VerlTool: Towards Holistic Agentic Reinforcement Learning with Tool Use

Anonymous authors
Paper under double-blind review

Abstract

Reinforcement Learning with Verifiable Rewards (RLVR) has demonstrated success in enhancing LLM reasoning capabilities, but remains limited to single-turn interactions without tool integration. While recent **A**gentic **R**einforcement **L**earning with **T**ool use (ARLT) approaches have emerged to address multi-turn tool interactions, existing works develop task-specific codebases that suffer from fragmentation, synchronous execution bottlenecks, and limited extensibility across domains. These inefficiencies hinder broader community adoption and algorithmic innovation. We introduce **VerlTool**, a unified and modular framework that addresses these limitations through systematic design principles. **VerlTool** provides four key contributions: **(1)** upstream alignment with VeRL ensuring compatibility and simplified maintenance, **(2)** unified tool management via standardized APIs supporting diverse modalities including code execution, search, SQL databases, and vision processing, **(3)** asynchronous rollout execution achieving near 2× speedup by eliminating synchronization bottlenecks, and **(4)** comprehensive evaluation demonstrating competitive performance across 6 ARLT domains. Our framework formalizes ARLT as multi-turn trajectories with multi-modal observation tokens (text/image/video), extending beyond single-turn RLVR paradigms. We train and evaluate models on mathematical reasoning, knowledge QA, SQL generation, visual reasoning, web search, and software engineering tasks, achieving results comparable to specialized systems while providing a unified training infrastructure. The modular plugin architecture enables rapid tool integration requiring only lightweight Python definitions, significantly reducing development overhead and providing a scalable foundation for tool-augmented RL research.

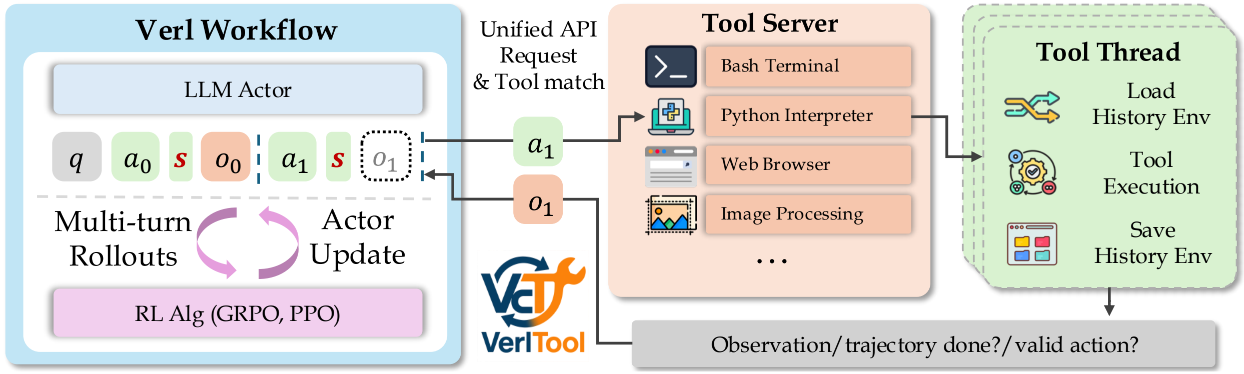


Figure 1: Overview of the **VerlTool**, a modularized and efficient framework for the **A**gentic **R**einforcement **L**earning with **T**ool Use (ARLT) training paradigm, where the RL workflow and tool execution are fully disaggregated for both efficiency and extensibility.

1 Introduction

> *"We shape our tools, and thereafter our tools shape us."* — Marshall McLuhan

Large language models (LLMs) such as OpenAI's O-series (Jaech et al., 2024) and DeepSeek-R1 (Guo et al., 2025) have recently achieved striking advances, surpassing top human performers in challenging domains like mathematics (AIME) and programming (LiveCodeBench (Jain et al., 2024), Codeforces (Quan et al., 2025)). A central driver of this progress is the paradigm of *reinforcement learning with verifiable rewards* (RLVR), which strengthens long-context reasoning during training. Through RLVR, LLMs exhibit emergent cognitive behaviors such as reflection, backtracking, and multi-step reasoning.

Yet these systems remain constrained in a fundamental way: they are unable to interact with the external world. Current LLM reasoning unfolds in a closed, single-turn setting without environmental feedback, often leading to brittle behaviors such as overthinking (Chen et al., 2024a) or hallucination (Yao et al., 2025). Conceptually, these models resemble a "brain in a vat", locked into self-contained simulations without grounding in interactive or physical reality.

To overcome this isolation, a parallel line of work explored augmenting LLMs with tool-use ability. Systems such as ToolFormer (Schick et al., 2023) and OpenHands (Wang et al., 2024b) extend models with supervised training on synthetic tool-use data, enabling practical interaction with code interpreters, search engines, or APIs. However, they primarily rely on imitation learning. They lack the agentic autonomy needed to learn directly from feedback and refine their behavior adaptively in open-ended environments.

Recent research begins to bridge this gap by combining tool use with RLVR, giving rise to a new paradigm we term **ARLT**—**A**gentic **R**einforcement **L**earning with **T**ool use. In ARLT, LLMs can actively engage with external tools such as code execution environments (Li et al., 2025c), search engines (Jin et al., 2025), image manipulators (Su et al., 2025), and domain-specific APIs (Feng et al., 2025). This interaction transforms training into a multi-turn, feedback-rich process that not only improves efficiency and reduces token usage but also fosters more robust agentic behaviors.

However, enabling ARLT poses significant challenges from a systems perspective. First, *rollout efficiency* becomes critical: multi-tool trajectories unfold asynchronously, with different tools producing results at varying speeds, demanding scalable asynchronous execution. Second, *tool management* remains fragmented: existing ARLT codebases are often tailored to specific tools, making it difficult to extend or reproduce results. Finally, *multimodal support* is still underdeveloped: while most RL frameworks focus narrowly on text, emerging multimodal reasoning agents (e.g., Pixel-Reasoner (Su et al., 2025)) require handling tool outputs that include images, videos, or other structured modalities in a unified design.

These barriers have slowed community progress, limiting reproducibility, extensibility, and algorithmic innovation. To address them, we introduce VerlTool: an open-source, user-friendly, and efficient framework built on top of VeRL (Sheng et al., 2024), designed explicitly for ARLT that supports both text and multi-modal training. Unlike prior systems, VerlTool enables multi-turn, stateful agentic training with tool use through four key contributions:

- **Upstream Alignment.** VerlTool inherits VeRL as a submodule, ensuring compatibility with upstream updates. This modular separation between RL training and agentic interaction simplifies maintenance and accelerates framework evolution.
- **Unified Tool Management.** We introduce a dedicated tool server with standardized interaction APIs, supporting diverse tools such as code execution, search, SQL/tabular reasoning, and vision utilities. Adding a new tool requires only a lightweight Python definition file, streamlining extensibility for both training and evaluation.
- **Asynchronous Rollouts.** By interacting with tool servers on a trajectory-by-trajectory basis rather than synchronously batch by batch, VerlTool eliminates idle waiting time. This design yields over 2× speedup during rollout execution.
- **Diverse ARLT Tasks.** We have implemented and tested VerlTool on six ARLT tasks, including Math, Search Retrieval, SQL, Visual Reasoning, Web Browsing, and SWE-Bench, achieving com-

petitive performance with previous baselines while trained in a unified framework. We also present common findings in the agentic RL setting across these tasks.

In summary, VerlTool provides a principled, extensible, and efficient framework for ARLT, bridging the gap between isolated LLM reasoning and interactive agentic intelligence. By combining upstream-aligned RL infrastructure, unified tool integration, asynchronous execution, and diverse tasks, it paves the way for scalable research and practical deployment of LLMs as tool-using agents.

2 Related Work

2.1 Reinforcement Learning for Agentic Tool Use

The integration of reinforcement learning with tool use has emerged as a powerful paradigm for developing adaptive LLM agents. Early tool-calling approaches relied on prompt-based orchestration (Yao et al., 2022; Lu et al., 2023; Shen et al., 2023), building on Chain-of-Thought reasoning (Wei et al., 2022) and multi-agent frameworks (Wu et al., 2024; Hong et al., 2023) for training-free tool invocation. While instruction-tuned models (Schick et al., 2023; Kong et al., 2023; Gou et al., 2023) learned structured calling patterns through supervised learning, they remained largely static and limited to single-turn interactions.

Recent work has demonstrated the advantages of reinforcement learning for tool use, enabling models to optimize their tool-calling policies based on execution outcomes and environmental feedback (Li et al., 2025c; Feng et al., 2025; Moshkov et al., 2025; Wang et al., 2025a). This paradigm, which we refer to as *Agentic Reinforcement Learning with Tool use (ARLT)*, extends beyond single-turn verification to support long-horizon, multi-turn interactions. Key characteristics of ARLT include: (1) credit assignment across sequential tool calls, (2) explicit handling of observation tokens from tool responses, and (3) integration with robust, failure-aware execution environments (Plaat et al., 2025; Ke et al., 2025).

This shift from static instruction-following to dynamic, feedback-driven learning has shown effectiveness across diverse domains, including mathematical reasoning with code execution, information retrieval, natural language to SQL generation, and visual reasoning tasks. These applications require agents to probe environments iteratively, adapt to tool feedback, and refine their strategies—capabilities that are difficult to achieve through purely supervised approaches.

2.2 Agentic RL Training Frameworks

Table 1: Tool support comparison across different frameworks (up to update until August 23, 2025). RAGEN and ROLL focus on the puzzle environments like bandit, which we did not list here.

Framework	**FAISS Search**	**Python Executor**	**Web Search**	**Bash Terminal**	**SQL Executor**	**Image Processing**
OpenRLHF (Hu et al., 2024)	✓	✓	×	×	×	×
VeRL (Sheng et al., 2024)	✓	✓	×	×	×	×
ROLL (Wang et al., 2025b)	×	×	×	×	×	×
RAGEN (Wang et al., 2025c)	×	×	×	×	×	×
Slime (THUDM, 2024)	✓	×	×	×	×	×
AReaL (Fu et al., 2025)	✓	×	✓	×	×	×
SkyRL (Cao et al., 2025)	×	✓	×	✓	✓	×
VerlTool(ours)	✓	✓	✓	✓	✓	✓

The success of Reinforcement Learning from Verifier Rewards (RLVR) has motivated the development of various frameworks to support scalable RL training for language models. Established synchronous frameworks include OpenRLHF (Hu et al., 2024) and VeRL (Sheng et al., 2024), which employ Ray-based distributed computing to manage training workflows. Fully asynchronous frameworks such as AReaL (Fu et al., 2025), ROLL (Wang et al., 2025b), and Slime (THUDM, 2024) also emerged to address scalability challenges.

As shown in Table 1, existing frameworks exhibit varying degrees of tool support. Traditional RL frameworks such as OpenRLHF and VeRL provide basic support for search and code execution tools but lack comprehensive multi-modal capabilities. ROLL focuses primarily on core RL training without extensive tool integration, while AReal supports search functionality but has limited executor capabilities. SkyRL (Cao et al., 2025) offers broader tool support, including bash terminals and SQL executors, but requires complex containerized environments that introduce deployment overhead. The limited tool coverage in existing frameworks has led to the development of domain-specific systems (e.g., Search-R1, PixelReasoner, and ToolRL) as task-specific extensions. However, these implementations typically feature hard-coded tool integrations that limit their extensibility and adaptability to new domains. As evident from Table 1, there remains a need for frameworks that provide comprehensive, extensible support for diverse tool types while maintaining ease of deployment and development.

Table 2: Comparison of agentic reinforcement learning frameworks. The table shows key features including feedback format support, asynchronous rollout capabilities, tool execution characteristics, resource management systems, and vision-language model (VLM) compatibility. VerlTool (separated by dashed line) demonstrates comprehensive support across all evaluated dimensions.

Agentic RL Frameworks	Tool/Env Feedback Format	Async Rollout	Independent Tool Server	Stateful Tool Execution?	Tool Resource Management	VLM Support?
OpenRLHF (gem)	Text	✗	✓	✗	-	✗
VeRL (MCP)	Text	✓	✗	✗	-	✓
ROLL (gem)	Text	✗	✓	✗	-	✗
RAGEN	Reward	✗	-	✓	-	✗
SLIME	Text	✓	✗	✗	-	✗
AREAL	Text	✓	✗	✗	-	✓
SKYRL	Text	✓	✓	✓	k8s	✗
Agent-Lightning	Text	✓	✓	✗	-	✗
VerlTool	Text/ Multi-modal/ Reward	✓	✓	✓	Ray	✓

Apart from functionality advantages, VerlTool also excels at diverse technical dimensions, such as stateful tool execution, tool resource management, and support of diverse tool feedback formats. As Table 2 clearly demonstrates, our framework addressed several core technical constraints that previous works suffer from, including **Tool Resource Management**, **Support for Stateful Tool Execution**, and **Broader Tool Feedback**:

Stateful Tool Execution. A key distinction is VerlTool's **per-trajectory stateful environment**. Existing frameworks (GEM, ROLL, OpenRLHF, SLIME) treat tools as stateless functions: each call is independent and cannot reference memory from previous steps. Agentic RL tasks such as multi-step code execution, iterative SQL debugging, or image refinement fundamentally require persistent state. VerlTool supports this through lightweight trajectory-scoped environments that maintain and update state across the entire rollout.

Tool Resource Management. Many prior frameworks rely on **OS-level multiprocessing** for tool execution. This approach lacks flexible scheduling and offers no control over CPU/GPU contention across concurrent trajectories. Modern tools (Python interpreters, image/video operations, search retrievers) introduce substantial and heterogeneous resource demands. VerlTool manages these resources explicitly via **Ray actors**, enabling isolation, fault tolerance, batching, throttling, and cross-node scalability, while also controlling the lifecycle of per-trajectory stateful environments (creation, reuse, and cleanup) uniformly.

Diverse Forms of Tool Feedback. Unlike prior frameworks that restrict tools to text-only outputs, VerlTool supports **multimodal** feedback (images, videos, etc) and integrates these observations directly into the trajectory token stream. Moreover, the tool server can emit **action-level rewards** (e.g., execution correctness, SQL validity, retrieval quality) immediately upon each tool call, enabling fine-grained credit assignment and stable multi-turn optimization. This combination of multimodal observations and per-action reward signals is not supported in existing ARLT frameworks and is essential for complex, interleaved tool-use tasks.

3 VerlTool Framework

In this section, we formulate the conceptual foundation of the ARLT paradigm starting from the original RLVR setting. We then elaborate on how VerlTool serves as a practical implementation on the server side for **A**gentic **R**einforcement **L**earning with **T**ool use (**ARLT**).

3.1 Preliminaries

RLVR Reinforcement learning with verifiable reward (RLVR) optimizes the language model using a predefined verifiable reward via the following objective:

$$\max_{\pi_\theta} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_\theta(\cdot|x)} \left[R_\phi(x, y) \right] - \beta \, \mathbb{D}_{\text{KL}} \left[\pi_\theta(y \mid x) \, \| \, \pi_{\text{ref}}(y \mid x) \right], \tag{1}$$

where π_θ denotes the policy LLM, π_{ref} is the reference LLM, R_ϕ is the verifiable reward function, and $\mathbb{D}_{\text{KL}}$ is the KL divergence. The input x is drawn from the dataset $\mathcal{D}$, and y is the corresponding single-turn output. A typical verifiable reward function is defined as:

$$R_\phi(x, y) = \begin{cases} 1 & \text{if match}(y, y_g) \\ -1 & \text{otherwise} \end{cases} \tag{2}$$

where y_g is the ground-truth answer and $\text{match}(\cdot, \cdot) \in \{1, 0\}$ is a verification function that determines whether the generated answer y matches y_g. This function can be implemented using either rule-based approaches (Wang et al., 2024b) or model-based verifiers (Ma et al., 2025).

