Reinforcement Learning for Self-Improving Agent with Skill Library

Jiongxiao Wang[1*] **Qiaojing Yan**[2] **Yawei Wang**[2] **Yijun Tian**[2] **Soumya Smruti Mishra**[2]
Zhichao Xu[2] **Megha Gandhi**[2] **Panpan Xu**[2†] **Lin Lee Cheong**[2]
[1]University of Wisconsin–Madison; [2]AWS Agentic AI
jwang2929@wisc.edu; {qiaojiny, yawenwan, yijunt, soumish, xzhichao, ganmegha, xupanpan, lcheong}@amazon.com

Abstract

Large Language Model (LLM)-based agents have demonstrated remarkable capabilities in complex reasoning and multi-turn interactions but struggle to continuously improve and adapt when deployed in new environments. One promising approach is implementing skill libraries that allow agents to learn, validate, and apply new skills. However, current skill library approaches rely primarily on LLM prompting, making consistent skill library implementation challenging. To overcome these challenges, we propose a Reinforcement Learning (RL)-based approach to enhance agents' self-improvement capabilities with a skill library. Specifically, we introduce Skill Augmented GRPO for self-Evolution (SAGE), a novel RL framework that systematically incorporates skills into learning. The framework's key component, Sequential Rollout, iteratively deploys agents across a chain of similar tasks for each rollout. As agents navigate through the task chain, skills generated from previous tasks accumulate in the library and become available for subsequent tasks. Additionally, the framework enhances skill generation and utilization through a Skill-integrated Reward that complements the original outcome-based rewards. Experimental results on AppWorld demonstrate that SAGE, when applied to supervised-finetuned model with expert experience, achieves 8.9% higher Scenario Goal Completion while requiring 26% fewer interaction steps and generating 59% fewer tokens, substantially outperforming existing approaches in both accuracy and efficiency. Our code is available at https://github.com/amazon-science/SAGE.

1 Introduction

Large language model (LLM)-based agents have been widely applied to automate complex tasks through active environmental interactions, including coding agent (Yang et al., 2024; Novikov et al., 2025), deep research (OpenAI, 2025), assistant agent (Yao et al., 2024; Chen et al., 2025), and web browsing (Yao et al., 2022; Zhou et al., 2024). To enhance the performance of these multi-turn interactive agents, researchers have successfully integrated reinforcement learning (RL) techniques into their frameworks (Chen et al., 2025; Qi et al., 2025; Zhou et al., 2025a; Wang et al., 2025a). Recent advances, particularly in reinforcement learning with verifiable rewards (RLVR) (Shao et al., 2024; Guo et al., 2025), have enabled effective end-to-end agent training for improved performance (Jin et al., 2025). However, despite RL's effectiveness, significant limitations persist: RL-trained agents are often limited to specific training scenarios (Zheng et al., 2024a; Li et al., 2024). When deployed in new environments, they struggle to demonstrate continual learning capabilities to effectively utilize valuable on-going experiences for future tasks.

To address these limitations, one potential solution is enabling agents to transform their previous interaction experiences into reusable skills, which can be stored in a skill library for future reference. When agents encounter similar tasks, these previously acquired skills can be leveraged through experience replay to improve task success rates, particularly in scenarios that were not encountered during training but experienced during deployment. Furthermore, since each skill is composed of a list of actions, utilization of these skills can enhance agent efficiency by condensing complex action sequences into reusable operations.

Recent research has made significant strides in enabling agents to compose reusable skills (Wang et al., 2024a; Cai et al., 2024; Nguyen et al., 2024; Wang et al., 2024c; Zheng et al., 2025; Wang et al., 2025b). For instance, Wang et al. (2025b) focuses on inducing high-level web browsing skills from successful action trajectories, transforming primi-

*Work was done during an internship at AWS Agentic AI.
†Corresponding author: Panpan Xu, Email: xupanpan@amazon.com

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1529–1550
July 2-7, 2026

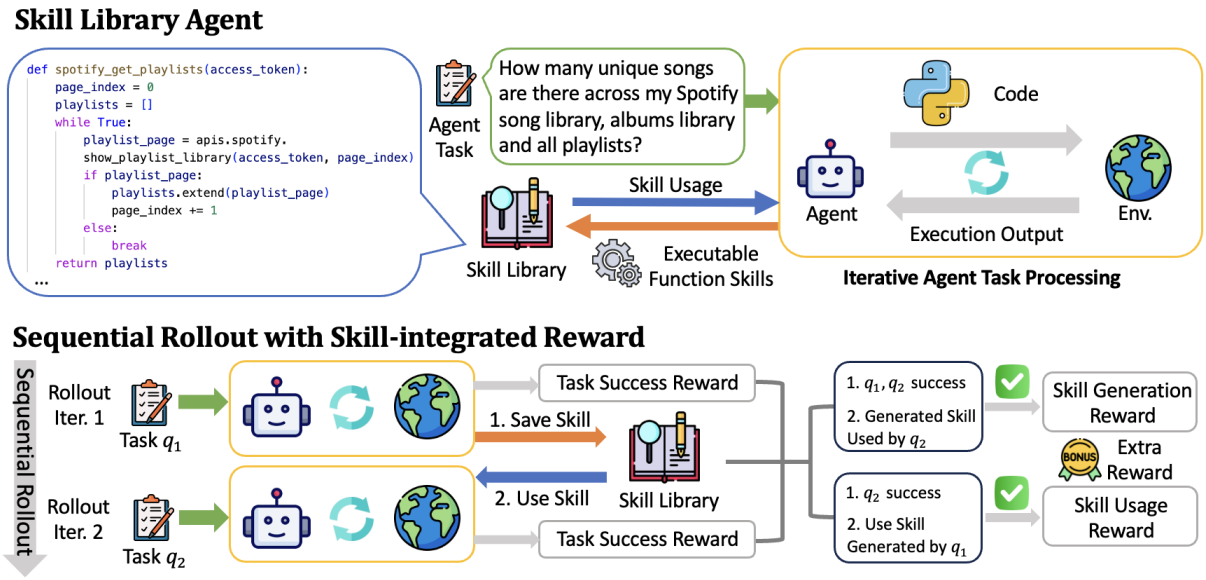

Figure 1: Illustration for Skill Library Agent and Sequential Rollout with Skill-integrated Reward.

tive actions like *click* and *search* into more complex operations such as *search product*. While these works have demonstrated the effectiveness of skill libraries for deployed agents, they predominantly rely on manually crafted prompts for skill generation and utilization. This prompt-based approach, however, is inherently constrained by the instruction following capabilities of the base model, limiting both the quality and adaptability of the skill library.

In this paper, we explore enhancing agents' self-improvement capabilities through RL with a skill library. Given the diverse frameworks of agents for various tasks, we focus specifically on tool-using agents that interact with environments through API calls and receive corresponding feedback. Our implementation extends the CodeAct framework (Wang et al., 2024b), which enables agents to compose code by combining multiple APIs with basic programming constructs (such as for loops) to solve complex tasks. Building upon this foundation, we develop a specialized framework for self-improving agents with skill library (referred to as **skill library agents** for simplicity). Unlike previous frameworks such as Agent Skill Induction (Wang et al., 2025b), which typically define reusable skills only after task completion, we implement a unified format for both task solving and skill generation, following Nguyen et al. (2024). In our approach, when the agent model interacts with the environment through APIs, it generates programmatic functions that can be saved as skills and subsequently called to execute, rather than using multiple APIs directly. Given that in-context prompting struggles to adapt open-source models to this newly proposed skill library agent, supervised fine-tuning (SFT) is first applied before RL using high-quality trajectories collected through expert experience generated by advanced LLMs.

Based on the SFT model, we propose a novel RL framework for skill library agents. Traditional RL approaches typically consider rewards only for individual examples, limiting their scope to ongoing task performance. For skill library agents, the focus is on developing high-quality, reusable skills while improving accurate skill usage from the library for task optimization. To realize this, we extend Group Relative Policy Optimization (GRPO) (Shao et al., 2024) and propose Skill Augmented GRPO for self-Evolution (SAGE), consisting of **Sequential Rollout** and **Skill-integrated Reward**. Instead of single-task trajectories, the agent is trained with chains of similar tasks. We implement the Sequential Rollout process on the task chain, where skills generated in the previous tasks are preserved and made available for use in the following ones. Under this framework, the final Skill-integrated Reward is computed as the sum of two components: the verifiable outcome-based reward and an extra reward for high-quality skill generation and utilization.

To evaluate the effectiveness of SAGE for skill library agents, we conducted experiments on the AppWorld dataset (Trivedi et al., 2024), where agents interact with their environment through API docu-

mentation lookup, API calls, and logical programming constructs to solve complex practical tasks. AppWorld utilizes the Scenario Goal Completion (SGC) metric, which measures the success rate of scenarios where all three similar tasks within a scenario are successfully completed. This metric effectively evaluates how well generated skills transfer across similar tasks. After applying SAGE to Qwen2.5-32B-Instruct, we observe significant improvements compared to prompting-based approaches. Our method achieves more than 3x SGC score (60.7% vs. 19.6%) while requiring less than half of generated tokens (1,475 vs. 2,988). Analysis of skill utilization reveals that agents trained with SAGE demonstrate more than 2x success rates (0.72 vs. 0.31) when utilizing learned skills.

Furthermore, our approach achieves state-of-the-art performance on both Test Normal and Test Challenge datasets compared to previous training methods which also applied RL but without a skill library. For example, our method achieves 72.0% Task Goal Completion (TGC) and 60.7% SGC with an average of 12.1 interaction steps and 1,475 generated tokens on the Test Normal set. This represents substantial improvement over baseline training with GRPO, which achieved 69.2% TGC and 51.8% SGC while requiring 16.4 average steps and 3,613 tokens. These improvements demonstrate the effectiveness of our approach in enhancing both task performance and efficiency through the skill library agent.

