ToolOmni: Enabling Open-World Tool Use via Agentic learning with Proactive Retrieval and Grounded Execution

Anonymous ACL submission

Abstract

Large Language Models (LLMs) enhance their problem-solving capability by utilizing external tools. However, in **open-world** scenarios with massive and evolving tool repositories, existing methods relying on static embedding retrieval or parameter memorization of tools struggle to align user intent with tool semantics or generalize to unseen tools, respectively, leading to suboptimal accuracy of open-world tool retrieval and execution. To address these, we present **ToolOmni**, a unified agentic framework that enables LLMs for open-world tool use by proactive retrieval and grounded execution within a reasoning loop. First, we construct a cold-start multi-turn interaction dataset to instill foundational agentic capabilities via Supervised Fine-Tuning (SFT). Then, we introduce open-world tool learning based on a **Decoupled Multi-Objective GRPO** algorithm, which simultaneously optimizes LLMs for both tool retrieval accuracy and execution efficacy in online environments. Extensive experiments demonstrate that ToolOmni achieves state-of-the-art performance both in retrieval and execution, surpassing strong baselines by a significant margin of **+10.8%** in end-to-end execution success rate, while exhibiting exceptional robustness and generalization capabilities.

1 Introduction

Tool Learning with LLM achieves higher accuracy, efficiency, and autonomy in problem solving by combining the strengths of specialized tools and foundational models (Nakano et al., 2021; Yao et al., 2022, 2023; Schick et al., 2023; Qin et al., 2024; Wu et al., 2023). Efforts in this field have predominantly focused on teaching models to effectively use tools via demonstration-based learning, typically leveraging Supervised Fine-Tuning(SFT) on curated expert trajectories (Nakano et al., 2021; Yao et al., 2022; Schick et al., 2023), or feedback-based learning, which aligns model behavior with feedback from both environment and human through Reinforcement Learning (Schulman et al., 2017; Christiano et al., 2017; Nakano et al., 2021; Baker et al., 2022; Li et al., 2025b; Qian et al., 2025). However, in **open-world** scenarios characterized by massive and dynamically updated tool repositories, models must not only understand how to use tools but also master the ability to search and select the correct ones.

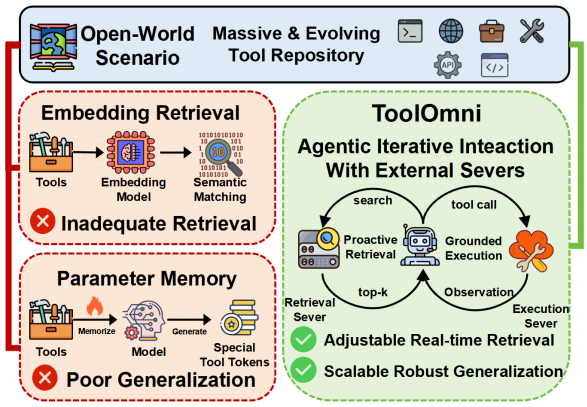

Figure 1: Motivation for **ToolOmni** in Open-World Scenarios: Embedding retrieval methods struggle with *Massive tools*, often resulting in low retrieval accuracy due to shallow matching; Parameter memory methods fail to adapt to *Evolving tools*, suffering from poor generalization to unseen tools. **ToolOmni** overcomes these limitations via a unified agentic framework that couples **Proactive Retrieval** with **Grounded Execution** , enabling effective open-world tool use.

As shown in Fig.1, to help narrow the scope of relevant tools, prevailing solutions typically adopt a pipeline approach, employing embedding models to retrieve relevant tools based on query similarity before passing them to the execution agent (Qin et al., 2024; Chen et al., 2024; Xu et al., 2024). However, this paradigm operates passively, which decouples the retrieval process from the agent's reasoning, preventing the execution model from proactively participating in tool selection or refining the search based on task-specific needs. Conse-

quently, these methods often struggle to effectively **align user intent** with the functionally essential tools in complex scenarios. Alternatively, some approaches (Wang et al., 2025; Schick et al., 2023; Su et al., 2025) fine-tune models to internalize tool documentation into parametric knowledge. While effective, this paradigm requires expensive retraining whenever the toolset updates, severely limiting **generalizability in dynamic environments**.

Recently, there has been a growing interest in agentic training frameworks that leverage Reinforcement Learning with Verifiable Rewards(RLVR) to unify reasoning with active environment interaction. Building on algorithms like Group Relative Policy Optimization(GRPO) (Shao et al., 2024) and Proximal Policy Optimization(PPO) (Schulman et al., 2017), recent works (Jin et al., 2025b; Xue et al., 2025; Qian et al., 2025; Li et al., 2025b) have demonstrated that LLMs can be trained to iteratively interact with the external environment, effectively invoking tools and leveraging feedback to enhance both performance and generalization. Nevertheless, these works often restrict LLMs to a limited toolset such as search engines and code executors, which constrains their applicability to diverse and dynamic open-world scenarios.

To address the above issues, we propose a unified agentic framework **ToolOmni** that integrates proactive retrieval and grounded execution into a unified end-to-end process, scaling agentic capabilities to dynamic, open-world tool scenarios. To build ToolOmni, we first conduct **Tool Learning cold start** phase with a high-quality hybrid dataset that integrates both retrieval and execution trajectories. This stage enables the model to acquire the foundational capabilities required for effective tool interaction. Building on this foundation, the second stage adopts an **Open World Tool Learning** process based on an enhanced Group Relative Policy Optimization algorithm. Unlike naive GRPO, which relies on a single reward, our framework treats retrieval and execution as interconnected yet distinct sub-tasks, where we compute task-specific rewards and advantages for retrieval and execution independently, and integrate them into a single optimization, enabling the synchronized optimization of both capabilities. This is crucial because decoupling allows us to provide finer-grained process supervision, ensuring that both retrieval recall and execution reasoning are optimized precisely without mutual interference.

To evaluate the effectiveness of ToolOmni, we conduct extensive experiments on the ToolBench benchmark. The results demonstrate that ToolOmni achieves superior performance in both tool retrieval and task execution, particularly in open-world scenarios with massive candidate tools**(+11.9%)**. Additionally, ToolOmni exhibits exceptional robustness to unseen instructions and tools, demonstrating that it learns universal tool-use mechanisms rather than relying on rigid memorization. The contributions of this work can be summarized as follows:

- We introduce **ToolOmni**, an end-to-end tool agentic framework that integrates proactive tool retrieval with grounded execution within a unified reasoning loop.
- We propose a two-stage training strategy that integrates a supervised cold-start for foundational tool retrieval and execution with GRPO-based RL for the subsequent synchronized optimization.
- Extensive experiments demonstrate that **ToolOmni** not only achieves superior performance in both tool retrieval and execution, but also shows strong robustness and generalizability to unseen domains.