GRPO (Shao et al., 2024) is a widely adopted RL algorithm designed to optimize the objective in Equation 1. In the single-turn RL case, the trajectory is simply the LLM generation $\tau = \{y\}$. The GRPO objective is given by:

$$J_{\text{GRPO}}(\theta) = \frac{1}{G} \sum_{i=1}^{G} \frac{1}{|\tau_i|} \sum_{t=1}^{|\tau_i|} \min \left[r_{i,t}(\theta) \cdot \hat{A}_{i,t}, \, \text{clip} \left(r_{i,t}(\theta), 1 - \epsilon, 1 + \epsilon \right) \cdot \hat{A}_{i,t} \right], \tag{3}$$

where $r_{i,t}(\theta)$ is the token-level importance ratio and $\hat{A}_{i,t}$ is the normalized advantage across all tokens:

$$r_{i,t}(\theta) = \frac{\pi_\theta(\tau_{i,(t)} | \tau_{i,<t})}{\pi_{\text{old}}(\tau_{i,(t)} | \tau_{i,<t})}, \quad \hat{A}_{i,t} = \frac{R_\phi(x, y) - \text{mean}(\{R_\phi(\tau_1), \ldots, R_\phi(\tau_G)\})}{\text{std}(\{R_\phi(\tau_1), \ldots, R_\phi(\tau_G)\})}. \tag{4}$$

3.2 Agentic Reinforcement Learning with Tool Use

ARLT. In the agentic RL setting, rollouts are *multi-turn* instead of single-turn, and the agent can *interact with tools* to receive external observations during the reasoning process. Thus, the trajectory can be written as $\tau = \{a_0, o_0, \ldots, a_{n-1}, o_{n-1}, a_n\}$, where a_i denotes the LLM-generated action tokens and o_i denotes the observation tokens returned by a tool call. Here, n is the total number of interaction steps.

To determine whether an action a_i invokes a specific tool, we assume that each a_i (for $0 \leq i < n$) ends with a stop token $s \in \mathbb{S}_k$, where $\mathbb{S}_k$ is the predefined set of stop tokens for tool $T_k \in \mathbb{T}$. For example, $\mathbb{S}_{\text{CI}} = \{\texttt{```output}, \texttt{</python>}\}$ for a code interpreter tool, or $\mathbb{S}_{\text{search}} = \{\texttt{</search>}\}$ for a search tool. The complete set of stop tokens is the union over all invoked tools: $\mathbb{S} = \bigcup_{k=1}^{|\mathbb{T}|} \mathbb{S}_k$.

The introduction of observation tokens o_i makes ARLT fundamentally different from the agentic RL defined in RAGEN (Wang et al., 2025c), where the agent only receives scalar rewards through environmental interaction. Moreover, the observation tokens are off-policy with respect to the current LLM π_θ being optimized, which can destabilize training (Jin et al., 2025). Therefore, these tokens are typically masked out during policy optimization. Let T_j be the token index of the first token in action segment a_j, then the GRPO loss for ARLT becomes:

$$J_{\text{GRPO-ARLT}}(\theta) = \frac{1}{G}\sum_{i=1}^{G}\frac{1}{\sum_{j=0}^{n}|a_j|}\sum_{j=0}^{n}\sum_{t=T_j}^{T_j+|a_j|}\min\left[r_{i,t}(\theta)\cdot\hat{A}_{i,t},\ \text{clip}\left(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon\right)\cdot\hat{A}_{i,t}\right], \tag{5}$$

3.3 Framework Design

Challenges. Building a general RL training framework that supports various tools is inherently challenging due to the additional overhead introduced by tool interactions. First, prior ARLT works are designed around a single tool and tightly couple the tool interaction logic with the core RL training loop, making extending or substituting tools challenging (Li et al., 2025c). This fragmentation increases the development burden for researchers seeking to experiment with novel tools or multi-tool scenarios. Second, these systems often rely on synchronous rollout mechanisms that process trajectories per-batch (Jin et al., 2025). In such settings, the tool interaction phase is triggered only after all actions a_i in a batch have been generated, resulting in idle bubbles and inefficient utilization of computational resources, especially on GPUs. To address these issues, we propose VERLTOOL, a general-purpose ARLT framework designed to support various tools as modular plugins via a unified API. Our goal is to minimize the integration overhead for community developers and provide a more efficient and extensible infrastructure for training LLMs with tool-use capabilities.

Overview. As shown in Figure 1, **VerlTool** adopts a modular and decoupled architecture consisting of two main components: the **VeRL Workflow** and the **Tool Server**, connected via a unified API. This separation enables independent management of RL training and tool execution while preserving efficient communication between them.

The **VeRL Workflow** handles all reinforcement learning activities, including multi-turn rollouts and actor updates. The LLM actor interacts with the external environment by generating a sequence of actions $\{a_0, a_1, \dots\}$, each potentially triggering a tool interaction. Once an action is identified as tool-invoking (via matching a predefined stop token), it is sent to the **Tool Server** along with auxiliary metadata. The observation o_i returned by the tool is then appended to the rollout, enabling observation-aware agent behavior and reward computation.

Asynchronous Rollout Design. A key feature of VERLTOOL is its support for fully *asynchronous rollouts*, which avoids the inefficiency of traditional synchronous batch-based frameworks. In such a setting, tool calls are processed only after the entire batch has completed generating their respective actions a_i, resulting in idle "bubbles" in GPU and CPU utilization. In contrast, VERLTOOL enables each trajectory to interact with the tool server independently and immediately upon finishing its action generation, as shown in Figure 2. This design ensures that tool execution latency does not block the entire batch, significantly improving throughput and system utilization in large-scale distributed settings. As shown in Table 4, the actor and environment evolve concurrently, achieving near 2 times speedup for the rollout stage.

Modular Tool-as-Plugin Design. As illustrated in Figure 3, VERLTOOL adopts a modular plugin system that cleanly abstracts tool interaction as an interface between the LLM actor and its external environment. Each tool is implemented as a subclass of a unified `BaseTool`, enabling seamless registration and extensibility. During rollouts, the actor's action a_i is parsed by `parse_action` to determine whether it invokes a tool; valid calls are routed to the appropriate module, which retrieves the trajectory state via `load_env`. The tool then executes its `conduct_action`, returning the observation o_i, a validity flag, and a termination flag for next action generation. We also maintain per-trajectory environments through lightweight state dictionaries, updated via `update_env` and cleared at the end of an episode with `delete_env`. By decoupling tool logic from the training workflow, developers can add new tools with minimal overhead, while the framework dynamically manages their execution across threads or distributed workers.

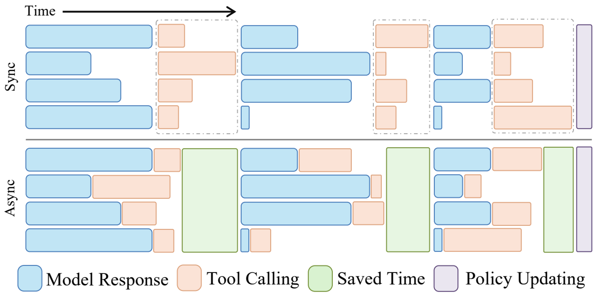
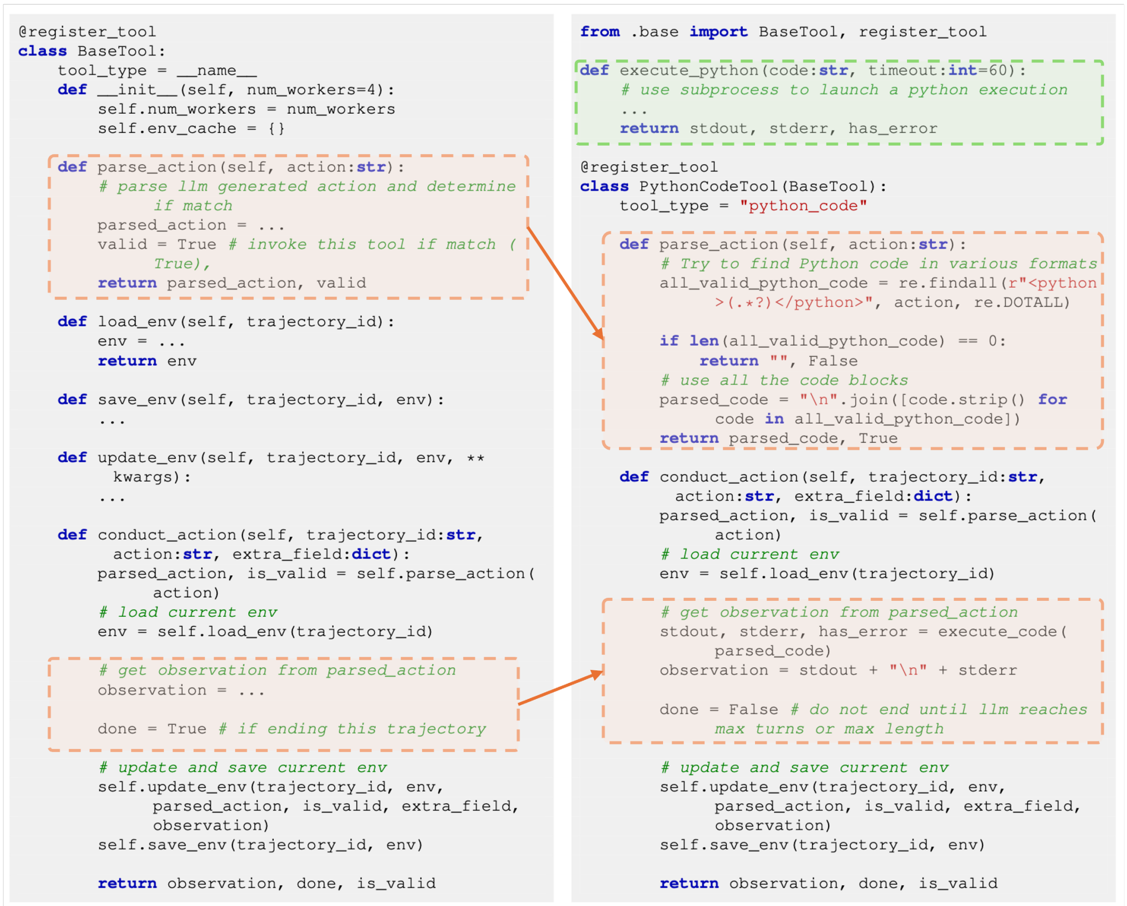


Figure 2: Visualization of the Async Rollout pipeline design and its effect in saving time.

Table 3: Performance comparison of Synchronous vs Asynchronous approaches. Experiments conducted on 8 H100 GPUs.

Metrics	Math-TIR	SQL	DeepSearch
Turns	4	5	5
Sync (s)	87	111	193
Async (s)	66	91	98
Speed Up (×)	**1.32**	**1.22**	**1.97**



```
@register_tool
class BaseTool:
    tool_type = __name__
    def __init__(self, num_workers=4):
        self.num_workers = num_workers
        self.env_cache = {}

    def parse_action(self, action:str):
        # parse llm generated action and determine
              if match
        parsed_action = ...
        valid = True # invoke this tool if match (
              True),
        return parsed_action, valid

    def load_env(self, trajectory_id):
        env = ...
        return env

    def save_env(self, trajectory_id, env):
        ...

    def update_env(self, trajectory_id, env, **
          kwargs):
        ...

    def conduct_action(self, trajectory_id:str,
          action:str, extra_field:dict):
        parsed_action, is_valid = self.parse_action(
              action)
        # load current env
        env = self.load_env(trajectory_id)

        # get observation from parsed_action
        observation = ...

        done = True # if ending this trajectory

        # update and save current env
        self.update_env(trajectory_id, env,
              parsed_action, is_valid, extra_field,
              observation)
        self.save_env(trajectory_id, env)

        return observation, done, is_valid
```

```
from .base import BaseTool, register_tool

def execute_python(code:str, timeout:int=60):
    # use subprocess to launch a python execution
    ...
    return stdout, stderr, has_error

@register_tool
class PythonCodeTool(BaseTool):
    tool_type = "python_code"

    def parse_action(self, action:str):
        # Try to find Python code in various formats
        all_valid_python_code = re.findall(r"<python
              >(.*?)</python>", action, re.DOTALL)

        if len(all_valid_python_code) == 0:
            return "", False
        # use all the code blocks
        parsed_code = "\n".join([code.strip() for
              code in all_valid_python_code])
        return parsed_code, True

    def conduct_action(self, trajectory_id:str,
          action:str, extra_field:dict):
        parsed_action, is_valid = self.parse_action(
              action)
        # load current env
        env = self.load_env(trajectory_id)

        # get observation from parsed_action
        stdout, stderr, has_error = execute_code(
              parsed_code)
        observation = stdout + "\n" + stderr

        done = False # do not end until llm reaches
              max turns or max length

        # update and save current env
        self.update_env(trajectory_id, env,
              parsed_action, is_valid, extra_field,
              observation)
        self.save_env(trajectory_id, env)

        return observation, done, is_valid
```

Figure 3: Example of code design for adding a new tool in VerlTool via the plugin interface.

Tokenization. A practical challenge in multi-turn agentic RL is how to tokenize tool observations and concatenate them with preceding LLM actions. Two strategies exist: (i) tokenize the action and observation strings separately, then append their token sequences; or (ii) concatenate the raw strings first and tokenize jointly. Although they yield the same sequence most of the time, discrepancies may arise for some specific combinations, as illustrated in Figure 5.

