SEARL: Joint Optimization of Policy and Tool Graph Memory for Self-Evolving Agents

Xinshun Feng[1*] **Xinhao Song**[1,2*] **Lijun Li**[1*†]
Gongshen Liu[2] **Jing Shao**[1†]
[1]Shanghai Artificial Intelligence Laboratory [2]Shanghai Jiaotong University
{fengxinshun, songxinhao, lilijun, shaojing}@pjlab.org.cn
lgshen@sjtu.edu.cn

Abstract

Recent advances in Reinforcement Learning with Verifiable Rewards (RLVR) have demonstrated significant potential in single-turn reasoning tasks. With the paradigm shift toward self-evolving agentic learning, models are increasingly expected to learn from trajectories by synthesizing tools or accumulating explicit experiences. However, prevailing methods typically rely on large-scale LLMs or multi-agent frameworks, which hinder their deployment in resource-constrained environments. The inherent sparsity of outcome-based rewards also poses a substantial challenge, as agents typically receive feedback only upon completion of tasks. To address these limitations, we introduce a `Tool-Memory` based self-evolving agentic framework **SEARL**. Unlike approaches that directly utilize interaction experiences, our method constructs a structured experience memory that integrates planning with execution. This provides a novel state abstraction that facilitates generalization across analogous contexts, such as tool reuse. Consequently, agents extract explicit knowledge from historical data while leveraging inter-trajectory correlations to densify reward signals. We evaluate our framework on knowledge reasoning and mathematics tasks, demonstrating its effectiveness in achieving more practical and efficient learning[1].

1 Introduction

Recent advances in benchmarking agentic capabilities, such as GAIA (Mialon et al., 2023), Humanity's Last Exam (HLE) (Phan et al., 2025), and WebArena (Zhou et al., 2023), have revealed that direct prompting for large language models (LLMs) remains insufficient for solving complex, long-horizon tasks (Hu et al., 2025b; Lu et al., 2025). A central challenge in this paradigm is determining the tool set available to the agent: the availability of appropriate tools is often a prerequisite for task completion. Most existing frameworks (Yao et al., 2023) adopt a static design, predefining a large, fixed tool list from which the agent must select and invoke tools, potentially limiting adaptability and generalization across diverse tasks.

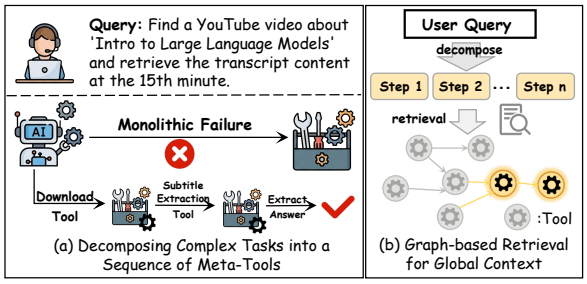

Figure 1: Limitations of existing paradigms. Monolithic tool synthesis overwhelms LLMs, necessitating a divide-and-conquer strategy, while current memory designs suffer from a lack of structural connectivity.

Recent works typically address this challenge from two perspectives. First, tool-generation methods like Alita (Qiu et al., 2025) and STELLA (Jin et al., 2025b) autonomously create tools but store them in unstructured repositories. This lack of structure leads to high-level abstractions, limiting both reusability and fine-grained composition. Furthermore, given the limited reasoning capabilities of smaller-scale LLMs, generating monolithic tools often results in failure; consequently, decomposing complex tasks into subtasks and creating corresponding tools proves to be a more effective strategy. Second, RL-based or experience-driven approaches (Tang et al., 2025a) leverage past trial data but often overlook the explicit dependency relations essential for complex reasoning. To bridge these gaps, we propose the `Tool Graph`, a structured memory where tools serve as nodes and execution dependencies as edges (Figure 1). This graph evolves continuously, providing strong inductive

[*]Equal contribution.
[†]Corresponding Author
[1]Code available at https://github.com/circles-post/SEARL

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 24518–24535

biases to improve generalization and planning.

While agentic reinforcement learning (Singh et al., 2025) has emerged as a prominent paradigm, it faces significant limitations. First, their reward designs typically prioritize trajectory-level success or format correctness, largely neglecting step-level feedback on reasoning quality. Although some methods incorporate process-level rewards, they often rely on heuristic designs or are tailored to specific domains (Feng et al., 2025), which restricts their applicability in general scenarios. Second, they focus on improving the intricacies of the models, ignoring the potential of expanding external memory to achieve long-term improvement. Motivated by these gaps, we propose SEARL, a reinforcement learning framework in which both the policy model and the tool-based memory evolve jointly during training.

Our framework enables the joint evolution of the agent's memory and policy, surpassing methods that optimize either component in isolation. It comprises two key components: (i) An environment augmented with a dynamic `Tool Graph` that supports continuous tool creation and reuse, serving as an evolving structured memory that progressively accumulates problem-solving capabilities. (ii) A tool-memory-aware policy optimization algorithm fine-tuned via trajectory- and step-level credits, which adapts the LLM to effectively navigate and leverage the growing graph structure. Through this joint optimization, the agent exhibits *self-evolution*, becoming increasingly competent as it encounters and solves more complex tasks. In summary, the key contributions of this work are as follows:

- We introduce **SEARL**, a new paradigm that jointly optimizes both policy parameters and external tool-based memory, enabling agents to continuously acquire, refine, and reuse problem-solving capabilities.
- We propose a `Tool-Memory-Aware` training algorithm that extends step-level advantages with memory-anchored clustering, providing fine-grained credit assignment for tool creation and execution.
- We formalize the `Tool Graph Memory` as a structured, persistent representation of tool knowledge and inter-tool dependencies, and show that its growth improves generalization and planning in complex tasks.

Experimental results validate the effectiveness of our method, showing that it empowers small LLMs with robust self-evolving capabilities driven by an ever-expanding tool memory.

2 Preliminary

Definition 1 (Self-Evolving Agent)**.** *A self-evolving agent improves its problem-solving capabilities through experience. This evolution typically follows two paradigms:* ***tool-based evolution****, which optimizes a set of tools $\mathcal{T}$ for specific tasks, and* ***memory-based evolution****, which leverages a repository of past successful trajectories $\mathcal{M}$. Ideally, this continuous adaptation enables the agent to generalize its reasoning processes to novel, unseen states without requiring manual retraining.*

Definition 2 (Tool Memory-Enhanced MDP)**.** *We define a Tool Memory-Enhanced Markov Decision Process as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{T}_G \rangle$. Here, $\mathcal{S}$ denotes the state space, $\mathcal{A}$ the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ the state transition dynamics, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function. Crucially, $\mathcal{T}_G = (V, E)$ represents the graph-structured tool memory, where $V \subset \mathcal{S} \times \mathcal{A} \times \mathbb{R}$ denotes the set of stored interaction tuples, and E encodes correlations derived from trajectories.*

Problem Formulation. Given a task description x drawn from distribution $\mathcal{D}$ at time step t, the agent observes state $s_t \in \mathcal{S}$ and generates a textual action a_t, and transitions to s_{t+1}. A complete interaction trajectory is denoted as $\tau = \{(s_i, a_i, r_i)\}_i^N$. The agent operates under an LLM-based policy $\pi_\theta(a_t | s_t, x, \mathcal{T}_G)$, parameterized by θ. The training objective seeks to maximize the expected reward while regularizing deviation from a reference policy π_{ref} via KL divergence:

$$\begin{aligned} &\max_{\pi_\theta} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_\theta(\cdot|x;\mathcal{T}_G)} \left[r_\phi(x, y) \right] - \\ &\beta \, \mathbb{D}_{\text{KL}} \left[\pi_\theta(y \mid x; \mathcal{T}_G) \,\|\, \pi_{\text{ref}}(y \mid x; \mathcal{T}_G) \right], \end{aligned} \tag{1}$$

The trajectory generation process $p(\tau)$ is decomposed into four sequential phases: (1) **Retrieve**: A retrieval function μ selects the most relevant tool T_t from memory $\mathcal{T}_G$; (2) **Reuse**: The agent decides whether to utilize the retrieved tool for the current action; (3) **Creation**: The agent determines if a new, universal tool should be defined and added to memory; and (4) **Transition**: The environment evolves to the next state based on the dynamics.