2 Related Works

2.1 Tool Retrieval in Open-World Scenarios

In open-world scenarios, a common approach for tool retrieval is to employ an embedding model (Shi et al., 2025; Robertson et al., 2009; Qin et al., 2024; Qu et al., 2024) that retrieves top-k relevant tools based on semantic similarity, narrowing the scope of candidate tools. Some works (Chen et al., 2024; Xu et al., 2024; Zheng et al., 2024; Kachuee et al., 2025) improve retrieval performance by using LLM to rewrite queries or expand tool documentation. Alternatively, another method trains the LLM to encode tool information into its parametric knowledge, enabling the model to directly generate corresponding tool identifiers to accomplish retrieval (Wang et al., 2025; Su et al., 2025). Our approach also utilizes embedding-based retrieval, yet transforms the interaction paradigm where the agent proactively formulates queries and invokes the embedding model as an executable tool.

2.2 LLM Tool Execution

Numerous studies have focused on augmenting LLMs with external tools to enhance their spe-

cialization and efficiency in solving complex tasks (Liang et al., 2024; Xu et al., 2023; Schick et al., 2023; Yao et al., 2022; Nakano et al., 2021). ReAct (Yao et al., 2023) establishes a Thought-Action-Observation loop, where the model generates explicit reasoning traces to justify and orchestrate tool execution in an interleaved manner. ToolLLM (Qin et al., 2024) adopts a tree search framework (DFSDT), allowing the model to explore multiple execution paths and back-track based on tool feedback to solve multi-step tasks. More recently, agentic training frameworks that leverage RLVR have been employed to enhance the LLM's ability to use external tools (Jin et al., 2025b; Xue et al., 2025; Qian et al., 2025; Li et al., 2025b,a). These studies treat external search and code execution as executable tools, facilitating an agentic, end-to-end reasoning paradigm for complex task solving. Inspired by them, ToolOmni broadens this scope to open-world by unifying proactive tool discovery and execution into an end-to-end process for complex task resolution.

3 Methodology

3.1 Problem Formulation

The goal of open-world tool-use agents is to address a user query Q by interacting with a large-scale tool repository $\mathcal{T} = \{t_1, t_2, \ldots, t_N\}$, where N is the number of candidate tools. Given the massive scale of $\mathcal{T}$ (e.g., $N > 10,000$), ToolOmni models the open-world tool-use process as a cascaded retrieval-execution framework driven by the policy π_θ. At each turn t, the agent first performs an iterative *Proactive Retrieval* phase to identify a task-complete set of tools, followed by a *Grounded Execution* phase to invoke them. The trajectory sequence τ and final answer a can be formulated as follows:

$$\tau = \Big\{ \underbrace{(r_t^{ret}, \alpha_t^{ret}, \mathcal{T}_{sub})}_{\text{Proactive Retrieval}}, \underbrace{(r_t^{exe}, \alpha_t^{exe}, o_t)}_{\text{Grounded Execution}} \Big\}_{t=1}^{T},$$
$$a \sim \pi_\theta(Q, \tau) \tag{1}$$

where the cascaded retrieval-execution framework at each turn is determined by the policy:

$$\tau = \begin{cases} (r_t^{ret}, \alpha_t^{ret}) \sim \pi_\theta(\cdot \mid s_t, Q), & \text{Ret.} \\ (r_t^{exe}, \alpha_t^{exe}) \sim \pi_\theta(\cdot \mid s_t, Q, \mathcal{T}_{sub}), & \text{Exec.} \end{cases}$$

At each turn t, the agent's state s_t consists of the history of all previous actions and the corresponding observations, i.e., $s_t = (a_1, s_1, ..., a_{t-1}, o_{t-1})$. In the Proactive Retrieval phase, the agent iteratively interleaves reasoning r_i^{ret} and actions α_i^{ret}, where each action α_i^{ret} either queries q_i to retrieve relevant tools or finalizes the task-complete toolset $\mathcal{T}_{sub}$. After finalizing the toolset $\mathcal{T}_{sub}$, the agent enters the Grounded Execution phase, where it interleaves reasoning r_t^{exe} and tool-call actions α_t^{exe} grounded in the retrieved tool documentation to generate observations o_t or the final answer a.

3.2 Cold-start Tool Learning

To endow ToolOmni with basic tool-use capabilities, we first perform a SFT phase, serving as a cold-start initialization for the model.

Data Curation. Our training dataset is derived from ToolBench (Qin et al., 2024) and carefully curated to support both retrieval and execution phases. For tool retrieval, we first select a subset of 80,000 queries to train a specialized retrieval-only model. Using this model, we perform rejection sampling to remove low-quality instances, resulting in a high-quality corpus of approximately 28,000 retrieval trajectories. For tool execution, we extract around 33,000 trajectories from the ToolBench training set, including both correct and incorrect execution paths. To ensure data quality, we employ Qwen-2.5-32B as an automated judge to rigorously validate each trajectory. Further details are presented in AppendixA.

SFT Objective. Given the dataset of high-quality trajectories $\mathcal{D} = \{(\tau_{ret}, \tau_{exe})\}$, we optimize the model using the standard cross-entropy loss:

$$\mathcal{L}_{SFT} = -\sum_{(x,y)\in\mathcal{D}} \log \pi_\theta(y \mid x) \tag{2}$$

Following prior works (Jin et al., 2025b; Huang et al., 2025; Sun et al.; Jin et al., 2025a), we compute token-level losses exclusively on the agent-generated reasoning traces r and actions α, masking all external observations o. This strategy prevents the model from predicting environmental dynamics, thereby stabilizing the policy gradient optimization.

3.3 Open-world Tool Learning

While SFT establishes basic tool-use capabilities, it relies on imitating high-quality labeled trajectories, restricting exploration of diverse trajectories and limiting scalability to large-scale, open-world tool scenarios. To bridge this gap, we introduce open-world tool learning based on GRPO, which enables

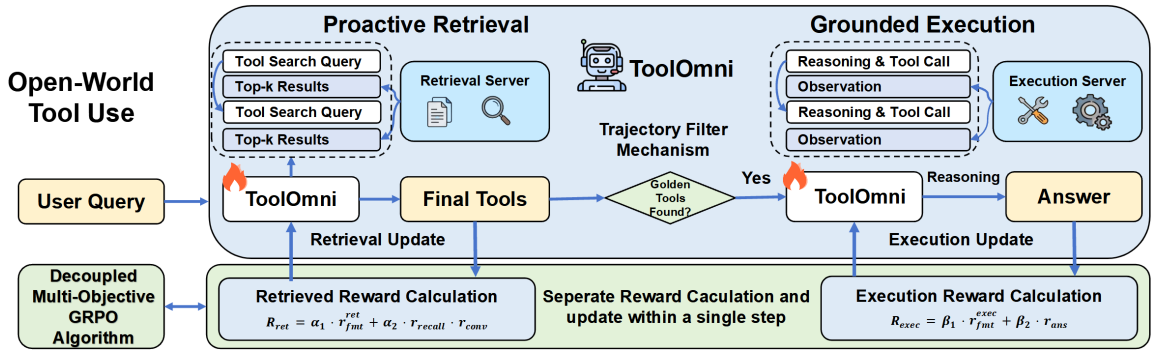

Figure 2: Overview of the **ToolOmni** framework. The pipeline operates in two decoupled phases: **Proactive Retrieval**: The agent iteratively interacts with the retrieval server to curate a candidate tool set. **Grounded Execution**: With retrieval results, the agent performs reasoning and tool invocation to generate the final answer.

the agent to optimize its proactive retrieval and grounded execution actions through iterative trial and error.