The action string "`</python>`" and observation "`\n<result>`" produce consistent tokens under the first strategy (*On-Policy*), whereas the second one merges boundary symbols into a different token id (e.g., `29,198` vs. `397`), which changed the LLM-generated contents (*Off-Policy*). To avoid such inconsistencies,

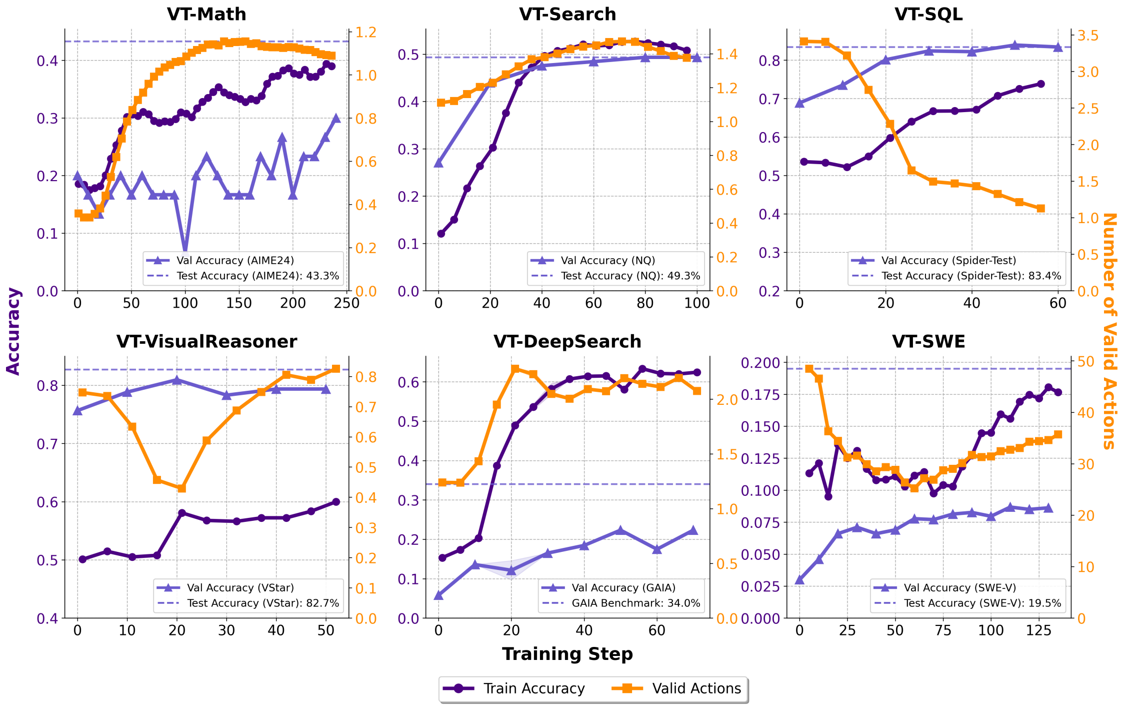


Figure 4: Training dynamics using VerlTool on all 6 tasks. For each task, the corresponding test benchmarks are AIME24, NQ, Spider-Test, VStar, GAIA, and SWE-Verified. All models are trained and evaluated based on **VerlTool** framework. The actual evaluation performance (purple dash) can be higher due to the train-eval settings difference. The number of actions is averaged over all sampled responses in each batch.

we adopt the first approach and consistently maintain a token list prefix during rollout, ensuring stable alignment across multiple rollout turns.

4 Experiments

Parallel Tool Server Backend. To support high-throughput and scalable execution of tool interactions, VerlTool offers two types of parallel execution backends within the Tool Server:

Multi-threading: For small-scale or lightweight tool calls, VerlTool employs Python's `ThreadPoolExecutor` to parallelize calls across multiple worker threads.

Ray-based Asynchronous Execution: To deal with resource-intensive tools for better resource management, VerlTool optionally supports integration with Ray (Moritz et al., 2017), enabling distributed and fault-tolerant tool execution across machines or GPU nodes. This design provides robust scalability for long-horizon or computationally intensive tool calls.

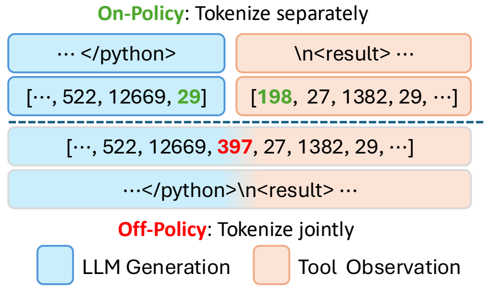


Figure 5: Tokenization of LLM generated content "...`</python>`" and tool observation "`\n<result>`..." can produce different token lists using Qwen2.5 tokenizer under different strategies.

Error Handlling. VerlTool allows timeout for tool execution on both the RL and tool-server side, coupled with a tool-call Retrying Mechanism. To deal with erroneous trajectories, we devised two options: Penalizing and Loss Marking. Please see the detailed introduction in Appendix C.

Table 5: Results on Math-related benchmarks with Python interpreter tool. The best results are indicated in **bold** and the second-best results are underlined.

Model	GSM8K	MATH 500	Minerva Math	Olympiad Bench	AIME24	AMC23	Avg.
Qwen2.5-Math-7B-Base/Instruct							
Qwen2.5-Math-7B-Instruct	**95.2**	83.0	**37.1**	41.6	16.7	70.0	57.3
Qwen-2.5-Math-7B-Instruct-TIR	88.8	80.2	26.8	41.6	30.0	52.5	53.3
SimpleRL-Zoo-7B	94.6	82.4	29.0	50.5	30.0	62.5	58.2
ToRL-7B	92.7	82.2	33.5	49.9	43.3	65.0	61.1
VT-Math-zero (GRPO)	91.8	**83.2**	31.6	50.5	**43.3**	70.0	61.7
VT-Math-zero (DAPO)	92.1	82.8	34.9	**51.6**	36.7	**75.0**	**62.2**

Table 6: Results of on knowledge-QA benchmarks. $^{\dagger}/^{\star}$ represents in-domain/out-domain datasets.

Model	General QA			Multi-Hop QA				Avg.
	NQ†	TriviaQA*	PopQA*	HotpotQA†	2wiki*	Musique*	Bamboogle*	Avg.
Qwen2.5-3b-Base/Instruct								
Direct Inference	10.6	28.8	10.8	14.9	24.4	2.0	2.4	13.4
Search-R1-base (GRPO)	42.1	58.3	41.3	29.7	27.4	6.6	12.8	31.2
Search-R1-base (PPO)	40.6	58.7	43.5	28.4	27.3	4.9	8.8	30.3
VT-Search-zero (GRPO)	45.4	61.6	**48.1**	32.4	30.8	7.6	15.2	34.4
VT-Search-zero (DAPO)	**45.8**	**62.3**	46.5	**33.0**	**31.1**	**8.2**	15.2	**34.6**
Qwen2.5-7b-Base/Instruct								
Direct Inference	13.4	40.8	14.0	18.3	25.0	3.1	12.0	18.1
Search-R1-base (GRPO)	39.5	56.0	38.8	32.6	29.7	12.5	36.0	35.0
Search-R1-base (PPO)	48.0	63.8	45.7	43.3	38.2	**19.6**	43.2	43.1
VT-Search-zero (GRPO)	**49.3**	**66.2**	**50.2**	**44.8**	**45.3**	19.3	**46.4**	**45.9**
VT-Search-zero (DAPO)	48.3	63.4	48.2	42.6	39.2	18.0	38.4	41.2

4.1 Experiment Setup

With a modular, plug-and-play design, VERLTOOL equips an agent with tools spanning multiple domains and modalities as shown in Table 14. In this section, we report experiment results in six ARLT tasks, including VT-Math (Table 5), VT-Search (Table 6), VT-SQL (Table 7), VT-VisualReasoner (Table 8), VT-DeepSearch (Table 8), and VT-SWE (Table 7), demonstrating VERLTOOL's compatibility with various tools. Please see details of training, evaluation, and detailed discussion in Appendix F.

4.2 Results

Training on VerlTool achieves competitive results. Models trained using VERLTOOL consistently match or exceed existing baselines across all six tasks. VT-Math achieves a 62.2% average performance on mathematical benchmarks, surpassing expert models on multiple benchmarks such as AIME24, AMC23, and Olympiad Bench. VT-Search reaches 45.9% accuracy on knowledge QA, surpassing Search-R1 by 10.9%. In terms of the NL2SQL task, VT-SQL matches specialized systems such as SkyRL-SQL. VT-VisualReasoner achieves 82.7% on V* Bench while VT-DeepSearch reaches 34.0% on GAIA. These figures demonstrate that trained under our unified framework, agents could achieve competitive task-specific performance compared to separate divergent code bases.

Multi-modal tools are well-supported. VERLTOOL's modular design enables seamless integration of a wide range of tools under a unified API interface. We demonstrated this capability through the implementation of text-based tools, including Python and SQL Interpreter, Local Retriever and Web Search Tools, and visual processing tools that operate on image and video modality (image operations and video frame selections). Moreover, our framework is equipped with system-level tools such as the bash terminal and

Table 7: Results on NL2SQL (left) and SWE-Verified (right) benchmarks in terms of pass rates.

Model	**Spider**	
Split	**Dev**	**Realistic**
Reasoning without Tool		
GPT-4o	70.9	-
DeepSeekCoder-6.7B-Instruct	63.2	-
OpenCoder-8B-Instruct	59.5	-
Qwen2.5-Coder-7B-Instruct	73.4	-
Tool Integrated Reasoning		
OmniSQL-7B	81.2	63.9
SkyRL-SQL-7B (GRPO)	**83.9**	81.1
VT-SQL (Qwen-2.5-Coder-7B-Instruct based)		
+ GRPO	**83.9**	**81.3**

Model	**SWEBench**
OpenHands Scaffold	
Qwen3-8B	3.6
OpenHands-7B-Agent	11.0
SkyRL-v0 (Cao et al., 2025)	
Qwen3-8B Based	9.4
OpenHands-7B-Agent Based	14.6
R2E Gym Scaffold	
Qwen3-8B	10.4
VT-SWE (Qwen3-8B Based)	
+GRPO	**19.5**

Table 8: Results on visual reasoning (left) and agentic search benchmarks (right).

Model	**V* Bench**
Reasoning without Tool	
GPT-4o	62.8
Gemini-2.5-Pro	79.2
Qwen2.5-VL-7B-Instruct	70.4
Video-R1-7B	51.2
Tool Integrated Reasoning	
Visual Sketchpad (GPT-4o)	80.4
IVM-Enhanced (GPT-4V)	81.2
Pixel-Reasoner-7B	**84.3**
VT-VisualReasoner (Qwen2.5-VL-7B-Instruct Based)	
+ GRPO-Acc	78.8
+ GRPO-Complex	82.7

Model	**GAIA**	**HLE**
Reasoning without Tool		
DeepSeek-R1-671B	25.2	**8.6**
GPT-4o	17.5	2.6
Qwen3-8B	20.4	4.6
Tool Integrated Reasoning (Qwen3-8B)		
Vanilla RAG	20.4	5.8
Search-o1	21.4	6.4
WebThinker	22.3	6.6
ReAct	23.3	4.6
VT-DeepSearch (Qwen3-8B Based)		
+ Snippets-Only	32.0	7.8
+ QwQ-32B	**34.0**	8.4

file operation tools. Visual reasoning experiments demonstrate that our agents could dynamically manipulate images and process visual information iteratively, enabling complex multi-modal workflows that were unsupported in existing single-modality RL-training frameworks, which only work on the text level.

Dynamics of Tool Usage across Tasks. Tool usage patterns exhibit substantial variation across different domains, with mathematical tasks typically requiring $1 \sim 4$ interactions while software engineering tasks may extend to over 100 interactions. Importantly, models do not spontaneously develop effective tool-use capabilities without appropriate reward design and initialization strategies. For instance, without VT-VisualReasoner's sophisticated reward mechanism, the frequency of tool actions gradually diminishes to zero within a few reinforcement learning (RL) steps.

The evolution of tool usage during training demonstrates task-specific characteristics that reflect the underlying utility of tool interactions. In VT-SQL settings, the number of actions decreases rapidly after several dozen RL steps, as the model learns that SQL executors are non-essential for most straightforward queries. Through training, the model develops a preference for responses requiring fewer tool calls by gradually memorizing expected execution results, thereby reducing the need for verification through SQL executors. Conversely, in VT-DeepSearch settings, tool usage increases dramatically during training because problem-solving fundamentally depends on search capabilities. Unlike simpler SQL results, the extensive and information-rich content returned by search tools cannot be easily memorized, necessitating increased tool invocations for effective task completion. This divergent behavior underscores how the intrinsic value of tool assistance shapes learning dynamics across different computational domains.

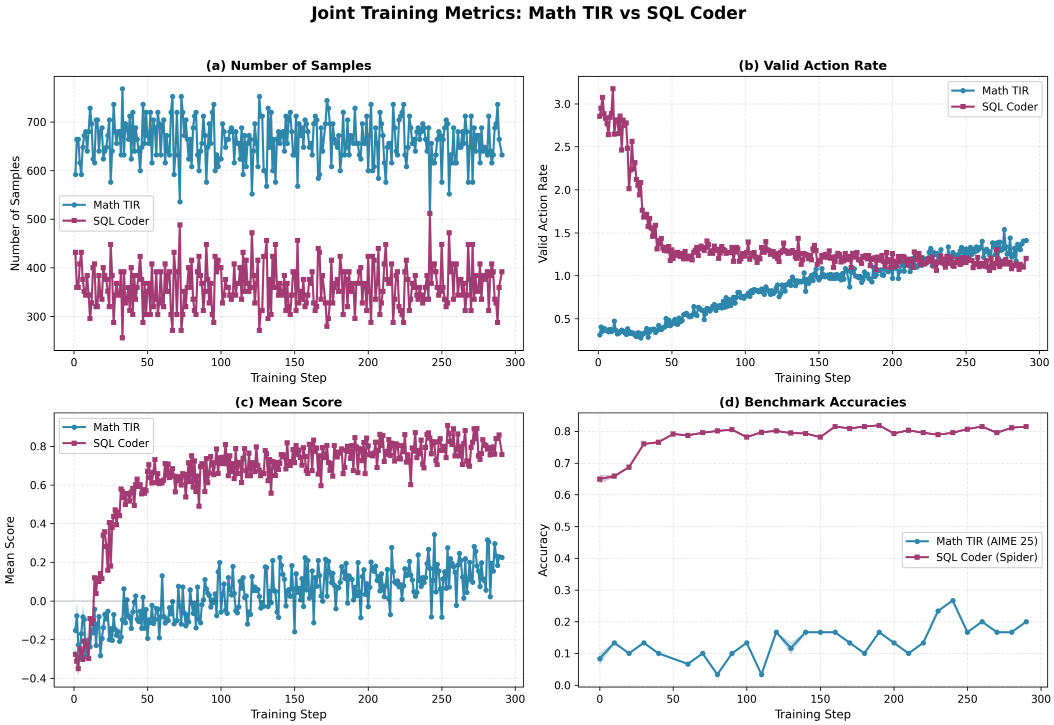


Figure 6: The joint training metrics on Qwen-2.5-Coder-Instruct-7B using two types of tools and tasks at the same time. Joint benefit is shown on both tasks.

Emerging behaviours of Agentic RL. Agentic models trained using VerlTool emerged sophisticated behaviors, including self-correction, iterative refinement, and strategic tool selection in the multi-round tool-calling environment provided by our framework. For instance, mathematical agents verify computations and backtrack from errors, search agents refine queries based on retrieved information, and software agents develop debugging strategies combining code analysis and incremental fixes. These capabilities represent genuine agentic problem-solving that extends beyond simple function-calling invocations. We present corresponding case studies in Appendix J.

Multi-Tool Training within Single Model We conducted experiments on training a single model to utilize tools with diversified types. We combine Math-TIR with SQL-Coder as Math-SQL multi-tool experiments. And combine Math and Visual Reasoning as Math-Visual Training. Although we attempted to train a model with more than two types of tools and tasks at a time, all the results show that the training collapsed. The detailed analysis and explanation of the reason for the training collapse is located at Appendix G.

Math-SQL: We first train Qwen-2.5-Coder-Instruct-7B on Math-TIR and SQL Coder tasks jointly. Results are shown in Table 9a and Figure 6. We see a joint benefit on downstream task performance due to the similarity of the Math and SQL tasks.

Math-Visual: We then train Pixel-Reaoner-7b-WarmStart on Math-TIR and Visual-Reasoner tasks jointly. Results are shown in Table 9b and Figure 7. However, the joint multi-tool training doesn't show enough benefit for either task.