2 Related Work

LLM-based Agent. Recent advancements in instruction-tuned LLMs have enabled them to follow user instructions for interacting with external environments as autonomous agents. Various frameworks have been developed to enhance these backbone LLMs' capabilities in performing agentic tasks. Yao et al. (2023) pioneered a "reason-then-act" pipeline, guiding LLMs to generate interactive actions for agents after a text reasoning process. Erdogan et al. (2025) extended this approach by incorporating an additional planning phase. Additionally, Wang et al. (2024b) demonstrated that generating executable Python code and then run it within a code interpreter could significantly improve agent performance. To further enhance agent capabilities, researchers have applied both supervised fine-tuning (Schick et al., 2023; Chen et al., 2023; Zeng et al., 2024; Zhang et al., 2024) and reinforcement learning (Song et al., 2024; Bai et al., 2024; Wang et al., 2025a; Chen et al., 2025; Qi et al., 2025; Zhou et al., 2025a) to develop more effective and adaptive agent backbone LLMs.

Self-Improving Agent with Skill Library. While RL algorithms have enabled agents to self-improve through valuable experiences explored during rollouts (Zhou et al., 2025b; Putta et al., 2024; Qi et al., 2025), enabling continuous self-improvement after deployment, especially in new environments, remains challenging. Wang et al. (2024a) pioneered the use of a skill library to record successful behaviors for later retrieval in Minecraft exploration. Subsequently, numerous studies have demonstrated the effectiveness of such skill libraries across various agentic tasks, including web exploration (Wang et al., 2024c; Zheng et al., 2025; Wang et al., 2025b), computer control (Zheng et al., 2024b; Wu et al., 2024), and math problems (Nguyen et al., 2024). These skills can take two forms: natural language experience memories serving as a reference, or executable skills that can be directly implemented in the environment. In this paper, we focus on leveraging RL to enhance agents' self-improvement capabilities by teaching them to generate executable skills for the skill library during test time.

3 Method

In this section, we first introduce our skill library agent and distinguish it from existing approaches. We then present our novel RL framework SAGE, which is specifically designed to integrate the skill library during training process. An illustrative figure of the skill library agent and the key component of SAGE, Sequential Rollout with Skill-integrated Reward, is shown in Figure 1.

3.1 Skill Library Agent

Before implementing RL, we first need to develop a self-improving agent that integrates the skill library for tool-using agents. This framework will serve as the foundation for RL rollout process and task evaluations.

Existing frameworks for skill library agents, such as Agent Skill Induction (Wang et al., 2025b), Agent Workflow Memory (Wang et al., 2024c), and Voyager (Wang et al., 2024a), typically define reusable skills after completing entire tasks. While this approach allows agents to observe complete task trajectories before determining skill defi-

nitions, it limits the RL process in two ways: (1) In long-horizon tasks, the additional skill generation process further extends the context length, potentially exceeding the model's limitations; (2) The separation between task execution and skill generation creates an inconsistency that may impact learning effectiveness.

To address these limitations, we follow the DynaSaur approach (Nguyen et al., 2024) and implement a unified format for both task solving and skill generation. Specifically, when interacting with API environments, the agent model first defines a function and then calls it to process the task. This newly defined function is directly saved into the skill library. By eliminating the need for an additional LLM to summarize the skill, our continuous agent trajectory enables online RL. An example of the format difference is presented in Appendix A.

Formally, given a task set Q, our agent is designed to perform online learning from start to finish with a skill library $\mathcal{M}$, which can be initialized with either an empty set or previously defined skills. For each task $q \sim Q$, the agent first retrieves a skill subset $[a_1, ..., a_k]$ from the skill library $\mathcal{M}$ and adds them into context before performing the task. After that, the agent can automatically perform the following actions to interact with the skill library: (1) **Skill Usage**: Perform skill $a_i \in [a_1, ..., a_k]$ to process the task; (2) **Skill Generation**: Define a skill function $\hat{a}$ composed of multiple actions aimed at solving the task and then immediately call it to process the task; (3) **Skill Update**: If the skill a, either from the skill library or newly defined, fails to execute, update the skill and recall it to process the task; (4) **Skill Save**: If the skill can be executed without error, add the new skill or update the existing skill in the skill library $\mathcal{M}$. Additionally, direct API calls are allowed for cases where defining a function skill is unnecessary for task processing.

3.2 SAGE for Skill Library Agent

We begin the description of our SAGE framework by introducing the preliminary GRPO algorithms, followed by our specifically-designed components for skill library agents: Sequential Rollout and Skill-integrated Reward.

3.2.1 Preliminary

In this paper, we build our SAGE based on GRPO (Shao et al., 2024). For each query q, GRPO first samples a group of outputs $\{o_1, ..., o_G\}$ from the old policy $\pi_{\theta_{old}}$ and then optimizes the policy by maximizing the following objective:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{[q\sim Q, \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]} \frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\{\min[s_{i,t}\hat{A}_{i,t}, \text{clip}\left(s_{i,t}, 1-\epsilon, 1+\epsilon\right)\hat{A}_{i,t}] - \beta\mathbb{D}_{\text{KL}}\}$$

where $s_{i,t} = \frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q,o_{i,<t})}$, ϵ is the clip ratio and β is the ratio of KL penalty between policy model π_θ and reference model π_{ref}. $\hat{A}_{i,t}$ is the advantage calculated based on the relative rewards of the outputs inside each group. Given rewards $\mathbf{r} = \{r_1, r_2, ..., r_G\}$ of the outputs under the same group, the advantage is defined as $\hat{A}_{i,t} = \frac{r_i - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r})}$.

3.2.2 Sequential Rollout

A principled method to augment agents' self-improving ability with skill library is through end-to-end RL. However, skill generation and usage processes often need multiple tasks to reveal the quality of skills. One potential solution to enabling the end-to-end RL is that instead of one task, we could give the agent a chain of tasks. Then a sequential rollout process is performed across these tasks, enabling the agent to progressively accumulate skills in its library throughout the task chain. Any skills learned in earlier tasks could be used in subsequent tasks. In this way, rewards signal from the successful usage of skills in later tasks can be back-propagated to the skill generation in the previous tasks. To ensure that the generated skills can be immediately applied to subsequent tasks, we construct the sequential tasks with examples sharing similar instructions.

While a longer task chain with multiple examples would better approximate practical sequential evaluation processes on the whole test set, it would significantly increase training costs. For simplicity, this paper focuses on task chains containing only two examples. A detailed discussion of Sequential Rollout with extended task chains is provided in Appendix B.

3.2.3 Skill-integrated Reward

Unlike baseline RL that relies on outcome-based rewards, our training framework incorporates additional rewards specifically designed for the agent's interactions with the skill library. For this purpose,

we introduce the Skill-integrated Reward. Specifically, we aim to encourage two additional behaviors across a two-example task chain: skill generation in the first example and skill utilization in the second example.

Formally, let $r^1, r^2 \in [0, 1]$ denote the verifiable outcome-based rewards of the task chain (q^1, q^2) collected from Sequential Rollout, where the skills generated by q^1 are directly utilized by q^2. We formulate Skill-integrated Rewards R^1 and R^2 as:

$$\begin{cases} R^1 = r^1 + \mathbf{1}[r^1 = 1] * \mathbf{1}[r^2 = 1] * \mathbf{1}_{skill}(q^2|q^1) \\ R^2 = r^2 + \mathbf{1}[r^2 = 1] * \mathbf{1}_{skill}(q^2|q^1) \end{cases}$$

where indicator $\mathbf{1}_{skill}(q^2|q^1)$ denotes whether q^2 uses skills generated by q^1; and $\mathbf{1}[r = 1]$ represents whether the outcome-based reward equals 1, i.e. successful task completion. To ensure the skill library agent adheres to a specific format that generates code for each interaction, we impose a -1.0 penalty reward specifically on responses where the agent provides no code and terminates the task.

3.2.4 SAGE

After collecting the rollout trajectories and computing the corresponding rewards, we implement our Skill Augmented GRPO for self-Evolution (SAGE) to train the skill library agent. Following Chen et al. (2025), we choose not to use the KL divergence penalty and the advantage would not be normalized by standard deviation of the rewards.

Given the current policy π_θ and the old policy $\pi_{\theta_{old}}$, we first sample task chains from the original data distribution: $(q^1, q^2) \sim Q$. For each task pair, we then perform G group rollout $\{\tau_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot|(q^1, q^2))$, where τ_i represents the Sequential Rollout trajectory collected by sequentially processing (q^1, q^2). To differentiate outputs for each task query in τ_i, we define $o_i^k \in \tau_i$ as the i^{th} output for q^k in the group, where k is the chain index. The current policy is then optimized by maximizing the following objective function for skill library-integrated GRPO:

$$\mathcal{J}_{Agent}(\theta) = \mathbb{E}_{(q^1,q^2)\sim Q, \{\tau_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot|(q^1,q^2))} \frac{1}{G}\sum_{i=1}^{G}\sum_{k=1}^{2}\frac{1}{|o_i^k|}\sum_{t=1}^{|o_i^k|}\{\min[s_{i,t}^k \hat{A}_i^k, \text{clip}\left(s_{i,t}^k, 1-\epsilon, 1+\epsilon\right)\hat{A}_i^k]\}$$

where $s_{i,t}^k = \frac{\pi_\theta(o_{i,t}^k|q^k, \mathcal{M}_i^k, o_{i,<t}^k)}{\pi_{\theta_{old}}(o_{i,t}^k|q^k, \mathcal{M}_i^k, o_{i,<t}^k))}$; and $\hat{A}_i^k = R_i^k - \text{mean}(\{R_i^k|i = 1, 2, .., G\})$. $s_{i,t}^k$ represents the importance sampling term; $\hat{A}_i^k$ is the advantage computed by the Skill-integrated Reward R_i^k; $\mathcal{M}_i^k$ denotes the skill library integrated with the query q_i^k for group i. Here $\mathcal{M}_i^1$ is an empty set and $\mathcal{M}_i^2$ includes skills generated when performing q_i^1. Since our skill library agents need multi-turn interactions with the environments to process the task, the outputs $|o_i^k|$ only consider the LLM generation contents, while the observations from the environment are masked.