3.3.1 Proactive Tool Retrieval

The agentic interaction begins with the proactive tool retrieval phase. Unlike existing tool learning approaches that rely on a single-turn passive retrieval, ToolOmni autonomously determines whether and what to retrieve. Formally, given the user instruction Q, the policy π_θ analyzes the user's intents and then formulates a tool search query q_t encapsulated within special tags: <search> q_t </search>. Upon receiving q_t, the retrieval server encodes it using a pre-trained embedding model $E(\cdot)$ and then performs retrieval. It computes the cosine similarity between the query vector E_{q_t} and the pre-indexed tool embeddings, returning the top-k candidates:

$$\mathcal{T}^t_{ret} = \underset{\tau \in \mathcal{T}}{\text{top-k}} \left(\cos(E_\tau, E_{q_t}) \right), \tag{3}$$

wehre $\cos(\cdot)$ denotes cosine similarity function. ToolOmni iterates proactive retrieval, autonomously generating multiple search queries as needed with real-time retrieval. Once a complete set of tools sufficient for the task has been collected, ToolOmni selects useful tools from the candidates and arranges them into a sorted subset that is finalized within <tool_call> and </tool_call> tags.

$$\mathcal{T}_{sub} = \pi_\theta(P_{ret}, Q, \bigcup_t \mathcal{T}^t_{ret}), \tag{4}$$

3.3.2 Grounded Tool Execution

After obtaining the sorted tool subset $\mathcal{T}_{sub}$, ToolOmni is instructed via the execution prompt $Pexec$ to enter the execution phase. We employ a strategic trajectory filtering procedure to ensure both the training stability of the execution policy and the solvability of the queries. Specifically, we only retain trajectories where the generated subset $\mathcal{T}_{sub}$ successfully recall all ground-truth tools ($\mathcal{T}_{gold} \subseteq \mathcal{T}_{sub}$). Based on these valid retrieval results, the policy π_θ then conducts multi-turn interleaved reasoning and tool invocation to solve the user's query.

Specifically, at each step t, ToolOmni first conducts reasoning inside <reasoning> and </reasoning> tags. Subsequently, it invokes a tool by explicitly specifying the target function name and its required arguments, e.g., <tool_call> {"tool_name": "genderize","tool_input": {"name": "john"}} </tool_call>. To ensure stable and efficient online reinforcement learning, we deploy an *LLM-based Tool Simulator* that emulates the environment's feedback (Guo et al., 2024). The simulator produces realistic execution results for tool invocations, which are enclosed within <information> and </information> tags and appended to the ongoing context. The iterative cycle of reasoning and tool invocation continues until it derives the final solution, which is ultimately presented within <answer> and </answer> tags. Finally, the grounded tool execution process can be formulated as:

$$y = \pi_\theta(P_{exec}, Q, \mathcal{T}_{sub}, O_{tool}). \tag{5}$$

Given the open-ended nature of tool execution, rigid rule-based matching is insufficient to verify the final answer. Instead, we utilize a reward model to evaluate each trajectory holistically. A positive reward is assigned when the agent successfully invokes the required tools and produces a correct final answer.

3.3.3 Open-world Tool Use Rewards

In the RL process, the reward functions are utilized to steer the model towards desired properties. Below, we design retrieval and execution rewards to induce the model to perform effective proactive retrieval and reliable grounded execution.

The retrieval reward R_{ret} comprises three weighted components: ensuring format correctness, maximizing recall of ground-truth tools, and promoting effective conversion of retrieved information:

$$R_{ret} = \alpha_1 \cdot r_{fmt}^{ret} + \alpha_2 \cdot r_{recall} \cdot r_{conv}. \quad (6)$$

Here, $r_{fmt}^{ret} \in \{0, 1\}$ ensures format compliance; $r_{rec} \in [0, 1]$ measures the recall of ground-truth tools $\mathcal{T}_{gold}$ within the cumulative retrieved set $\mathcal{T}_{ret}$; and $r_{conv} \in [0, 1]$ rewards the proportion of these retrieved tools that are ultimately selected into the final set, encouraging effective utilization.

The execution reward R_{exec} is similarly formulated as a weighted combination of format and outcome correctness:

$$R_{exec} = \beta_1 \cdot r_{fmt}^{exec} + \beta_2 \cdot r_{ans}. \quad (7)$$

The term $r_{fmt}^{exec} \in \{0, 1\}$ enforces structural validity of the reasoning path and tool invocations, while $r_{ans} \in \{0, 1\}$ is a binary signal from the reward model indicating the correctness of the answer.

3.3.4 Online Policy Optimization

Group-Relative Advantage Estimation. For each query x, the agent generates a group of G outputs $\{o_1, \ldots, o_G\}$ sampled from the current policy π_θ. We compute the advantages for retrieval and execution independently using the group-relative standard:

$$A_{task}^{(i)} = \frac{R_{task}^{(i)} - \text{mean}(\{R_{task}\}_G)}{\text{std}(\{R_{task}\}_G) + \epsilon} \quad (8)$$

where $task \in \{ret, exec\}$. By normalizing rewards within the group specifically for each subtask, we isolate the learning signals, preventing the sparsity of execution rewards from destabilizing the retrieval learning process.

Decoupled Policy Gradient. Based on the estimated advantages, we optimize the policy π_θ by maximizing the surrogate objective. Consistent with the SFT stage, we apply token-level masking to ensure that gradients are back-propagated solely through the agent-generated tokens. The final optimization objective is formulated as:

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|y_i|}\sum_{t=1}^{|y_i|}\mathbb{I}(t \in \mathcal{M}_{task}) \right.$$
$$\left. \min\left(\rho_{i,t}(\theta)\hat{A}_i, \text{clip}(\rho_{i,t}(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_i\right)\right]$$

Optimization Stability. To further stabilize the training, we implement two key optimizations for GRPO in this context: **(1)Separated Update**: Instead of summing the gradients directly, we perform the updates for retrieval and execution sequentially within a single step. This prevents gradient conflict where the magnitude of one objective overwhelms the other; **(2)Selective Rollout**: As detailed in Sec. 3.3.2, we enforce a strict filtering mechanism where the execution generation is initiated only when the retrieval stage successfully recalls all golden tools ($\mathcal{T}_{gold} \subseteq \mathcal{T}_{ret}$). By excluding invalid retrieval instances prior to rollout, we ensure that the execution policy is trained exclusively on high-quality, grounded contexts.