Formally, this decomposition is expressed as:

$$p(\tau) = \prod_{t=0}^{N-1} \underbrace{\mu(T_t|s_t, \mathcal{T}_G)}_{\text{(1)Retrieve}} \underbrace{p_{LLM}(a_t|s_t, T_t)}_{\text{(2)Reuse}} \times \underbrace{p_{LLM}(T_{t+1}|a_t, T_t)}_{\text{(3)Creation}} \underbrace{\mathcal{P}(s_{t+1}|s_t, a_t)}_{\text{(4)Transition}} \quad (2)$$

where N denotes the maximum steps, and T_t is the pertinent tool retrieved from $\mathcal{T}_G$.

3 Self-Evolving Agent Reinforcement Learning

This section details the proposed method, including the overall workflow, the design of the reward function, the advantage estimation and policy optimization algorithm, and the tool-based memory structure for retrieving and reusing relevant tools.

3.1 Structured Trajectory Generation

Our approach formalizes the decision-making process as a structured sequence of meta-reasoning stages, explicitly annotated to enable fine-grained control. Initially, a global plan is generated to delineate the high-level strategy and the execution order of subtasks. Departing from methods that treat subtask execution as a monolithic opaque block, we decompose the process into distinct phases encompassing tool retrieval, reasoning, and external actions. Specifically, we define four step-level components, each delimited by XML-style tags (e.g., `<tool_call>`), with the entire sequence encapsulated within a `<subtask>` tag.

- **Planning**: Decomposes the task into high-level steps to formulate an overall strategy at the start of a task.
- **Retrieve**: Selects the most relevant tool T_t from the tool-based memory $\mathcal{T}_G$ using a retrieval policy $\mu(T_t \mid s_t, \mathcal{T}_G)$, providing context for the upcoming action.
- **Think**: Performs internal deliberation conditioned on the state, deciding whether to use a specific tool or produce a direct answer.
- **Action**: Generates an output, which can be either an `<answer>` (a textual response) or a `<tool_call>` command for code execution. To simplify tool reuse and unify the creation process, the MCP tool creation is also defined as a regular tool calling.

3.2 Reward Shaping

During reinforcement learning, the agent is guided by a composite reward signal that combines task completion with feedback on the agent's interaction with tool-based memory. The overall signal consists of a sparse outcome reward that reflects whether the final answer solves the task, and a dense, process-level reward tied to planning, tool usage, and code execution.

Outcome Reward ($R(\tau)$). A binary signal awarded at the conclusion of a trajectory. Specifically, $R(\tau) = r_s$ if the task is successfully completed with the correct answer, and 0 otherwise. In our experimental setting, the constant r_s is set to 1.

Behavioral Reward (r_t^{τ}) We use a dense reward, assigned at each step t, to incentivize locally beneficial behaviors. To guide the agent's behavior toward a desired direction, we design distinct rewards for different actions.

- **Planning Reward (r_{planning})**: Awarded if the generated plan can be successfully parsed into a complete sequence of subtasks with executable subplans.
- **Tool Creation Reward (r_{creation})**: Granted when the agent creates a new MCP tool that conforms to the required registration format, encouraging a meaningful extension of tool-based memory $\mathcal{T}_G$.
- **Tool Execution Reward ($r_{\text{execution}}$)**: The tool call generated is awarded when it executes successfully and returns a valid output, promoting reliable and verifiable behavior.

Format Reward (r_t^{format}). A positive reward $+\lambda_{\text{format}}$ is granted at step t if the model output conforms to the required structure.

3.3 Advantage Estimation with Tool-Memory-Aware Policy Optimization

While group-based RL has proven effective for single-turn tasks, extending it to multi-step agent settings faces significant credit assignment challenges. Existing step-level approaches (Feng et al., 2025) attempt to mitigate this by grouping identical states. However, this relies heavily on frequent state re-visitation (e.g., in GUI environments). In more general, open-ended environments, the state space is vast and continuous, rendering such precise state matching impractical. Compounding this

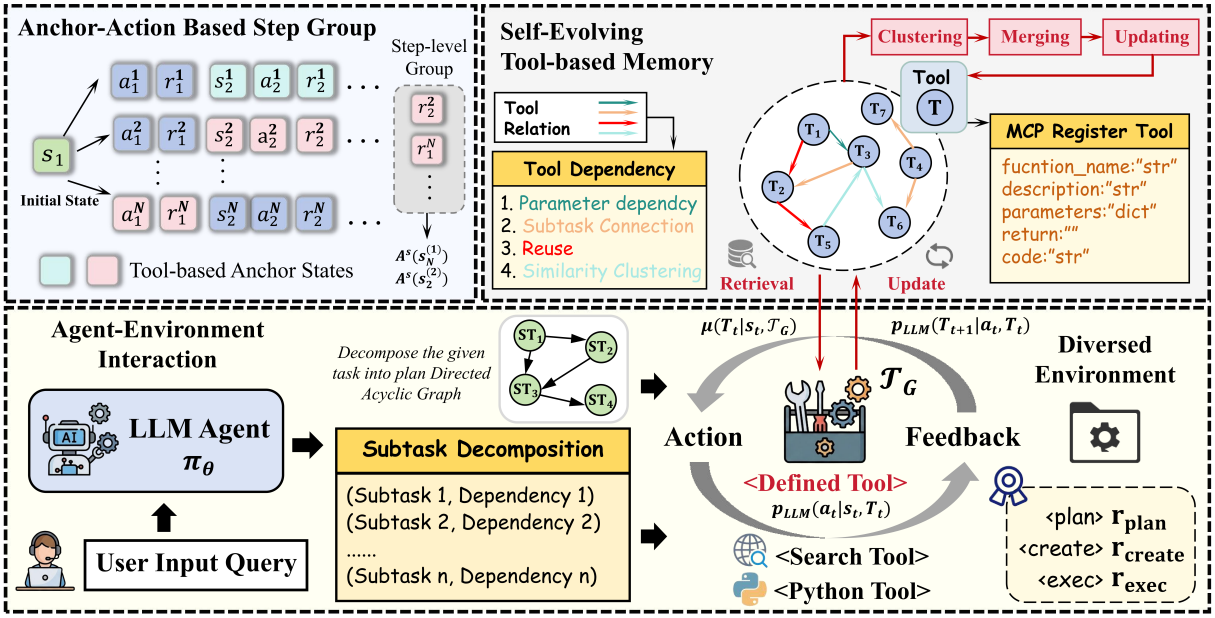

Figure 2: Overview of the proposed framework. The architecture consists of three components: (1) Agent-Environment Interaction (Bottom): decomposes tasks and generates structured trajectories via planning, retrieval, and execution. (2) Anchor-Action Step Grouping (Top-Left): leverages tool-based anchors to enable fine-grained, step-level advantage estimation. (3) Self-Evolving Tool-based Memory (Top-Right): A graph-structured memory $\mathcal{T}_G$ that dynamically retrieves, clusters, and updates MCP tools based on trajectory dependencies.

issue, basic step-level reward designs are susceptible to reward hacking, often lacking a reliable correlation with actual reasoning quality. Furthermore, current frameworks offer limited mechanisms for auditing these risks or implementing effective mitigation measures.

To overcome these limitations, we propose our algorithm in this section, combining step-level rewards with tool-based memory $\mathcal{T}_G$. Instead of grouping by raw environment states, we define the tool utilization as the **anchors** during advantage computation. We leverage a two-level advantage structure: (i) episode-level relative advantages capture the global effectiveness of entire trajectories, providing a stable task-level learning signal, and (ii) tool-based memory-anchored step advantages deliver fine-grained credit for tool creation, reuse, and tool-execution decisions.