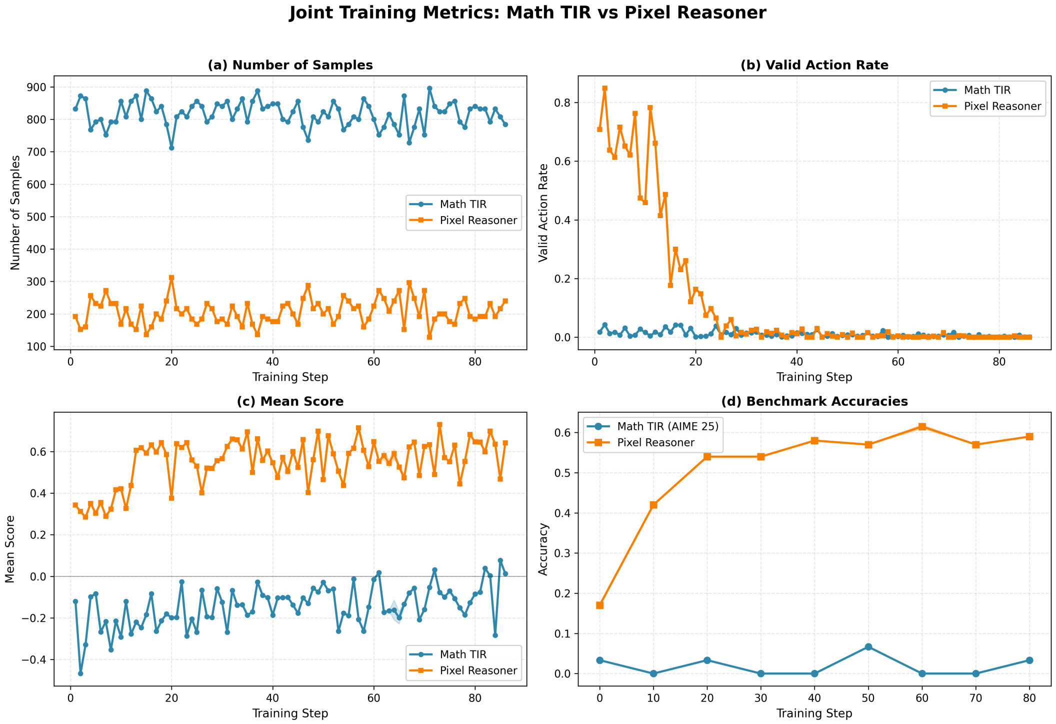


Figure 7: The joint training metrics on Pixel-Reaoner-7b-WarmStart checkpoint using two types of tools and tasks at the same time. No joint benefit shown on either task.

(a) Comparison of performance on training 290 steps. "Single-Tool" denotes training Qwen2.5-Coder-Instruct for Spider-relistic and Qwen-2.5-Math for AIME. "Multi-Tool" trains Qwen2.5-Coder-Instruct on the SQL-Coder and Math-TIR jointly.

Method (290 steps)	Spider -realistic	AIME 24	AIME 25
Single-Tool	81.3	**28.75**	19.17
Multi-Tool	**81.5**	23.33	**26.67**

(b) Comparison of performance on training 290 steps. "Single-Tool" denotes the Qwen2.5-VL-Instruct training for V-Star and Qwen-2.5-Math for AIME. The multi-tool experiment trains Qwen2.5-VL-Instruct on both the V-Star and Math-TIR jointly.

Method (80 steps)	V-Star	AIME 24	AIME 25
Single-Tool	**81.3**	**28.75**	**19.17**
Multi-Tool	59.0	0	3

5 Conclusion

We propose VerlTool, addressing key limitations of Agentic Reinforcement Learning with Tool use (ARLT) models' training. Our framework features a unified and modular systematic design, providing multi-modal tool management through standardized API designs, while maintaining high-efficiency model training featuring asynchronous rollout execution. Our system extends traditional single-turn reinforcement learning with verifiable rewards to ARLT domains, featuring robust system designs and upstream-aligned with VeRL. The framework is extensively examined across six domains featuring diverse tool integrations and modalities. As evidenced by extensive evaluation, agents trained through our framework demonstrated competitive performance compared to specialized systems, while unified under our training infrastructure. We present VerlTool as a scalable foundational training infrastructure to the RL community and hope our contributions could facilitate the advancement of ARLT research.

References

Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. Sadga: Structure-aware dual graph aggregation network for text-to-sql. *ArXiv*, abs/2111.00653, 2021. URL https://api.semanticscholar.org/CorpusID:240353884.

Shiyi Cao, Sumanth Hegde, Dacheng Li, Tyler Griggs, Shu Liu, Eric Tang, Jiayi Pan, Xingyao Wang, Akshay Malik, Graham Neubig, Kourosh Hakhamaneshi, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Skyrl-v0: Train real-world long-horizon agents via reinforcement learning, 2025.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Trans. Mach. Learn. Res.*, 2023, 2022. URL https://api.semanticscholar.org/CorpusID:253801709.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do not think that much for 2+3=? on the overthinking of o1-like llms. *ArXiv*, abs/2412.21187, 2024a. URL https://api.semanticscholar.org/CorpusID:275133600.

Zhipeng Chen, Yingqian Min, Beichen Zhang, Jie Chen, Jinhao Jiang, Daixuan Cheng, Wayne Xin Zhao, Zheng Liu, Xu Miao, Yang Lu, Lei Fang, Zhongyuan Wang, and Ji-Rong Wen. An empirical study on eliciting and improving r1-like reasoning models. *arXiv preprint arXiv:2503.04548*, 2025.

Zhiyu Zoey Chen, Jing Ma, Xinlu Zhang, Nan Hao, An Yan, Armineh Nourbakhsh, Xianjun Yang, Julian J. McAuley, Linda Ruth Petzold, and William Yang Wang. A survey on large language models for critical societal domains: Finance, healthcare, and law. *ArXiv*, abs/2405.01769, 2024b. URL https://api.semanticscholar.org/CorpusID:269587715.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, et al. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*, 2025.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazar'e, Maria Lomeli, Lucas Hosseini, and Herv'e J'egou. The faiss library. *ArXiv*, abs/2401.08281, 2024. URL https://api.semanticscholar.org/CorpusID:267028372.

Joshua M Epstein and Robert Axtell. *Growing artificial societies: social science from the bottom up*. Brookings Institution Press, 1996.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms, 2025. URL https://arxiv.org/abs/2504.11536.

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, Tongkai Yang, Binhang Yuan, and Yi Wu. Areal: A large-scale asynchronous reinforcement learning system for language reasoning, 2025. URL https://arxiv.org/abs/2505.24298.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John Robert Woodward, Jinxia Xie, and Pengsheng Huang. Towards robustness of text-to-sql models against synonym substitution. *ArXiv*, abs/2106.01065, 2021a. URL https://api.semanticscholar.org/CorpusID:235293739.

Yujian Gan, Xinyun Chen, and Matthew Purver. Exploring underexplored limitations of cross-domain text-to-sql generalization, 2021b.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17:1132–1145, 2023a. URL https://api.semanticscholar.org/CorpusID:261276437.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *ArXiv*, abs/2312.10997, 2023b. URL https://api.semanticscholar.org/CorpusID:266359151.

Gemini. Gemini: A family of highly capable multimodal models, 2024. URL https://arxiv.org/abs/2312.11805.

Nigel Gilbert. *Agent-based models*. Sage Publications, 2019.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*, 2023.

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

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *ArXiv*, abs/2504.11456, 2025. URL https://api.semanticscholar.org/CorpusID:277787455.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

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

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*, 2023.

Jian Hu, Xibin Wu, Zilin Zhu, Xianyu, Weixun Wang, Dehao Zhang, and Yu Cao. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*, 2024.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43:1 – 55, 2023. URL https://api.semanticscholar.org/CorpusID:265067168.

Wenlong Huang, P. Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *ArXiv*, abs/2201.07207, 2022. URL https://api.semanticscholar.org/CorpusID:246035276.

Tatsuro Inaba, Hirokazu Kiyomaru, Fei Cheng, and Sadao Kurohashi. Multitool-cot: Gpt-3 can use multiple external tools with chain of thought prompting. *ArXiv*, abs/2305.16896, 2023. URL https://api.semanticscholar.org/CorpusID:258947061.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents. *ArXiv*, abs/2504.07164, 2025. URL https://api.semanticscholar.org/CorpusID:277667306.

Pengcheng Jiang, Jiacheng Lin, Lang Cao, Runchu Tian, SeongKu Kang, Zifeng Wang, Jimeng Sun, and Jiawei Han. Deepretrieval: Hacking real search engines and retrievers with large language models via reinforcement learning. *ArXiv*, abs/2503.00223, 2025. URL https://api.semanticscholar.org/CorpusID:276742133.

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. *ArXiv*, abs/2305.06983, 2023. URL https://api.semanticscholar.org/CorpusID:258615731.

Bowen Jin, Jinsung Yoon, Jiawei Han, and Sercan Ö. Arik. Long-context llms meet rag: Overcoming challenges for long inputs in rag. *ArXiv*, abs/2410.05983, 2024. URL https://api.semanticscholar.org/CorpusID:273229050.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *ArXiv*, abs/2503.09516, 2025. URL https://api.semanticscholar.org/CorpusID:276937772.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Yu Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. Dense passage retrieval for open-domain question answering. *ArXiv*, abs/2004.04906, 2020. URL https://api.semanticscholar.org/CorpusID:215737187.

Zixuan Ke, Fangkai Jiao, Yifei Ming, Xuan-Phi Nguyen, Austin Xu, Do Xuan Long, Minzhi Li, Chengwei Qin, PeiFeng Wang, Silvio Savarese, Caiming Xiong, and Shafiq Joty. A survey of frontiers in llm reasoning: Inference scaling, learning to reason, and agentic systems. *Trans. Mach. Learn. Res.*, 2025, 2025. URL https://api.semanticscholar.org/CorpusID:277781085.

Geunwoo Kim, Pierre Baldi, and Stephen Marcus McAleer. Language models can solve computer tasks. *ArXiv*, abs/2303.17491, 2023. URL https://api.semanticscholar.org/CorpusID:257834038.

Yilun Kong, Jingqing Ruan, Yihong Chen, Bin Zhang, Tianpeng Bao, Shiwei Shi, Guoqing Du, Xiaoru Hu, Hangyu Mao, Ziyue Li, Xingyu Zeng, and Rui Zhao. Tptu-v2: Boosting task planning and tool usage of large language model-based agents in real-world systems. *ArXiv*, abs/2311.11315, 2023. URL https://api.semanticscholar.org/CorpusID:265294410.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. *ArXiv*, abs/2005.11401, 2020. URL https://api.semanticscholar.org/CorpusID:218869575.

Aitor Lewkowycz, Anders Johan Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=IFXTZERXdM7.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. Codes: Towards building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data*, 2:1 – 28, 2024. URL https://api.semanticscholar.org/CorpusID:267938784.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. Omnisql: Synthesizing high-quality text-to-sql data at scale. *ArXiv*, abs/2503.02240, 2025a. URL https://api.semanticscholar.org/CorpusID:276774742.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. *ArXiv*, abs/2301.07507, 2023. URL https://api.semanticscholar.org/CorpusID:255998567.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, Weizhou Shen, Junkai Zhang, Dingchu Zhang, Xixi Wu, Yong Jiang, Ming Yan, Pengjun Xie, Fei Huang, and Jingren Zhou. Websailor: Navigating super-human reasoning for web agent. *ArXiv*, abs/2507.02592, 2025b. URL https://api.semanticscholar.org/CorpusID:280078605.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated rl, 2025c. URL https://arxiv.org/abs/2503.23383.

Shu Liu, Sumanth Hegde, Shiyi Cao, Alan Zhu, Dacheng Li, Tyler Griggs, Eric Tang, Akshay Malik, Kourosh Hakhamaneshi, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Skyrl-sql: Matching gpt-4o and o4-mini on text2sql with multi-turn rl, 2025. Notion Blog.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yu xin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. A survey of text-to-sql in the era of llms: Where are we, and where are we going? *IEEE Transactions on Knowledge and Data Engineering*, 2024. URL https://api.semanticscholar.org/CorpusID:271843296.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. *ArXiv*, abs/2304.09842, 2023. URL https://api.semanticscholar.org/CorpusID:258212542.

Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun Ma, and Wenhu Chen. General-reasoner: Advancing llm reasoning across all domains. *ArXiv*, abs/2505.14652, 2025. URL https://api.semanticscholar.org/CorpusID:278768680.

Xinji Mai, Haotian Xu, Weinong Wang, Jian Hu, Yingying Zhang, Wenqiang Zhang, et al. Agent rl scaling law: Agent rl with spontaneous code execution for mathematical problem solving. *arXiv preprint arXiv:2505.07773*, 2025.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Hannaneh Hajishirzi, and Daniel Khashabi. When not to trust language models: Investigating effectiveness and limitations of parametric and non-parametric memories. *arXiv preprint arXiv:2212.10511*, 7, 2022.

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

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging ai applications. *ArXiv*, abs/1712.05889, 2017. URL https://api.semanticscholar.org/CorpusID:34552495.

Ivan Moshkov, Darragh Hanley, Ivan Sorokin, Shubham Toshniwal, Christof Henkel, Benedikt Schifferer, Wei Du, and Igor Gitman. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset, 2025. URL https://arxiv.org/abs/2504.16891.

OpenAI. Introducing gpt-4.1 in the api. https://openai.com/index/gpt-4-1/, 2025.

Simone Papicchio, Simone Rossi, Luca Cagliero, and Paolo Papotti. Think2sql: Reinforce llm reasoning capabilities for text2sql. *ArXiv*, abs/2504.15077, 2025. URL https://api.semanticscholar.org/CorpusID:277955819.

C.A.I. Peng, Xi Yang, Aokun Chen, Kaleb E. Smith, Nima M. Pournejatian, Anthony B Costa, Cheryl Martin, Mona G. Flores, Ying Zhang, Tanja Magoc, Gloria P. Lipori, Duane A. Mitchell, Naykky M. Singh Ospina, Mustafa Mamon Ahmed, William R. Hogan, Elizabeth A. Shenkman, Yi Guo, Jiang Bian, and Yonghui Wu. A study of generative large language model for medical research and healthcare. *NPJ Digital Medicine*, 6, 2023. URL https://api.semanticscholar.org/CorpusID:258841310.

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

Aske Plaat, Max J. van Duijn, Niki van Stein, Mike Preuss, Peter van der Putten, and Kees Joost Batenburg. Agentic large language models, a survey. *ArXiv*, abs/2503.23037, 2025. URL https://api.semanticscholar.org/CorpusID:277451794.

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, and Sercan Ö. Arik. Reasoning-sql: Reinforcement learning with sql tailored partial rewards for reasoning-enhanced text-to-sql. *ArXiv*, abs/2503.23157, 2025. URL https://api.semanticscholar.org/CorpusID:277452634.

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

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*, 2025.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Jirong Wen. Tool learning with large language models: A survey. *ArXiv*, abs/2405.17935, 2024. URL https://api.semanticscholar.org/CorpusID:270067624.

Shanghaoran Quan, Jiaxin Yang, Bowen Yu, Bo Zheng, Dayiheng Liu, An Yang, Xuancheng Ren, Bofei Gao, Yibo Miao, Yunlong Feng, Zekun Wang, Jian Yang, Zeyu Cui, Yang Fan, Yichang Zhang, Binyuan Hui, and Junyang Lin. Codeelo: Benchmarking competition-level code generation of llms with human-comparable elo ratings. *ArXiv*, abs/2501.01257, 2025. URL https://api.semanticscholar.org/CorpusID:275212089.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *ArXiv*, abs/2305.18290, 2023. URL https://api.semanticscholar.org/CorpusID:258959321.

Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Xingyu Zeng, and Rui Zhao. Tptu: Task planning and tool usage of large language model-based ai agents. *ArXiv*, abs/2308.03427, 2023. URL https://api.semanticscholar.org/CorpusID:265381326.

Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach.* pearson, 2016.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *ArXiv*, abs/2302.04761, 2023. URL https://api.semanticscholar.org/CorpusID:256697342.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017. URL https://api.semanticscholar.org/CorpusID:28695052.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Jun-Mei Song, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *ArXiv*, abs/2402.03300, 2024. URL https://api.semanticscholar.org/CorpusID:267412607.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yue Ting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *ArXiv*, abs/2303.17580, 2023. URL https://api.semanticscholar.org/CorpusID:257833781.

Zhuocheng Shen. Llm with tools: A survey. *ArXiv*, abs/2409.18807, 2024. URL https://api.semanticscholar.org/CorpusID:272968969.

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

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2503.05592.

Alex Su, Haozhe Wang, Weiming Ren, Fangzhen Lin, and Wenhu Chen. Pixel reasoner: Incentivizing pixel-space reasoning with curiosity-driven reinforcement learning. *ArXiv*, abs/2505.15966, 2025. URL https://api.semanticscholar.org/CorpusID:278789415.

Shuang Sun, Huatong Song, Yuhao Wang, Ruiyang Ren, Jinhao Jiang, Junjie Zhang, Fei Bai, Jia Deng, Wayne Xin Zhao, Zheng Liu, et al. Simpledeepsearcher: Deep information seeking via web-powered reasoning trajectory synthesis. *arXiv preprint arXiv:2505.16834*, 2025.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

THUDM. slime: A llm post-training framework aiming at scaling rl. https://github.com/THUDM/slime, 2024. Software framework for LLM post-training with reinforcement learning scaling.

H. Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *ArXiv*, abs/2212.10509, 2022a. URL https://api.semanticscholar.org/CorpusID:254877499.

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

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. In *Annual Meeting of the Association for Computational Linguistics*, 2019. URL https://api.semanticscholar.org/CorpusID:207863446.

Haozhe Wang, Long Li, Chao Qu, Fengming Zhu, Weidi Xu, Wei Chu, and Fangzhen Lin. To code or not to code? adaptive tool integration for math language models via expectation-maximization. *arXiv preprint arXiv:2502.00691*, 2025a.

Hongru Wang, Yujia Qin, Yankai Lin, Jeff Z. Pan, and Kam-Fai Wong. Empowering large language models: Tool learning for real-world interaction. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, pp. 2983–2986, New York, NY, USA, 2024a. Association for Computing Machinery. ISBN 9798400704314. doi: 10.1145/3626772.3661381. URL https://doi.org/10.1145/3626772.3661381.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning, 2023. URL https://arxiv.org/abs/2310.03731.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *ArXiv*, abs/2212.03533, 2022. URL https://api.semanticscholar.org/CorpusID:254366618.

Weixun Wang, Shaopan Xiong, Gengru Chen, Wei Gao, Sheng Guo, Yancheng He, Ju Huang, Jiaheng Liu, Zhendong Li, Xiaoyang Li, et al. Reinforcement learning optimization for large-scale learning: An efficient and user-friendly scaling library. *arXiv preprint arXiv:2506.06122*, 2025b.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, 2024b. URL https://api.semanticscholar.org/CorpusID:271404773.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Monica Lam, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Fei-Fei Li, Lijuan Wang, Yejin Choi, and Manling Li. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *ArXiv*, abs/2504.20073, 2025c. URL https://api.semanticscholar.org/CorpusID:278170861.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, F. Xia, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903, 2022. URL https://api.semanticscholar.org/CorpusID:246411621.

Michael Wooldridge. Intelligent agents. *Multiagent systems: A modern approach to distributed artificial intelligence*, 1:27–73, 1999.

Penghao Wu and Saining Xie. V?: Guided visual search as a core mechanism in multimodal llms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13084–13094, 2024.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Zejun Ma, and Bo An. Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. https://simpletir.notion.site/report, 2025. Notion Blog.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. Gpt4tools: Teaching large language model to use tools via self-instruction. *ArXiv*, abs/2305.18752, 2023. URL https://api.semanticscholar.org/CorpusID:258967184.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629, 2022. URL https://api.semanticscholar.org/CorpusID:252762395.

Zijun Yao, Yantao Liu, Yanxu Chen, Jianhui Chen, Junfeng Fang, Lei Hou, Juanzi Li, and Tat-Seng Chua. Are reasoning models more prone to hallucination? *ArXiv*, abs/2505.23646, 2025. URL https://api.semanticscholar.org/CorpusID:278996592.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for reasoning. *ArXiv*, abs/2502.03387, 2025. URL https://api.semanticscholar.org/CorpusID:276116748.

Tao Yu, Rui Zhang, Kai-Chou Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Z Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *ArXiv*, abs/1809.08887, 2018. URL https://api.semanticscholar.org/CorpusID:52815560.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mammoth: Building math generalist models through hybrid instruction tuning, 2023. URL https://arxiv.org/abs/2309.05653.

Bohan Zhai, Canwen Xu, Yuxiong He, and Zhewei Yao. Excot: Optimizing reasoning for text-to-sql with execution feedback. In *Annual Meeting of the Association for Computational Linguistics*, 2025. URL https://api.semanticscholar.org/CorpusID:277321668.

Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, and Shuming Shi. Siren's song in the ai ocean: A survey on hallucination in large language models. *ArXiv*, abs/2309.01219, 2023. URL https://api.semanticscholar.org/CorpusID:261530162.

Appendix

A Tool Server implementation details and efficiency analysis **23**
A.1 Tool Server Design Issues and Our Solutions 23
A.2 Tool Server Efficiency Analysis 23
A.3 How does VerlTool isolate tool environments? 23

B Cross-Framework Comparison in Features **25**
B.1 Cross-framework Comparison in Runtime analysis under same workloads 25

C Error Handling Strategies **25**
C.1 Step-level Timeout and Retrying Mechanism 26
C.2 Trajectory-Level Erroneous Handling 26

D Tool Execution Metrics **26**

E Ablation on Tokenization Strategies **27**
E.1 Concurrent Work from VLLM Confirms the Importance of "No Retokenization" 27
E.2 Our Additional Experiments Validating the Same Phenomenon 27

F Detailed Experiment Setup **29**
F.1 Mathematical Reasoning with Python Executor (VT-Math) 29
F.2 Knowledge QA with Search Retriever (VT-Search) 29
F.3 Multi-Turn SQL Query Generation (VT-SQL) 29
F.4 Visual Reasoning with Image Operations (VT-VisualReasoner) 30
F.5 Agentic Web Search (VT-DeeepSearch) 30
F.6 Software Engineering Benchmark (VT-SWE) 30
F.7 Supported Tools 31
F.8 Training and Evaluation Configurations 31

G Multi-Tool Training within Single Model **32**

H Occasional Performance Gain Discussion **32**
H.1 Parameter Differences 32
H.2 Scaling Effects Analysis on the Maximum Response Length 33

I More related works **34**
I.1 Organization 34
I.2 Tool-Integrated Reasoning 34
I.3 From Tool Integration to Agentic LLMs 34

I.4 Reinforcement Learning with Verifiable Reward (RLVR) 35
I.5 Agentic Reinforcement Learning with Tool Use (ARLT) 35

J Case Study **37**

K The use of Large Language Models **42**

A Tool Server implementation details and efficiency analysis

We detailed the tool server's implementation and performance in this section. Our system avoids all GIL bottlenecks by design, achieving low latency for light-weight tools and higher throughput for long-running tools. Overall, the results demonstrated our tool server implementation's high efficiency and scalability, showing that it is suitable for large-scale agentic RL workloads

A.1 Tool Server Design Issues and Our Solutions

Python's GIL on multithreaded tool execution could potentially impact the tool execution efficiency and overall performance of the framework. To mitigate this issue, VERLTOOL's design is not constrained by the GIL thanks to the deliberate separation of *request dispatch*, *process-based execution*, and *ray actor-based resource management*.

First, multithreading is used only for lightweight request dispatch. After dispatching, each tool-worker operates in one of two modes:

(1) for resource-intensive tools, the worker launches a new OS process via `subprocess`, fully bypassing the GIL (e.g., `Python Interpreter`, `Image Processing`);

(2) for lightweight, I/O-bound tools, execution is handled via `asyncio`, which does not block the GIL (e.g., `search-retriever`, `Google-Search`).

Second, many heavier tools are managed as **Ray Actors**, each running in an isolated actor process with explicitly allocated CPU/GPU resources. These processes never share a Python interpreter or GIL, offering robust parallelism and fine-grained resource control.

The execution model is summarized below:

Tool Resource Type	Server Mode	Execution Backend	GIL Impact
Resource-intensive (CPU/GPU)	Threading Ray Actor	OS process via `subprocess` ray process with resources	None (new process) None (actor process)
I/O-bound, lightweight	Threading Ray Actor	`asyncio` event loop ray process with resources	None (non-blocking) None (actor process)

Table 10: Execution paths of VERLTOOL's tool server and their interaction with the Python GIL.

A.2 Tool Server Efficiency Analysis

We conducted a detailed efficiency analysis of the tool-server backend. Across both lightweight and realistic workloads, the results show that VERLTOOL's tool server maintains *stable latency*, *high throughput*, and *graceful degradation* under increasing load.

For lightweight Python calls (Table 11), the server achieves an average response time of **1.04s** at its optimal configuration (1024 max concurrency, 256 request concurrency), and remains stable even under extreme stress (4096 concurrent requests) with a bounded average latency of **9.1s**.

For realistic long-running tasks with randomized 1–10s execution time (Table 12), the tool server matches baseline performance at moderate concurrency and achieves a **9.7% higher throughput** at high concurrency (1024), confirming that kernel pooling and process-based isolation efficiently handle heterogeneous, long-duration tool calls. These measurements validate the practicality of VERLTOOL's resource-managed tool execution design and its ability to support large-scale, highly parallel agentic RL workloads.

A.3 How does VerlTool isolate tool environments?

Virtualization and Isolation. VERLTOOL adopts a *layered virtualization strategy* tailored to the security and performance needs of each tool type. Instead of relying on a single mechanism, the framework com-

Table 11: Performance comparison for lightweight Python execution (`print("hello world")`). The tool-server demonstrates efficient handling of simple operations across varying concurrency levels. At the optimal configuration (1024 max concurrency, 256 request concurrency), the average response time is **1.04s** with maximum latency of only 1.43s, showing excellent scalability. Performance degrades gracefully under extreme load (4096 requests) but remains stable with 9.1s average response time.

Mode	**Tool-Server Max Concurrency**	**Request Concurrency**	**Response Time (s)**			**Note**
			Min	**Avg**	**Max**	
Ray	128	256	0.29	3.47	20.00	Under-provisioned
	512	256	0.38	1.25	2.15	Good
	1024	256	0.36	**1.04**	1.43	**Optimal**
	4096	256	0.48	1.15	1.44	Over-provisioned
	4096	512	0.28	1.46	2.20	Scaling up
	4096	1024	0.26	2.81	3.99	Heavy load
	4096	2048	0.41	5.81	8.67	Very heavy load
	4096	4096	0.47	9.10	16.03	Extreme load

Table 12: Performance comparison for realistic computation simulation using `import time; import random; time.sleep(random.randint(1,10)); print('Hello from stress test!')`. This benchmark simulates real-world scenarios with variable execution time (1-10s random sleep). **Key Finding**: The tool-server (VerlTool Server) achieves **9.7% higher throughput** (83.11 vs 75.79 req/s) at high concurrency (1024) while maintaining comparable latency. At moderate concurrency (256), both approaches perform similarly, indicating that the tool-server's benefits become pronounced under higher load conditions. The tool-server's kernel pooling approach efficiently manages concurrent long-running tasks without significant overhead.

Backend	**Max Concurrency**	**Request Concurrency**	**Response Time (s)**			**Throughput (req/s)**
			Min	**Avg**	**Max**	
Baseline	256	256	1.29	6.01	10.63	23.54
VerlTool Server	256	256	1.16	6.23	11.64	21.54
Baseline	1024	1024	1.30	6.69	11.96	75.79
VerlTool Server	1024	1024	1.22	7.31	11.18	**83.11**

bines lightweight sandboxes for high-throughput RL workloads with heavier OS-level environments when full system access is necessary.

Code-execution tools (e.g., Python environments used in Math-TIR or DeepSearch) run inside *firejail* sandboxes, providing process and filesystem isolation with negligible overhead. **Browser-based tools** use Ray-managed Playwright sessions, where each request is served in an isolated browser context to avoid cross-trajectory interference and ensure deterministic behavior. For **software engineering tasks**, VerlTool employs *Docker-based SWE environments* that support persistent per-trajectory state and strong OS-level isolation, similar to real-world development setups.

In the SWE Docker environment, *sudo* can be enabled safely due to container-level isolation; however, our tasks do not require elevated privileges. VerlTool follows a strict *least-privilege* policy, allocating stronger permissions only when demanded by task semantics. For typical ARLT tasks, unprivileged containers and firejail sandboxes are sufficient, ensuring both safety and training efficiency.

Overall, this multi-layer design achieves (i) scalable lightweight isolation, (ii) strong OS-level virtualization when needed, and (iii) a unified interface for tool developers without exposing host-level privileges.

B Cross-Framework Comparison in Features

Compared to VERL MCP. The MCP interface in VERL exposes tools through a **stateless JSON–RPC protocol** designed for single-turn interactions. It does not maintain trajectory-level state (e.g., persistent Python variables, SQL cursors, image buffers), nor does it support multimodal outputs beyond plain JSON strings.

Simply adding more tools to MCP cannot replicate VerlTool's requirements of (1) stateful tool environments, (2) multimodal observation tokens.

Compared to AREAL. Our framework differentiates from AREAL in the following aspects. Firstly, AREAL provides an asynchronous RL training engine. However, offers **no unified tool-server abstraction**, no multimodal I/O format, and no mechanism for managing stateful tool calls. Moreover, it has **no well-designed tool-resource management strategies** for heterogeneous tools (e.g., Python sandbox, image processing, search retriever).

Secondly, different to VerlTool, extending AREAL would require building precisely the components that VerlTool provides: (1) multimodal token handling, (2) a trajectory-scoped tool environment, and (3) a managed execution backend.

Finally, VerlTool's ecosystem positioning is fundamentally different from AREAL and serves as a **complementary** role. Designed as a drop-in extension for the widely used veRL training stack, VerlTool's upstream alignment reduces the adoption cost and difficulties for researchers and machine learning engineers who are already familiar with VERL, and also ensures long-term compatibility as the VERL framework continues to evolve.

Compared to SkyRL. SkyRL integrates several task-specific tools, but its vision pipeline is **not a general multimodal API**, and cross-tool interactions are not unified under a single plugin interface. Its reliance on Kubernetes for resource management is **powerful but heavyweight**, making reproduction and extension difficult in typical research settings. Furthermore, tools are not exposed as a uniform abstraction, requiring substantial custom logic for each new tool type. VerlTool instead offers lightweight plugin definitions via `BaseTool`, unified across text, SQL, search, and multimodal tools.