4 Experiment

In this section, we conduct extensive experiments on the ToolBench benchmark to answer the following Research Questions (RQs): **RQ1:** How does ToolOmni perform in retrieving relevant tools from massive, open-world repositories compared to existing baselines? **RQ2:** Can ToolOmni effectively execute complex, multi-step tasks in an end-to-end manner, surpassing pipeline and unified approaches? **RQ3:** Does the proposed framework demonstrate robustness against retrieval noise and generalization capabilities across unseen tools and domains? **RQ4:** What are the individual contributions of the core components (e.g., iterative retrieval, training stages) and hyperparameters to the overall performance?

4.1 Experimental Setup

We evaluate ToolOmni on **ToolBench** (Qin et al., 2024; Guo et al., 2024), testing across three difficulty levels (I1–I3) and three generalization splits (Instruction, Tool, Category) to assess robustness. Retrieval performance is measured by **NDCG@**k ($k \in \{1, 3, 5\}$), while end-to-end execution is evaluated using **Solvable Pass Rate (SoPR)** and **Win Rate (SoWR)**, computed by a GPT-5 judge. We initialize our model with **Qwen3-4B-Instruct** (Yang

Method	I1			I2			I3			Avg.
	@1	@3	@5	@1	@3	@5	@1	@3	@5	
In-Domain										
BM25*	29.46	31.12	33.27	24.13	25.29	27.65	32.00	25.88	29.78	28.73
EmbSim*	63.67	61.03	65.37	49.11	42.27	46.56	53.00	46.40	52.73	53.35
Re-Invoke*	69.47	-	61.10	54.56	-	53.79	59.65	-	59.55	59.69
IterFeedback	71.64	71.29	76.31	62.65	55.58	60.62	73.85	64.06	69.02	67.22
ToolGen	69.47	72.26	79.12	46.77	53.58	62.45	77.06	76.44	**85.48**	69.18
ToolRetriever	81.91	82.05	**85.57**	75.92	69.61	**75.40**	79.82	72.42	77.11	77.76
ToolOmni	**86.07**	**84.49**	85.56	**81.50**	**73.13**	74.11	**84.40**	**76.81**	77.25	**80.37**
Multi-Domain										
BM25*	22.77	22.64	25.61	18.29	20.74	22.18	10.00	10.08	12.33	18.29
EmbSim*	54.00	50.82	55.86	40.84	36.67	39.55	18.00	17.77	20.70	37.13
ToolGen	68.84	72.00	78.76	46.77	53.55	62.43	75.68	**75.26**	**84.48**	68.64
IterFeedback	71.77	70.49	74.82	63.18	55.41	61.32	67.43	59.23	63.14	65.20
ToolRetriever	81.03	80.88	84.52	75.91	69.53	**75.21**	77.52	69.15	74.24	76.44
ToolOmni	**86.07**	**84.09**	**84.80**	**80.63**	**72.96**	73.91	**78.90**	71.44	71.82	**78.29**

Table 1: Main results of tool retrieval performance (NDCG@k %) on ToolBench. **ToolOmni** achieves the best performance across most metrics. The best performance is boldfaced, while the second-best performance is underlined. **Note**: Methods marked with * report results cited from ToolGen (Wang et al., 2025).

et al., 2025) and train it via the proposed decoupled GRPO ($G = 5, T = 1.0$) on 8 NVIDIA H100 GPUs. For comparison, we benchmark against competitive baselines across both tasks. **For retrieval**, we consider sparse (BM25 (Robertson et al., 2009)), dense (ToolRetriever (Qin et al., 2024),EmbSim[1]), and refinement methods (IterFeedback (Xu et al., 2024), Re-Invoke (Chen et al., 2024)), alongside the generative ToolGen (Wang et al., 2025). **For execution**, we evaluate pipeline agents (GPT, ToolLLaMA (Qin et al., 2024)) paired with ToolRetriever, and the unified generative model ToolGen (Wang et al., 2025). Further details are provided in **Appendix B**.

4.2 Tool Retrieval Performance(RQ1)

We evaluate retrieval under two settings: *In-Domain*, where the search space is restricted to the specific subset of tools relevant to the current test group; and *Multi-Domain*, where the agent must identify the correct tools from the entire repository of over **16,000** APIs. Regarding the mechanism, baselines typically employ static one-shot retrieval. ToolGen uses generative retrieval, while IterFeedback performs multi-turn query refinement. Similarly, ToolOmni adopts an agentic iterative retrieval paradigm. To ensure a fair comparison, we align the search budget of ToolOmni with IterFeedback, limiting both to a maximum of 4 retrieval turns.

As shown in Table 1, **ToolOmni** achieves superior performance across the majority of metrics, particularly in the most rigorous Multi-Domain setting where it attains the highest average NDCG of **78.29%**. It significantly outperforms strong baselines like ToolRetriever and ToolGen in top-1 (@1) and top-3 (@3) precision, demonstrating its superior ability to accurately pinpoint the "golden tools" from a massive repository.

Notably, ToolOmni occasionally scores slightly lower on **NDCG@5**. This is an expected side effect of our *proactive selection mechanism*. Unlike standard retrievers that always return a fixed top-k list, ToolOmni autonomously decides when to stop searching. If a task requires only 1 or 2 tools, our model outputs a concise set rather than padding it with irrelevant candidates to fill the top-5 slots. Thus, this performance gap reflects a preference for efficiency and precision over merely maximizing recall metrics.

4.3 Tool Execution Performance(RQ2)

We assess execution performance in two scenarios: **(1)With Golden Truth** and **(2)With Retriever** (end-to-end setting). For the end-to-end comparison, pipeline baselines are paired with ToolRetriever to perform a single initial search. ToolOmni and ToolGen utilize their internal retrieval mechanisms. To ensure fair comparison during execution, we enforce a uniform interaction budget: the maximum number of tool execution turns is capped at 6 for all models.

