L0: REINFORCEMENT LEARNING TO BECOME GENERAL AGENTS

L0

Junjie Zhang [1,*], **Jingyi Xi** [1,*], **Zhuoyang Song**[1,*],
Junyu Lu[1], **Yuhua Ke**[1], **Ting Sun**[1], **Yukun Yang**[1,2],
Jiaxing Zhang[1,†], **Songxin Zhang**[1,†], **Zejian Xie**[1,†]

[1]Lionrock AI Lab, China Merchants Research Institute Of Advanced Technology
[2]Department of Statistics and Data Science, Southern University of Science and Technology

June 28, 2025

ABSTRACT

Training large models to function as autonomous agents for multi-turn, long-horizon tasks presents significant challenges in scalability and training efficiency. To address this, we introduce L-Zero (L0), a scalable, end-to-end training pipeline for general-purpose agents. Featuring a low-cost, extensible, and sandboxed concurrent agent worker pool, L0 lowers the barrier for applying reinforcement learning in complex environments. We also introduce NB-Agent, the agent scaffold within L0, which operates in a "code-as-action" fashion via a Read-Eval-Print-Loop (REPL). We evaluate L0 on factuality question-answering benchmarks. Our experiments demonstrate that a base model can develop robust problem-solving skills using solely Reinforcement Learning with Verifiable Rewards (RLVR). On the Qwen2.5-7B-Instruct model, our approach yields substantial gains on SimpleQA from 30% to 80% and on HotpotQA from 22% to 41%. We have open-sourced the entire L0 system, including our L0 series models, the NB-Agent, a complete training pipeline, and the corresponding training recipes on GitHub.

Keywords AI Agents • Reinforcement Learning • Code-as-Action • Verifiable Rewards

1 Introduction

Recent progress in reinforcement learning (Hu 2025; Yu et al. 2025) has enabled Large Language Models (LLMs) to become active agents, capable of performing complex, multi-step tasks by interacting with external environments and tools. Pioneering works have demonstrated impressive capabilities by integrating search functionalities with the models' intrinsic reasoning, allowing agents to dynamically query for information and ground their responses in external knowledge. For instance, Search-o1 (Li et al. 2025) incorporates an agentic retrieval-augmented generation (RAG) mechanism so that the model dynamically retrieves external knowledge whenever it encounters uncertainty. SearchR1 (Jin et al. 2025) framework teaches an LLM to generate multiple search queries during step-by-step reasoning and retrieve real-time information, yielding large accuracy gains on QA tasks.

However, existing works have severe limitations in multi-turn RL pipelines, as they frame long-horizon reasoning as a single-step bandit problem with only final-answer rewards, ignoring intermediate signals. Similarly, strict prompt templates are often used that allow just one tool call per reasoning turn. Such designs make it difficult for the agent to coordinate multiple tools or refine its behavior through turn-level feedback. In practice, real-world agentic tasks typically require orchestrating heterogeneous actions (e.g. web search, database queries, code execution) over many steps and preserving context or memory across those steps. It indicates that training these agents for long-horizon, stateful interactions remains a significant challenge, demanding both expressive agent architectures capable of managing internal state and scalable RL infrastructure that can handle resource-intensive, multi-turn rollouts. This prevailing

*Equal Contribution
†Corresponding Author

gap motivates our research toward bridging advanced agent designs and effective, scalable reinforcement learning frameworks.

To address this prevailing gap, we introduce L-Zero (L0), a scalable, end-to-end pipeline for training general-purpose agents. We first present the architectural heart of our system, the NB-Agent, a novel agent scaffold that operates using a "code-as-action" paradigm (Wang et al. 2024) within an interactive, stateful Python environment. We then detail the end-to-end reinforcement learning framework designed to train this agent, highlighting three core innovations: an agentic policy gradient tailored for complex, multi-token actions; a multi-faceted reward model based on verifiable outcomes; and a highly scalable, sandboxed infrastructure engineered for robust, parallelized agent rollouts. Finally, we present a comprehensive empirical evaluation of L0 on several challenging question-answering benchmarks. Our experiments demonstrate that the agentic scaffold alone provides a strong structural prior for reasoning, which is then significantly amplified by our RL training, leading to substantial performance gains over strong supervised baselines.