B.1 Cross-framework Comparison in Runtime analysis under same workloads

We hereby evaluate the actual per-step training time in different frameworks under the same workload setting for Math-TIR framework. Results show VerlToolachieves the fastest per-step wall-clock time on 8 H100 GPUs.

Table 13: Wall-clock time analysis across different frameworks on Math-TIR tasks using the same setting. Each step samples 8 responses for 1024. The per-step time is averaged over the first 10 steps.

FrameWork	per step time
SimpleTIR	268
verl-GEM	206
ROLL-GEM	-
AReal	237
VerlTool	**83**

C Error Handling Strategies

VerlTool provides the following two error handling strategies:

C.1 Step-level Timeout and Retrying Mechanism

For tool-heavy tasks, each tool action is assigned a strict per-step timeout (e.g., 90 seconds in VT-SWE) to prevent slow or stalled tools from blocking long-horizon rollouts.

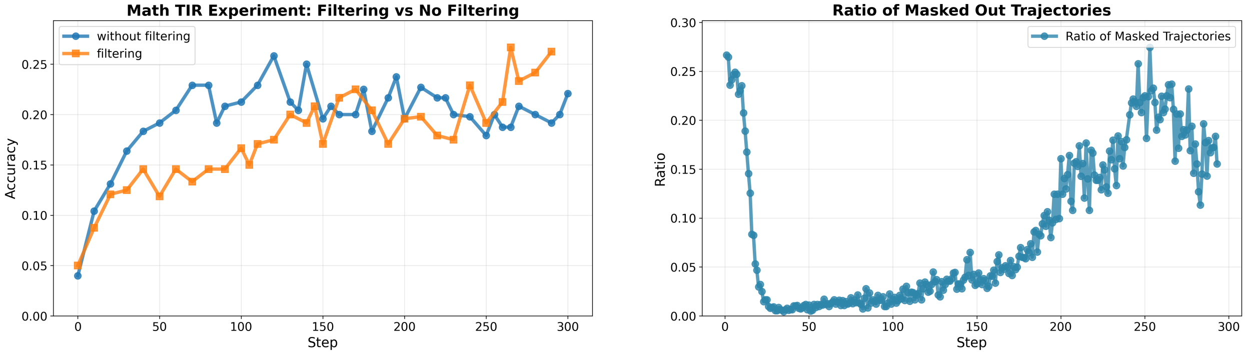
The tool server follows a *best-effort delivery* model: under heavy resource pressure (e.g., near-OOM), it may drop or refuse requests to stay alive. To distinguish between (1) a tool that is genuinely slow and (2) a request that was dropped due to overload, we use two timeout layers. The tool-server timeout is set slightly shorter than the trainer-side timeout, so tool-side delays trigger tool-server timeouts, while dropped requests appear as trainer-side timeouts. On the trainer side, users can adjust the retry behavior using the MAX_RETRY option, allowing the system to recover from transient failures.

To further stabilize training when resources are tight, VerlTool also allows users to limit the number of trajectories rolled out concurrently. This caps peak CPU/GPU/memory usage on the tool server and reduces the chance that retries fail due to resource exhaustion.

C.2 Trajectory-Level Erroneous Handling

There are multiple ways to deal with erroneous trajectories. One simple way is to mask out the loss for those erroneous trajectories or those that do not produce a final answer. As shown in Figure 8, we implement Simple-TIR in VerlTool with minimal modification and get the same conclusion.

Similar strategy is also emploied in VT-SWE. As detailed in Appendix C.1 of the main paper, trajectories that encounter *tool errors*, *exceed length limits*, or *rolling time exceed 30 minutes* are removed from gradient computation using the same masking interface and are assigned a reward of zero under GRPO. This prevents corrupted or incomplete trajectories from contaminating credit assignment while preserving a clean learning signal for successful episodes.



(a) The effects of filtering the erroneous trajectories or not on AIME-25 (avg@8) accuracy in Math-TIR experiments. Filtering trajectories like SimpleTIR (Xue et al., 2025) stabilizes the training.

(b) The ratio of masked out erroneous trajectories during training. The ratio starts high and decreases as the model improves, then increases to a stable 20%, which is the same observation as SimpleTIR.

Figure 8: Analysis of trajectory filtering in Math-TIR experiments. This not only demonstrates VerlTool is easy to modify and integrate more RL strategies like Simple-TIR, and proves that filtering the erroneous trajectories can stabilize the training.

D Tool Execution Metrics

The detailed breakdown of time spent on model inference versus tool execution is presented in this section. As shown in Figure 9, tool-call latency initially rises due to a large number of inefficient code written by the model, then rapidly decreases as the policy learns to issue more well-formed tool calls (middle plot). After stabilization, tool execution and model-generation times remain comparable throughout training. This confirms that (1) tool use does not dominate end-to-end cost, (2) asynchronous execution amortizes tool latency effectively, and (3) model improvements directly translate into more efficient tool usage and higher final accuracy on AIME (right plot).

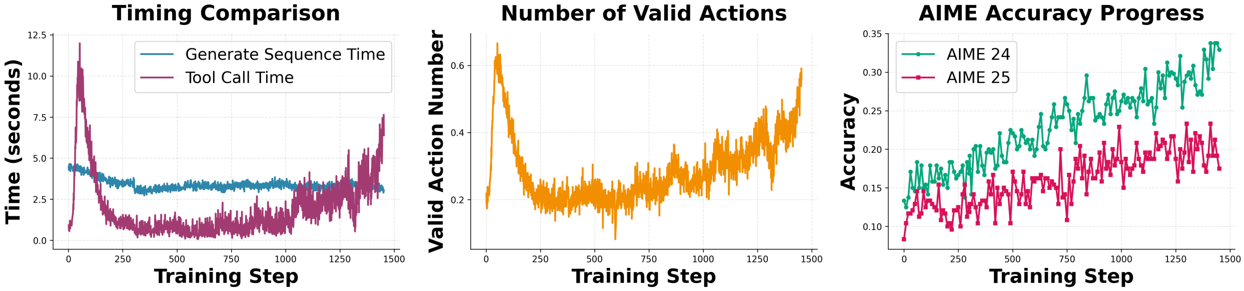


Figure 9: Training metrics for Math-TIR tasks on Qwen-1.5B-Math model

E Ablation on Tokenization Strategies

In this section, we provide additional evidence on why the tokenization strategy used in VerlTool—specifically, avoiding retokenizing model outputs—is important for stable agentic RL. We show that (1) concurrent work has independently validated the same issue, and (2) our own experiments confirm that retokenization significantly affects both training stability and downstream performance.

E.1 Concurrent Work from VLLM Confirms the Importance of “No Retokenization”

vLLM published a detailed analysis ("No More Retokenization Drift: Returning Token IDs via the OpenAI Compatible API Matters in Agent RL", Oct 22, 2025) after the formation of this paper and highlighted two key issues: (1) the same string can produce *different token ID sequences* when retokenized, and (2) this retokenization drift can cause *rapid collapse* in agentic RL training. Figure 10 summarizes their findings.

These observations are fully consistent with what we raised in our paper (Figure 11), despite being discovered independently and after the completion of our paper. This strengthens the significance and correctness of the issue we identified.

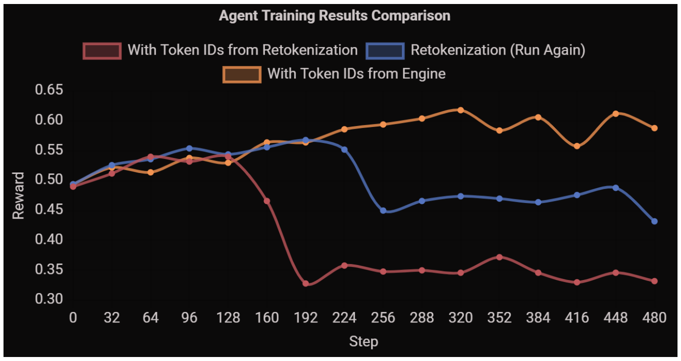


Figure 10: Retokenization drift analysis from the Agent Lightning blog.

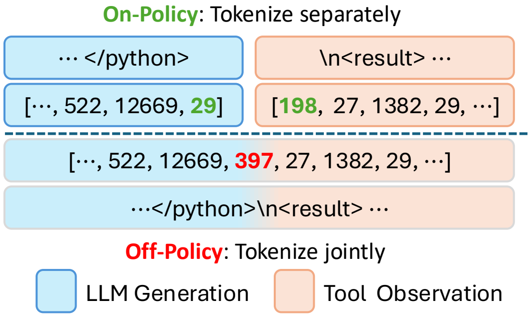


Figure 11: Retokenization boundary issue highlighted in our paper.

E.2 Our Additional Experiments Validating the Same Phenomenon

To directly measure the impact of retokenization on agentic RL using VerlTool framework, we conducted additional experiments on the SQL task (Spider). The results in Figure 12 show three effects:

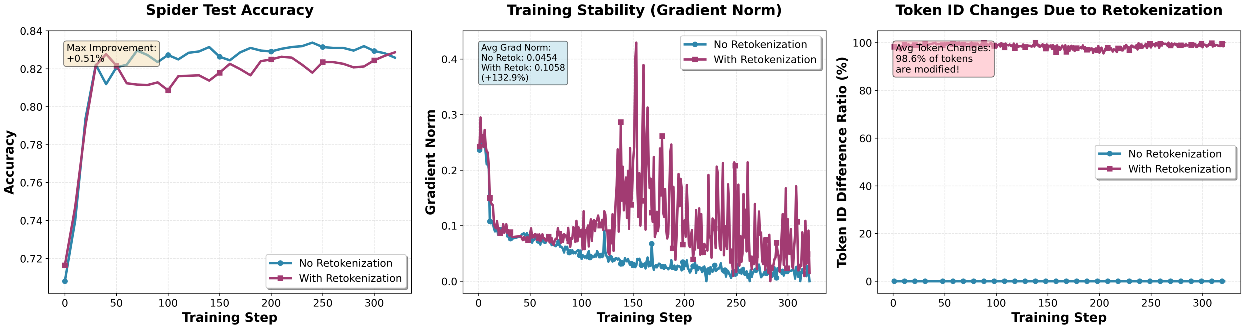


Figure 12: Ablation of tokenization strategies in our SQL agentic RL experiment. Retokenization causes instability and performance degradation.

- **(1) Performance drift:** The retokenized model exhibits inconsistent or degrading reward dynamics, matching the collapse behavior noted by vLLM. In contrast, using the original token IDs provided by the model yields smooth and monotonic improvement.
- **(2) Training instability:** Retokenization inflates the gradient norm by **132.9%** on average, indicating unstable policy updates.
- **(3) Large token-ID mismatch:** Retokenization modifies **98.6%** of token IDs during training, demonstrating that token boundary inconsistencies are pervasive and not rare corner cases.

Together, these results show that retokenization does not merely create cosmetic differences—it substantially alters the underlying training trajectory and destabilizes policy optimization.

F Detailed Experiment Setup

We evaluate VerlTool across six diverse domains to demonstrate its effectiveness in tool-augmented reasoning. Each task domain presents unique challenges and requires different tool integration strategies, allowing us to comprehensively assess the framework's adaptability and performance.

F.1 Mathematical Reasoning with Python Executor (VT-Math)

Mathematical reasoning tasks often involve complex computations that are prone to numerical errors when performed purely through natural language reasoning. To address this limitation, we integrate a Python code interpreter tool that enables agents to execute mathematical calculations reliably and verify intermediate results. We train a mathematical-coding agent that issues Python snippets to a sandboxed interpreter and processes execution traces.

We use DeepMath (He et al., 2025) as our training dataset. The reward function combines answer accuracy with tool usage incentives:

$$R_{\text{acc}}(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if match}(\mathbf{y}, \mathbf{y}_g) \\ -1 & \text{otherwise} \end{cases}, \quad R_{\text{tool}}(\mathbf{x}, \mathbf{y}) = \begin{cases} 0 & \text{if match}(\mathbf{y}, \mathbf{y}_g) \\ -0.25 & \text{otherwise} \end{cases} \tag{6}$$

where the final reward is $R_{\text{math}} = R_{\text{acc}}(\mathbf{x}, \mathbf{y}) + R_{\text{tool}}(\mathbf{x}, \mathbf{y})$. This design encourages the model to explore Python executor usage for problem-solving while maintaining an accuracy focus.

We evaluate on multiple mathematical benchmarks: MATH-500 (Hendrycks et al., 2021), Olympiad (He et al., 2024), Minerva (Lewkowycz et al., 2022), GSM8K (Cobbe et al., 2021), AMC, AIME24, and AIME25, using Math-Evaluation-Harness[1] for standardized eval.

F.2 Knowledge QA with Search Retriever (VT-Search)

Question answering tasks often require access to external knowledge beyond the model's parametric memory, particularly for factual queries and multi-hop reasoning. We integrate a FAISS-based search retriever tool that enables agents to query a local knowledge base and extract relevant information for answering complex questions.

Following prior work (Jin et al., 2025; Song et al., 2025), we integrate an E5 retriever (Wang et al., 2022) and index the 2018 Wikipedia dump (Karpukhin et al., 2020). The agent alternates between search operations and reasoning steps to construct comprehensive answers.

For this task, we apply accuracy as the primary reward:

$$R_{\text{search}}(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if match}(\mathbf{y}, \mathbf{y}_g) \\ -1 & \text{otherwise} \end{cases} \tag{7}$$

We evaluate using Exact Match scores on General Q&A benchmarks (NQ (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), PopQA (Mallen et al., 2022)) and Multi-hop Q&A benchmarks (HotpotQA (Yang et al., 2018), 2Wiki (Ho et al., 2020), MuSiQue (Trivedi et al., 2022b), Bamboogle (Press et al., 2022)).

F.3 Multi-Turn SQL Query Generation (VT-SQL)

Natural language-to-SQL (NL2SQL) conversion requires understanding database schemas and translating natural language queries into executable SQL commands. This task benefits from tool integration as it allows iterative query refinement based on execution feedback and error correction.

We assess SQL Executor adaptability using the SkyRL-SQL training set (Liu et al., 2025) with Qwen2.5-7B-Instruct as the base model. The agent translates natural language questions into executable SQL, given schema hints and tool-calling instructions.

---

[1] https://github.com/ZubinGou/math-evaluation-harness

The reward function focuses solely on execution accuracy:

$$R_{\text{sql}}(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if match}(\mathbf{y}, \mathbf{y}_g) \\ -1 & \text{otherwise} \end{cases} \tag{8}$$

Following standard conventions, we evaluate execution accuracy (EX) on SPIDER-1.0 (Yu et al., 2018) (Dev and Test splits), SPIDER-DK (Gan et al., 2021b), and SPIDER-SYN (Gan et al., 2021a).

F.4 Visual Reasoning with Image Operations (VT-VisualReasoner)

Traditional visual reasoning tasks are conducted primarily in the text modality, where models cannot dynamically process images as actions. To address this limitation, we implement image operation tools that enable agents to zoom into specific image regions, select key frames, and perform other visual manipulations to enhance reasoning over dense visual information, following the Pixel-Reasoner (Su et al., 2025) approach.