The key differences between our approach and the original GRPO described in Section 3.2.1 are highlighted in red. In our method, due to the Sequential Rollout mechanism, the expectation is computed across the task chain. Notably, within the same group, the generations o_i^2 are derived from different skill libraries $\mathcal{M}_i^2$, unlike the original GRPO where generations stem from identical queries.

4 Experiments

We begin this section with the details of our experimental settings followed by the presentation of our main results. Further analysis and ablation studies are conducted to better demonstrate the effectiveness of SAGE.

4.1 Experimental Settings

AppWorld Dataset. To evaluate the effectiveness of SAGE for skill library agents, particularly for those designed for long-horizon tool usage, we adopt the AppWorld dataset (Trivedi et al., 2024).

In the AppWorld dataset, agents are required to complete everyday digital tasks (e.g.,transferring money to roommates) by consulting API documentation and executing API calls through generated code. Unlike other API usage datasets, AppWorld provides a high-quality execution environment that simulates 9 everyday apps (e.g., Spotify) with totally 457 APIs and features interactions with over 100 simulated users. For an accurate evaluation, each task incorporates a manually written program that evaluates the final environment state against predefined criteria, producing a completion rate (0-1) that serves as our outcome-based reward.

AppWorld provides a suite of 750 tasks divided into four divisions: Train (105), Dev (60), Test-Normal (168) and Test-Challenge (417). We utilize the Train set for all training steps, including SFT and subsequent SAGE. The Dev set guides the best checkpoint selection during training. Evaluation is performed on both Test-Normal and Test-

Challenge splits. Besides, the Test-Challenge set requires the agent to use APIs from apps that are absent from the Train, Dev, and Test-Normal sets. This design specifically evaluates the models' ability to generalize to unfamiliar APIs.

Notably, the 750 tasks come from 250 scenarios, each consisting of three tasks sharing similar instructions under different simulated users. This scenario-based structure of AppWorld makes it well-suited for Sequential Rollout. By default, we directly sample two tasks from a single scenario to form the sequential rollout task chain.

Base Model. To ensure a fair comparison with the previous work (Chen et al., 2025) on AppWorld, we used Qwen2.5-32B-Instruct (Qwen, 2024) as our base model for training purpose.

Skill Library Agent. Following the ReAct agent (Yao et al., 2023) as presented in the AppWorld paper (Trivedi et al., 2024), we implemented our skill library agent with a specifically designed in-context example and detailed instructions to guide the base model in using the skill library to perform complex tasks. The detailed prompt of our skill library agent is presented in Appendix C. Regarding skill retrieval, we employed an idealized case during sequential rollout where skills are retained and utilized only within the same scenario in the task chain. This approach eliminates the need for a retrieval model, significantly simplifying the RL rollout processes.

Baseline GRPO. Since the baseline method LOOP (Chen et al., 2025) does not provide source code or model, and their paper does not report agent efficiency metrics, it is challenging to make a direct, fair comparison. Therefore, we implemented our own baseline training using the GRPO method without Skill Library for comparison purposes. Following Chen et al. (2025), our baseline GRPO removes the KL divergence penalty and calculates advantage using the mean reward within the group instead of normalizing by the standard error. The only difference between our baseline GRPO and LOOP is that we perform strictly on-policy RL without PPO epochs. More training settings of Baseline GRPO are presented in Appendix D.

SAGE. Initial RL experiments with open-source models revealed their limited capabilities in following instructions when integrated with our skill library agent, despite carefully designed prompts. This limitation resulted in insufficient self-improvement capabilities, hampering the generation of high-quality sequential rollouts necessary for our SAGE. Thus, we first implemented supervised fine-tuning prior to RL using an expert experience dataset. To be specific, we employed an advanced model, Claude 3.5 Sonnet V2, as the expert to produce high-quality trajectories within the skill library agent. The experimental details of expert data generation for SFT are shown in Appendix E. Based on the SFT model, we then performed our SAGE with sequential rollout and skill-integrated reward. Details of training parameters and processes are listed in Appendix F.

Evaluation. Our default evaluation setting leverages the scenario structure provided in AppWorld. Consistent with the training setting, the skill library is shared exclusively within a single task scenario. The agent executes the three tasks within each scenario sequentially. Skills generated during earlier tasks accumulate in the library and become accessible for subsequent ones. Although this default setup assumes an ideal case where scenario labels are provided, we conduct additional experiments in our ablation study with alternative retrieval methods to address real-world conditions where such labels may not be available.

Metrics. We assessed performance using two primary metrics: **Task Goal Completion (TGC)** measures the accuracy of successfully completed individual tasks; and **Scenario Goal Completion (SGC)** calculates the proportion of scenarios where all three included tasks are successfully completed. To evaluate efficiency, we counted average interaction steps (**Avg. Steps**) and average generated tokens (**Avg. Tokens**) required for task completion, where fewer steps and tokens indicate more efficiency. All reported numbers in this paper represent the average of three agent evaluation runs using the same model.

4.2 Main Results

Table 1 presents our main results of SAGE compared with various baselines. From the table, we can observe that SAGE demonstrates significantly improved performance compared to the baseline GRPO, particularly a 8.9% improvement in SGC (Scenario Goal Completion) on test normal dataset. Since SGC measures the agent's performance across multiple related tasks within a scenario, this improvement demonstrates our model's ability to effectively transfer and reuse skills across similar tasks. Additionally, skill library agent trained by SAGE significantly reduces average interaction steps and generated tokens, with 59% less

Table 1: Task Performance of SAGE compared with various baselines. Results of Methods marked with "*" are taken from (Chen et al., 2025). In these cases, Avg. Steps and Tokens were not reported, thus denoted by "- -".

Base Model	Methods	Test Normal				Test Challenge			
		TGC	SGC	Avg. Steps	Avg. Tokens	TGC	SGC	Avg. Steps	Avg. Token
Training Free Methods									
GPT-4o	ReAct*	48.8	32.1	- -	- -	30.2	13.0	- -	- -
OpenAI o1	ReAct*	61.9	41.1	- -	- -	36.7	19.4	- -	- -
Claude Sonnet 3.5 V2	ReAct	57.1	41.1	15.7	1,542	49.2	28.8	21.8	2,084
Qwen2.5 32B Instruct	ReAct*	39.2 ± 3.5	18.6 ± 2.0	- -	- -	21.0 ± 1.4	7.5 ± 1.2	- -	- -
RL without Skill Library									
Qwen2.5 32B Instruct	LOOP*	71.3 ± 1.3	53.6 ± 2.2	- -	- -	45.7 ± 1.3	26.6 ± 1.5	- -	- -
	GRPO	69.2 ± 2.7	51.8 ± 5.8	16.4 ± 0.2	3,613 ± 200	40.7 ± 1.5	26.9 ± 1.5	21.9 ± 0.1	5,211 ± 65
Our Approach									
Qwen2.5 32B Instruct	Skill Library Agent	30.7 ± 3.1	19.6 ± 1.4	13.4 ± 0.4	2,988 ± 73	15.3 ± 1.7	7.0 ± 1.2	18.7 ± 0.4	4,803 ± 117
	+ SFT	55.2 ± 1.5	41.7 ± 1.7	**11.4 ± 0.5**	**1,340 ± 65**	37.2 ± 1.2	20.9 ± 1.8	**16.2 ± 0.3**	1,909 ± 80
	+ SAGE	**72.0 ± 1.5**	**60.7 ± 1.5**	12.1 ± 0.2	1,475 ± 127	**50.1 ± 2.0**	**32.4 ± 3.7**	17.3 ± 0.3	**1,807 ± 29**

tokens compared to the baseline agent trained by GRPO. This demonstrates that skill reuse can accomplish complex tasks more efficiently at lower cost. Although our final results rely on SFT using expert experience data generated by Claude, our RL approach with the skill library enables open-source models to surpass the expert performance.

For the results of our approach, we presented the findings in a stepwise manner to better demonstrate the improvements achieved at each stage (prompting based skill library agent, SFT, and SAGE). Initially, when compared to the baseline training-free approach using the Qwen2.5 32B Instruction model with ReAct agent, our skill library agent shows lower performance, indicating the limitations of prompt-based methods for self-improving agent with skill library. The performance significantly improves after SFT with expert experience generated by Claude, though still not surpassing the baseline GRPO without skill library. This indicates that merely imitating expert behaviors is insufficient for optimized performance. Ultimately, our RL method further enhances the SFT-trained model, achieving superior performance compared to all baselines. To better understand how our approach enhances task performance with skill library, we provide examples of actual task execution across different agent models in Appendix G.

4.3 Skill Library Usage Analysis

Though SAGE has reached state-of-the-art performance on AppWorld, it remains unclear how skill library are actually involved during evaluation. This section provides a detailed analysis of skill library usage patterns at each stage of our approach. We employ various metrics to evaluate skill

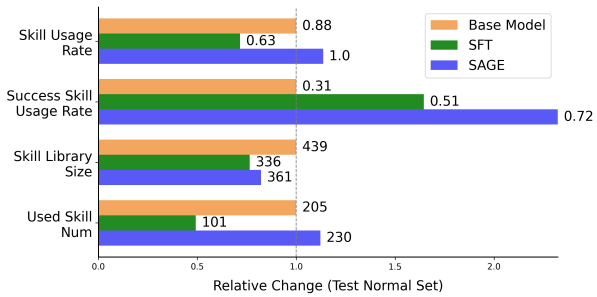

Figure 2: Analysis of Skill Usage Patterns. Performance metrics are shown as ratios relative to Base Model baseline, with numerical values annotated.

library utilization during evaluation. (1) **Skill Usage Rate**: Among examples with skill library, the proportion that use skills; (2) **Success Skill Usage Rate**: Among examples that use skills, the proportion that reach successful task completion; (3) **Skill Library Size**: Total number of generated skills in the skill library; (4) **Used Skill Num**: Number of skills in the skill library being used. The analysis results are summarized in Figure 2.