2 NB-Agent

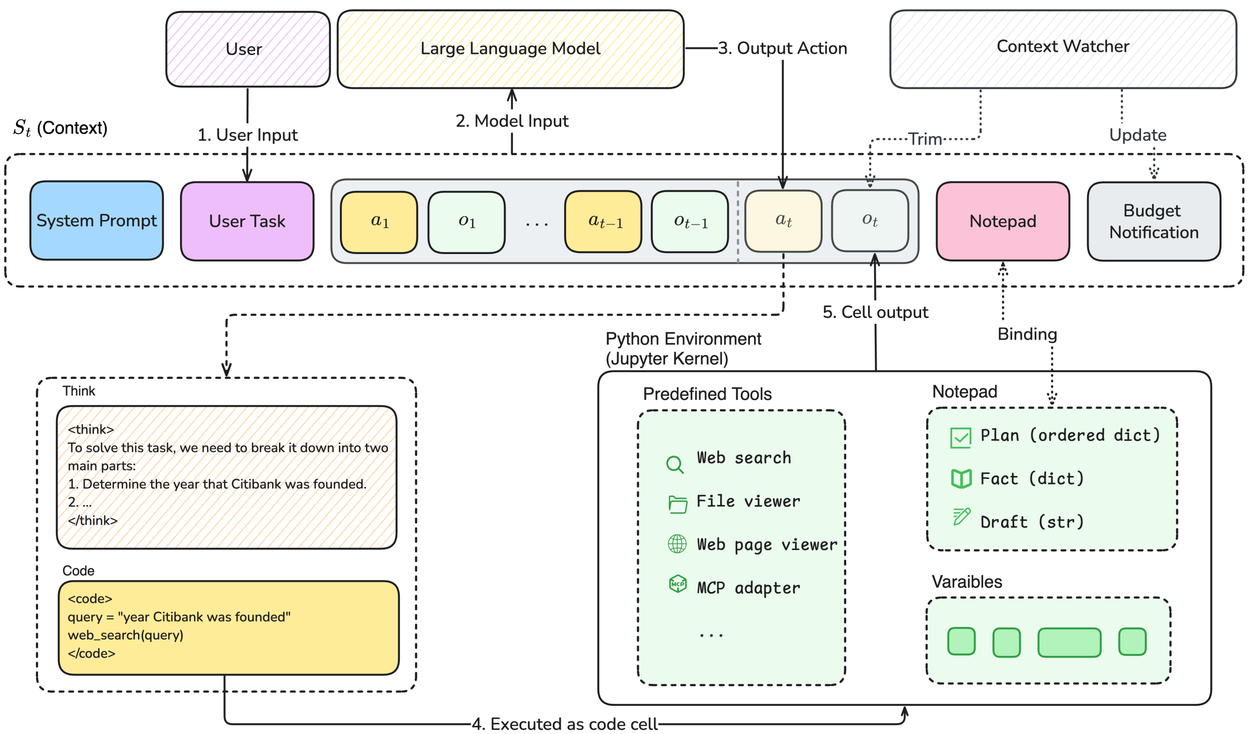

Figure 1: NB Agent

Inspired by prior work that champions code as a general-purpose action space for autonomous agents, we introduce the NB-Agent. Our agent operates within a cyclical process as detailed in Algorithm 1 and Figure 1. The loop initiates with a user-defined task and continues until a definitive answer is submitted.

As illustrated, upon receiving a task, the NB-Agent enters a "Think-Code-Observe" loop. In each cycle, the LLM generates a `<think>` trace to reason about the next steps, followed by a `<code>` block containing Python code. This code is then executed within a code cell, mimicking a human's interaction with a Jupyter Notebook. The output from the cell execution is captured, wrapped in an `<output>` tag, and serves as the observation for the next cycle.

The `context_watcher` module is responsible for managing the context window of the LLM. If the conversational history exceeds the context length, it will strategically truncate previous actions and outputs. A notification regarding the remaining token and step budget is also sent to the LLM to inform its planning. This loop continues until the agent invokes the submit_final_answer tool to terminate the process and return the result.

2.1 Agent's Environment: An Interactive Python Kernel

We ground our agent's operation within a standard Python environment, specifically a Jupyter kernel. This design choice is pivotal for two reasons. First, it provides a robust and stateful execution environment where the agent can

Algorithm 1 Agentic Loop

Input: Initial state $s_0 =$ [system_prompt, user_task]
Ensure: Final result when task is completed
$t \leftarrow 1$
while True **do**
  $s_t \leftarrow$ context_watcher$(s_{t-1}, a_{t-1}, o_{t-1})$
  $a_t \leftarrow (a_{\text{think},t}, a_{\text{code},t}) \leftarrow \Pi_\theta(s_t)$
  $o_t \leftarrow$ execute_code_in_environment(s_t, a_t)
  if is_final_answer_submitted(o_t) **then**
    break
  end if
  $t \leftarrow t + 1$
end while
return final result

define variables, install new libraries, and maintain a persistent session state, mirroring the workflow of a human data scientist.