Episode-Level Relative Advantages. The episode-level relative advantage is computed over a group of N trajectories $\{\tau_i\}_{i=1}^{N}$ rolled out under the same task and initial state. For each trajectory, we utilize the total return $R(\tau_i) = R_{\text{orm}}(\tau_i) + \sum_{t=1}^{T} r_t^{(i)}$ as a holistic measure of task completion quality, where R_{orm} denotes the rule-based score and $\sum_{t=1}^{T} r_t^{(i)}$ aggregates process-level rewards. The resulting set of trajectory-return pairs forms an *episode-level* group:

$$\mathcal{G}^E = \{(\tau_i, R(\tau_i))\}_{i=1}^{N} \tag{3}$$

The episode relative advantage $A^E(\tau_i)$ for each τ_i can be formalized as:

$$A_E(\tau_i) = \frac{R(\tau_i) - \frac{1}{N}\sum_j R(\tau_j)}{\sqrt{\frac{1}{N}\sum_j \left(R(\tau_j) - \frac{1}{N}\sum_k R(\tau_k)\right)^2}} \tag{4}$$

Step-Level Relative Advantages. While the episode-level relative advantage provides a coarse signal, it cannot distinguish the contributions of individual actions within a trajectory. To provide fine-grained credit assignment, we construct step-level groups using tool-based memory anchors rather than raw environment states. Specifically, let $\mathcal{U} = \{g_1, g_2, \ldots, g_{|\mathcal{U}|}\}$ denote the set of all distinct MCP tools appearing across the trajectory group $\{\tau_1, \tau_1, \ldots, \tau_N\}$. For each MCP tool g in $\mathcal{U}$, we collect all actions associated with t into a group.

$$\mathcal{G}_S(g) = \left\{ \left(a_t^{(i)}, R_t^{(i)}\right) \right\}_{(i,t)\in\mathcal{I}_g} \tag{5}$$

where $R_t^{(i)} = \sum_{k=t}^{T} \gamma^{k-t} r_k^{(i)}$ denotes the discounted return-to-go for the i-th trajectory starting from step t drawn inspiration from (Feng et al.,

2025). After all trajectories are generated, we perform a post-processing merge operation on the tool-based memory $\mathcal{T}_G$: newly created MCP tools are compared against existing ones using a similarity metric over their name and description. If the similarity exceeds a threshold, the new tool is merged with the most similar existing tool, and $\mathcal{T}_G$ is updated accordingly. This procedure defines a unique memory anchor g for each equivalence class of tools. All steps interacting with anchor g are aggregated into the same step-level group $\mathcal{G}_S(g)$. This grouping unifies credit assignment across trajectories and temporal contexts, ensuring that advantage estimation captures the specific utility of the MCP tool, isolated from unrelated contextual variances. Once these step-level groups are formed, the *step relative advantage* for each $g \sim \mathcal{U}$ and each action $a_t^{(i)} \in G^S(g)$ can be formalized as:

$$\mu_{\mathcal{G}} = \frac{1}{|\mathcal{G}_S(g)|} \sum_{(a,r)\in\mathcal{G}_S(g)} r,$$
$$A_S(a_t^{(i)}) = \frac{R_t^{(i)} - \mu_{\mathcal{G}}}{\sqrt{\frac{1}{|\mathcal{G}_S(g)|}\sum_{(a,r)\in\mathcal{G}_S(g)} (r-\mu_{\mathcal{G}})^2}} \tag{6}$$

where the denominator provides standard-deviation normalization over the group returns.

This tool-based, memory-anchored advantage evaluates the relative utility of actions associated with the same MCP tool across varying trajectories. While agents address distinct tasks and generate diverse trajectories, they operate within analogous state subspaces when utilizing the same specific tool. Consequently, by integrating MCP tool usage into training, we effectively abstract the unbounded real-world state space into a finite set defined by the toolset $\mathcal{T}_G$. When combined with the episode-level advantage, this approach offers a complementary optimization signal: the episode-level term provides a coarse, trajectory-wide guide, whereas the tool-level advantage assigns fine-grained credit specifically to tool-related decisions. Detailed policy optimization can be found in Appendix E.

3.4 Tool-Based Memory

In this section, we formally describe the operation and evolution of our `Tool-Based Memory`, as a directed graph $\mathcal{T}_G = (V, E)$, where nodes V denote registered MCP tools and edges E encode step-level dependencies. Serving as an external memory, the lifecycle of this graph encompasses four phases: **Subgraph Extraction**, **Tool Registration**, **Tool Retrieval**, and **Memory Update**.

Subgraph Extraction. During the `plan` phase, the task is decomposed into subtasks $\{ST_k\}_{k=1}^m$, forming a dependency graph G_{plan}. We project this structure onto the tool space via a mapping ϕ to derive a task-specific memory subgraph $\mathcal{T}_G^{(i)}$:

$$\begin{aligned}\mathcal{T}_G^{(i)} &= \big(V^{(i)}, E^{(i)}\big),\\ V^{(i)} &= \big\{\phi(ST_k) \mid k = 1, \ldots, m\big\},\\ E^{(i)} &= \big\{(\phi(u), \phi(v)) \mid (u, v) \in E_{\text{plan}}\big\}\end{aligned} \tag{7}$$

where $E^{(i)}$ preserves the trajectory-level execution order. Crucially, we instruct the model to generate dedicated, modular tools for specific subtasks rather than monolithic solvers. This granularity not only improves training stability but also ensures the resulting tools capture foundational operations, enhancing their reusability across diverse tasks.

Tool Registration and Retrieval. Tool registration is facilitated by the `mcp creation` tool. During trajectory execution, whenever the agent invokes this tool, we verify its execution status; successfully executed instances are tentatively added to a candidate pool. At the end of each training iteration, to prevent redundancy, we calculate cumulative rewards across rollouts and select only the tools associated with the highest rewards for final registration. For retrieval, given a sequence of decomposed subplans $[p_1, p_2, \ldots, p_n]$, we employ a dedicated model to identify the most relevant tools by evaluating the alignment between the content of each subplan and the tool descriptions. Detailed procedures are provided in Appendix F.

Memory Update and Consolidation. Once task-specific subgraphs $\{\mathcal{T}_G^{(i)}\}$ are constructed, they are integrated into the global memory $\mathcal{T}_G$ through a unified merge operation. This process involves two parallel mechanisms: semantic node merging and structural edge consolidation. First, to determine tool equivalence, we compute the semantic embedding $\mathbf{e}(v)$ for each tool. The similarity between a new tool $v \in V^{(i)}$ and an existing tool $v' \in V$ is measured via the cosine similarity of their normalized embeddings: $\text{sim}(v, v') = \tilde{\mathbf{e}}(v)^\top \tilde{\mathbf{e}}(v') \in [-1, 1]$. If $\text{sim}(v, v') \geq \delta$, v is merged with v'; otherwise, v is registered as a new node, δ is a predefined threshold. Simultaneously, directed edges representing trajectory-level precedence (i.e., subtask ST_p precedes ST_q) are incorporated to preserve

causal structures. Crucially, when tools are merged, their incident edges are automatically redirected to the consolidated node, effectively accumulating dependency patterns across diverse trajectories. The global memory update is formalized as:

$$\mathcal{T}_G \leftarrow \text{Merge}\Big(\mathcal{T}_G, \{\mathcal{T}_G^{(i)}\}_{i=1}^{N}\Big), \tag{8}$$

This procedure ensures that $\mathcal{T}_G$ evolves into a persistent repository that captures both tool-level semantics and sequential dependencies. The detailed algorithm is provided in Appendix D.

4 Experiment

4.1 Datasets

To comprehensively evaluate the effectiveness of our SEARL in training agents using a tool, we conduct experiments on the following three types of long-horizon reasoning tasks:

- **Mathematical Reasoning**: AIME2024 (American Invitational Mathematics Examination, 2024), MATh500 (Lightman et al., 2023), and GSM8K (Hendrycks et al., 2021).
- **Knowledge-Intensive Reasoning**: including WebWalker (Wu et al., 2025a); as well as three Wikipedia-based open-domain QA tasks: HotpotQA (Yang et al., 2018), 2WikiMultihopQA (Ho et al., 2020), and Musique (Trivedi et al., 2022), and bamboogle (Press et al., 2022).

