SKILLS: <...>
RELATION_TYPE: <...>
REASON: <...>
END

CONSTRAINTS FOR SCREEN
- selected: candidates worth a relation
attempt.
- rejected: candidates not worth it (coarse
evidence).
- skills: a pair or small group per entry.
- reason: short, evidence-based.
- No assignment labels, no needs_full_content.

CONSTRAINTS FOR DECIDE
- One DECIDE block per candidate, ready for
RELATION BUILDER.
- If coarse evidence is thin, read full skill
content first.
- Pick ASSIGNMENT: DOWNWARD (extract) or
UPWARD (abstract).
- Fill only the fields for the chosen branch:
    * UPWARD -> ABSTRACT_PATTERN.
    * DOWNWARD -> RELATION_TYPE, then:
        - SHARED_PART -> SHARED_PART.
        - SUBSET -> SUBSET_OF (narrower vs
        broader).
        - MERGE -> MERGED_SKILL (canonical name
        ).
```

Prompt Template for Relation Builder The Relation Builder constructs the assigned relations from full skill contents. For *Downward Atomic Extraction*, it extracts, reuses, merges, or canonicalizes atomic skills and adds explicit reuse references; for *Upward Abstract Induction*, it creates abstract parent skills that capture shared task schemas.

Relation Builder Prompt

```
You are the RELATION BUILDER in SKILLPYRAMID.
You receive a final assignment from the
RELATION ANALYZER and the involved `SKILL.md`
contents.

Your job is to construct the actual reusable
relation while preserving the skill library's
correctness. Be conservative and evidence-
grounded.

INPUT

ASSIGNMENT: <DOWNWARD_ATOMIC_EXTRACTION |
UPWARD_ABSTRACT_INDUCTION>
SKILLS: <comma-separated skill names>
RELATION_TYPE: <SHARED_PART | SUBSET | MERGE,
only for DOWNWARD_ATOMIC_EXTRACTION>
SUBSET_OF: <narrower skill -> broader skill,
only for RELATION_TYPE SUBSET>
MERGED_SKILL: <proposed canonical skill name,
only for RELATION_TYPE MERGE>
SHARED_PART: <common reusable capability, only
 for RELATION_TYPE SHARED_PART>
ABSTRACT_PATTERN: <high-level pattern, only
for UPWARD_ABSTRACT_INDUCTION>
REASON: <brief evidence-grounded explanation>
END

The caller also provides the full `SKILL.md`
contents of the involved skills.

TOP-LEVEL ASSIGNMENTS

DOWNWARD_ATOMIC_EXTRACTION: Build or reuse a
lower-level atomic capability. This assignment
 contains three relation types: SHARED_PART,
SUBSET, and MERGE.
- SHARED_PART: Extract the shared reusable
operation into one lower-level atomic skill.
Then rewrite each source skill so it delegates
 exactly that shared operation through a reuse
 reference.
- SUBSET: Rewrite only the broader skill.
Wherever it reimplements the covered subskill,
 replace that local portion with an explicit
reuse reference to the narrower skill.
- MERGE: Create one canonical lower-level
skill that preserves all useful, non-
conflicting details from the near-equivalent
source skills. The output must be a standalone
 skill, not a comparison.
```

```
UPWARD_ABSTRACT_INDUCTION: Create one higher-
level abstract skill that summarizes the
common task schema of the group. Do not merge
or replace the children. The abstract skill
should describe how lower-level skills compose
 to solve a class of tasks.

REUSE REFERENCE

Use exactly this format when a skill delegates
 to another skill:

[reuse skill: <skill name> | when: <short
trigger> | provides: <short capability>]

BUILD RULES
- The RELATION ANALYZER assignment is
authoritative. Do not change ASSIGNMENT or
RELATION_TYPE.
- Do not invent tools, scripts, files,
observations, actions, dependencies, or source
-skill capabilities.
- Preserve important workflows, constraints,
edge cases, and verification checks from
source skills.
- For RELATION_TYPE SHARED_PART, the new skill
 must be narrower than every source skill.
- For RELATION_TYPE SUBSET, rewrite only the
broader skill named in SUBSET_OF.
- For RELATION_TYPE MERGE, combine compatible
details into one canonical skill and do not
preserve duplicate standalone variants.
- For UPWARD_ABSTRACT_INDUCTION, the new skill
 must be higher-level guidance, not a concrete
 action script.
- Rewritten skills must remain understandable
as standalone `SKILL.md` files.
- Add reuse references only near the affected
passages.
- Use UPPERCASE for ASSIGNMENT names,
RELATION_TYPE names, and output field labels
when labels are required.

OUTPUT FORMAT

Output skill content only. Do not include
analysis, markdown fences, JSON, or commentary
 unless the caller explicitly requests
structured metadata.