Second, and more critically, it leverages the Read-Eval-Print Loop (REPL) as a natural and effective mechanism for the agent's interaction cycle. The agent's generated code (read) is executed by the kernel (eval), and the output is captured (print) and returned as an observation. This "action-observe" paradigm inherent to REPL provides immediate feedback, allowing the agent to iteratively refine its approach based on concrete results.

2.2 Overcoming Context Limitations

A primary challenge for large language model-based agents is the finite context window. We introduce two key innovations to address this:

Context-Variable Binding via a "NotePad": We establish a bidirectional link between the agent's transient context and persistent Python variables through a structured NotePad tool. This NotePad is a mutable object within the Python environment, comprising dedicated modules for planning, fact storage, and drafting. By manipulating this object through code, the agent can actively control what information is preserved beyond the immediate context window, effectively giving it a manageable, long-term memory.

REPL-driven Information Retrieval: The REPL interface allows the agent to not only execute actions but also to manage its memory. The agent can save critical information (e.g., API results, intermediate findings) to Python variables using simple commands and can retrieve it in later steps as needed. This mechanism transforms the Python environment into a dynamic and accessible external memory, enabling the agent to handle long-term dependencies and complex data structures efficiently.

3 End-to-End Agentic Reinforcement Learning

Training the NB-Agent with RLVR requires a framework built specifically for long-horizon, interactive tasks. To this end, we developed a comprehensive framework for end-to-end agentic reinforcement learning based on verl. As illustrated in Figure 2, our approach is built on three core innovations designed to overcome the unique challenges of this domain: - A novel agentic policy gradient that redefines an "action" to encompass a entire sequence of reasoning and code generation. - A verifiable reward model that provides multi-faceted, automated feedback on final answer correctness, format compliance, and code execution. - A scalable, decoupled infrastructure engineered to handle the resource-intensive demands of parallel, multi-turn agent rollouts.

3.1 Agentic Policy Gradient

In contrast to conventional language modeling where an action corresponds to generating a single token, our agentic framework defines an action as a complete sequence of tokens that constitutes a meaningful step. This necessitates a corresponding adaptation of the policy gradient method. Therefore, we formulate a policy gradient that operates at the level of these action sequences. The policy gradient, $\hat{g}$, is estimated as the expectation over timesteps t

$$\hat{g} = \hat{\mathbb{E}}_t[\nabla \log \pi_\theta(a_t, s_t)\hat{A}_t]$$

Since $\Pi_\theta(a_t, s_t)$ is the probability of generating an action sequence under state s_t, this can be decomposed into the product of conditional probabilities for each token in the sequence:

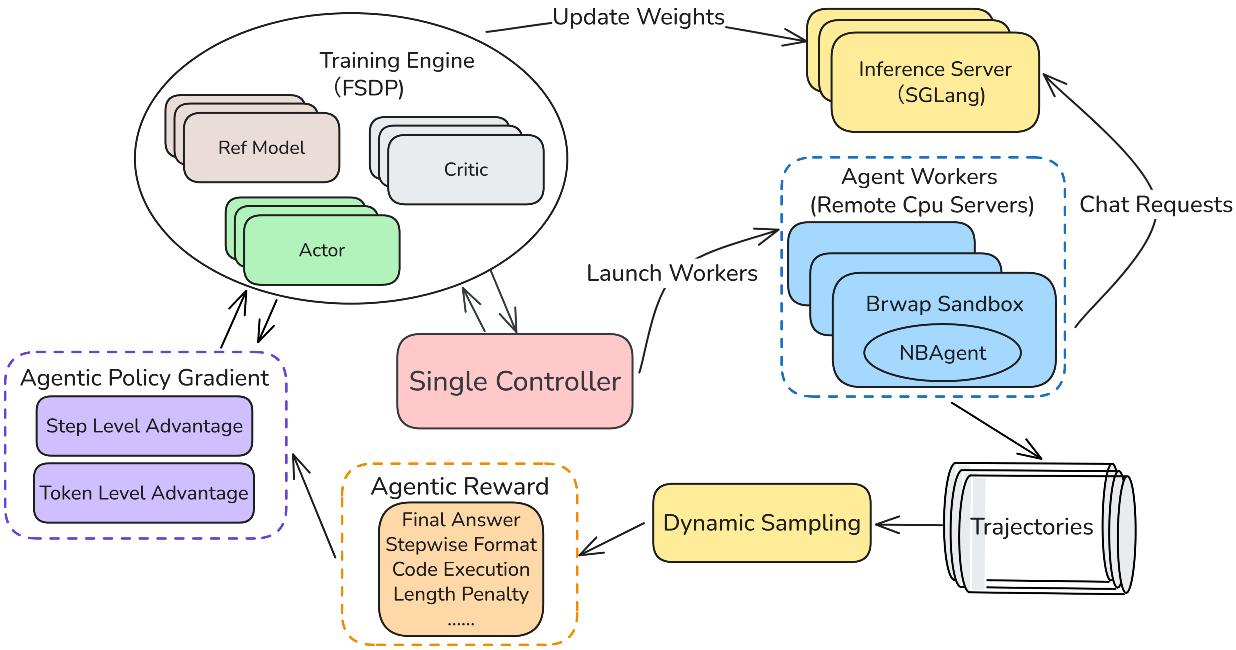

Figure 2: Agent RL Architecture

$$\pi_\theta(a_t, s_t) = P_\theta(a_t|s_t) = \prod_{i=1}^{|a_t|} P_\theta(tok_i|[s_t, tok_0, ..., tok_{i-1}])$$

, where tok_i is the i-th token generated in a_t.

Then, the gradient can be rewritten as:

$$\hat{g} = \hat{\mathbb{E}}_t[\nabla \log \prod_{i=1}^{|a_t|} P_\theta(tok_i|[s_t, tok_0, ..., tok_{i-1}])\hat{A}_t]$$

$$= \hat{\mathbb{E}}_t[\sum_{i=1}^{|a_t|} \nabla \log P_\theta(tok_i|[s_t, tok_0, ..., tok_{i-1}])\hat{A}_t]$$

In practice, we adopt the token-level normalization strategy proposed by DAPO to address the contribution of longer trajectories:

$$\hat{g} = \frac{1}{\sum_{j=1}^{N} \sum_{t=1}^{T_n} |a_t|} \sum_{n=1}^{N} \sum_{t=1}^{T_n} \hat{A}_t \sum_{i=1}^{|a_t|} \nabla \log P_\theta(tok_i|[s_t, tok_0, ..., tok_{i-1}])$$

, where N is the number of samples in a batch, T_n is num of steps of n-th trajectory, and $|a_t|$ is the length of the action sequence at time step t.

For the advantage estimation, $\hat{A}_t$, we adopt the approach from REINFORCE++:

$$\hat{A}_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

where $r_{t'}$ is the reward at time step t' and γ is the discount factor.

And we do a batch-step-wise advantage normalization to stabilize the training:

$$\hat{A}_t = \frac{\hat{A}_t - \mathbb{E}[\hat{A}_t]}{\sqrt{\mathrm{Var}[\hat{A}_t] + \epsilon}}$$

where the expectation and variance are computed over all timesteps of all trajectories within the batch, and ϵ is a small constant to avoid division by zero.

Our implementation adheres to a strictly on-policy training regime, meaning we do not use corrective mechanisms like importance sampling or PPO-style clipping. Furthermore, in line with common practices, we include an explicit KL-divergence penalty term in the objective function.

To enhance exploration and stabilize training, we employ a dynamic sampling strategy inspired by DAPO. This procedure involves generating multiple trajectories from the same input query and randomly discard trajectories that yield either zero or maximum core rewards.

3.2 Verified Reward for Agentic RL

To objectively measure the quality of a trajectory $\tau = (s_1, a_1), \ldots, (s_T, a_T)$, we design a verifiable reward $R(\tau)$ that combines multiple criteria for multi-step agentic training. This reward is verifiable in the sense that it is computed automatically from the agent's outputs and environment feedback, without requiring human judgments. Formally, $R(\tau)$ is composed of three components:

Final Answer Correctness This term rewards the agent for the quality of its submitted answer. For tasks with a ground-truth solution, the reward, R_{final}, is calculated as a weighted sum of three metrics:

$$R_{final} = 0.5 \cdot \text{EM} + 0.4 \cdot \text{F1} + 0.1 \cdot I(\text{has\_answer})$$

where: - **EM** is the Exact Match score (1 for a perfect match, 0 otherwise). - **F1** is the token-level F1 score, which measures content overlap. - I(**has_answer**) is a binary indicator that is 1 if the agent submits an answer and 0 otherwise.