Following ARPO (Dong et al., 2025b), we adopt the same data split settings for all benchmarks, ensuring consistency and comparability of results.

4.2 Baselines

To evaluate the effectiveness, we compare SEARL with common trajectory-level RL algorithms for training LLM-based tool-use agents, including TIR Prompting, GRPO (Shao et al., 2024), DAPO (Yu et al., 2025), REINFORCE++ (Hu, 2025), and ARPO (Dong et al., 2025b). GiGPO (Feng et al., 2025) is excluded from the experiments due to its incompatibility with our task settings.

4.3 Training Settings

For mathematical and multi-hop knowledge reasoning tasks, we utilize the 10,000 open-source RL training samples from Tool-star (Dong et al., 2025a) as our training dataset. The agent is equipped with two fundamental tools: a Python interpreter and a Wikipedia search interface, which is modified based on Chai et al. (2025). Notably, to ensure resource efficiency, we implement the local Wikipedia search server proposed in (Jin et al., 2025a). Detailed training configurations and specific prompts are provided in the Appendix C.

4.4 Evaluation Metrics

For all benchmarks, we adopt an LLM-as-Judge evaluation to ensure a consistent and reliable assessment across diverse tasks. Specifically, we employ Qwen3-32B as the judge model to assess binary correctness against ground-truth solutions, for its high alignment with human evaluation. The corresponding prompt is provided in Appendix G. We report results using pass@1 accuracy. Model predictions are post-processed by isolating the content between `<answer>` and `</answer>` tags, followed by extracting from `\boxed{...}`.

4.5 Main Results

Table 1 presents the comparative performance of SEARL against strong baselines across mathematical reasoning and multi-hop question answering benchmarks. The results indicate that our self-evolving tool-based framework outperforms policy gradient methods in knowledge-intensive reasoning tasks, while demonstrating superior generalization capabilities in complex mathematical problem-solving, thereby distinguishing itself.

Superiority of Structured Memory in Multi-Hop Reasoning. On multi-hop QA datasets (HotpotQA, 2wiki, Bamboogle), SEARL consistently outperforms or matches strong baselines. This gap is pronounced in tasks requiring the composition of information from disjoint sources. We attribute this to the Tool Graph Memory, acting as a persistent external knowledge structure. A challenge in our setting is that the local Wikipedia search retrieves massive irrelevant context. Baselines (e.g., GRPO), relying on transient context windows, struggle to filter this noise, often updating policies based on noisy trajectories. In contrast, our agent leverages graph structure to decompose queries and isolate precise evidence. This structured retrieval ensures intermediate steps are grounded, significantly reducing hallucinations and maintaining coherence.

Robustness and Generalization in Mathematical Reasoning. In the mathematical domain, results highlight a trade-off: robustness on standard tasks

Methods	Mathematical Reasoning			Multi-Hop QA				Avg Rank
	GSM8K	MATH500	AIME24	HotpotQA	2wiki	Musique	Bamboogle	
TIR Prompt	0.2259	0.0540	0.0000	0.2300	0.1250	0.0350	0.1200	5.29
GRPO	**0.8870**	**0.7360**	0.1333	0.2150	0.3450	**0.0900**	0.1600	2.43
DAPO	0.8059	0.5520	0.1333	**0.3350**	0.3500	0.0650	0.2480	3.00
Reinforce++	0.8658	0.6800	0.1000	0.1100	0.2600	0.0000	0.0080	4.57
ARPO	0.8241	0.6480	**0.3333**	0.1400	0.2200	0.0650	0.1760	3.57
SEARL	0.8620	0.6820	**0.3333**	**0.3350**	**0.3600**	**0.0900**	**0.3040**	**1.43**

Table 1: Performance comparison across mathematical reasoning and multi-hop QA benchmarks. **Bold** indicates the best performance, while underlined denotes the second best. The Avg Rank column represents the average ranking across all tasks (lower is better).

versus exceptional generalization on complex problems. On benchmarks like GSM8K and MATH500, SEARL remains competitive. We acknowledge that for simpler problems, autonomous tool generation may introduce minor procedural noise where monolithic reasoning suffices. However, this slight overhead is justified by the model achieving the highest performance on AIME24, matching that of ARPO. On this benchmark requiring novel solution paths, our method establishes a significant lead. While baselines may overfit to static patterns of easier datasets, our self-evolving mechanism dynamically constructs tools to decompose intricate problems, demonstrating superior adaptability.

4.6 Ablation Study and Analysis

In this section, we investigate the impact of the training algorithm on learning dynamics and evaluate the contribution of each component.

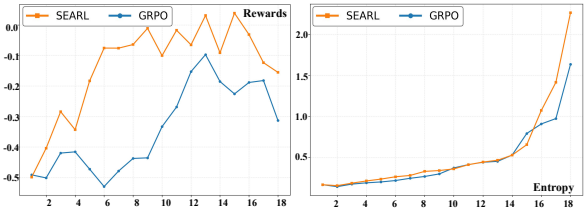

Figure 3: Comparison of training rewards and entropy between GRPO and SEARL.

Learning Dynamics. Figure 3 illustrates the evolution of overall training rewards and entropy, benchmarking SEARL against the GRPO baseline under identical workflow conditions. SEARL training rewards outperforms GRPO throughout the process, suggesting that it effectively leverages step-grouped advantages to derive more informative feedback. SEARL maintains higher entropy levels throughout training, indicating sustained exploration capabilities. Notably, the training rewards remain predominantly negative. Drawing inspiration from (Lee et al., 2025), we deliberately impose strict negative penalties to deter redundant tool invocations and failed creation attempts.

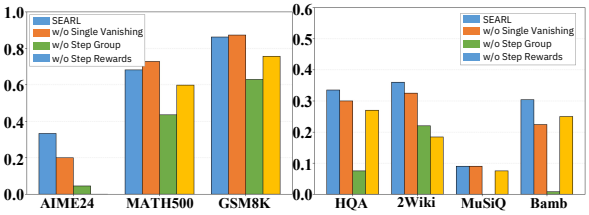

Figure 4: Ablation study illustrating the impact of removing different components.

Component Analysis. To validate the effectiveness of our design, we conduct an ablation study by selectively removing three key components: (i) *Single Vanishing*, which bypasses group-level advantage estimation when a group contains only a single element; (ii) *Step-level Grouping*, which removes the grouping strategy entirely; and (iii) *Step Rewards*, which eliminates fine-grained process-level reward feedback. The results are presented in Figure 4. We observe that removing Step-level Grouping leads to the most significant performance degradation across the majority of datasets (e.g., AIME24 and Bamb), underscoring its critical role in accurate advantage estimation. Similarly, the absence of Step Rewards results in a noticeable drop, confirming the necessity of dense supervision signals. In contrast, while the Single Vanishing mechanism has a relatively smaller impact, it remains essential for maintaining overall stability.