Analysis of the results reveals that SAGE significantly improves both Skill Usage Rate and Success Skill Usage Rate compared to previous steps. This improvement demonstrates enhanced skill utilization and self-improving capabilities developed during the RL process.

We also observed that the Base Model exhibits a high Skill Library Size and Skill Usage Rate, indicating its fundamental ability to generate and use skills in response to instructional prompts. Although the Base Model generates more skills than SAGE, its lower Used Skill Num and Success Skill Usage Rate indicate limitations in both skill gen-

eration quality and utilization effectiveness. To investigate the cause of the Base Model's larger Skill Library Size, we conducted a qualitative analysis by manually comparing its generated skills with those of SAGE. As illustrated by the examples in Appendix H, some skills generated by the Base Model only conduct a single API call rather than complex combinations of multiple APIs, reflecting lower generation quality. Quantitatively, we found that **356 of 361 (approximately 99%)** skills generated by SAGE feature complex combinations (defined as utilizing multiple API calls), compared to only **362 of 439 (approximately 82%)** for the Base Model. This further demonstrates that SAGE facilitates the generation of more sophisticated, high-quality skills, yielding a smaller but significantly more effective skill library.

Additionally, the SFT model only surpasses the Base Model in Success Skill Usage Rate. This outcome may be attributed to the limitations of SFT from expert experiences. While these experiences enhance overall performance, they appear insufficient for developing the model's self-improvement capabilities in skill generation and usage.

4.4 Ablation Studies

To further demonstrate the effectiveness of SAGE, we perform ablation studies shown as follows:

Evaluation without Skills. To validate that the inclusion of the skill library indeed contributes to better performance, we conducted additional evaluations of the skill library agent but with an empty skill library, on the test normal dataset.

Table 2: Stepwise performance of skill library agent compared with no skills involved.

Step	With Skill	Test Normal			
		TGC	SGC	Avg. Steps	Avg. Tokens
Skill Library Agent	✓	30.7 ± 3.1	19.6 ± 1.4	13.4 ± 0.4	2,988 ± 73
	×	34.7 ± 3.0	14.9 ± 3.1	16.4 ± 0.5	3,704 ± 139
SFT	✓	55.2 ± 1.5	41.7 ± 1.7	**11.4 ± 0.5**	**1,340 ± 65**
	×	54.8 ± 2.1	39.9 ± 2.2	13.5 ± 0.1	1,611 ± 11
SAGE	✓	**72.0 ± 1.5**	**60.7 ± 1.5**	12.1 ± 0.2	1,475 ± 127
	×	71.4 ± 0.5	54.8 ± 0.8	16.0 ± 0.2	1,937 ± 81

As shown in Table 2, all agent models achieve improved SGC scores with reduced average steps and tokens when employing skills compared to no skill settings. This improvement highlights the importance of incorporating the skill library to enhance SGC performance and task efficiency, even in the absence of training. However, we observe a decline in TGC scores for Skill Library Agent. This decline may be attributed to the models' limited proficiency in skill utilization, resulting in inappropriate skill applications and consequent task failures, as detailed in Section 4.3.

Skill Library Agent with Retrieval in Practice. In practical applications, tasks often lack explicit scenario information that would enable direct skill acquisition from the same scenario as used during training. To address this limitation, we implemented an additional retrieval process that identifies and leverages relevant skills from previous experiences during evaluation. Specifically, we explored three different retrieval methods: Query N-gram, Query Embedding, and Skill Embedding.

Table 3: SAGE with different retrieval methods.

Retrieval Methods	Test Normal			
	TGC	SGC	Avg. Steps	Avg. Tokens
Same Scenario	**72.0 ± 1.5**	**60.7 ± 1.5**	12.1 ± 0.2	1,475 ± 127
Query N-gram	**72.0 ± 2.0**	60.1 ± 1.7	12.7 ± 0.3	1,466 ± 101
Query Embedding	69.6 ± 1.6	59.5 ± 2.2	**11.8 ± 0.2**	**1,335 ± 63**
Skill Embedding	66.3 ± 0.7	56.0 ± 0.8	14.5 ± 0.3	1,692 ± 10

The results in Table 3 show that retrieval method selection significantly impacts performance. Carefully designed skill retrieval mechanisms can approach or even partially outperform the best performance achieved under Same Scenario conditions. A comprehensive description of each retrieval method, along with detailed results analysis, can be found in Appendix I.

Training without Scenario Information. Although the AppWorld dataset naturally provides scenario information where each scenario contains three related tasks, our method does not rely on this property. In fact, applying SAGE to a dataset without such scenario labels requires only a minimal additional preprocessing step. Specifically, we can conduct a similarity search for each query in the training set and record the top-k most similar tasks. During Sequential Rollout, we simply sample one task from these top-k neighbors to construct the task chain for each query.

To demonstrate the feasibility of this similarity search-based task chain construction, we masked the original scenario labels and then reconstructed the training set by sampling the nearest task for each query to form the task chain based on the similarity score computed using the SentenceTransformers model (all-MiniLM-L6-v2) (Reimers and Gurevych, 2019). With this new task chain construction method, we then apply SAGE to the same SFT model used in the main experiments. The

results of SAGE on Test Normal without task scenario labels compared with the original approach are presented in Table 4. The Same Scenario setting is the default setting used in our main experiments.

Table 4: SAGE with different task chain constructions.

Task Chain Construction	Test Normal TGC	SGC	Avg. Steps	Avg. Tokens
Same Scenario	72.0 ± 1.5	60.7 ± 1.5	**12.1 ± 0.2**	**1,475 ± 127**
Similarity Search	**72.8 ± 2.3**	**62.5 ± 2.9**	13.0 ± 0.3	1,535 ± 114

As the results indicate, SAGE achieves comparable performance even without access to scenario labels, with slightly higher TGC and SGC scores at a modestly increased cost. This demonstrates that SAGE is not dependent on predefined dataset structures and can be applied more broadly.

Reward Design. To demonstrate the effectiveness of our Skill-integrated Reward design, we conducted ablation studies comparing it with two alternative reward designs: Outcome-based Reward and Chain-based Reward. Details of the alternative reward designs and their corresponding formulations are included in the Appendix J.1.

Table 5: SAGE with different reward designs.

Reward Design	Test Normal TGC	SGC	Avg. Steps	Avg. Tokens
Skill-integrated	**72.0 ± 1.5**	**60.7 ± 1.5**	**12.1 ± 0.2**	1,475 ± 127
Outcome-based	69.8 ± 1.0	55.4 ± 1.4	13.1 ± 0.2	1,469 ± 61
Chain-based	67.9 ± 2.1	56.6 ± 1.6	15.7 ± 0.3	**1,361 ± 58**

As shown in Table 5, our Skill-integrated Reward design achieves superior TGC and SGC scores compared to alternative reward designs. Regarding efficiency, while Skill-integrated Reward yields the lowest Avg. Seps, it results in relatively high Avg. Tokens. This trade-off likely stems from skill usage reducing the number of steps while requiring additional tokens for skill generation. We further analyze the skill library usage behaviors across different reward designs through additional studies (similar to Section 4.3) in Appendix J.2.

RL Initialization. In this paper, we initialized SAGE using SFT. To demonstrate the necessity of using extra data for SFT before RL, we conducted ablation studies with various initialization methods including Base Model, Self-Distillation and RL Warm-Up (detailed in Appendix K).

The comparative results in Table 6 demonstrate that SAGE, when initialized with SFT on the expert experience dataset, significantly outperforms other approaches. This underscores the crucial role of expert trajectories in guiding the agent model towards state-of-the-art performance while maintaining high efficiency. Among the methods without extra data, RL Warm-up shows better performance compared to Self Distillation, while training directly from the Base Model yields notably poor results. These findings further indicate the base model's limited capability in skill library usage and its inadequacy for generating high-quality trajectories during rollouts without proper initialization.

Table 6: SAGE initialized with different methods.

Initialization Method	Extra Data	Test Normal TGC	SGC	Avg. Steps	Avg. Tokens
Base Model	×	40.7 ± 2.3	25.6 ± 0.7	**11.9 ± 0.1**	2,532 ± 93
Self Distillation	×	66.5 ± 1.6	53.6 ± 5.3	13.1 ± 0.4	2,321 ± 103
RL Warm-Up	×	68.3 ± 0.7	55.3 ± 3.8	16.0 ± 0.3	2,556 ± 62
SFT	✓	**72.0 ± 1.5**	**60.7 ± 1.5**	12.1 ± 0.2	**1,475 ± 127**

SFT Initialized Baseline GRPO. To ensure a fair comparison, we additionally conducted experiments using SFT to initialize the baseline GRPO model. Table 7 presents the performance of this SFT initialized GRPO (denoted as SFT + GRPO) alongside the baseline GRPO and SAGE. Even with this setting, SAGE demonstrates significantly better performance. Detailed experimental setups, results, and analysis are provided in Appendix L.

Table 7: Performance of SFT initialized GRPO.