Table 2 details the execution results. With

[1]OpenAI's sentence embedding model:`text-embedding-3-large`

Method	Solvable Pass Rate (SoPR)				Solvable Win Rate (SoWR)			
	I1	I2	I3	Avg.	I1	I2	I3	Avg.
With Ground-Truth Tools								
GPT-3.5	56.60	47.80	**54.64**	53.01	-	-	-	-
ToolLlama-v2	43.60	45.80	39.30	42.90	38.65	55.66	32.79	42.37
ToolLlama-v1	24.20	29.20	12.30	21.90	28.22	42.45	16.39	29.02
ToolGen (3B)	40.80	43.40	21.30	35.17	39.26	47.17	14.75	33.73
ToolGen (7B)	44.20	42.90	32.00	39.70	35.58	45.28	16.39	32.42
ToolOmni	**60.70**	**50.90**	52.50	**54.70**	**52.76**	**56.60**	**40.98**	**50.11**
With Retriever (End-to-End)								
GPT-3.5	51.43	41.19	34.43	42.35	43.56	46.23	29.51	39.77
ToolLlama-v2	43.10	46.70	40.20	43.33	42.94	50.00	14.75	35.90
ToolLlama-v1	28.05	29.20	13.90	23.72	26.38	33.96	11.48	23.94
ToolGen (3B)	37.40	40.60	30.30	36.10	36.20	29.25	9.84	25.10
ToolGen (7B)	39.60	40.60	23.00	34.40	30.06	29.25	16.39	25.23
ToolOmni	**59.20**	**58.10**	**45.10**	**54.13** ↑10.8	**50.31**	**57.55**	**42.62**	**50.16** ↑10.5

Table 2: End-to-end execution performance on ToolBench. We report both **Solvable Pass Rate (SoPR %)** and **Solvable Win Rate (SoWR %)**. ToolOmni demonstrates robust superiority, significantly outperforming GPT-3.5 and ToolLlama-v2 in end-to-end settings.

Method	Tool Gen.	Category Gen.	Average
GPT-3.5	50.60	42.10	46.35
ToolLlama-v2	39.20	39.40	39.30
ToolGen (7B)	33.20	40.10	36.65
ToolOmni	**52.20**	**55.95**	**54.08**

Table 3: Generalization performance (SoPR %) in the **End-to-End** setting. We report performance on Tool Gen. and Category Gen.. **ToolOmni** achieves the best generalization robustness.

Golden Tools, **ToolOmni** attains the highest average SoPR of **54.70%**, surpassing GPT-3.5 (53.01%) and ToolLlama-v2 (42.90%), validating its superior reasoning capabilities. In the End-to-End setting, this advantage widens significantly. ToolOmni achieves **54.13%** SoPR, outperforming the GPT-3.5 pipeline (42.35%) by **+11.78%** and doubling the gain over the unified model ToolGen (36.10%). These results confirm the efficacy of our decoupled optimization strategy.

Furthermore,**ToolOmni** demonstrates robust superiority regarding the response quality metric **Solvable Win Rate (SoWR)** In the end-to-end setting, it achieves an average SoWR of **50.16%**, significantly surpassing the strongest open-source baseline ToolLlama-v2 (35.90%) and the GPT-3.5 pipeline (39.77%). This indicates that ToolOmni not only correctly solves more queries but also generates more precise and coherent answers.

4.4 Robustness Analysis(RQ3)

Robustness Analysis on Generalization Splits. As presented in Table 3, **ToolOmni** demonstrates exceptional robustness against distribution shifts. In the **Tool Generalization** setting (unseen tools within known categories), it achieves a SoPR of **52.20%**, effectively adapting to new API signatures. More impressively, in the rigorous **Category Generalization** setting (entirely novel domains), **ToolOmni** attains a remarkable score of **55.95%**. This performance surpasses a strong pipeline baseline ChatGPT (42.10%) by a substantial margin of **+13.85%**. This result indicates that while generative baselines like ToolGen tend to overfit to the specific tool seen during training, ToolOmni successfully learns the *universal meta-skills* of tool usage—such as parameter inference and error recovery. Consequently, it can transfer its reasoning capabilities to completely new domains without requiring domain-specific fine-tuning.

Robustness against Retrieval Noise. To test resilience, we injected adversarial tools that were retrieved by a dense model but excluding the ground truth—into the context at levels $N \in \{0, 5, 10, 15\}$. As shown in Figure 4, **ToolLlama-v2** degrades monotonically (39.3% $\rightarrow$ 20.5%), revealing its susceptibility to semantic distraction. In contrast, **ToolOmni** exhibits an adaptive resilience pattern. After an initial decline, its accuracy significantly **rebounds** to **58.2%** at $N = 15$. This indicates that as the candidate pool expands, our agent adaptively identifies valid *alternative tools* within the adversarial set, showcasing a flexible reasoning capability that transcends rigid ground-truth matching.

Settings	Ret. NDCG	Exec. SoPR
Full	**78.3**	**52.5**
w/o SFT	77.8	43.4
w/o RL	49.3	23.8

Table 4: Ablation on training stages. **Full ToolOmni** integrated with SFT as well as RL achieves the optimal results.

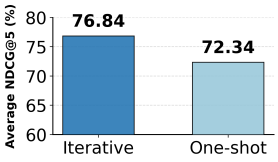

Figure 3: Ablation on retrieval strategy. **Iterative Retrieval** boosts NDCG@5 over the One-shot baseline.

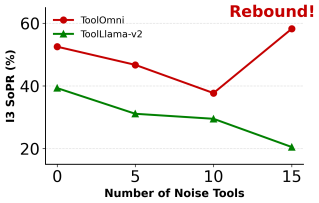

Figure 4: Robustness against adversarial noise on complex I3 tasks.

Top-k	Avg. NDCG
1	62.8
3	75.8
5	**78.3**
7	77.6
9	73.2

Table 5: Ablation study on retrieval strategy. "5" is the default setting.

4.5 Ablation Study(RQ4)

To validate the contribution of our core components—specifically the iterative retrieval mechanism and the multi-stage training pipeline—we compare ToolOmni against three variants: (1) **w/o Iterative Retrieval**, a static baseline restricted to single-round retrieval; (2) **w/o RL**, the model trained solely via SFT; and (3) **w/o SFT**, the model trained directly via RL from the base LLM, skipping the cold-start phase.

Impact of Training Stages. Table 4 disentangles the impact of training stages. For **Retrieval**, the removal of RL causes a drastic drop in NDCG (-28.99%), while removing SFT has a negligible impact (-0.51%), indicating that retrieval quality is primarily driven by RL-based exploration. Conversely, for **Execution**, the *w/o SFT* variant suffers a significant performance gap compared to the full model (43.40% vs. 52.50%). This confirms that SFT is essential as a cold-start mechanism, providing the necessary reasoning foundation that enables RL to optimize for complex tool-use scenarios effectively.

Impact of Iterative Retrieval. We further analyze the effectiveness of the retrieval strategy by comparing the Average NDCG@5 across all splits. As shown in Figure 3, the **Iterative Retrieval** mechanism employed by ToolOmni achieves an average score of **76.84%**, surpassing the static **One-shot Retrieval** baseline (72.34%) by **+4.5%**. This indicates that the ability to iteratively refine search queries allows the agent to dynamically adjust its search intent and filter out noise, effectively pinpointing the most utility-oriented tools required for execution.

4.6 Hyperparameter Sensitivity(RQ4)

Sensitivity to Retrieval Count (k). We investigate how the number of retrieved tools provided to the agent affects overall performance. Table 5 reports the Average NDCG across different retrieval counts $k \in \{1, 3, 5, 7, 9\}$.