This composite score provides a nuanced signal that strongly rewards perfect answers (EM weight), gives partial credit for substantial correctness (F1 weight), and lightly incentivizes the agent to provide a response rather than terminating without one.

Stepwise Format Compliance: This term gives reward for adhering to the required `<think></think>`and `<code></code>` output structure at each step. The agent is incentivized to output well-formed reasoning and code blocks that match the template (e.g. properly using the tags and not mixing thought text into code).

Code Execution & Correctness: This component rewards the agent for producing code actions that execute successfully and achieve their intended effect. If a `<code>` action compiles/runs without errors, the reward is positive.

3.3 Scalable Agent RL Infra

Training agents for long-horizon tasks that involve real-world interactions and code execution introduces significant infrastructure challenges, particularly concerning resource allocation and variable task completion times. To address this, we developed a scalable and robust infrastructure engineered with several key design principles:

- **Decoupled Architecture:** Our infrastructure is built on a decoupled architecture where **a pool of Agent Workers**, managed on remote CPU servers, handles all environment interaction and trajectory collection. This pool's workload is precisely controlled via a `max_concurrency` parameter to manage system load. Workers communicate with a dedicated Inference Server, running SGLang, via stateless HTTP requests to query the latest policy for actions. This separation of concerns—CPU-based environment simulation from GPU-based inference and training—prevents computational bottlenecks and allows each component to be scaled independently.
- **Lightweight, Isolated Agent Environments:** Each Agent Worker is encapsulated within **a secure sandbox using Bubblewrap**. This technology provides strong process isolation with minimal performance overhead. Critically, unlike traditional container solutions like Docker, Bubblewrap operates without requiring root privileges, which both enhances security and simplifies deployment across the distributed worker pool, making it ideal for large-scale parallelization.
- **Agent Execution Control:** To ensure system stability and consistent data throughput despite variations in task complexity, we implement **robust execution controls**. These include a timeout mechanism that

terminates workers exceeding a `time threshold` and a `maximum retry` limitation for tasks that repeatedly fail.

Collectively, these design choices create a high-throughput, scalable, and fault-tolerant agent rollout pipeline, providing the stable foundation required for effective large-scale reinforcement learning.

4 Experiment

4.1 Evaluation Datasets and Metrics

We comprehensively evaluate L0 on a set of open-domain multi-hop question answering tasks, categorized as follows: **HotpotQA** (Yang et al. 2018) represents the first large-scale dataset designed explicitly for multi-hop reasoning, requiring models to reason across multiple Wikipedia paragraphs; **2WikiMultiHopQA** (Ho et al. 2020) explicitly provides reasoning paths for assessing multi-step inference capabilities; **Musique** (Trivedi et al. 2022) consists of 2-to-4-hop reasoning questions constructed from five existing single-hop datasets; **Bamboogle** (Press et al. 2023) comprises complex queries that Google frequently answers incorrectly, intended to test models' cross-domain combinational reasoning abilities; **SimpleQA** (Wei et al. 2024) focus on short, fact-seeking queries.

We adopt Exact Match (EM), F1-score and LLM-as-a-judge (LJ) as our evaluation metric, considering a prediction correct if its normalized form exactly matches any one of the normalized ground-truth answers.

4.2 Baselines

To systematically evaluate the effectiveness of L0, we compare our approach against three categories of baseline methods: (1) **Basic Prompting Engineering:** This category includes Direct Prompting, Chain-of-Thought (CoT), and standard Retrieval-Augmented Generation (RAG). (2) **Advanced RAG Methods:** We consider RAgent (Jayasundara, Arachchilage, and Russello 2024) and Search-o1 (Li et al. 2025). RAgent employs a retrieval-based agentic policy-generation framework to systematically organize internal knowledge, while Search-o1 integrates an agent-based search workflow into large-scale reasoning processes. (3) **Reinforcement Learning-based Methods:** This category includes DeepSeek-R1 (Guo et al. 2025), Search-R1 (Jin et al. 2025), and ZeroSearch (Sun et al. 2025). In DeepSeek-R1, the policy model performs deep reasoning relying solely on internal knowledge. Search-R1 allows the policy model to interact repeatedly with an actual search engine during inference. ZeroSearch further incorporates simulated search during training to incentivize the LLM's ability to leverage real-world search engines effectively.