The reward design incorporates both accuracy-oriented and compositional complexity measures:

$$\begin{aligned} R_{\text{visualreasoner}}(\mathbf{x}, \mathbf{y}) &= r(\mathbf{x}, \mathbf{y}) + \alpha \cdot r_{\text{curiosity}}(\mathbf{x}, \mathbf{y}) + \beta \cdot r_{\text{penalty}}(\mathbf{y}), && (9) \\ \text{where} \quad r_{\text{curiosity}}(\mathbf{x}, \mathbf{y}) &= \max(H - \text{RaPR}(\mathbf{x}), 0) \cdot \mathbf{1}_{\text{PR}}(\mathbf{y}) && (10) \\ r_{\text{penalty}}(\mathbf{y}) &= \min(N - \mathbf{n}_{\text{vo}}(\mathbf{y}), 0) && (11) \end{aligned}$$

where $\text{RaPR}(\mathbf{x})$ denotes the ratio of responses that invoke tool calls and $\mathbf{n}_{\text{vo}}(\mathbf{y})$ denotes the number of actions that response $\mathbf{y}$ invokes. Hyperparameters are set as $H = 0.3$, $N = 1$, $\alpha = 0.5$ and $\beta = 0.05$. We train two variants using accuracy-only reward and the original complexity-driven reward, denoted as “GRPO-acc” and “GRPO-complex” respectively.

We use the official training dataset from Pixel-Reasoner and evaluate primarily on V-Star (Wu & Xie, 2024), which assesses MLLM visual search capabilities.

F.5 Agentic Web Search (VT-DeeepSearch)

Open-web question answering requires real-time information retrieval and multi-step reasoning over diverse web sources. GAIA (Mialon et al., 2023) and HLE (Phan et al., 2025) are representative benchmarks testing these capabilities. We implement a Web Search tool using Google Search API through SERPER with caching, enabling agents to perform dynamic information gathering and synthesis from online sources.

We apply both accuracy and tool-usage rewards to encourage effective search behavior:

$$R_{\text{deepsearch}}(\mathbf{x}, \mathbf{y}) = R_{\text{acc}}(\mathbf{x}, \mathbf{y}) + R_{\text{tool}}(\mathbf{x}, \mathbf{y}), \quad \text{where } R_{\text{tool}}(\mathbf{x}, \mathbf{y}) = \begin{cases} 0.1, & \text{if tool is called} \\ 0, & \text{if no tool call} \end{cases} \tag{12}$$

We use 1K mixed training examples from SimpleDeepSearcher (Sun et al., 2025) and Web-Sailor (Li et al., 2025b), following the setting in Dong et al. (2025). Starting from Qwen3-8B, we evaluate on GAIA and HLE (text-only) benchmarks. We retrieve top-k URLs for each query and use the returned snippets as content during RL training. For evaluation, we employ two settings: “Snippet-Only” aligns with training conditions using only snippet content, while “QwQ-32B” uses a browser agent to summarize raw content from retrieved URLs.

F.6 Software Engineering Benchmark (VT-SWE)

Software engineering tasks require code understanding, localization, debugging, and modification capabilities that benefit from iterative execution and testing. We integrate bash terminal and code execution tools to enable agents to interact with software development environments effectively.

We build on the R2E-Gym scaffold (Jain et al., 2025) and its training dataset R2E-Lite, using Qwen3-8B in no-think mode as the base model. The reward function is defined strictly by task completion accuracy: an agent must terminate normally and pass all verification tests to receive a reward of 1; otherwise, the reward is 0:

$$R_{\text{swe}}(\mathbf{x},\mathbf{y}) = \begin{cases} 1 & \text{if execution terminates successfully and all tests pass} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

We evaluate performance on the SWE-Verified benchmark, measuring the agent's ability to resolve software engineering tasks and pass verification tests.

For training and evaluation, we maintain a cluster of eight servers (each with 64 CPU cores and 200 GB memory), orchestrating sandbox services via Kubernetes. Each task runs inside the official Docker image provided by R2E-Lite, initialized with 1 CPU and 2 GB memory, elastically scalable up to 2 CPUs and 4 GB memory. We observed that the main bottleneck lies in disk I/O during Docker initialization. To stabilize training, we therefore allocate more CPU and memory resources than are minimally required.

The modular architecture of VerlTool, which separates training from environment services, allows us to scale sandbox environments efficiently. Each environment interaction is given a 90-second timeout, reward computation has a 300-second timeout, and the maximum time per trajectory is capped at 20 minutes. Any trajectory that times out, encounters an exception, or exceeds the length limit is assigned a reward of 0, and its gradients are masked during updates.

F.7 Supported Tools

Table 14: Currently supported tools of VerlTool framework.

Tools	Description	Related works
Code Interpreter	Execute Python code	ToRL (Li et al., 2025c)
Faiss Search	Vector similarity search for documents	Search-R1 (Jin et al., 2025)
Web Search API	Real-time web search and retrieval	SimpleDeepSearch (Sun et al., 2025)
Image Processing	Image resize, video frame selection	PixelReasoner (Su et al., 2025)
Bash Terminal	Execute shell commands	R2E-Gym (Jain et al., 2025)
SQL Executor	Database queries and data management	SkySQL (Liu et al., 2025)
MCP Interface	Model Context Protocol for external tool	ToolRL (Feng et al., 2025)

F.8 Training and Evaluation Configurations

Table 15 summarizes the detailed configurations for each task during training and evaluation. Due to configuration differences across tasks, there may be gaps between validation curves and final downstream evaluation performance, as illustrated in Figure 4.

The RL training parameters vary across tasks to accommodate different complexity levels and interaction patterns. Math-TIR and Pixel-Reasoner use smaller batch sizes due to computational constraints, while Search-R1 employs larger batch sizes for stable retrieval learning. The agentic tool use parameters reflect task-specific requirements: Math-TIR typically requires single-turn interactions, while SWE tasks may require up to 100 interaction turns for complex debugging scenarios.

Evaluation parameters are configured to balance comprehensive assessment with computational efficiency. Temperature settings range from 0.0 for deterministic tasks like SQL generation to 0.6 for creative tasks requiring exploration. Maximum turn limits reflect task complexity, with simple QA tasks limited to 2-5 turns while software engineering tasks allow up to 100 turns for thorough problem resolution.

G Multi-Tool Training within Single Model

We attribute the collapse of models trained with three or more tool types to two factors: (1) the lack of an effective cold-start phase, which leads the policy to explore invalid tool–task combinations early in training, and (2) conflicting optimization objectives across heterogeneous tasks, which produce unstable gradients and mutually interfering reward signals. While single-tool and closely related two-tool settings (e.g., Math-SQL) can share representations, broader multi-tool mixtures require additional prior knowledge, tool-specific warm-up stages, or hierarchical routing to avoid destructive interference. We leave these techniques for future work.

H Occasional Performance Gain Discussion

In this section, we discuss the possible reasons behind the occasional performance gains between our agents and baselines. We conclude that this is mainly due to differences in **Training Parameter Settings**, **Tool-Calling Formats**, and **Maximum Response Length**, with detailed analysis.

H.1 Parameter Differences

Parameter difference can have a noticeable impact on agents' performance. For VT-Search-zero, our training parameter differs from the original Search-R1 in the following key aspects. Firstly, we set the maximum tool response length to 2048 while the original Search-R1 uses 500. As discussed below, this parameter has a significant impact on the performance of local dense retrieval agents. Additionally, we set the group size in GRPO as 16 while Search-R1 is 5. For VT-VisualReasoner, it falls slightly short of the PixelReasoner, largely because our agent was trained for only 50 steps, whereas the official baseline is obtained from a full 400-step model.

Table 15: Training and evaluation configurations across all six tasks.

Tasks	**VT-Math**	**VT-Search**	**VT-SQL**	**VT-VisualReasoner**	**VT-DeepSearch**	**VT-SWE**
			RL Training Parameters			
Rollout BS	128	512	256	128	128	32
N Samples	16	16	5	8	16	8
Gradient BS	128	64	256	128	128	32
Temperature	1.0	1.0	0.6	1.0	1.0	1.0
Top P	1.0	1.0	0.95	1.0	1.0	1.0
Learning Rate	1e-6	1e-6	1e-6	1e-6	1e-6	2e-6
Val Temperature	0.0	0.0	0.0	0.0	0.0	0.0
Val Top P	1.0	1.0	0.95	1.0	1.0	1.0
			Agentic Tool Use Parameters			
Max Turns	1	2	5	3	5	100
MTRL	✘	✘	✘	✔	✘	✔
Max Prompt Length	1024	4096	4096	16384	2048	10240
Max Response Length	3072	4096	4096	16384	8196	22528
Max Action Length	2048	2048	2048	2048	8196	10240
Max Observation Length	512	1024	1024	8192	4096	10240
Action Stop Tokens	‘‘‘output	</search>, </answer>	</sql>	</tool_call>	</python>, </search>	</function>
			Evaluation Parameters			
Temperature	0.6	0.0	0.0	0.0	0.6	1.0
Top P	0.95	1.0	1.0	1.0	0.95	1.0
Max Turns	4	2	5	5	10	100
Max Prompt Length	1024	4096	4096	16384	2048	-
Max Response Length	3072	4096	4096	16384	32768	40960
Max Action Length	2048	2048	2048	4096	16483	-
Max Observation Length	512	1024	1024	8192	4096	-

Table 16: Key parameter differences between our VT-* agents and the original baselines.

Agent	**Parameter**	**Ours**	**Original Baseline**
VT-Search-zero	Max tool response length l	2048	500
	GRPO group size	16	5
VT-VisualReasoner	Training steps	50	400

H.2 Scaling Effects Analysis on the Maximum Response Length

For context-dependent tasks, scaling the response length can significantly impact the agent's performance. For the Search-R1 (local dense retrieval) task, we scaled the maximum tool response length l from 512 to 4096. As shown in the table below, setting l to 512 yields suboptimal performance, whereas scaling to 1024 and 2048 results in noticeable performance gains. However, scaling further has little improvement in model performance. This is because shortening l risks truncating the information that models could use to answer questions correctly. On the other hand, as the returned documents are ranked by semantic relevance, including less related documents may distract the model and hinder it from identifying the most crucial clue.

Table 17: Effect of scaling the maximum tool response length l on Search-R1 Task, best-performing metrics are highlighted in bold.

Benchmarks	l**=512**	l**=1024**	l**=2048**	l**=4096**
NQ	0.234	**0.475**	0.473	0.469
TriviaQA	0.445	0.634	**0.637**	0.628
PopQA	0.179	0.469	0.485	**0.488**
HotpotQA	0.206	0.351	0.348	**0.352**
2Wiki	0.259	0.324	0.331	**0.334**
Musique	0.043	0.090	0.087	**0.093**
Bamboogle	0.088	**0.232**	0.200	0.192

I More related works

I.1 Organization

In this section, we first establish the definition of Agentic Large Language Models (Agentic LLMs). Then, prior tool integration efforts in LLMs and the shift from single-turn, prompt-driven tool-calling to instruction tuning/RL-based multi-turn agentic interaction are reviewed. Further, we introduce various representative domain-specific tasks that are proven to benefit from developing corresponding tool-use oriented agents. We then distinguish Reinforcement Learning with Verifiable Rewards (RLVR) from Agentic Reinforcement Learning with Tool Use (ARLT). Finally, we survey existing systems for training RL-based tool-using agents and position our work: VerlTool.

I.2 Tool-Integrated Reasoning

Augmenting Large Language Models (LLMs) with external tools has become a prominent approach to address limitations in parametric reasoning and enable more complex task solving (Shen, 2024). Early approaches focused on *prompt-based integration*, where systems like HuggingGPT (Shen et al., 2023), Chameleon (Lu et al., 2023), and MultiTool-CoT (Inaba et al., 2023) used tool manuals, demonstrations, or structured Chain-of-Thought templates to orchestrate sequential tool invocations. While these methods offer plug-and-play convenience, they face challenges when adapting to complex, multi-step tasks due to their reliance on static prompting strategies.

A complementary line of work explores *instruction tuning*, where models are explicitly trained to recognize tool formats and generate appropriate function calls. Representative systems include ToolFormer (Schick et al., 2023), which uses bootstrapped annotations to teach tool usage patterns, GPT4Tools (Yang et al., 2023), which distills tool-use traces from more capable models, and LIMO (Ye et al., 2025), which demonstrates that targeted examples can elicit extended reasoning chains. However, these supervised approaches primarily provide static guidance and lack mechanisms for dynamic error correction based on tool execution feedback.

In contrast, reinforcement learning approaches enable models to develop adaptive tool-calling strategies through interaction-based training. Our work builds on this direction by employing GRPO training to enhance models' capabilities for self-reflection and iterative refinement in response to tool feedback.

I.3 From Tool Integration to Agentic LLMs

Large language models (LLMs) demonstrated exceptional flexibility and generality with model parameter and training-data scaling (Team et al., 2024; Qwen et al., 2025; OpenAI, 2025). With recent research advancements dramatically enhancing their capability in reasoning, information retrieval, and instruction-following (Ke et al., 2025), current research trends have shifted from developing simple LLM-based tool-callers to empowering LLMs with versatile interaction capability to facilitate their agentic actions in the world (Plaat et al., 2025).

Agentic LLMs. As one of the central concepts in artificial intelligence (Russell & Norvig, 2016), Decision-making, identifying environmental changes, communication, and acting on one's goal or will are generally defined as common traits of Agency (Epstein & Axtell, 1996; Wooldridge, 1999; Gilbert, 2019). Following well-established conventions, we denote Agentic LLMs as models that reason, act through tools, and interact over one or multiple turns, maintaining state and revising plans according to environmental observations, framing agent behavior beyond passive text-generation, and motivating the possession of improved planning and tool-use capabilities.

Agentic Tool-calling Acquisition. As one fundamental capability, tool-calling has been widely studied as one of the most effective ways of empowering Agentic LLMs with the capability to act in exposed environments. Early attempts involve prompt-based orchestration, which structures function calls in a training-free manner with specific instructions and tool-calling schemas (Yao et al., 2022; Lu et al., 2023), exploiting

LLMs' instruction-following capability through Chain-of-Thought technique (Wei et al., 2022) and explicit task decomposition (Kim et al., 2023; Huang et al., 2022) or multi-agent orchestration (Shen et al., 2023; Ruan et al., 2023). Instruction-tuning based tool callers learn function-call schemas and appropriate tool-call choices through supervised traces (Schick et al., 2023; Kong et al., 2023; Gou et al., 2023), improving problem-solving capabilities while suffering from a lack of generality and remaining largely single-turn (Qu et al., 2024). Reinforcement-Learning-based acquisition further enhances tool-using generality and multi-turn behaviors (Li et al., 2025c; Feng et al., 2025) through problem-solving outcomes and tool-feedback, facilitating environment exploration, self-reflection, error-corrections, and enhanced performance on reasoning-intensive tasks through informative tool responses. (Moshkov et al., 2025).

I.4 Reinforcement Learning with Verifiable Reward (RLVR)

Conventional RLHF pipelines, such as DPO (Rafailov et al., 2023) and PPO (Schulman et al., 2017), optimize answer-level quality and require reward models with substantial sizes. With the introduction of GRPO in DeepSeekMath (Shao et al., 2024), as a critic-free variant of PPO, it enables stabilized RL training on long reasoning chains and enables the integration of multi-turn tool responses and verifiable rewards in reinforcement learning. Existing works have explored the potential of extending GRPO training through the inclusion of rule-based verifiable rewards such as format-based rewards and exact-match based comparisons, as well as tool-calling responses. They demonstrated significant success in developing expert agents in multiple domains with tool-calling and self-reflection capabilities.