The performance follows a clear **inverted U-shaped trajectory**. At low values ($k = 1$), the model suffers significantly (62.84%), as the constrained context often fails to include the necessary ground-truth tools (low recall). As k increases to 5, performance peaks at **78.29%**, indicating an optimal balance where the context is rich enough to cover required functionalities without being overwhelming. However, increasing k further to 9 leads to a noticeable decline (73.16%). This degradation suggests that an excessively long context introduces irrelevant "noise tools," which dilutes the agent's attention and complicates the reasoning process. Thus, our default setting of $k = 5$ proves to be the most robust configuration for open-world scenarios.

5 Conclusion

In this paper, we presented **ToolOmni**, a comprehensive agentic framework that enables open-world tool use via agentic learning with proactive retrieval and grounded execution. To address the inherent challenges of sparse rewards and sequential dependencies in this domain, we proposed a novel **Decoupled Multi-Objective GRPO** algorithm. Extensive experiments on the ToolBench benchmark demonstrate that **ToolOmni** achieves state-of-the-art performance in retrieval and end-to-end generation. Notably, our agent exhibits exceptional robustness against retrieval noise and strong generalization capabilities across unseen tools and domains. These results validate that equipping LLMs with the ability to iteratively reason about tool selection and execution is the key to scalable and robust tool learning. Future work will explore extending this decoupled framework to multi-modal tools and more complex, long-horizon planning tasks.

Limitations

While ToolOmni demonstrates robust performance, we acknowledge two primary limitations. First, our current experimental environment relies on an **LLM-simulated sandbox** (MirrorAPI) rather than live, real-world API interactions. Although this ensures reproducibility, it may not fully capture the latency, rate limits, and unpredictable error states inherent in genuine open-world deployments. Second, our execution evaluation depends on **LLM-as-a-Judge**. While scalable, such automated judges can exhibit biases—such as favoring specific response styles—and stochastic instability, potentially affecting the granularity of the metrics. Future work will focus on deploying ToolOmni in live environments and developing more rigorous, verifiable evaluation protocols.

References

Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. 2022. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654.

Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammad-Hossein Bateni, Chen-Yu Lee, and Tomas Pfister. 2024. Re-invoke: Tool invocation rewriting for zero-shot tool retrieval. In *EMNLP (Findings)*, pages 4705–4726. Association for Computational Linguistics.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. *arXiv preprint arXiv:2403.07714*.

Ziyang Huang, Xiaowei Yuan, Yiming Ju, Jun Zhao, and Kang Liu. 2025. Reinforced internal-external knowledge synergistic reasoning for efficient adaptive search agent. *arXiv preprint arXiv:2505.07596*.

Bowen Jin, Jinsung Yoon, Priyanka Kargupta, Sercan O Arik, and Jiawei Han. 2025a. An empirical study on reinforcement learning for reasoning-search interleaved llm agents. *arXiv preprint arXiv:2505.15117*.

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

Mohammad Kachuee, Sarthak Ahuja, Vaibhav Kumar, Puyang Xu, and Xiaohu Liu. 2025. Improving tool retrieval by leveraging large language models for query generation. In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*, pages 29–38.

Xiaoxi Li, Wenxiang Jiao, Jiarui Jin, Guanting Dong, Jiajie Jin, Yinuo Wang, Hao Wang, Yutao Zhu, Ji-Rong Wen, Yuan Lu, and 1 others. 2025a. Deepagent: A general reasoning agent with scalable toolsets. *arXiv preprint arXiv:2510.21618*.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025b. Torl: Scaling tool-integrated rl. *arXiv preprint arXiv:2503.23383*.

Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, and 1 others. 2024. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *Intelligent Computing*, 3:0063.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, and 1 others. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

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

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *ICLR*. OpenReview.net.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2024. Towards completeness-oriented tool retrieval for large language models. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 1930–1940.

Stephen Robertson, Hugo Zaragoza, and 1 others. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

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

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. 2025. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. *arXiv preprint arXiv:2503.01763*.

Yunyue Su, Jingshuai Zhang, Bowen Fang, Wen Ye, Jinghao Zhang, Bowen Song, Weiqiang Wang, Qiang Liu, and Liang Wang. 2025. Toolscaler: Scalable generative tool calling via structure-aware semantic tokenization. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 556–578.

Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. Zerosearch: Incentivize the search capability of llms without searching, 2025. *URL https://arxiv. org/abs/2505.04588*.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. 2025. Toolgen: Unified tool retrieval and calling via generation. In *ICLR*. OpenReview.net.

Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. 2023. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*.

Qiancheng Xu, Yongqi Li, Heming Xia, and Wenjie Li. 2024. Enhancing tool retrieval with iterative feedback from large language models. *arXiv preprint arXiv:2406.17465*.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. 2023. On the tool manipulation capability of open-source large language models. *arXiv preprint arXiv:2305.16504*.

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. 2025. Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. *arXiv preprint arXiv:2509.02479*.

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

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *ICLR*. OpenReview.net.

Yuanhang Zheng, Peng Li, Wei Liu, Yang Liu, Jian Luan, and Bin Wang. 2024. Toolrerank: Adaptive and hierarchy-aware reranking for tool retrieval. *arXiv preprint arXiv:2403.06551*.

A Data Curation

A.1 ToolBench

ToolBench (Qin et al., 2024) serves as a pioneering open-source benchmark for tool learning, constructed by scraping real-world APIs from RapidAPI. It encompasses a massive repository of **16,464 APIs** distributed across **49 distinct categories** (e.g., Social Media, E-commerce, Weather), reflecting a true open-world environment. Each API is documented with a JSON-based schema specifying its name, description, and calling parameters.

To generate high-quality instruction-tuning data, ToolBench utilizes a **Depth-First Search Decision Tree(DFSDT)** mechanism powered by ChatGPT. This process efficiently explores the vast action space to synthesize diverse and valid solution paths, resulting in over 126k tool-use instructions.

Crucially, these instructions cover a wide spectrum of complexity, ranging from single-tool tasks (I1) to intricate scenarios requiring the composition of multiple tools from the same category (I2) or across different collections (I3). This hierarchical diversity makes it an ideal testbed for evaluating an agent's generalization and reasoning capabilities.

A.2 Source and Difficulty Stratification

Our dataset is derived from the **ToolBench** corpus. To construct a balanced and challenging dataset, we first assess the difficulty of each query. We employ **ToolRetriever** to perform a preliminary dense retrieval, returning the top-5 tool candidates for each user query. An instance is classified based on recall success:

- **Easy Data**: All ground-truth "golden tools" are successfully retrieved within the top-5 candidates.

- **Hard Data**: At least one golden tool is missed by the retriever.

We then perform stratified sampling, selecting **60,000 hard instances** and **20,000 easy instances**, totaling **80,000 queries**. This set serves as the foundational pool for our subsequent processing and is also utilized as the prompt source for the Reinforcement Learning (RL) stage.