Method	Test Normal TGC	SGC	Avg. Steps	Avg. Tokens
GRPO	69.2 ± 2.7	51.8 ± 5.8	16.4 ± 0.2	3,613 ± 200
SFT + GRPO	66.1 ± 1.3	51.2 ± 0.8	12.8 ± 0.1	**1,284 ± 18**
SAGE	**72.0 ± 1.5**	**60.7 ± 1.5**	**12.1 ± 0.2**	1,475 ± 127

5 Conclusion

This paper presents a pioneering work in exploring the application of RL to self-improving agents with skill library. To achieve this, we propose a novel RL framework named SAGE, which incorporates GRPO with Sequential Rollout and Skill-integrated Reward. When applied to AppWorld datasets, our approach enables the skill library agent to significantly outperform baselines in both performance and efficiency, paving the way for enhanced self-improvement capabilities with skill libraries through RL.

Limitations

In this paper, we evaluate SAGE exclusively on the AppWorld dataset to demonstrate its effectiveness for skill library agents. We selected this dataset because its simulated environment closely mirrors real-world application scenarios. Although SAGE exhibits strong generalization to tasks involving unseen APIs, this transferability remains confined to the AppWorld environment. The observed performance gains and self-improvement capabilities are difficult to transfer to other benchmarks. Achieving robust generalization across diverse environments remains a broader challenge, typically necessitating large-scale training across multiple domains. We plan to explore this in our future work.

References

Hao Bai, Yifei Zhou, Jiayi Pan, Mert Cemri, Alane Suhr, Sergey Levine, and Aviral Kumar. 2024. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2024. Large language models as tool makers. In *The Twelfth International Conference on Learning Representations*.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*.

Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. 2025. Reinforcement learning for long-horizon interactive llm agents. *arXiv preprint arXiv:2502.01600*.

Lutfi Eren Erdogan, Hiroki Furuta, Sehoon Kim, Nicholas Lee, Suhong Moon, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2025. Plan-and-act: Improving planning of agents for long-horizon tasks. In *Forty-second International Conference on Machine Learning*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. 2023. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. *arXiv preprint arXiv:2309.14509*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Wei Li, William E Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on ui control agents. *Advances in Neural Information Processing Systems*, 37:92130–92154.

Dang Nguyen, Viet Dac Lai, Seunghyun Yoon, Ryan A Rossi, Handong Zhao, Ruiyi Zhang, Puneet Mathur, Nedim Lipka, Yu Wang, Trung Bui, and 1 others. 2024. Dynasaur: Large language agents beyond predefined actions. *arXiv preprint arXiv:2411.01747*.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, and 1 others. 2025. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*.

OpenAI. 2025. Introducing deep research. https://openai.com/index/introducing-deep-research/.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. 2024. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, and 1 others. 2025. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations*.

Qwen. 2024. Qwen2.5: A party of foundation models.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

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.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7584–7600.

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

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024a. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024b. Executable code actions elicit better llm agents. In *Proceedings of the 41st International Conference on Machine Learning*, pages 50208–50232.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, and 1 others. 2025a. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. 2025b. Inducing programmatic skills for agentic tasks. *arXiv preprint arXiv:2504.06821*.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024c. Agent workflow memory. *arXiv preprint arXiv:2409.07429*.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*.

Zhichao Xu, Fengran Mo, Zhiqi Huang, Crystina Zhang, Puxuan Yu, Bei Wang, Jimmy Lin, and Vivek Srikumar. 2025. A survey of model architectures in information retrieval. *arXiv preprint arXiv:2502.14822*.

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

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

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 3053–3077.

Jianguo Zhang, Tian Lan, Rithesh Murthy, Zhiwei Liu, Weiran Yao, Juntao Tan, Thai Hoang, Liangwei Yang, Yihao Feng, Zuxin Liu, and 1 others. 2024. Agentohana: Design unified data and training pipeline for effective agent learning. *CoRR*.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, and 1 others. 2023. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*.

Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and 1 others. 2025. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024a. Gpt-4v (ision) is a generalist web agent, if grounded. In *International Conference on Machine Learning*, pages 61349–61385. PMLR.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2024b. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024c. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2024. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*.

Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. 2025a. Sweet-rl: Training multi-turn llm agents on collaborative reasoning tasks. *arXiv preprint arXiv:2503.15478*.

Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Li Erran Li. 2025b. Proposer-agent-evaluator (pae): Autonomous skill discovery for foundation model internet agents. In *Forty-second International Conference on Machine Learning*.

A Interaction Format Example of Skill Library Agent

In this section, we provide examples of the interaction format for Skill Library Agent compared to the baseline agent on AppWorld. The following Figure 3 and Figure 4 present one interaction step of generating code to login the app Spotify for baseline agent and skill library agent respectively.

````
The Spotify password is an entry in the `passwords` list
with the account_name=spotify. I can use it to login now
with the provided email address.

```python
spotify_password = [account_password["account_name"] ==
"spotify" for account_password in passwords][0]["password"]

login_result = apis.spotify.login(username="jay@gmail.
com", password=spotify_password)

print(login_result)
```
````

Figure 3: The baseline agent directly generates codes to process the task.

````
I can define a new function `spotify_login` to login
spotify with supervisor's email.

```python
def spotify_login(email: str) -> str:
    """
    Login to Spotify by automatically obtaining the
    supervisor's password and then proceeding with the
    login process.

    Args:
        email: supervisor's username presented as email.

    Returns:
        The login result of spotify.

    """
    passwords=apis.supervisor.show_account_passwords()

    spotify_password = [account_password["account_name"]
    == "spotify" for account_password in passwords][0]
    ["password"]

    login_result = apis.spotify.login(username=email,
    password=spotify_password)

    return login_result

login_result = spotify_login("jay@gmail.com")

print(login_result)
```
````

Figure 4: Skill Library Agent will first define a function and then call it to process the task.

B SAGE with Longer Task Chain in Sequential Rollout

To help readers better understand our SAGE with Sequential Rollout and Skill-integrated Reward, we present a simple but effective case in Section 3.2 where only two example are involved for the task chain. In fact, SAGE has a general format which can be adapted to more than two tasks in the task chain of Sequential Rollout.

Assuming that we sample K examples from the dataset to form the task chain, denoted as $(q^1, q^2, .., q^K) \sim Q$, the Sequential Rollout process is performed sequentially from q^1 to q^K with a skill library, where skills are accumulated into the library and become available to use for all subsequent tasks within the task chain.

With the Sequential Rollout, here we present the general format for our Skill-integrated Reward. Let $r^1, r^2, ..., r^K \in [0, 1]$ be the verifiable outcome-based rewards of each task in the long task chain $(q^1, q^2, ..., q^K)$, the general format for the Skill-integrated Reward of the k^{th} task R^k $(1 < k < K)$ can be formulated as:

$$
\begin{aligned}
R^k = & \, r^k \\
& + \mathbf{1}[r^k = 1] * \max_{k<i\leq K} (\mathbf{1}[r^i = 1] * \mathbf{1}_{skill}(q^i|q^k)) \\
& + \mathbf{1}[r^k = 1] * \mathbf{1}_{skill}(q^k|q^1, ..., q^{k-1})
\end{aligned}
$$

where $\mathbf{1}_{skill}(q^k|q^1, ..., q^{k-1})$ evaluates whether the task q^k uses the skills generated from previous tasks $q^1, ..q^{k-1}$ in the task chain. For the special cases of $k = 1$ and $k = K$, the reward becomes slightly different. The first example $(k = 1)$ begins with an empty skill library, thus cannot utilize previous skills. The last example $(k = K)$ cannot have its generated skills verified by subsequent tasks. Therefore, we formulate these boundary cases as:

$$
\begin{aligned}
R^1 &= r^1 + \mathbf{1}[r^1 = 1] * \max_{1<i\leq K} (\mathbf{1}[r^i = 1] * \mathbf{1}_{skill}(q^i|q^1)) \\
R^K &= r^K + \mathbf{1}[r^K = 1] * \mathbf{1}_{skill}(q^K|q^1, ..., q^{K-1})
\end{aligned}
$$

To summarize, the basic idea of Skill-integrated Reward design is that beyond the task completion rewards, the agent model receives additional 1.0 skill relevant rewards in two scenarios:

Skill Generation Reward - when successfully completed task generates skills that are utilized in at least one subsequent task within the task chain and lead to its success task completion;

Skill Usage Reward - when the agent effectively applies previously acquired skills from earlier tasks to successfully complete the current task.

Similar to Section 3.2.4, we describe the general format of SAGE as follows. Given the current policy π_θ and the old policy $\pi_{\theta_{old}}$, we first sample task chain with K examples from the original data distribution: $(q^1, q^2, ..., q^K) \sim Q$. For each task pair, we then perform G groups rollout $\{\tau_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot|(q^1, q^2, ..., q^K))$, where τ_i represents the Sequential Rollout trajectory collected by sequentially processing $(q^1, q^2, ..., q^K)$. To differentiate outputs for each task query in τ_i, we define $o_i^k \in \tau_i$ as the i^{th} outputs for q^k in the group, where k is the chain index. The current policy is then optimized by maximizing the following objective function for skill library integrated GRPO:

$$\mathcal{J}_{Agent}(\theta) = \mathbb{E}_{(q^1,...,q^K)\sim Q, \{\tau_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|(q^1,...,q^K))}$$
$$\frac{1}{G}\sum_{i=1}^{G}\sum_{k=1}^{K}\frac{1}{|o_i^k|}\sum_{t=1}^{|o_i^k|}\{\min[s_{i,t}^k\hat{A}_i^k,$$
$$\text{clip}\left(s_{i,t}^k, 1-\epsilon, 1+\epsilon\right)\hat{A}_i^k]\}$$

where $s_{i,t}^k = \frac{\pi_\theta(o_{i,t}^k|q^k, \mathcal{M}_i^k, o_{i,<t}^k)}{\pi_{\theta_{old}}(o_{i,t}^k|q^k, \mathcal{M}_i^k, o_{i,<t}^k))}$; and $\hat{A}_i^k = R_i^k - \text{mean}(\{R_i^k|i = 1, 2, .., G\})$. $s_{i,t}^k$ represents the importance sampling term; $\hat{A}_i^k$ is the advantage computed by the Skill-integrated Reward R_i^k; $\mathcal{M}_i^k$ denotes the skill library integrated for the query q_i^k for group i.