4.7 Implementation of the Tool Graph

Figure 5 illustrates the paradigm shift enabled by our framework in handling complex reasoning tasks. In contrast to the baseline, which relies on generating disposable, monolithic tools with high computational complexity ($O(n^2)$), our method

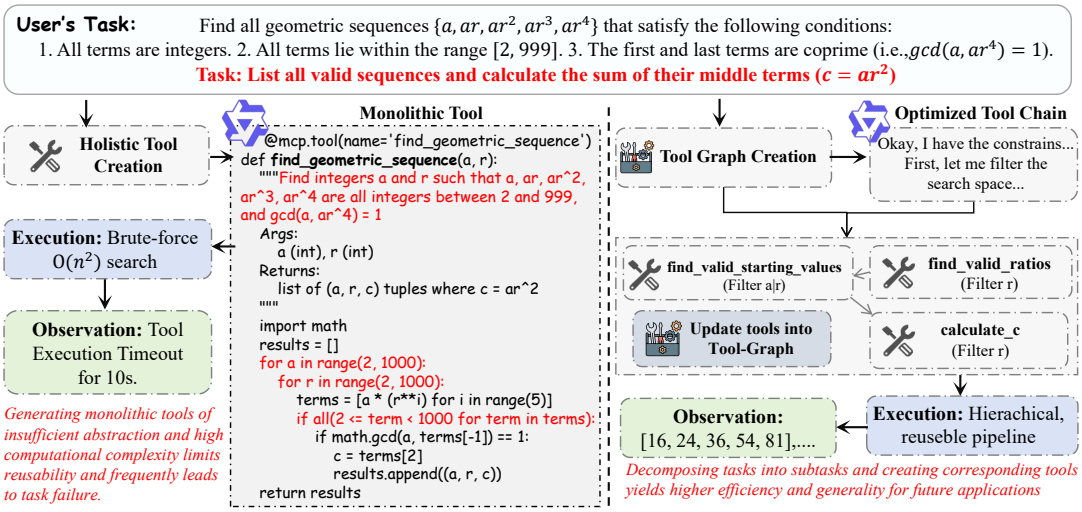

Figure 5: Comparison on constrained geometric sequence optimization. **(Left)** The baseline uses a monolithic tool causing $O(n^2)$ brute-force timeouts. **(Right)** Our method employs a modular tool chain to prune the search space, significantly improving efficiency and reusability.

produces consistently more accurate and mathematically rigorous solutions through modular tool evolution. Instead of naively iterating through all possible combinations, our agent identifies key constraints to optimize its reasoning strategy. It dynamically adjusts its approach by breaking the task into a dependency graph of sub-problems, creating specialized tools for each logical stage. This divide-and-conquer strategy enables the agent to enforce constraints hierarchically, filtering invalid candidates at the earliest possible step. As a result, the agent avoids the pitfalls of unstructured brute-force execution, achieving high-efficiency problem solving while populating the tool memory with robust, reusable modules for future tasks.

4.8 Evolving Dynamics of Tool Graph

To better understand the evolving dynamics of our designed tool memory, we extract a representative tool subgraph across four different training steps and analyze its structural progression, as illustrated in Figure 6. Here, N and E denote the number of nodes and edges, respectively, and the black dashed lines indicate the dependencies between tools. Overall, three distinct functional tool clusters emerge over time. During the early stages of training, the tool graph consists of small, disjoint subgraphs. As training progresses, these isolated components become connected through tool reuse and merging techniques. This evolution not only increases the structural complexity of the graph but also integrates experience from different domains. Beyond inter-cluster connections, the overall graph

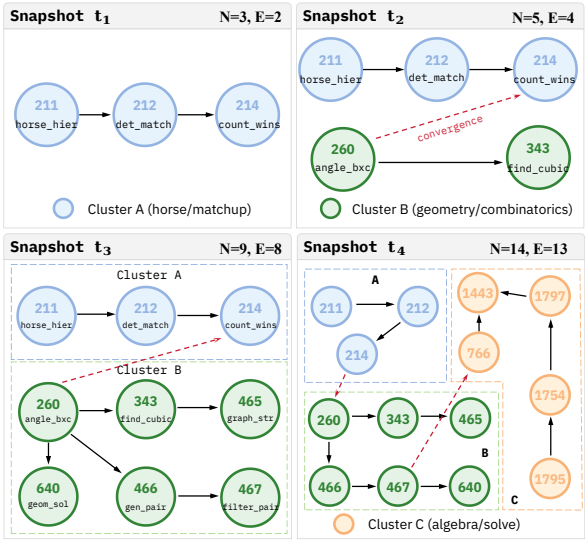

Figure 6: Structural Growth of Tool Subgraphs from Single-Path Reuse to Multi-Branch Convergence.

complexity also grows significantly, as evident in Snapshot t_3. In the later stages of training, the graph complexity continues to increase as distinct clusters merge together, ultimately equipping the agent with diverse, cross-disciplinary experiences.

5 Related Work

Self-Evolving Agents. Self-evolving agents aim to overcome the static limitations of LLMs through continual adaptation (Gao et al., 2025a). Existing research explores adaptation across multiple dimensions, including model parameter updates (Zhou et al., 2025; Hu et al., 2025a), long-term memory

expansion (Liang et al., 2024; Zhang et al., 2023), and autonomous tool creation or reuse (Qiu et al., 2025; Wang et al., 2024). Such evolution occurs either dynamically during inference or via continual learning (Qu et al., 2024). Notable systems like Alita (Qiu et al., 2025), SE-Agent (Lin et al., 2025), and Agent KB (Tang et al., 2025b) have demonstrated capabilities in dynamic tool generation, trajectory refinement, and cross-domain knowledge transfer. Most approaches treat tool evolution and policy learning as independent modules.

Agentic Reinforcement Learning. Reinforcement learning has become a central paradigm for improving agent decision-making in multi-turn environments, addressing challenges like long-horizon credit assignment and sparse rewards (Wu et al., 2025b; Mialon et al., 2023; Wei et al., 2025; Zhuang et al., 2025; Xu et al., 2026). While early trajectory-level methods like GRPO (Guo et al., 2025) struggled with coarse feedback signals, subsequent work improved stability through group-based advantage estimation (Feng et al., 2025) and structured rewards that evaluate reasoning quality and tool efficiency (Dong et al., 2025a). Recent efforts have scaled training to longer horizons (Gao et al., 2025b) and integrated hierarchical planning (Zhang, 2025). Despite the advances, most methods focus on optimizing policy parameters while neglecting persistent external memory. We bridge the gap by coupling policy optimization with the growth of a structured *Tool Graph Memory*, allowing agents to refine the decision policy and accumulate durable capabilities simultaneously.

6 Conclusion

We have presented a robust paradigm for building self-evolving agents capable of autonomous tool creation and fine-grained credit assignment. Specifically, we introduce the Tool Graph Memory, a dynamic mechanism that not only stores executable tools but also captures their causal dependencies and usage contexts. Coupled with anchor-based advantage estimation and designed process rewards, this memory enables the agent to efficiently generalize learned skills to novel tasks. Our extensive experiments confirm that this joint optimization of policy and memory yields significant gains. By enabling agents to build and refine their own Tool Memory over time, this work takes a significant step toward developing truly autonomous, open-ended generalist agents.

Limitations

While SEARL demonstrates substantial improvements in multi-hop reasoning and remains competitive in mathematical tasks, several limitations persist. First, a performance gap remains between our approach and other methods on the GSM8K and MATH500 datasets. This suggests that the overhead of generating tools for simple problems may hinder basic reasoning capabilities. Second, the toolset developed during training may limit the model's adaptability to other contexts, such as direct search or highly specialized domains. Furthermore, due to the scale of the model, many of the generated tools remain trivial and too simplistic to be effectively reused by other LLMs. Finally, despite careful design, the reward function may still incentivize superficial reward hacking. This underscores the need for further refinement to better align agent incentives with genuine task correctness and reasoning depth.

Acknowledgements

We thank the anonymous reviewers and the area chair for their constructive comments. The authors of this paper were supported by Shanghai Artificial Intelligence Laboratory.

References

American Invitational Mathematics Examination. 2024. Aime 2024.

Bytedance-Seed-Foundation-Code-Team, :, Yao Cheng, Jianfeng Chen, Jie Chen, Li Chen, Liyu Chen, Wentao Chen, Zhengyu Chen, Shijie Geng, Aoyan Li, Bo Li, Bowen Li, Linyi Li, Boyi Liu, Jiaheng Liu, Kaibo Liu, Qi Liu, Shukai Liu, and 37 others. 2025. Fullstack bench: Evaluating llms as full stack coders. *Preprint*, arXiv:2412.00535.