A.3 Retrieval Training Data

From the initial 80,000 queries, we train a specialized retrieval-only model to generate candidate search trajectories. We then apply **Rejection Sampling** on these generations, filtering out trajectories where the model fails to locate the correct tools. This rigorous filtering yields a high-quality corpus of approximately **28,000 retrieval trajectories**, which is used for the retrieval component of the SFT stage.

A.4 Execution Training Data

For the execution phase, we process the same pool of 80,000 instances to align with our agentic format. Since original ToolBench data lacks explicit reasoning steps, we employ **Qwen-2.5-72B-Instruct** to augment the data by generating detailed reasoning paths before each tool invocation. To ensure correctness, we use **Qwen-2.5-32B** as an automated judge to verify the execution results. To improve the model's discrimination ability during training, we construct a mixed dataset comprising:

- **Positive Samples (70%)**: Trajectories verified as correct.
- **Negative Samples (30%)**: Trajectories containing errors (e.g., hallucinated parameters or wrong tool selection).

This process results in a final execution dataset of approximately **33,000 trajectories**.

A.5 Final Dataset Composition

The **SFT Dataset** is the union of the 28,000 retrieval trajectories and 33,000 execution trajectories. The **RL Dataset** utilizes the initial pool of 80,000 queries as prompts to drive the online exploration and optimization process.

B Experiment Setup

B.1 Datasets and Metrics

Our experiments are conducted on ToolBench (Qin et al., 2024; Guo et al., 2024), a comprehensive real-world tool benchmark containing 16000+ real-word apis. Following the split of (Qin et al., 2024), the test queries are categorized into three complexity levels: **I1** (single-tool instructions), **I2** (intra-category multi-tool instructions), and **I3** (intra-collection multi-tool instructions). To rigorously evaluate robustness, these scenarios are further stratified into three generalization splits: **Instruction (Inst)** generalization (unseen queries with seen tools), **Tool** generalization (unseen tools within known categories), and **Category (Cate)** generalization (unseen tools from entirely new domains).

We evaluate tool retrieval performance with **NDCG@k** (with $k \in \{1, 3, 5\}$). For end-to-end evaluation, we report two key metrics: **Solvable Pass Rate (SoPR)**, which measures the percentage of queries successfully solved by the agent, and **Solvable Win Rate (SoWR)**, which indicates the proportion of answers that outperform those generated by a reference baseline. Both metrics are computed using GPT-5 Mini as an automated judge to ensure consistent and scalable evaluation.

B.1.1 Retrieval Test Dataset

The retrieval evaluation is conducted on the official ToolBench test set. Table 6 details the distribution of test queries across the three instruction complexity levels.

Table 6: Statistics of the **ToolBench Retrieval Test Set**. The dataset is categorized by instruction complexity (I1: Single-tool, I2: Intra-category, I3: Intra-collection).

Complexity Level	Number of Queries
I1 (Single-tool)	796
I2 (Intra-category)	573
I3 (Intra-collection)	218
Total	**1587**

B.1.2 Execution Test Dataset

For the end-to-end execution evaluation, we utilize the curated **Solvable Test Queries** from StableToolBench to ensure robust assessment. The test set comprises six distinct generalization subsets, covering different levels of difficulty (G1, G2, G3) and generalization types (Instruction, Category, Tool). Table 7 presents the detailed statistics.

B.2 Comparison Baselines.

To validate the effectiveness of ToolOmni, we benchmark it against a comprehensive set of competitive methods. For tool retrieval, we com-

Table 7: Statistics of the **StableToolBench Solvable Test Set**. Queries are stratified by generalization difficulty (G1, G2, G3) and type.

Difficulty	Subset Name	Count
G1	Instruction Gen. (I1)	163
	Tool Gen. (I1)	158
	Category Gen. (I1)	153
G2	Instruction Gen. (I2)	106
	Category Gen. (I2)	124
G3	Instruction Gen. (I3)	61
Total		**765**

pare with traditional sparse retrievers like BM25 (Robertson et al., 2009) and dense retrievers such as ToolRetriever (Qin et al., 2024). Additionally, we evaluate advanced query refinement strategies, including Re-Invoke (Chen et al., 2024) and Iter-Feedback (Xu et al., 2024), as well as ToolGen (Wang et al., 2025), which adopts a generative paradigm for retrieval. For tool execution, our baselines encompass both proprietary and open-source models: we employ ChatGPT as strong zero-shot references, and compare against ToolLLaMA (Qin et al., 2024), the state-of-the-art SFT baseline on ToolBench. We also include ToolGen (Wang et al., 2025) again to assess the performance of unified generative frameworks in end-to-end scenarios.

B.3 Implementation Details.

We initialize ToolOmni upon the Qwen3-4B-Instruct (Yang et al., 2025). Regarding the reward configuration, we set the format weight to 0.2 and the performance weight to 0.8 for both phases (i.e., $\alpha_1 = 0.2, \alpha_2 = 0.8$ for retrieval; $\beta_1 = 0.2, \beta_2 = 0.8$ for execution) For the reinforcement learning stage, we configure the GRPO group size to $G = 5$ and set the sampling temperature to 1.0 to encourage exploration during training. The retrieval module employs ToolRetriever as the dense embedding model, while the execution environment is simulated using MirrorAPI (Guo et al., 2024) to provide stable and low-latency feedback. All models are trained for a single epoch on 8 NVIDIA H100 GPUs.

C Case Study

To provide a qualitative understanding of ToolOmni's superiority, we present comprehensive case studies across both Open-Domain and Oracle settings. In the **Open-Domain scenario** (Fig.5), we demonstrate how ToolOmni's proactive iterative retrieval effectively filters out noise and locates critical tools that pipeline baselines often miss, thereby preventing downstream hallucinations. In the **With Golden Truth setting** (Fig.6), we highlight the agent's robust reasoning capabilities. Specifically, Case 2 illustrates how ToolOmni autonomously diagnoses API errors (e.g., missing tokens) and strategically pivots to alternative tools. Together, these cases validate that ToolOmni is not merely a tool invoker, but a resilient problem solver capable of navigating the complexities of open-world environments.

D Learning Algorithm of ToolOmni

Algorithm 1 outlines the complete training procedure. A critical feature of our approach is the **Filtered Rollout mechanism** (Line 6-9), which acts as a quality gate. By initiating execution rollouts only when the retrieval phase successfully recalls the golden tools, we ensure that the execution policy is trained exclusively on grounded, solvable contexts. This prevents the model from learning "hallucination shortcuts" to compensate for missing information. Furthermore, the **Seperated Optimization** (Phase 3) allows the retrieval and execution modules to evolve at their own pace, guided by their respective specialized rewards, thereby stabilizing the overall learning dynamics in the complex open-world environment.