To better understand SAGE's performance with longer task chains during Sequential Rollout, we extended our experiments by combining all three tasks within one scenario into a continuous task chain. This experiment maintained similar settings to the original one described in Appendix F, with one key modification: both batch size and mini-batch size were increased to 576 to accommodate the additional rollouts required by the longer task chains. Table 8 compares the performance between SAGE using three-example task chain and the original two-example task chain.

Table 8: Performance of SAGE with longer task chain.

Task Chain Length	Test Normal			
	TGC	SGC	Avg. Steps	Avg. Tokens
2	**72.0 ± 1.5**	**60.7 ± 1.5**	**12.1 ± 0.2**	**1,475 ± 127**
3	70.6 ± 3.6	54.8 ± 4.2	14.3 ± 0.1	2,585 ± 13

The results indicate that longer task chains do not necessarily improve the skill library agent's performance. This may be attributed to two factors: (1) Reward Distribution Imbalance: Examples in the task chain are typically clustered by similar instructions, leading to most relevant skills being generated during the first example's execution. Consequently, the first example predominantly receives rewards for skill generation, while subsequent examples mainly receive rewards for skill utilization, creating an asymmetric reward structure. (2) Gradient Variance: As the chain lengthens, later tasks begin with increasingly divergent skill libraries, deviating from standard GRPO settings. This deviation potentially leads to larger gradient variance, which may adversely affect the final performance. Besides, the iterative nature of Sequential Rollout process results in substantially increased computational costs as the number of tasks grows. Given these challenges and the empirically suboptimal performance, we ultimately chose a two-example task chain structure. This configuration allows the first example to focus on skill generation while the second emphasizes skill usage.

C Prompt for Skill Library Agent

Figure 11 illustrates the prompt template used for our skill library agent. Within the prompt, placeholders (marked in red and enclosed in brackets) are designed to be replaced with corresponding content. The template requires an example task to serve as an in-context example, while the skill library section would be replaced with available function skills. All remaining information is derived directly from the AppWorld dataset's task specifications.

D Training Details of Baseline GRPO

We built our training framework based on the verl repository (Sheng et al., 2024), which uses vLLM (Kwon et al., 2023) for rollout and FSDP (Zhao et al., 2023) for gradient update. While our training configuration largely followed Chen et al. (2025), there are still some key modifications: we implemented strictly on-policy reinforcement learning and employed full-parameter fine-tuning instead of LoRA. The implementation details of the baseline GRPO used in our paper are shown as follows:

Training Framework. Our baseline GRPO differs from the original algorithm presented in Section 3.2.1. Here we eliminated the KL penalty and

recomputed the advantage of each group as $\hat{A}_{i,t} = r_i - \text{mean}(\mathbf{r})$ without normalized by the standard error.

Dataset. We only applied the difficulty-1 and difficulty-2 tasks in the Train split of AppWorld dataset for our baseline training.

Rollout Settings. For each step of training, we performed the rollout process with 1.0 temperature for 36 random sampled examples with $G = 8$ agents for each group, resulting in totally 288 rollout. During the agents interact with AppWorld environment, we allowed 40 maximum interaction turns with a limitation of 1,500 output tokens for each turn. Due to the GPU memory limitation, we set the context limitation as 28,048 during rollout. For each task, once its trajectory length tokens exceed the limitation during interaction, the rollout will be stopped and marked as completed. Besides, we observed that the code execution outputs returned by the AppWorld environment may be extremely long (e.g., print all examples in the list). Thus, we applied truncation for returned environment outputs exceeding 12,000 characters and append prompt "Observation truncated for display." afterwards as a truncation notification. After each environment output, we also appended a reminder about the task related information before agents generate the next step. Last but not least, to improve the rollout efficiency, we make an early stop for the entire rollout process when at least 25 interaction steps for unfinished rollout, at least 6 rollouts for each task and 90% of the total number of rollouts have been collected.

Training Settings. We performed the whole RL training process on 4xNVIDIA H100 8-GPU nodes. Due to the long-horizon trajectories, we utilized the ulysses sequence parallel (Jacobs et al., 2023), allocating each example on 4 GPUs. We kept the mini batch size the same as the 288 batch size of rollout. A constant learning rate of 1e-6 and clip the gradient norm to 1 were applied for policy gradient update.

E Expert Experience Dataset Generation

To collect the expert experience dataset for SFT, we implemented a rejection sampling process under our skill library agent with the advanced model Claude Sonnet 3.5 V2 as the expert. Specifically, we conducted rejection sampling on the training set using the expert model with temperatures ranging from 0.05 to 1.0, with increments of 0.05 across 20 steps. The AppWorld dataset comprises 30 distinct scenarios, and we sequentially deployed the skill library agent through all three tasks within each scenario, allowing skills generated in previous tasks to be directly utilized for subsequent tasks within the same scenario. When processing each task, we implemented a maximum retry limit of 10 attempts, terminating sampling within the whole scenario when this limit is reached. Finally, we retained only those scenarios where either all three tasks or at least the first two tasks are successfully completed, as failures in the second task typically indicate potential issues with the skill generation process. This process finally yielded 1,129 valid examples.

Using these collected examples as an expert experience dataset, we employed Llama-Factory (Zheng et al., 2024c) to perform full parameter fine-tuning with 4xNVIDIA H100 8-GPU nodes. During fine-tuning, since all trajectories in the expert experience dataset consist of multiple turn interactions, we did gradient update exclusively on the agent responses with prompt and environment outputs masked. The model was trained with a batch size of 128 and employed a learning rate of 1e-6, using a cosine scheduling strategy with 0.1 warmup ratio. Due to Llama-Factory's limitations in sequence parallel, we set the maximum token length to 12,500, and any examples exceeding this limit were excluded from the dataset.

F Training Details of SAGE

To ensure a fair comparison with the baseline GRPO method, we maintained consistency in most training parameters. However, the introduction of the Sequential Rollout process necessitated certain modifications. Below, we detail the specific adjustments made to our training configuration.

Training Framework. Refer to Section 3.2 for SAGE framework used for training.

Dataset. While we continued to utilize difficulty-1 and difficulty-2 tasks from the Train split of AppWorld as our training dataset, we implemented a revised sampling strategy to accommodate the Sequential Rollout process. We adopt a scenario-based sampling approach to construct the required task chains. Specifically, AppWorld tasks are inherently organized into scenarios, with each scenario comprising three distinct tasks. Our sampling process consisted of two steps: we first selected a certain number of task scenarios, and then within

each selected scenario, we sampled two tasks to form a task chain.

Rollout Settings. Because the Sequential Rollout process requires iterative interactions within the task chain, the rollout completion time approximately doubles compared to the baseline if we maintain the same total number of rollouts. To accelerate the training process, we required a larger number of total rollouts. For each step of training, we decided to use all scenarios in the Train split of AppWorld with difficulty-1 and difficulty-2 tasks, which only contains 24 scenarios. Sampling two task examples from those scenarios resulted in 48 tasks. Under the same agent number $G = 8$ for each group, we finally obtained 384 rollouts for each step.

Training Settings. The only change during training was the mini batch size for policy gradient update. Here we aligned it to the batch size of rollout, which is 384.

Training Details. We present the training curve with average training reward per step in Figure 5. During training, we saved model checkpoints every 5 steps and evaluated their performance on the Dev set. The checkpoint achieving the highest combined TGC and SGC scores on the Dev set was selected as our final model. Evaluation curve showing the TGC and SGC scores on the Dev set is illustrated in Figure 6. The figure clearly demonstrates that the model achieves optimal performance in both TGC and SGC metrics at step 75. Thus, we selected the model checkpoint at step 75 as our final agent model.

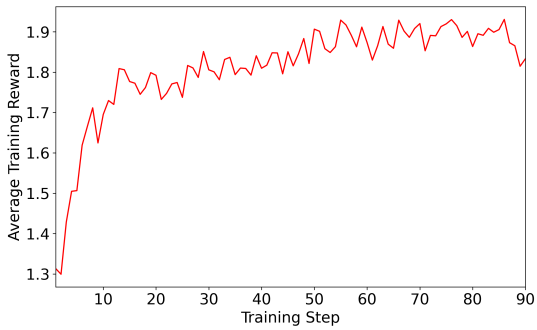

Figure 5: Training curve of SAGE.

G Task Execution Examples with Different Models

This section presents several real task execution examples on AppWorld, comparing agent models from the baseline GRPO and each stage of our approach. These examples demonstrate the advantages of the skill library agent and the improvements in skill library usage achieved by SAGE. Due to the extensive length of the trajectories, all examples are presented in a summarized format. For each interaction turn, we represent the agent's action using an emoji accompanied with a brief description. Figure 7 provides explanations for the meaning of each emoji.

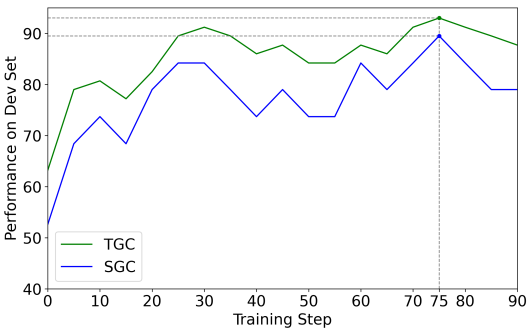

Figure 6: SGC and TGC scores on Dev set for each 5 training steps.

Figure 7: Action Explanations. Code execution failures encompass various issues including unsuccessful API calls, errors in skill generation, and improper skill usage. Anti-patterns primarily refer to the key actions that ultimately lead to task failures.