During the evaluation process, all baseline methods consistently employ SerpAPI as the search engine, with their performance and experoment settings adopted from those reported in ZeroSearch.

4.3 Experiment Settings

We employ Qwen2.5-7B-Instruct, Qwen2.5-32B-Instruct and Qwen3-4B-Thinking [yang2025qwen3] as backbone models, configuring the generation parameters with a max-tokens of 32,768 and a top-p sampling threshold of 0.9. To simulate realistic retrieval scenarios, we use Google Web Search via SerpAPI as the external search engine and retrieve webpage content using the Jina Reader API. Furthermore, we introduce a novel data filtering and stratification strategy based on objectivity, temporal stability, and question difficulty, selecting a total of 20K QA pairs from the training sets of 2WikiMultihopQA (Ho et al. 2020), TriviaQA (Joshi et al. 2017), NQ (Kwiatkowski et al. 2019), and HotpotQA (Yang et al. 2018) datasets for reinforcement learning.

4.4 Main Result

In Table 1, we report two configurations of our method: L0-Scaffold (NB-Agent + Qwen, without RL training), and L0-RL (NB-Agent + Qwen, training with AgentRL).

The L0-Scaffold configuration instantiates the NB-Agent's structured reasoning template but without RL training, beating the Direct Answer (11.87%) and RAG (17.33%) on average. This scaffold explicitly separates reasoning, query generation, and answering in a multi-turn loop, which enforces a structured decision-making process to enhance the transparency and reliability. It is noteworthy that due to the inherent complexity of the Scaffold architecture, the performance prior to RL training is comparable to that of Search-o1.

L0-RL further applies reinforcement learning to boost performance by teaching the model to use the equipped tools and memory management effectively. Compared with L0-Scaffold, L0-RL demonstrates a substantial performance improvement, with the average score dramatically increasing from 18.57% to 38.28%. By training the agent to learn from the verifiable outcomes of its self-generated code, AgentRL teaches it to master its own tools, manage memory effectively, and develop sophisticated, self-correcting strategies. The agent evolves from a structured but naive operator into a highly autonomous and proficient problem-solver.

L0-RL's gains compare favorably to other RL-driven LLM Agents, outperforming the Search-R1 and ZeroSearch by a clear margin on average, indicating that the L0 scaffold provides a far richer and more expressive environment for reinforcement learning. While other methods train agents to learn when to call a single tool (e.g., a search engine), our framework trains the agent to become a programmatic problem-solver, learning how to compose actions, manage state, and reason within a structured environment. This synergy between an advanced agent architecture and a tailored RL training process enables a more capable and generalizable form of agentic intelligence.

Finally, we analyze performance across three different backbone models to understand the effects of scale and inherent model capabilities, with results shown in Table 2. As expected, scaling the model size from 7B to 32B elevates the baseline performance of L0-Scaffold from 20.88% to 43.50% on EM score, showing that a more stronger foundation model is better able to leverage the structured environment provided by the scaffold. More interestingly, the Qwen-3-4B-Thinking model, which is pre-trained for robust tool-use capabilities, shows the most pronounced improvement after RL training. Although its L0-Scaffold score starts lower (14.78% on EM and 18.65% on F1), it achieves an L0-RL score of 44.67% on EM and 54.36% on F1. This suggests a powerful synergy that models with inherent reasoning and tool-use abilities are exceptionally receptive to AgentRL training. They can more effectively utilize the fine-grained learning signals to unlock their full potential, resulting in even more dramatic performance enhancements.

Table 1: Main results of different methods. The best performance is set in bold.

Method	**Multi-Hop QA**			
	HotpotQA	**Musique**	**Bamboogle**	**Avg.**
Qwen-2.5-7B-Base/Instruct				
Direct Answer	16.40	4.80	14.40	11.87
CoT	16.20	6.60	24.00	15.60
RAG	25.80	9.40	16.80	17.33
RA-Agent	19.60	7.60	28.00	18.40
Search-o1	17.00	8.60	30.40	18.67
R1-base	21.00	9.80	27.78	19.53
R1-instruct	21.60	8.40	25.00	18.33
Search-R1-base	31.20	18.20	30.56	26.65
Search-R1-inst	32.80	17.40	26.39	25.53
ZeroSearch-base	32.00	18.00	33.33	27.78
ZeroSearch-inst	34.60	**18.40**	27.78	26.93
Our Methods				
L0-Scaffold	22.03	8.33	31.20	20.52
L0-RL	**40.63**	16.60	**57.60**	**38.28**

Table 2: Main results of L0 using different LLMs as backbone.