Jiajun Chai, Guojun Yin, Zekun Xu, Chuhuai Yue, Yi Jia, Siyu Xia, Xiaohan Wang, Jiwen Jiang, Xiaoguang Li, Chengqi Dong, Hang He, and Wei Lin. 2025. Rlfactory: A plug-and-play reinforcement learning post-training framework for llm multi-turn tool-use. *Preprint*, arXiv:2509.06980.

Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. 2025a. Tool-star: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410*.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, and 1

others. 2025b. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2025. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, and 1 others. 2025a. A survey of self-evolving agents: On path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*.

Jiaxuan Gao, Wei Fu, Minyang Xie, Shusheng Xu, Chuyi He, Zhiyu Mei, Banghua Zhu, and Yi Wu. 2025b. Beyond ten turns: Unlocking long-horizon agentic search with large-scale asynchronous rl. *arXiv preprint arXiv:2508.07976*.

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*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*.

Jian Hu. 2025. Reinforce++: A simple and efficient approach for aligning large language models. *arXiv preprint arXiv:2501.03262*.

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jianguang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. 2025a. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pages 496–507.

Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Qiguang Chen, and 1 others. 2025b. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation. *arXiv preprint arXiv:2505.23885*.

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

Ruofan Jin, Zaixi Zhang, Mengdi Wang, and Le Cong. 2025b. Stella: Self-evolving llm agent for biomedical research. *arXiv preprint arXiv:2507.02004*.

Sangyun Lee, Brandon Amos, and Giulia Fanti. 2025. Banel: Exploration posteriors for generative modeling using only negative rewards. *arXiv preprint arXiv:2510.09596*.

Xuechen Liang, Yangfan He, Yinghui Xia, Xinyuan Song, Jianhui Wang, Meiling Tao, Li Sun, Xinhang Yuan, Jiayi Su, Keqin Li, and 1 others. 2024. Self-evolving agents with reflective and memory-augmented abilities. *arXiv preprint arXiv:2409.00872*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Jiaye Lin, Yifu Guo, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, Mingguang Chen, Daxin Jiang, Binxing Jiao, Chen Hu, and 1 others. 2025. Se-agent: Self-evolution trajectory optimization in multi-step reasoning with llm-based agents. *arXiv preprint arXiv:2508.02085*.

Pan Lu, Bowen Chen, Sheng Liu, Rahul Thapa, Joseph Boen, and James Zou. 2025. Octotools: An agentic framework with extensible tools for complex reasoning. *arXiv preprint arXiv:2502.11271*.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, and 1 others. 2025. Humanity's last exam. *arXiv preprint arXiv:2501.14249*.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2022. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*.

Jiahao Qiu, Xuan Qi, Tongcheng Zhang, Xinzhe Juan, Jiacheng Guo, Yifu Lu, Yimin Wang, Zixin Yao, Qihan Ren, Xun Jiang, and 1 others. 2025. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution. *arXiv preprint arXiv:2505.20286*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. From exploration to mastery: Enabling llms to master tools via self-driven interactions. *arXiv preprint arXiv:2410.08197*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. *URL https://arxiv.org/abs/2402.03300*, 2(3):5.

Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. 2025. Agentic reasoning and tool integration for llms via reinforcement learning. *arXiv preprint arXiv:2505.01441*.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, and 1 others. 2025a. Agent kb: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, and 1 others. 2025b. Agent kb: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. 2024. Toolgen: Unified tool retrieval and calling via generation. *arXiv preprint arXiv:2410.03439*.

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, and 1 others. 2025. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. *arXiv preprint arXiv:2505.16421*.

Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, and 1 others. 2025a. Webwalker: Benchmarking llms in web traversal. *arXiv preprint arXiv:2501.07572*.

Jiaqi Wu, Qinlao Zhao, Zefeng Chen, Kai Qin, Yifei Zhao, Xueqian Wang, and Yuhang Yao. 2025b. Gap: Graph-based agent planning with parallel tool use and reinforcement learning. *arXiv preprint arXiv:2510.25320*.

Zihang Xu, Haozhi Xie, Ziqi Miao, Wuxuan Gong, Chen Qian, and Lijun Li. 2026. Stable adaptive thinking via advantage shaping and length-aware gradient regulation. *arXiv preprint arXiv:2602.22556*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

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)*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Danyang Zhang, Lu Chen, Situo Zhang, Hongshen Xu, Zihan Zhao, and Kai Yu. 2023. Large language models are semi-parametric reinforcement learning agents. *Advances in Neural Information Processing Systems*, 36:78227–78239.

Yanfei Zhang. 2025. Agent-as-tool: A study on the hierarchical decision making with reinforcement learning. *arXiv preprint arXiv:2507.01489*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Yifei Zhou, Sergey Levine, Jason Weston, Xian Li, and Sainbayar Sukhbaatar. 2025. Self-challenging language model agents. *arXiv preprint arXiv:2506.01716*.

Yuchen Zhuang, Di Jin, Jiaao Chen, Wenqi Shi, Hanrui Wang, and Chao Zhang. 2025. Workforceagent-r1: Incentivizing reasoning capability in llm-based web agents via reinforcement learning. *arXiv preprint arXiv:2505.22942*.

A Dataset Statics

A.1 Mathematical Reasoning Benchmarks

- **AIME24 (American Invitational Mathematics Examination, 2024)** serves as a rigorous benchmark for evaluating mathematical reasoning. It comprises 30 challenging problems derived from the American Invitational Mathematics Examination, spanning diverse domains such as algebraic equations and geometric puzzles. Due to its high complexity and the richness of its problem types, AIME24 is widely adopted to assess the reasoning performance of advanced models.

- **MATH500 (Lightman et al., 2023)** is a curated subset of 500 challenging problems selected by OpenAI from the larger MATH dataset. These problems cover a broad spectrum of mathematical disciplines, including algebra, geometry, calculus, and number theory, with difficulty levels ranging from high school to collegiate standards. It is frequently used in academic research to evaluate the problem-solving capabilities of various reasoning models.

- **GSM8K (Hendrycks et al., 2021)** consists of high-quality grade-school math word problems released by OpenAI. Solving these problems typically requires 2 to 8 steps of multi-step reasoning involving basic arithmetic operations. This dataset is primarily used to test the logical consistency and fundamental mathematical competencies of models.

A.2 Knowledge-Intensive reasoning benchmarks

- **HotPotQA (Yang et al., 2018)** is a pivotal benchmark for multi-hop question answering. Sourced entirely from Wikipedia, it provides a rich, structured knowledge base designed to evaluate the ability of LLMs to perform complex reasoning and process information across multiple supporting documents.

- **2WikiMultihopQA (Ho et al., 2020)** is specifically constructed to assess multi-step reasoning capabilities. It challenges natural language processing models to answer complex queries by integrating and synthesizing evidence from disjoint Wikipedia articles, ensuring that models cannot rely on single-document retrieval alone.

- **MuSiQue (Trivedi et al., 2022)** serves as a highly challenging benchmark aimed at pushing the boundaries of multi-hop reasoning. By minimizing reasoning shortcuts, it encourages the development of models that go beyond simple information retrieval, requiring deeper semantic understanding and rigorous logical synthesis to derive correct answers.

- **Bamboogle (Press et al., 2022)** evaluates reasoning capabilities using "Google-proof" questions that resist direct search engine lookup. It focuses on queries where the answer must be derived by combining information from multiple distinct sources. This benchmark is crucial for distinguishing between genuine multi-source synthesis and reliance on parametric memory or simple retrieval heuristics.

B Baseline Descriptions

- **GRPO (Guo et al., 2025)** is a reinforcement learning algorithm based on policy optimization, designed to balance stability, sample efficiency, and theoretical guarantees. By introducing the concept of group-based relative advantage, it simplifies gradient estimation while preserving the theoretical assurance of monotonic policy improvement. GRPO is versatile and applicable to tasks in both continuous and discrete action spaces.