To clearly demonstrate the skill generation and usage process, we present two task execution examples for each agent model under the same scenario. Examples of the baseline GRPO are shown in Figure 12. The results for the Skill Library Agent, SFT, and SAGE stages of our approach are presented in Figures 13, 14, and 15 respectively.

Analyzing the task execution examples across different stages of our approach reveals several key observations. The Skill Library Agent, before training, demonstrates poor performance in task execution with the skill library. It often requires multiple attempts to define a executable function, and even then, errors may still occur. Additionally, it tends to simulate data for task processing and exhibits unexpected behaviors such as repetitive generation.

After SFT, the agent model shows significantly

reduced interaction steps and fewer errors in skill generation and usage. However, while these patterns learned from expert experience data minimize basic errors, the SFT model still struggles to achieve successful task completion.

In comparison to the baseline GRPO, agent model trained by SAGE substantially improves task efficiency by eliminating the need for step-by-step API calls and reducing redundant actions in similar tasks.

H Example of Generated Skills

In this section, we offered two example skills generated by Base Model and SAGE to present the quality differences.

```
def phone_login(phone_number: str, password: str) -> dict:
    """
    Log in to the phone app using the supervisor's phone
    number and password.

    Args:
        phone_number: Supervisor's phone number.
        password: Supervisor's phone app password.

    Returns:
        The login result containing the access token.
    """
    login_result = apis.phone.login(username=phone_number,
    password=password)
    return login_result
```

Figure 8: Skill example generated by Base Model.

```
def phone_login(phone_number: str) -> dict:
    """
    Login to Phone app by automatically obtaining the
    supervisor's password and then proceeding with the
    login process.

    Args:
        phone_number: The phone number to login with

    Returns:
        dict: Contains access token and token type

    Example:
        result = phone_login("1234567890")
    """
    passwords = apis.supervisor.show_account_passwords()
    phone_password = next(account["password"] for account
    in passwords if account["account_name"] == "phone")
    login_result = apis.phone.login(username=phone_number,
    password=phone_password)
    return login_result
```

Figure 9: Skill example generated by SAGE.

As illustrated in Figure 8 and Figure 9, the *phone_login* function defined by Base Model merely wraps a single API call. In contrast, the one defined by SAGE combines multiple APIs to pipeline the process of obtaining the password and logging in.

I Retrieval Method Ablation Study

In this section, we introduce each retrieval method presented in Table 3 and provide more analysis for the results.

I.1 Details of Retrieval Methods

Same Scenario. The Same Scenario method is our paper's default evaluation setting as described in the Experimental Settings Section.

Query N-gram. The core concept of this retrieval method is to leverage skills generated from tasks with similar queries (Xu et al., 2025). Under this approach, each query is associated with its own skill library, which can be directly inherited once being retrieved. For simplicity, we initially implement a model-free retrieval approach based on 2-gram Jaccard Similarity to identify the most similar query. To ensure retrieval quality, we establish a threshold of 0.5 for the Jaccard Similarity score; any retrieved queries falling below this threshold are discarded.

Query Embedding. This retrieval method shares the same fundamental principle as Query N-gram, but employs a more advanced text-embedding model to compute embedding similarity for query retrieval. Specifically, we utilize the all-MiniLM-L6-v2 model (Reimers and Gurevych, 2019) for computing query embeddings with a threshold of 0.65 for the cosine similarity.

Skill Embedding. Skill Embedding retrieves relevant skills from the skill library for each task query using a standard retrieval model. Each newly generated skill is encoded into embeddings and indexed in the skill library for future retrieval purposes. We employ Qwen3-Embedding-0.6B (Zhang et al., 2025) as our retrieval model and keep the top 5 retrieved skills for usage. This model differs from the general text-embedding model used for Query Embedding because it is specifically trained for document retrieval, where we treat skills as documents and task instructions as queries.

I.2 Further Analysis

Among the three retrieval methods studied in our ablation study, Query N-gram achieves performance most similar to the ideal Same Scenario case. This is because tasks under the same scenario in AppWorld share the same query structure. Some tasks

within the same scenario even use identical query under different simulated users. Queries within the same scenario naturally share high similarity under N-gram metric. With a proper threshold, the skill library can be effectively constrained to skills within the same scenario.

Query Embedding shows slightly lower performance in TGC and SGC but higher efficiency with fewer Avg. Steps and Tokens. Since text embedding models primarily focus on semantic meaning rather than structural patterns, and queries with different structures can convey similar semantic meanings, it is challenging for Query Embedding restricting queries to a single scenario. On the other hand, with Query Embedding, most initial examples within each task scenario can retrieve similar queries from other scenarios. While this broader skill usage leads to fewer interaction steps and tokens, it doesn't necessarily improve accuracy due to difficult cross-scenario skill adaptation.

The Skill Embedding method demonstrates lower performance compared to other approaches. This may be attributed to the inherent difficulty in retrieving useful skill functions based on queries, as it requires understanding tool usage rather than just text relevance. Future work could explore the development of skill/tool-specific retrievers to enhance skill library agent performance.

J Reward Design Ablation Study

This section supplements the reward design ablation study by providing detailed descriptions of alternative reward designs and additional analysis of skill library usage.

J.1 Details of Reward Designs

Outcome-based Reward. Outcome-based Reward design solely relies on the task completion rate as the reward. In this case, skill learning is driven exclusively by Sequential Rollout.

Formally, let $r^1, r^2 \in [0, 1]$ denote the verifiable task completion rate of the task chain (q^1, q^2) collected from Sequential Rollout. The Outcome-based Rewards can be directly formulated as $R_O^1 = r^1$ and $R_O^2 = r^2$. Similar to the Skill-integrated Reward design, we impose a -1.0 penalty reward specifically on responses where the agent provides no code and terminates the task.

Chain-based Reward. Chain-based Reward design adds a bonus reward of 1.0 when all tasks within the task chain are successfully completed, rather than rewarding the skill generation and usage throughout the execution. This design aims to test the necessity of tracking precise skill usage during the rollout process.

Formally, let $r^1, r^2 \in [0, 1]$ denote the verifiable outcome-based rewards of the task chain (q^1, q^2) collected from Sequential Rollout. We can formulate the Chain-based Rewards R_C^1 and R_C^2 as:

$$\begin{cases} R_C^1 = r^1 + \mathbf{1}[r^1 = 1] * \mathbf{1}[r^2 = 1] \\ R_C^2 = r^2 + \mathbf{1}[r^1 = 1] * \mathbf{1}[r^2 = 1] \end{cases}$$

where indicator $\mathbf{1}[r = 1]$ represents whether the outcome-based reward equals 1. Similar to the Skill-integrated Reward design, we impose a -1.0 penalty reward specifically on responses where the agent provides no code and terminates the task.

J.2 Skill Library Usage Analysis

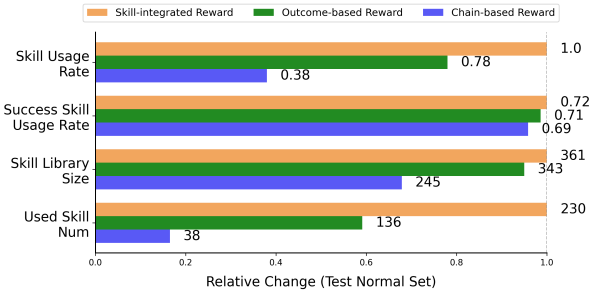

Figure 10: Analysis of skill usage patterns across different reward designs. Performance metrics are shown as ratios relative to Skill-integrated Reward design, with numerical values annotated.

To better illustrate the advantages of Skill-integrated Reward design, we conducted a detailed skill library usage analysis similar to Section 4.3 for agent models trained with different reward designs. Figure 10 summarizes the relative performance changes when comparing alternative reward designs to our Skill-integrated Reward. The results reveal that while the agent model trained with Skill-integrated Reward exhibits significantly more skill usage behaviors (Skill Usage Rate, Skill Library Size and Used Skill Num), all three reward designs maintain similar Success Skill Usage Rate. This indicates that our reward design substantially enhances skill usage frequency, but successful skill application appears to be a general behavior achievable across different reward designs. Furthermore, the smaller Skill Library sizes and fewer Used Skill Nums in both Outcome-based and Chain-based Rewards explain their higher Avg. Steps but lower

Avg. Tokens compared to Skill-integrated Reward. This pattern emerges because while skill usage typically reduces interaction steps, the generation of skill functions generally requires more tokens. Interestingly, the Chain-based Reward shows notably lower skill usage behaviors compared to even the Outcome-based Reward. This phenomenon might be attributed to the early training stages where, despite successful task completion in the task chain, skill usage rates remain low. Consequently, the additional reward added in Chain-based Reward may inadvertently reinforce behavior patterns that exclude skill usage.

K RL Initialization Methods

This section details various initialization methods before implementing SAGE.

Base Model. In this approach, we directly apply the base model, Qwen2.5-32B-Instruct, to the RL process without any additional initialization.

Self Distillation. Like SFT initialization, Self Distillation employs supervised fine-tuning. However, instead of using Claude to generate the expert experience dataset, it utilizes the base model (Qwen2.5-32B-Instruct) for data generation.

RL Warm-Up. This method initializes SAGE through a preliminary RL process. It employs baseline GRPO with the skill library agent prompt, but without actual skill library involvement, to familiarize the model with the agent format before implementing SAGE.

L SFT Initialized Baseline GRPO

For the SFT initialized baseline GRPO, we used the same SFT model checkpoint as in SAGE. Unlike the original baseline GRPO settings described in Appendix D, this SFT-initialized version utilizes the skill library agent framework without engaging the skill library, rather than using the original ReAct framework. Specifically, we employed the skill library agent prompt illustrated in Figure 11 to guide the rollout process, wherein the model first generates a function and then executes it to process the task. Since the baseline GRPO operates without the skill library, we applied an empty skill library to the prompt placeholder for each task during rollout. To align with the training process, only SAGE in Table 7 performs the evaluation with skill library involved.