For generated or rewritten skills, use this
format:

name: <kebab-case skill name>
description: <one-sentence description>

# <Skill Title>

## Purpose
<what capability or pattern this skill
provides>

## When to Use
<triggers and applicable conditions>

## Inputs / Expected Context
<required observations, task state, objects,
files, or prior context>

## Workflow
<step-by-step reusable procedure or high-level
 composition pattern>

## Verification
<how to confirm progress or completion>

## Constraints / Guardrails
<limits, failure modes, and unsupported
assumptions>

## Output
<expected result or capability provided>
```

Prompt Template for Skill Creator The Skill Creator generates task-specific skills grounded in the current pyramid. It first uses higher-level skills for *Framework Construction*, then lower-level skills for *Detail Instantiation*; stage-specific retrieval weights implement this priority.

Skill Creator Prompt

```
You are the SKILL CREATOR in SKILLPYRAMID. You
 may be called multiple times.
Each time you are called, you will be clearly
informed that you are currently in one of the
following two stages.

STAGES
- FRAMEWORK_CONSTRUCTION: Use higher-level or
abstract skills first to outline the task
structure, subgoals, decision points, success
criteria, and high-level solving logic.
- DETAIL_INSTANTIATION: Use lower-level or
atomic skills to fill executable operations,
required inputs, state checks, verification
rules, outputs, and reuse references, then
produce the final reusable skill.

Your job is to produce a reusable `SKILL.md`
body that is grounded in the supplied task
context, retrieved skills, and current
environment feedback. Be conservative and
evidence-grounded.

INPUT FOR FRAMEWORK_CONSTRUCTION
{
  "stage": "FRAMEWORK_CONSTRUCTION",
  "task_instruction": "<new task instruction>",

  "retrieved_skills": [
    {
      "name": "<skill name>",
      "description": "<skill description>",
      "level": "<optional level or role>",
      "relation_context": "<optional pyramid
      or graph context>",
      "relevance": "<optional exact, partial,
      or background relevance summary>"
    }
  ]
}

INPUT FOR DETAIL_INSTANTIATION
{
```

```
  "stage": "DETAIL_INSTANTIATION",
  "task_instruction": "<new task instruction>",

  "framework": "<FRAMEWORK_CONSTRUCTION output
  >",
  "retrieved_skills": [
    {
      "name": "<skill name>",
      "description": "<skill description>",
      "content": "<optional SKILL.md content>",

      "level": "<optional level or role>",
      "relation_context": "<optional pyramid
      or graph context>",
      "relevance": "<optional exact, partial,
      or background relevance summary>"
    }
  ]
}

GENERATION PROCESS
- In FRAMEWORK_CONSTRUCTION, use the stage-
ranked `retrieved_skills` to extract task
class, reusable strategy, decomposition,
decision points, and success criteria. Focus
on names, descriptions, relation context,
relevance summaries, and any supplied high-
level procedural content.
- FRAMEWORK_CONSTRUCTION produces a framework
only: candidate name, description, task class,
 high-level logic, subgoals, success criteria,
 reuse candidates, and cautions. Do not write
the final skill body in this stage.
- In DETAIL_INSTANTIATION, use the stage-
ranked `retrieved_skills` to extract concrete
operations, required inputs, state checks,
verification rules, outputs, and reusable
failure cautions. Focus on supplied `SKILL.md`
 content, exact reusable subtasks, and
environment-compatible action patterns.
- DETAIL_INSTANTIATION produces the final
reusable skill by filling the framework with
grounded executable detail and placing reuse
references near the substeps they support.
- If retrieved skills are only partially
related, reuse their reasoning pattern,
verification habits, and cautions, but do not
copy unsupported concrete objects, rooms,
product IDs, commands, or environment facts.

REUSE REFERENCE
Use exactly this format when a generated skill
 delegates to another skill:
[reuse skill: <skill name> | when: <short
trigger> | provides: <short capability>]

GENERATION RULES
- Follow the current STAGE's output format
only.
- Use exact names from retrieved/available
skills; invent nothing
  (skills, files, tools, actions, env facts).
- Final skill is self-contained, usable as a
SKILL.md body.
- Favor reusable workflow guidance over one-
off replay.
- Action verbs/commands only if in the
supplied action space.
- Live environment feedback overrides the
generated skill.
- If nothing relevant is retrieved, borrow
only style and cautions.
- Keep concrete details, constraints, and
failure cautions as evidence.
- UPPERCASE for STAGE names, process names,
and field labels.

OUTPUT A FOR FRAMEWORK_CONSTRUCTION
STAGE: FRAMEWORK_CONSTRUCTION
NAME: <concise kebab-case candidate skill name
>
DESCRIPTION: <one hedged sentence describing
when this skill may help>
TASK_CLASS: <reusable task class>
HIGH_LEVEL_LOGIC: <high-level solving logic
inferred from abstract or higher-level skills>
SUBGOALS: <ordered subgoals or decision points
>
SUCCESS_CRITERIA: <how the final skill should
define completion>
REUSE_CANDIDATES: <exact retrieved skill names
 and the role each may play>
CAUTIONS: <unsupported assumptions,
environment dependencies, or failure modes>
END

CONSTRAINTS FOR FRAMEWORK_CONSTRUCTION
- One FRAMEWORK_CONSTRUCTION block.
- No final CONTENT.
- REUSE_CANDIDATES: exact retrieved skill
names only.