Method	**HotpotQA**		**Musique**		**Bamboogle**		**SimpleQA**			**Avg.**	
	EM	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**	**F1**	**LJ**	**EM**	**F1**
Qwen-3-4B-Thinking											
L0-Scaffold	14.17	18.47	4.28	7.13	18.40	21.52	22.26	27.49	29.46	14.78	18.65
L0-RL	38.74	49.94	16.03	23.20	60.53	68.59	63.36	75.69	81.59	44.67	54.36
Qwen-2.5-7B-Instruct											
L0-Scaffold	22.03	29.91	8.33	14.65	31.20	36.15	21.94	28.68	30.45	20.88	27.35
L0-RL	40.63	52.42	16.60	25.40	57.60	68.05	61.68	75.12	80.40	44.13	55.25
Qwen-2.5-32B-Instruct											
L0-Scaffold	38.48	50.72	14.78	23.11	59.20	68.96	51.53	64.35	69.46	41.00	51.79
L0-RL	**46.14**	**59.16**	**20.22**	**30.03**	**65.07**	**73.70**	**67.58**	**81.28**	**87.29**	**49.75**	**61.04**

4.5 Analysis: Impact of Task Difficulty and Dynamic Sampling

To better understand the training dynamics of our AgentRL framework and the challenges posed by complex tasks, we analyze agent behavior under different data difficulty settings: (1) low-difficulty QA tasks. (2) high-difficulty QA tasks. (3) high-difficulty QA tasks trained using a Dynamic Sampling strategy. On the `easy_dataset`, the agent exhibits stable and predictable learning. In sharp contrast, training on the `hard_dataset` without any mitigation strategy leads to significant training instability and eventual model collapse.

Through RL with a reward on answer and execution accuracy, we observe that during training the policy quickly learns to eliminate unnecessary or uninformative queries. For example, the number of redundant search turns drops sharply as training progresses. It indicates the self-correction to learn the correct format and begin to eliminate unnecessary steps, thereby focusing its reasoning and retrieval more effectively. The average response length in Figure 3 (b) and the number of steps to completion in Figure 3 (d) both explode in the latter half of training. This indicates that the agent enters a state of inefficient, repetitive looping, generating excessively long and often incoherent thought processes in a failing attempt to find a solution. This ultimately leads to a degenerate policy where the agent can no longer produce well-structured or valid actions, causing the observed collapse in format and execution rewards. The `dynamic_sampling-hard` configuration demonstrates the effectiveness of our proposed mitigation strategy. By dynamically adjusting the sampling of tasks, we can maintain training stability even on the most challenging dataset. As shown in Figure 3 (a) and Figure 3 (c), both the Code Execution and Format Compliance rewards remain stable throughout the training process, avoiding the collapse seen in the standard hard_dataset configuration.