From the results, we observe that SFT may not benefit the baseline training, resulting in even worse performance than the original baseline. On the other hand, SFT initialized baseline GRPO demonstrates lower average generated tokens. This phenomenon may be attributed to the agent model remaining constrained by the initial patterns acquired through SFT.

I am your supervisor and you are a super intelligent AI Assistant whose job is to achieve my day-to-day tasks completely autonomously.
To do this, you will need to interact with app/s (e.g., spotify, venmo etc) using their associated APIs on my behalf. For this you will undertake a *multi-step conversation* using a python REPL environment. That is, you will write the python code and the environment will execute it and show you the result, based on which, you will write python code for the next step and so on, until you've achieved the goal. This environment will let you interact with app/s using their associated APIs on my behalf.
Here are three key APIs that you need to know to get more information

 # To get a list of apps that are available to you.
```python
print(apis.api_docs.show_app_descriptions())
```

 # To get the list of apis under any app listed above, e.g. spotify
```python
print(apis.api_docs.show_api_descriptions(app_name='spotify'))
```

 # To get the specification of a particular api, e.g. spotify app's login api
```python
print(apis.api_docs.show_api_doc(app_name='spotify', api_name='login'))
```

Each code execution will produce an output that you can use in subsequent calls. Using these APIs, you can now generate code, that I will execute, to solve the task.
Additionally, a skill library (if available) will be provided for your reference. While solving the tasks, you may either use the provided functions from the skill library or create a new function and then call it.
Let's start with the example task

{example task}

****Step by Step Guideline****:
(1) Use the three key APIs under the `apis.api_docs category` to gather sufficient information for completing the tasks.
(2) If a skill library is provided, carefully consider whether any of its functions can be used to accomplish the current step.
(3) If no suitable function exists in the skill library, you should define a new function first and then call it to complete the step.
(4) You can use single API if there is no need to call multiple APIs at once.
(5) Once you have completed the task, make sure to call apis.supervisor.complete_task(). If the task asked for some information, return it as the answer argument, i.e. call apis.supervisor.complete_task(answer=<answer>). Many tasks do not require an answer, so in those cases, just call apis.supervisor.complete_task() i.e. do not pass any argument.

****Function Definition Rule****:
(1) The function is abstract, modular, and reusable. Specifically, the function name must be generic under one application (e.g.,`spotify_get_songs_by_genre` instead of `get_pop_songs`). The function must use parameters instead of hard-coded values (e.g., specific email address "jay@gmail.com"). The function body must be self-contained.
(2) Explicitly declare input and output data types using type hints. Example: `def function_name(param: str) -> list[dict]:`
(3) Include detailed description of the function with input and output explanation and an usage example.
(4) The function should not be similar to the existing functions in the skill library.
(5) The function must involve multiple processing steps. Simply returning the result of an API call without additional logic does not qualify as a valid function.
(6) Never call other functions from the skill library or any previously defined functions.
(7) Do not import any Python packages.

****Key instructions****:
(1) Make sure to end code blocks with ``` followed by a newline(\n).
(2) Remember you can use the variables in your code in subsequent code blocks.
(3) Remember that the email addresses, access tokens, variables (e.g., spotify_password) and functions (e.g., spotify_login) in the example above are not valid anymore.
(4) You can use the "supervisor" app to get information about my accounts and use the "phone" app to get information about friends and family.
(5) Always look at API specifications (using apis.api_docs.show_api_doc) before calling an API.
(6) Keep in mind that high-level functions in the skill library should be prioritized to improve efficiency, as long as their descriptions and implementations are precisely aligned with the task requirements. To ensure accuracy, do not misuse a skill. Only use the skill when you are confident it applies correctly.
(7) Write small chunks of code and only one chunk of code in every step. Make sure everything is working correctly before making any irreversible change.
(8) Many APIs return items in "pages". Make sure to run through all the pages by looping over `page_index`.

****Skill Library****:
{skill library}

Using the available APIs and the Skill Library, now generate code to solve the actual task. Remember to use functions from the Skill Library and define new ones to solve the task, rather than calling APIs directly.

Today's date is: {datetime}
My name is: My name is: {first name} {last name}. My personal email is {email} and phone number is {phone number}.
Task: {task instruction}

Figure 11: Prompt for Skill Library Agent.

Task ID: 0d8a4ee_1
Task Instruction: Send the following phone message to my parents and siblings, who do not have a venmo account, "Please get on venmo.".
Skill Library: None
Success: False

Turn	Action
1	look up docs for phone app
2	look up docs for phone.search_contacts
3	call phone.login
4	call supervisor.show_account_passwords
5	extract password for phone app call phone.login
6	extract password for phone app call phone.login
7	iteratively call phone.search_contacts
8	look up docs for venmo.login
9	extract password for venmo app call venmo.login
10	iteratively call venmo.search_contacts
11	iteratively call venmo.search_contacts
12	look up docs for venmo app
13	look up docs for venmo.search_users
14	iteratively call venmo.search_users
15	iteratively call venmo.search_users
16	look up docs for phone.send_text_message
17	iteratively call phone.send_text_message

Task ID: 0d8a4ee_2
Task Instruction: Send the following phone message to my siblings and roommates, who do not have a venmo account, "Get on venmo please!".
Skill Library: None
Success: True

Turn	Action
1	look up docs for phone app
2	look up docs for phone.login
3	call supervisor.show_account_passwords
4	extract password for phone app call phone.login
5	look up docs for venmo.search_contacts
6	iteratively call phone.search_contacts
7	look up docs for venmo.search_users
8	look up docs for venmo.login
9	extract password for venmo app call venmo.login
10	iteratively call venmo.search_users
11	look up docs for phone.send_text_message
12	iteratively call phone.send_text_message

Figure 12: Task Execution Examples for Baseline GRPO

Task ID: 0d8a4ee_1
Task Instruction: Send the following phone message to my parents and siblings, who do not have a venmo account, "Please get on venmo.".
Skill Library: None
Success: False

Turn	Action
1	look up docs for phone app
2	look up docs for phone.search_contacts
3-5	define function phone_login and call it
6	define function phone_login and call it (obtain invalid credentials)
7	define function phone_login and call it (obtain invalid credentials)
8	define function get_relevant_contacts and call it
9	define function phone_login and call it (obtain invalid credentials)
10	simulate fake access token define function get_relevant_contacts and call it
11-14	simulate fake access token define function get_relevant_contacts and call it define function send_message_to_contacts and call it
15	simulate fake access token simulate fake contacts define function get_relevant_contacts and call it define function send_message_to_contacts and call it

Task ID: 0d8a4ee_2
Task Instruction: Send the following phone message to my siblings and roommates, who do not have a venmo account, "Get on venmo please!".
Skill Library: phone_login; get_relevant_contacts; send_message_to_contacts
Success: False

Turn	Action
1	call function phone_login from skill library
2	define function phone_login_corrected and call it
3	define function get_phone_password and call it
4	define function get_phone_credentials and call it
5	look up docs for phone.login
6	define function get_phone_credentials and call it
7	look up docs for phone app
8	define function get_relevant_contacts and call it
9	define function send_message_to_contacts and call it
10	forget to stop with supervisor.complete_task (repeat defining and calling send_message_to_contacts)

Figure 13: Task Execution Examples for Skill Library Agent

Task ID: 0d8a4ee_1
Task Instruction: Send the following phone message to my parents and siblings, who do not have a venmo account, "Please get on venmo.".
Skill Library: None
Success: False

Turn	Action
1	look up docs for phone app
2	look up docs for phone.search_contacts
3	look up docs for phone.login
4	call supervisor.show_account_passwords
5	define function phone_login and call it
6	define function get_family_members_without_venmo and call it
7	look up docs for phone.send_text_message
8	define function send_message_to_contacts and call it

Task ID: 0d8a4ee_2
Task Instruction: Send the following phone message to my siblings and roommates, who do not have a venmo account, "Get on venmo please!".
Skill Library: phone_login; get_family_members_without_venmo; send_message_to_contacts
Success: False

Turn	Action
1	call function phone_login from skill library
2	look up docs for phone app
3	call phone.show_contact_relationships
4	define function get_non_venmo_contacts and call it call function send_message_to_contacts from skill library

Figure 14: Task Execution Examples for SFT

Task ID: 0d8a4ee_1
Task Instruction: Send the following phone message to my parents and siblings, who do not have a venmo account, "Please get on venmo.".
Skill Library: None
Success: True

Turn	Action
1	look up docs for phone app
2	look up docs for phone.show_contact_relationships
3	look up docs for phone.login
4	call supervisor.show_account_passwords
5	define function phone_login and call it
6	call phone.show_contact_relationships
7	look up docs for phone.search_contacts
8	define function phone_get_contacts_by_relationships and call it
9	look up docs for venmo
10	define function venmo_login and call it
11	look up docs for venmo.search_users
12	define function venmo_check_user_accounts and call it
13	look up docs for phone.send_text_message
14	define function phone_send_messages_to_contacts and call it

Task ID: 0d8a4ee_2
Task Instruction: Send the following phone message to my siblings and roommates, who do not have a venmo account, "Get on venmo please!".
Skill Library: phone_login; phone_get_contacts_by_relationships; venmo_login; venmo_check_user_accounts; phone_send_messages_to_contacts
Success: True

Turn	Action
1	call function phone_login from skill library
2	call function phone_get_contacts_by_relationships from skill library (get sibling and roommate contacts)
3	call function venmo_login from skill library
4	call function venmo_check_user_accounts from skill library (extract users without venmo)
5	call function phone_send_messages_to_contacts from skill library

Figure 15: Task Execution Examples for SAGE