By integrating GRPO and verifiable rewards, advancements have been witnessed in a wide range of domain-specific tasks. ToRL (Li et al., 2025c) integrates Python Code Interpreters into the GRPO training of solving mathematical tasks, surpassing RL baselines which does not have tool-calling integration on multiple mathematical datasets. ToolRL (Qian et al., 2025) explored and analyzed the impact of reward design and tool choice on the effect of tool-integrated GRPO training, revealed the significance of tool-use reward design in boosting LLMs' tool calling and generalization capability, as well as achieving stable GRPO training. ReTool (Feng et al., 2025) integrated tool-calling in PPO training, resulting in a Python tool-enabled agent with strong mathematical problem-solving capability. ExCoT (Zhai et al., 2025) and Think2SQL (Papicchio et al., 2025) demonstrated preliminary success of utilizing GRPO and the response of SQL executors to enhance the base model's performance on natural language-to-SQL (NL2SQL) tasks through comprehensive reward design and challenging training problem-filtering. By integrating Faiss (Douze et al., 2024)-based local retrievers and API-based online search services, Search-R1 (Jin et al., 2025) and R1-Searcher (Song et al., 2025) equip base models with enhanced search tool-calling and retrieval capability, achieving superior performance across multiple retrieval-centric benchmarks.

I.5 Agentic Reinforcement Learning with Tool Use (ARLT)

Despite RLVR optimizing policies' behaviors using rule-based verifiable checks with the inclusion of single-turn tool-calling, current RLVR-based agentic models' limited capability in making real multi-turn, long-horizon interactions with tools and perform effective self-reflection and dynamic plan revising remains to be the gap to the generalist Agent-LLM. The inclusion of training the model over dynamic, long-term interactions while exposing it to intermediate tool responses could shape the agentic model's subsequent actions, pushing beyond single-turn verification toward long-horizon, interaction-centric RL training.

Therefore, we use Agentic Reinforcement Learning with Tool Use (ARLT) to denote reinforcement learning on dynamic, multi-turn trajectories in which tool responses are treated as environmental observations that condition future actions. ARLT therefore (i) assigns credit across tool calls rather than only at the final answer, (ii) handles observation tokens explicitly (e.g., masking non-model tokens during GRPO optimization), and (iii) relies on asynchronous, failure-aware executors for adapting potentially slow, stochastic, or error-prone (Plaat et al., 2025; Ke et al., 2025) tool calls. In contrast to RLVR, ARLT targets exploration, re-planning, and recovery from tool failures, and is naturally suited to settings where the problem-solving requires probing the environment before coming up with correct solutions. In this work, our framework is mainly evaluated on the following domain-specific tasks.

Mathematical Interactive Coding. Tool-integrated reasoning was first introduced to tackle computationally intensive mathematical problems by combining natural language reasoning with programming strategies (Chen et al., 2022; Yue et al., 2023; Jin et al., 2025; Song et al., 2025; Wang et al., 2024a; Chen et al., 2022). Building on this idea, Wang et al. (2023) proposed an iterative method that couples textual reasoning with code execution to cross-check the answers, improving the accuracy. More recently, Chen et al. (2025) incorporated code execution into reasoning through supervised fine-tuning on curated code-integrated CoT data. Yet this method is limited by its dependence on specific data distributions and cannot learn adaptive tool-use strategies—such as when and how to call tools—via reinforcement learning. To solve this, concurrent work, including ToRL (Li et al., 2025c) and ZeroTIR (Mai et al., 2025), applies ZeroRL to train agents for mathematical code interpreter use.

Agentic Search and Retrieval. Large language models (Gemini, 2024; OpenAI, 2025) possess a huge amount of intrinsic knowledge while struggling at domain-specific, knowledge-centric tasks (Chen et al., 2024b; Peng et al., 2023) and suffer from hallucination (Zhang et al., 2023; Huang et al., 2023). A common approach to mitigate this issue is by integrating search engines into the LLMs. Predominant approaches of search engine integration often fall within two categories: Retrieval Augmented Generation (RAG)-based (Lewis et al., 2020; Gao et al., 2023b) and Tool-calling based retrieval (Schick et al., 2023). As RAG relies on a separate retriever to extract documents in a single turn without the interaction of LLM, it faces challenges of retrieving irrelevant information or returning less useful context (Jin et al., 2024; Jiang et al., 2023). Conversely, Tool-calling-based retrieval enhances LLMs' capability of calling the search retriever as a tool either through prompting (Yao et al., 2022; Trivedi et al., 2022a), fine-tuning (Schick et al., 2023), or through reinforcement learning (Jin et al., 2025; Song et al., 2025; Jiang et al., 2025), while enhancing searching agents with multi-turn tool-calling-based retrieval capability remains under-explored.

Natural Language to SQL. Natural language-to-SQL (NL2SQL) refers to the task of generating database-specific query codes for extracting data of interest. Early efforts typically involve developing expert models with an encoder-decoder structure for achieving this goal, where the encoder fuses the database schema and the natural-language query, leaving the answer generation to the decoder module (Wang et al., 2019; Cai et al., 2021; Li et al., 2023; Liu et al., 2024). With the advances of sequence-to-sequence models, especially large language models, the NL2SQL tasks have seen a rapid transfer from simple sequence-to-sequence modeling into developing tabular understanding agents utilizing pre-trained large language models. Existing works mostly focus on prompt enhancement (Gao et al., 2023a), large-scale data synthesis and model finetuning (Li et al., 2024; 2025a), while some recent works explored the realm of Reinforcement Learning-driven NL2SQL agents (Papicchio et al., 2025; Pourreza et al., 2025) and NL2SQL agents featuring multi-round Tool Calling capabilities. (Liu et al., 2025)

J Case Study

In this section, we present case studies of representative tasks, featuring corresponding agents' outputs to illustrate Multi-Turn Tool Calling. The queries are randomly extracted from the underlying benchmarks. For readability, tag markers are colorized according to their functional categories:

1. Internal reasoning tags, `<think>...</think>` are colored in purple.
2. Tool-calling tags are formatted as `<python>/<sql>/<search>`.
3. Observation/Tool Response tags are colored as `<observation>/<information>/<result>`.
4. The final output tags are formulated as `<answer>/<solution>`.

We specifically chose the following four representative tasks for demonstration: **Mathematical Reasoning with Python Executor**, **Knowledge Q&A with Search Retriever**, **Multi-turn SQL Generation**, and **Agentic Web Search**. For a detailed explanation of tag definitions, please refer to Tab.15 and corresponding settings in Appendix's Sec. F.

For Agentic Web Search, Multi-turn SQL Generation, and Knowledge Q&A with Retriever, the text-heavy responses from the tools are truncated or simplified.

Table 18: Example answer demonstration of VT-Math

Example from DeepMath Dataset

Question:
Determine the function f given that it has a power series representation centered at 0, converging within the interval $(-1, 1)$, and there exists a sequence of distinct points (x_n) approaching 0 such that $f(x_n) = 0$ for all $n \in \mathbb{N}$. What is f?
Labeled Answer:
0

Model Output by VT-Math GRPO

Let's consider the problem. The function f has a power series representation centered at 0 that converges within the interval $(-1, 1)$. This means that $f(x)$ can be written as: $f(x) = \sum_{n=0}^{\infty} a_n x^n$ where the series converges for $|x| < 1$. We are given that there exists a sequence of distinct points (x_n) approaching 0 such that $f(x_n) = 0$ for all $n \in \mathbb{N}$. This implies that: $f(x_n) = \sum_{n=0}^{\infty} a_n x_n^n = 0$ Since the series converges at x_n and $x_n \to 0$, we can conclude that each $a_n = 0$ for all $n \geq 1$ because if any $a_k \neq 0$ for some $k \geq 1$, then the series would not converge to 0 as $x_n \to 0$. Therefore, the only possible term is the constant term a_0. This means we can write:

$$f(x) = a_0$$

Given that $f(0) = a_0 = 0$ (since $f(x_n) \to f(0) = 0$ as $x_n \to 0$), we have:

$$f(x) = 0$$

Therefore, the function f is the zero function. Let's put this understanding into a more formal version:
‘‘‘python

```python
from sympy import symbols, Function, Eq, solve

# Define the variable and the function
x = symbols('x')
f = Function('f')(x)
# Define the power series representation centered at 0
# Since we are given that there exists a sequence (x_n) approaching 0 such that f(x_n) = 0
    for all
# It implies that the function is the zero function
f_zero = 0
# Print the zero function
print(f_zero)
```

‘‘‘
‘‘‘output
result
0
‘‘‘
The power series representation of the function f converging within the interval $(-1, 1)$, given that there exists a sequence of distinct points (x_n) approaching 0 such that $f(x_n) = 0$ for all $n \in \mathbb{N}$, results in the function being the zero function. Therefore, the function f is $\boxed{0}$.

Table 19: Example answer demonstration of VT-SEARCH

Example from HotpotQA Dataset

Question:
What is the birth name of Nadeem Siddique's favorite boxer?
Labeled Answer:
Walker Smith Jr.

Model Output by VT-Search (GRPO)

<think>
To answer this question, I need to find out Nadeem Siddique's favorite boxer and then determine the birth name of that boxer.
</think>
<search>
Who is Nadeem Siddique's favorite boxer?
</search>
<information>
Doc 1(Title: "Nadeem Siddique") Nadeem Siddique Siddique was born in Bradford, Yorkshire, United Kingdom. He supports his local football team Bradford City and **his favorite boxer is Sugar Ray Robinson**...
Doc 2(Title: "Nadeem Siddique") in 2005 for the people of Palestine and Gaza whereby he raised £352,000. He stated 'Me and Junior have been doing loads of things for charities and with our popularity in Bradford we always raise quite a few quid...
Doc 3(Title: "Nadeem Siddique") 3 fights and 3 wins and is on the verge of a major championship fight. Siddique is the founder of 'Make the Weight' gym; the opening of the gym was attended by boxer Mike Tyson in whose...
</information>
From the search results, I found that Nadeem Siddique's favorite boxer is Sugar Ray Robinson.
<search>
What is the birth name of Sugar Ray Robinson?
</search>
<information>
Doc 1(Title: "Sugar Ray Robinson") Robinson was born **Walker Smith Jr**. in Ailey, Georgia, to Walker Smith Sr. and Leila Hurst. Robinson was the youngest of three children...
Doc 2(Title: "Sugar Ray Robinson") disease. He died in Los Angeles at the age of 67 and was interred in the Inglewood Park Cemetery, Inglewood, California. Robinson married Marjorie Joseph in 1938; the marriage was annulled the same year. Their son, Ronnie Smith...
Doc 3(Title: "Sugar Ray Robinson") Jackie Gleason, Nat King Cole, Joe Louis, and Lena Horne among others. Robinson was known as a flamboyant personality outside the ring. He combined striking good looks with charisma and a flair for the dramatic...
</information>
From the search results, I found that the birth name of Sugar Ray Robinson is Walker Smith Jr.
<answer> Walker Smith Jr. **</answer>**

Table 20: Example answer demonstration (SQL reasoning)

Example (Student–Pet DB)

Question:
What are the ids of the students who do not own cats as pets?
Labeled Answer:

```
SELECT stuid FROM student EXCEPT SELECT T1.stuid FROM student AS T1 JOIN has_pet AS T2 ON
    T1.stuid  =  T2.stuid JOIN pets AS T3 ON T3.petid  =  T2.petid WHERE T3.pettype  =  ’
    cat’
```

Model Output (Agentic SQL)

<think>I need to find the student IDs of students who do not own cats as pets. To achieve this, I will first identify all students who own cats and then exclude them from the list of all students.
</think>
<sql>

```
SELECT StuID FROM Has_Pet WHERE PetID IN (SELECT PetID FROM Pets WHERE PetType = ’cat’);
```

</sql>
<observation>
0
1001
<reminder>You have 5 turns left to complete the task.</reminder>
</observation>
<think>I have the list of student IDs who own cats. Now, I will use a NOT IN clause to exclude these IDs from the list of all students.
</think>
<sql>

```
SELECT StuID FROM Student WHERE StuID NOT IN (SELECT StuID FROM Has_Pet WHERE PetID IN (
    SELECT PetID FROM Pets WHERE PetType = ’cat’));
```

</sql>
<observation>
0
1002
1003
...
<reminder>You have 4 turns left to complete the task.</reminder>
</observation>
<think>I have the list of student IDs who do not own cats as pets. Since the question asks for the ids of the students, I can now output the final SQL as the solution.
</think>
<solution>

```
SELECT StuID FROM Student WHERE StuID NOT IN (SELECT StuID FROM Has_Pet WHERE PetID IN (
    SELECT PetID FROM Pets WHERE PetType = ’cat’));
```

</solution>

Table 21: Example answer demonstration of VT-DeepSearch

Example from Gaia Dataset

Question:
Find the value of x to the nearest tenth: Lx = (d/dx * (A * x-squared)) + 4-thousand'n'ninety-7 minus C Where L is the last two digits of the year of the Venezuelan Declaration of Independence, A is the number of colors in the TikTok logo as of July 2023, excluding black and white, and C is the height of the average woman in the Philippines according to a July 2023 Business Insider article, rounded to the nearest whole centimeter.
Labeled Answer:
563.9

Model Output by VT-DeepSearcher (GRPO)

<think> Okay, let's try to figure out this problem step by step. First, I need to understand what each part of the equation means and find the values for L, A, and C. Then I can plug them into the equation to solve for x.
The equation given is Lx = (d/dx * (A * x-squared)) + 4-thousand'n'ninety-7 minus C. Wait, the wording here is a bit confusing. Let me parse it again. Maybe it's L multiplied by x equals the derivative of (A times x squared) plus 4097 minus C. Or maybe there's a typo? Let me check the original question again.
The equation is written as: Lx = (d/dx * (A * x-squared)) + 4-thousand'n'ninety-7 minus C. Hmm, the "4-thousand'n'ninety-7" part might be a way to write 4097. So maybe it's 4097. So the equation is Lx equals the derivative of A x squared plus 4097 minus C.
<search> Venezuelan Declaration of Independence year </search>
<result>
The Venezuelan Declaration of Independence occurred on ****July 5, 1811****. The last two digits of this year are ****11****, which corresponds to L...
</result>
So L = 11.
<search> TikTok logo colors 2023 </search>
<result> The TikTok logo as of 2023 features ****two colors excluding black and white**** : pink (#FE2C55) and aqua/cyan (#25F4EE). Thus A = 2...
</result>
So A = 2.
<search> average height of women in Philippines 2023 Business Insider </search>
<result> The average height of women in the Philippines is ****149.6 cm****, rounded to ****150 cm****. Thus C = 150...
</result>
So C = 150.
Now the equation is:
$Lx = (d/dx(Ax^2)) + 4097 - C$.
Derivative of Ax^2 is $2Ax = 4x$.
Substituting $L = 11, A = 2, C = 150$:
$11x = 4x + 4097 - 150 = 4x + 3947$.
Simplify: $7x = 3947 \Rightarrow x = 3947/7$.
<python>

```
3947 / 7
```

</python>
<result>
563.9
</result>
<answer>
The final answer is $\boxed{563.9}$.
</answer>
</think>

K The use of Large Language Models

We have utilized Claude and ChatGPT's website clients primarily for writing enhancement tasks, including eliminating potential grammar errors, improving the logical coherence of existing content, and reorganizing tables and figures for better presentation. Additionally, we employed these tools for data visualization, such as generating Figure 4 from our collected data. We also leveraged ChatGPT's deep research capabilities to identify relevant works in agentic RL and infrastructure, which are comprehensively presented in section 2 and Appendix I.