- **DAPO (Yu et al., 2025)**, developed by ByteDance Labs, is an RL algorithm tailored to address the stability challenges of large-scale LLM training. It demonstrates superior performance in complex tasks such as mathematical reasoning and code generation. Its proposed "Clip-Higher" strategy effectively boosts entropy to encourage sample diversity. Furthermore, DAPO stabilizes the training process through mechanisms like dynamic sampling, token-level policy gradient loss, and overlong reward shaping.

- **REINFORCE++ (Hu, 2025)** represents a robust evolution of the classic REINFORCE algorithm, integrating multiple optimization strategies to mitigate high variance. It incorporates baseline subtraction and temporal difference (TD) estimation to stabilize gradient updates, enabling incremental learning without the need to await full trajectories. Additionally, it employs entropy regularization to prevent premature policy rigidity and encourage exploration.

- **ARPO (Dong et al., 2025b)** is an RL method specifically designed for multi-turn LLM agents. It features an entropy-based adaptive rollout scheme that dynamically intensifies sampling during steps with high uncertainty. Moreover, it incorporates a specialized advantage attribution mechanism to effectively assign credit across complex, branching tool-use interactions.

C Implementation Details

In this section, we detail the experimental training settings. We implement SEARL based on the RL-Factory framework (Chai et al., 2025). Crucially, to prevent the model from overfitting to deterministic tool outputs, we exclude tool execution results from the loss calculation; optimization is restricted solely to tokens involved in reasoning and tool invocations. For the environment, we utilize the Python sandbox from (Bytedance-Seed-Foundation-Code-Team et al., 2025) as the coding interface and a local Wikipedia search server (Jin et al., 2025a) for retrieval. To strike a balance between efficiency and performance, we limit search returns to the top-3 results and impose a 10-second timeout on tool calls to ensure training efficiency. All experiments are conducted using the Qwen3-4B model in standard generation mode (non-thinking). All experiments were performed on NVIDIA H200 GPUs, with each training epoch requiring approximately 10 hours. Specific hyperparameters are listed in Table 2.

Table 2: Agentic RL Training Hyperparameters.

Hyperparameter	Value
Backbone Model	Qwen-3-4B
top_p	0.98
rollout_num	8
temperature	0.7
repetition_penalty	1.05
max_turns	6
max_prompt_length	4096
max_response_length	2048
Global Batch Size	384 (64 $\times$ 6 GPUs)
Learning Rate	1.0×10^{-6}
Num Train Epochs	1.0

D Training Algorithms

Algorithm 1 SEARL Training Process

Require: Dataset $\mathcal{D}$, Initial Policy π_θ, Reference Policy π_{ref}, Initial Tool Graph $\mathcal{T}_G$, Learning rate η, Hyperparameter λ, Iterations K, Rollouts per task N

for iteration $k \leftarrow 1$ **to** K **do**
  Initialize trajectory buffer $\mathcal{B} \leftarrow \emptyset$
  Initialize candidate tool buffer $\mathcal{C}_{new} \leftarrow \emptyset$ ▷ Buffer for newly created tools
  Sample a batch of tasks X from $\mathcal{D}$
  for each task $x \in X$ **do**
    for rollout $i \leftarrow 1$ **to** N **do**
      Initialize trajectory $\tau_i \leftarrow \emptyset$
      for step $t \leftarrow 1$ **to** T **do**
        if $t == 1$ **then**
          Generate subtask plans $P = [p_1, \ldots, p_n]$ based on query x
        end if
        Retrieve tools $T_t \leftarrow \text{Retrieve}(P, \mathcal{T}_G)$
        Generate action $a_t \sim \pi_\theta(\cdot|s_t, T_t)$
        Execute a_t, observe reward r_t and next state s_{t+1}
        Append (s_t, a_t, r_t, T_t) to τ_i
        if a_t creates a new tool t_{new} **then**
          $\mathcal{C}_{new} \leftarrow \mathcal{C}_{new} \cup \{t_{new}\}$ ▷ Collect distinct tools separately
        end if
      end for
      $\mathcal{B} \leftarrow \mathcal{B} \cup \{\tau_i\}$ ▷ Collect trajectory for RL
    end for
  end for
  Memory Evolution:
  Filter valid tools from $\mathcal{C}_{new}$ based on execution success and rewards
  Register filtered tools into $\mathcal{T}_G$ via Merge and consolidation
  Advantage Estimation:
  for each episode group $\mathcal{G}^E$ in $\mathcal{B}$ **do**
    Compute total return $R(\tau) \leftarrow R_{orm} + \sum r_t$
    Compute Episode Relative Advantage A^E
    Identify unique MCP tools $\mathcal{U}$ involved in $\mathcal{G}^E$
    for each tool anchor $g \in \mathcal{U}$ **do**
      Aggregate step-level group $\mathcal{G}_S(g)$ from $\mathcal{B}$
      Compute Tool-Anchored Step Advantage A^S
    end for
    Compute final advantage $A_{total} \leftarrow A^E + \lambda \cdot A^S$
  end for
  Policy Update:
  Optimize π_θ by maximizing $\mathbb{E}_{\tau \sim \mathcal{B}}[A_{total} \log \pi_\theta]$
end for

E Policy Optimization

Inspired by GiGPO (Feng et al., 2025), we integrate the two levels of advantage signals into a unified metric for hierarchical credit assignment:

$$A(a_t^{(i)}) = A^E(\tau_i) + \omega \cdot A^S(a_t^{(i)}), \tag{9}$$

where $\omega \in \mathbb{R}_{\geq 0}$ is a weighting coefficient that balances the episode-level and step-level advantages. Specifically, $A^E(\tau_i)$ evaluates the relative quality of the entire episode compared to others

within its group, whereas $A^S(a_t^{(i)})$ provides fine-grained credit assignment for actions taken under analogous tool-use conditions. Together, they offer robust hierarchical supervision for the policy optimization of LLM agents. Consequently, the clipped policy optimization objective for SEARL is defined as:

$$\mathcal{J}_{\text{SEARL}}(\theta) = \mathbb{E}_{\substack{x\sim p(X) \\ \{\tau_i\}_{i=1}^{N}\sim\pi_{\theta_{\text{old}}}}} \Bigg[\frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \min\Big(\rho_\theta(a_t^{(i)}) A(a_t^{(i)}), \text{clip}\big(\rho_\theta(a_t^{(i)}), 1\pm\epsilon\big) A(a_t^{(i)})\Big)\Bigg] - \beta \mathbb{D}_{\text{KL}}\big(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x)\big),$$

where $\rho_\theta(a_t^{(i)}) = \frac{\pi_\theta(a_t^{(i)}|s_t^{(i)},x)}{\pi_{\theta_{\text{old}}}(a_t^{(i)}|s_t^{(i)},x)}$ is the importance weight, and β scales the KL penalty that enforces proximity to the reference policy π_{ref}.

F Tool Creation and Retrieval Details

```
@mcp.tool()
def create_and_execute_mcp(
    name: str,
    description: str,
    arguments: str,
    returns: str,
    code: str,
    inputs: Dict[str, Any],
    timeout: float = 15.0
) -> str:
    """
    MCP Creation and Execution Tool

    Create and immediately execute an
        MCP tool function.

    Args:
        name: MCP tool name (Function
            name)
        description: Tool description
        arguments: Argument description
            string (e.g., "a, b (int)")
        returns: Return value
            description
        code: Complete Python function
            implementation code
        inputs: Input arguments
            dictionary required for this
             function call (e.g., {"a":
            1, "b": 2})
        timeout: Execution timeout in
            seconds (default: 15.0)

    Returns:
        str: JSON formatted string
            containing creation status
            and execution result
             {
                 "creation_success":
                     bool,
                 "execution_result": any
                     ,
                 "stdout": str,
                 "stderr": str,
                 "error": str (optional)
             }
    """
    ...
```

Listing 1: MCP Creation and Execution Tool

F.1 Tool Creation

We enable the agent with the power of tool creation by introducing a predefined tool named `mcp creation tool`, thus utilizing the basic tool-calling ability of LLMs. The detailed code input can be found in Code 1. Different from solely creating tools, we directly execute the created tool to save reasoning steps in LLMs, and filter out the creation failed tools.