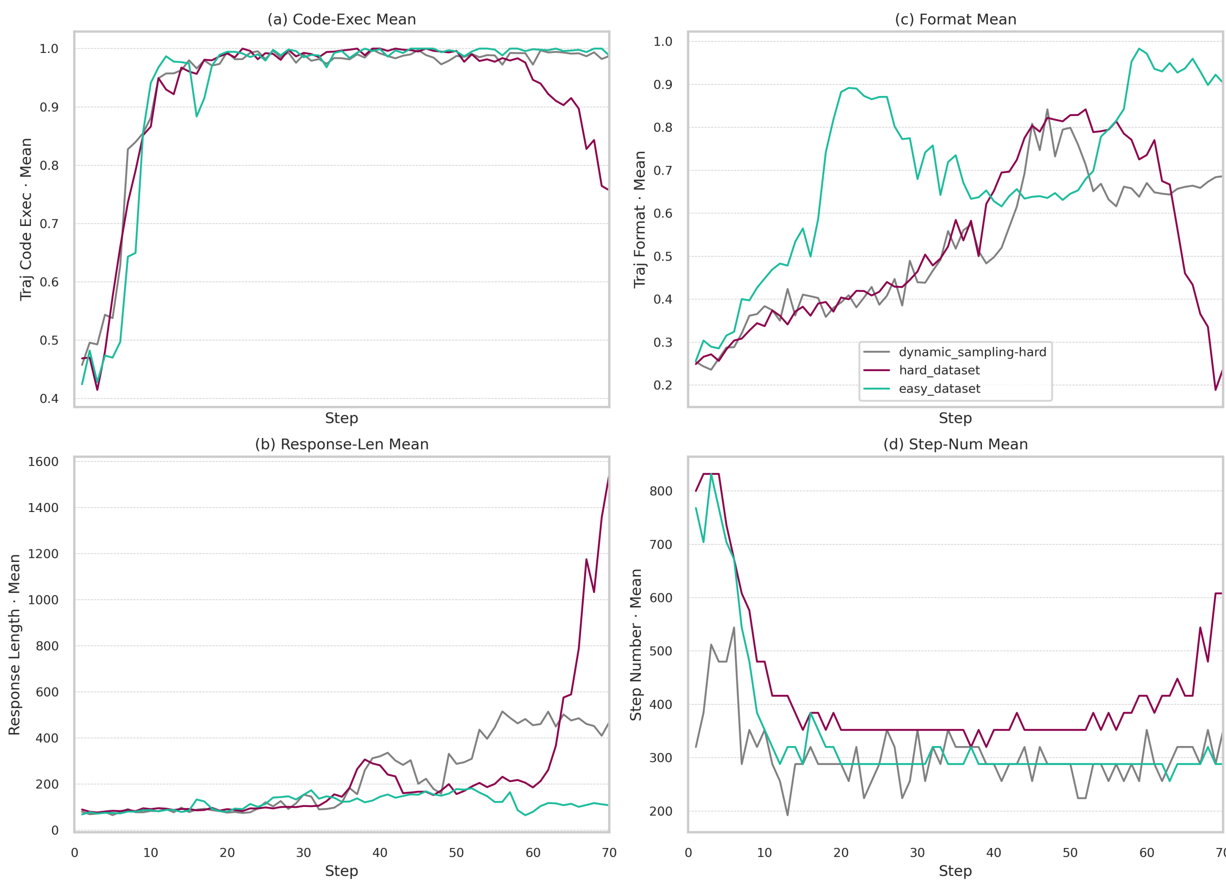

Figure 3: Agent RL Architecture

Guo, Daya, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, et al. 2025. "Deepseek-R1: Incentivizing Reasoning Capability in Llms via Reinforcement Learning." *arXiv Preprint arXiv:2501.12948*.

Ho, Xanh, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. "Constructing a Multi-Hop QA Dataset for Comprehensive Evaluation of Reasoning Steps." In *Proceedings of the 28th International Conference on Computational Linguistics*, 6609–25.

Hu, Jian. 2025. "REINFORCE++: A Simple and Efficient Approach for Aligning Large Language Models." *arXiv e-Prints*, arXiv–2501.

Jayasundara, Sakuna Harinda, Nalin Asanka Gamagedara Arachchilage, and Giovanni Russello. 2024. "RAGent: Retrieval-Based Access Control Policy Generation." *arXiv Preprint arXiv:2409.07489*.

Jin, Bowen, 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*.

Joshi, Mandar, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. 2017. "TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension." In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1601–11.

Kwiatkowski, Tom, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, et al. 2019. "Natural Questions: A Benchmark for Question Answering Research." *Transactions of the Association for Computational Linguistics* 7: 452–66.

Li, Xiaoxi, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025. "Search-O1: Agentic Search-Enhanced Large Reasoning Models." *arXiv Preprint arXiv:2501.05366*.

Press, Ofir, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2023. "Measuring and Narrowing the Compositionality Gap in Language Models." In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 5687–5711.

Sun, Hao, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. 2025. "Zerosearch: Incentivize the Search Capability of Llms Without Searching." *arXiv Preprint arXiv:2505.04588*.

Trivedi, Harsh, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. " MuSiQue: Multihop Questions via Single-Hop Question Composition." *Transactions of the Association for Computational Linguistics* 10: 539–54.

Wang, Xingyao, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. "Executable Code Actions Elicit Better LLM Agents." In *International Conference on Machine Learning*, 50208–32. PMLR.

Wei, Jason, Nguyen Karina, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. 2024. "Measuring Short-Form Factuality in Large Language Models." *arXiv Preprint arXiv:2411.04368*.

Yang, Zhilin, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. "HotpotQA: A Dataset for Diverse, Explainable Multi-Hop Question Answering." In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2369–80.

Yu, Qiying, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, et al. 2025. "DAPO: An Open-Source LLM Reinforcement Learning System at Scale." *CoRR*.