OUTPUT B FOR DETAIL_INSTANTIATION
Output exactly:
NAME: <concise kebab-case skill name>
DESCRIPTION: <one hedged sentence describing
when this skill may help>
CONTENT:
# <Skill Title>
## Purpose
<task class and intended outcome>

## Possible Strategies
<high-level plan, decomposed through abstract
skills when relevant>

## Reuse Plan
<reuse references for exact subtask coverage,
each near its trigger>

## Execution Checklist
<atomic operations, state checks, and
verification habits>

## Cautions
<failure modes, unsupported assumptions, and
when to switch tactics>
END

CONSTRAINTS FOR DETAIL_INSTANTIATION
- One final skill, structured by the
FRAMEWORK_CONSTRUCTION output.
- Fill operations, inputs, checks, outputs
from
  retrieved skills when supplied.
- Reuse references: exact names only, placed
near affected passages.
```

```
- No analysis, fences, JSON, or commentary
outside the skill format.
```

Prompt Template for ALFWorld Evaluation The ALFWorld evaluation prompt frames the agent as an embodied household assistant. It specifies the household environment, task goal, valid object-manipulation actions, and the required Thought/Action output format. For skill-based methods, retrieved skills are inserted through the {{skill_reference}} field.

System Prompt for ALFWorld

```
Interact with a household to solve a task.
Imagine you are an intelligent agent in a
household environment, and your goal is to
perform actions to complete the task.

At the beginning of the interaction, you will
be given a detailed description of the current
 environment and the task goal. At each turn,
you will be given the latest observation. You
should first think about the current situation
 and plan your next steps, then output your
action for this turn.

Your output must strictly follow this format:

Thought: <your thoughts>
Action: <your next action>

The available actions are:

1. go to {recep}
2. take {obj} from {recep}
3. move {obj} to {recep}
4. open {recep}
5. close {recep}
6. use {obj}
7. clean {obj} with {recep}
8. heat {obj} with {recep}
9. cool {obj} with {recep}

Here, {obj} and {recep} correspond to objects
and receptacles in the environment.

After each turn, the environment will provide
immediate feedback. Use this feedback to plan
your next steps. If the environment outputs "
Nothing happened", the previous action was
invalid, and you should try another valid
option.

{{skill_reference}}

Your response should use the following format:

Thought: <your thoughts>
Action: <your next action>
```

Prompt Template for ScienceWorld Evaluation The ScienceWorld evaluation prompt frames the agent as an assistant conducting scientific experiments. It defines the available rooms, experiment-oriented actions, object inspection and manipulation operations, and the required Thought/Action output format. Retrieved skill references are inserted through the {{skill_reference}} field when applicable.

System Prompt for ScienceWorld

```
You are a helpful assistant conducting
scientific experiments in an environment.

The environment contains several rooms:
kitchen, foundry, workshop, bathroom, outside,
 living room, bedroom, greenhouse, art studio,
 and hallway.

You should explore the environment and find
the items needed to complete the experiment.

You can teleport to any room in one step.

All containers in the environment have already
 been opened, so you can directly get items
from the containers.

The available actions are:

open OBJ: open a container
close OBJ: close a container
activate OBJ: activate a device
deactivate OBJ: deactivate a device
connect OBJ to OBJ: connect electrical
components
disconnect OBJ: disconnect electrical
components
use OBJ [on OBJ]: use a device or item
look around: describe the current room
examine OBJ: describe an object in detail
look at OBJ: describe a container's contents
read OBJ: read a note or book
move OBJ to OBJ: move an object to a container
pick up OBJ: move an object to the inventory
pour OBJ into OBJ: pour a liquid into a
container
mix OBJ: chemically mix a container
teleport to LOC: teleport to a specific room
focus on OBJ: signal intent on a task object
wait: take no action for 10 steps
wait1: take no action for one step

{{skill_reference}}

Your response should use the following format:

Thought: <your thoughts>
Action: <your next action>
```

Prompt Template for WebShop Evaluation The WebShop evaluation prompt frames the agent as a web-shopping assistant. It provides the current observation and available actions at each step, restricts actions to search[keywords] and click[value], and requires the Thought/Action output format. Retrieved skill references are in-

serted through the {{skill_reference}} field for skill-based methods.

System Prompt for WebShop

```
You are web shopping.

I will give you instructions about what to do.
 You must follow the instructions.

At each round, I will give you an observation
and a list of available actions. You should
choose one valid action based on the current
state and the instruction.

You may use the search action if search is
available. You may also click one of the
available clickable buttons.

An action must follow one of these formats:

search[keywords]
click[value]

The search keywords are up to you, but the
value in click[value] must exactly match one
of the available actions.

If the action is invalid, nothing will happen.

Design search keywords carefully so that they
are specific enough to find the requested
product.

{{skill_reference}}

Your response should use the following format:

Thought: <your thoughts>
Action: <your next action>
```