Upon creation, the generated tools $\mathtt{t}_i$ are not immediately registered in the global `Tool Graph` $\mathcal{T}_G$ to mitigate the risk of duplication across different rollouts of the same training sample. Instead, during the formal registration phase, we aggregate all trajectories $[\tau_1, \tau_2, \ldots, \tau_N]$ associated with the sample. We compute the cumulative reward for each individual trajectory—summing both outcome and process scores—and retain only the optimal trajectory τ_i with the highest reward. Subsequently, the extracted sub-plan graph G_{plan} and its corresponding tools are integrated into $\mathcal{T}_G$. In this structure, tools represent nodes while subtask dependencies serve as edges, forming a **connected component**. Notably, we employ an embedding model to encode the description of tool v into a semantic embedding $\mathbf{e}(v)$, which is then integrated as a feature of the node.

F.2 Tool Retrieval

In contrast to the complex lifecycle of tool creation, spanning invocation, generation, and registration, the tool retrieval procedure is significantly more streamlined. This process occurs exclusively during the system prompt generation stage, following the computation of plans. Given the derived subtask plan list $[p_1, p_2, \ldots, p_n]$, we first encode the textual description of each plan into embeddings $[\mathbf{e}_p^1, \mathbf{e}_p^2, \ldots, \mathbf{e}_p^n]$. We then utilize a retrieval model $\mathcal{R}$ to identify the top-k relevant tools from the Tool Graph $\mathcal{T}_G$ via graph traversal. Finally, these tools are appended to the system prompt. Crucially, this mechanism serves as a recommendation rather than

a constraint; the agent retains the autonomy to decide whether to invoke the retrieved tools. The detailed prompt for tool recommendation can be found in Appendix G.

F.3 Retrieval Model

To ensure robust and contextually relevant experience retrieval, we implement a **Hybrid Retrieval** framework that synthesizes the strengths of both sparse and dense retrieval mechanisms. This dual-stream approach captures relevance at different levels of abstraction:

Sparse Retrieval (Text-based). For surface-level term matching, we utilize traditional information retrieval techniques based on TF-IDF (Term Frequency-Inverse Document Frequency). This method represents textual content as sparse, high-dimensional vectors, quantifying the importance of terms relative to the corpus. It excels at identifying documents with significant keyword overlap, ensuring high precision when vocabulary alignment is strong.

Dense Retrieval (Semantic). To capture deeper contextual relationships beyond exact keyword matching, we employ a dense retrieval component. Specifically, we utilize the `sentence-transformers/all-MiniLM-L6-v2` model, a lightweight transformer-based encoder that maps sentences into a continuous vector space. By computing cosine similarity between embeddings, this method retrieves experiences that are semantically related even in the absence of lexical overlap.

Hybrid Fusion. To mitigate the limitations of individual methods, we fuse the results using a weighted ranking strategy. For a retrieved experience $e_i \in \mathcal{E}$, the final relevance score σ_i^{hyb} is computed as a linear combination of the sparse score σ_i^{text} and the dense score σ_i^{sem}:

$$\sigma_i^{\text{hyb}} = \alpha \cdot \sigma_i^{\text{text}} + (1 - \alpha) \cdot \sigma_i^{\text{sem}}, \qquad (10)$$

where $\alpha \in [0, 1]$ is a tunable parameter (setting $\alpha = 0.5$ in our settings) that balances the trade-off between lexical precision and semantic generalization. This hybrid mechanism ensures robustness against both syntactic variation and conceptual drift.

G Prompt

Initial Plan Generation

Create a step-by-step plan for the given task.
Instructions:

1. Break the task into subtasks (ST1, ST2, ST3, ...). If the task is indivisible, use only ST1.
2. Define subtask dependencies in a **##DAG_LIST**. Example: [(ST1, ST2)] means ST2 depends on ST1. For a single task, use [(ST1)].
3. For each subtask **##STn**, provide clear, actionable steps.
4. The plan must only contain concrete actions. Do not include explanations or simulated code.

SUBTASKS EXAMPLE:

```
##DAG_LIST
[(ST1, ST3), (ST1, ST2), (ST2, ST3)]
##ST1:xxx
1. xxx.
2. xxx.
##ST2:xxx
1. xxx.
2. xxx.
3. xxx.
4. xxx.
##ST3:xxx
1. xxx.
2. xxx.
3. xxx.
```

Previous is an example of generating subtasks, you are not required to solve the task within the plan itself; focus on outlining the steps.
Always verify your answers before providing them, provide the plan for verifying.
If you are uncertain about the task, you can plan for web search to gather more information.

Now, write a plan below to solve the task within {{max_turns-1}} steps.

System Prompt: Tool Creation Prompt

```
You are a step-by-step problem solver. For each step, follow this loop:
- <subtask> ST_X: {step} </subtask>
- <thinking> ... </thinking>: explain reasoning, note whether an MCP tool is needed
- <tool_call> ... </tool_call>: use plain Python if no MCP is needed; to create a new MCP, call
    create_and_execute_mcp tool.
Examples:
- Create a reusable MCP:
  <subtask> ST_1: Build quadratic solver MCP </subtask>
  <thinking> I need to create a quadratic equation solver to handle equations of the form ax^2+bx
    +c=0. </thinking>
  <tool_call>
  {
      "name": "create_and_execute_mcp",
      "arguments": {
          "name": "quadratic_solver",
          "description": "Solve ax^2+bx+c=0",
          "arguments": "a,b,c (float)",
          "returns": "roots list",
          "code": "def quadratic_solver(a,b,c):\n    import math\n    d=b*b-4*a*c\n    if d>0:\n
           r1=(-b+math.sqrt(d))/(2*a)\n        r2=(-b-math.sqrt(d))/(2*a)\n        return [r1,r2
    ]\n    if d==0:\n        r=-b/(2*a)\n        return [r,r]\n    return 'complex roots'",
          "inputs": {"a": 1, "b": -3, "c": 2}
      }
  }
  </tool_call>
Rules:
- Only create an MCP when it is reusable; at most one MCP per step; ensure a unique MCP name
- Do not reuse tool names as variables; each code block is state-isolated-redefine imports/vars/
    functions every step
- Repeat the cycle of <subtask> ST_X: {step} </subtask> <thinking> ... </thinking> <tool_call>
    ... </tool_call> until the task is solved.
- Final answer must be in <answer>\boxed{...}</answer> with only the boxed result
--- Here is the Task and Plan: ---
Task:
**{{question}}**
Plan:
**{{input_plan}}**
Now begin to solve the task according to the given plan. Now begin with ST_1.
If you solve the task correctly, you will receive a reward of $1,000,000.
```

System Prompt: Tool & Reasoning Format

```
You are a helpful assistant who can solve the given question step by step with the help of
    available tools. Given a question, you need to think about the reasoning process and then
    provide the answer. While thinking, you can invoke tools to search for information or
    perform calculations as needed.
# Reasoning and Answer Format
- The reasoning process should be enclosed within <think> </think> tags (if thinking mode is
    enabled).
- The final answer must be enclosed within <answer> </answer> tags.
- The final exact answer should be enclosed within \boxed{} with LaTeX format inside the <answer>
     tags.
# Example
<think>
I need to search for information about this topic first.
</think>
<tool_call>
{
    "name": "search",
    "arguments": {
        "query": "search_query_here"
    }
}
</tool_call>
[Tool execution result will be automatically returned here]
<think>
Now I have the information, let me calculate the result using Python.
</think>
<tool_call>
{
    "name": "execute_python_code",
    "arguments": {
        "code": "print(2+3)",
        "timeout": 10.0
    }
}
</tool_call>
[Tool execution result will be automatically returned here]
<think>
Based on the search results and calculations, I can now provide the final answer.
</think>
<answer>
The final answer is \boxed{answer here}
</answer>
Note: Tool execution results are automatically returned by the system. You do not need to include
     <tool_response> tags. Simply continue with your reasoning after the tool call.
```