E Prompts

To facilitate reproducibility, we provide the full system prompts utilized in our experiments. As illustrated in Figure 7, we design two distinct prompts tailored for the decoupled phases:

- The **Retrieval Prompt** (Left) instructs the agent to act as a "search copilot," employing an iterative loop of query generation and information synthesis to identify the optimal set of tools from the massive repository.
- The **Execution Prompt** (Right) guides the agent to function as an "AutoGPT," leveraging the retrieved tools within a grounded reasoning framework to solve the user's query step-by-step.

These prompts are used consistently across both the SFT data generation and the RL training stages.

Algorithm 1 Decoupled Multi-Objective GRPO Algorithm

Require: Policy π_θ; Dataset $\mathcal{D}$; Group size G; Learning rate η.
Ensure: Optimized policy π_{θ^*}.

for each training iteration **do**
  Sample a batch of queries $\{x_i\}_{i=1}^{B} \sim \mathcal{D}$.
  Phase 1: Group Rollout
  for each query x_i **do**
    for $j = 1$ to G **do**
      Generate retrieval trajectory $q_{i,j} \sim \pi_{\theta_{old}}(\cdot|x_i)$.
      Retrieve tools $\mathcal{T}_{i,j}$ using query $q_{i,j}$.
      if $\mathcal{T}_{gold} \subseteq \mathcal{T}_{i,j}$ **then** ▷ *Trajectory Filtering*
        Generate execution trajectory $e_{i,j} \sim \pi_{\theta_{old}}(\cdot|x_i, \mathcal{T}_{i,j})$.
      end if
    end for
  end for
  Phase 2: Decoupled Reward Calculation and Advantage Estimation
  for each query x_i and trajectory $q_{i,j}, e_{i,j}$ **do**
    Compute retrieval reward: $R_{ret}^{i,j} = \alpha_1 r_{fmt}^{ret} + \alpha_2 r_{recall} \cdot r_{conv}$.
    Estimate advantages for retrieval: $A_{ret}^{i,j} = \frac{R_{ret}^{i,j} - \mu(R_{ret}^{i})}{\sigma(R_{ret}^{i}) + \epsilon}$.
    Compute execution reward: $R_{exec}^{i,j} = \beta_1 r_{fmt}^{exec} + \beta_2 r_{ans}$.
    Estimate advantages for execution: $A_{exec}^{i,j} = \frac{R_{exec}^{i,j} - \mu(R_{exec}^{i})}{\sigma(R_{exec}^{i}) + \epsilon}$.
  end for
  Phase 3: Seperated Optimization
  Retrieval Update π_θ parameters: $\theta \leftarrow \theta + \eta \nabla_\theta(\mathcal{J}_{ret})$.
  Execution Update π_θ parameters: $\theta \leftarrow \theta + \eta \nabla_\theta(\mathcal{J}_{exec})$.
end for
return π_θ

Case 1: Open-Domain Scenario

Instruction: I want to explore different genres of movies and series available on streaming platforms. Can you provide me with the basic details and streaming availability of a movie or series with the keyword 'zombie' and released between the years 2000 and 2025? Also, fetch the details of the actors and crew associated with the title with Watchmode ID 3173903.

✖ Baseline (ToolLlama + ToolRetriever):

[ToolRetriever Retrieval Result (Top-5)]

- [ID:781] OTT - Search (Noise)
- **[ID:782] OTT - Title Details**
- **[ID:4580] Watchmode - Cast & Crew**
- [ID:778] OTT - Advanced Search (Wrong Tool for Streaming)
- [ID:13631] Watchmode - Person (Noise)

Missing Golden Tool: **[ID:7585] Streaming Availability - Get Basic**

[Execution Phase]

`Action 1: OTT-Search(title="zombie", page=1)`
Error: Used basic search; returned random titles like "Zombie Outbreak" without checking year/genre.

`Action 2: TitleDetails(imdbid="tt13411672")`
Result: Streaming information.

`Action 3: TitleDetails(id="3173903")`
Error: Wrong tool! Used 'TitleDetails' instead of 'CastCrew' for Watchmode ID. Returned unrelated anime "Jujutsu Kaisen".

Outcome: Hallucination. Falsely claims "Available on Netflix" and provides wrong cast info.

ToolOmni (Ours):

[Phase 1: Proactive Retrieval]

<search> fetch basic details and streaming availability of a movie or series </search>
<information> **[ID:782] OTT - Title Details** *Get basic information ...* </information>
<search> confirm the keyword 'zombie' and released between the years 2000 and 2025, specifically for streaming platforms. </search>
<information> **[ID:7585] Streaming Availability - Get Basic** *Get details of a show ...* </information>
<search> associated with Watchmode ID 3173903, specifically for streaming platforms and crew details. </search>
<information> **[ID:4580] Watchmode - Title Cast & Crew** *Return all people associated with the title ...* </information>
<final_tools> **ID: [782,7585,4580]** </final_tools>

[Phase 2: Grounded Execution]

<reasoning> To fulfill this request, I need to:1. Search for movies or series with the keyword 'zombie' and released between 2000 and 2025 using the OTT details search tool with a filter for release year.2. Fetch the basic details and streaming availability of such a movie or series using the Streaming Availability tool with a country parameter (e.g., US).3. Retrieve the cast and crew details for the title with Watchmode ID 3173903 using the Watchmode tool. </reasoning>
<tool_call> {"tool": "OTT details", "api": "Search", "input": {"title": "zombie", "start_year": 2000, "end_year": 2025}} </tool_call>
<information> *Found "Zombie (2000)", "Zombie Nightmare"...* </information>
<reasoning> Check streaming availability. </reasoning>
<tool_call> {"tool": "Streaming Availability", "api": "Get Basic", "input": {"country":"us", "imdb_id": "tt0238784"}} </tool_call>
<information> *Streaming Not Available* </information>
<reasoning> Get crew details </reasoning>
<tool_call> {"tool": "Watchmode", "api": "Title Cast & Crew", "input": {"id": "3173903"}} </tool_call>
<information> *Crew details ...* </information>
<reasoning> The search for movies or series with the keyword 'zombie' and released between 2000 and 2025 returned valid results, including both movies and TV movies. Additionally, the cast and crew details for the title with Watchmode ID 3173903 have been successfully retrieved. Although streaming availability was not found in the US, the user's request is satisfied with the exploration of genres and the cast/crew details for the specified title. </reasoning>
<answer> Provided movie list (2000-2025), streaming status and correct cast list. </answer>

Outcome: Solved

Figure 5: Qualitative comparison (Case Study). **Case 1** illustrates how pipeline baselines fail due to retrieval noise (selecting basic Search instead of Advanced Search) and missing tools (Streaming), leading to hallucinations. In contrast, **ToolOmni**'s proactive iterative retrieval precisely locates the golden toolset, enabling correct execution and grounded response generation. *Note: The iterative process shown is condensed for brevity.*

Case 3: Complex Reasoning With Golden Tools

Instruction: I'm a movie enthusiast and I'm interested in watching critically acclaimed films. Can you suggest some award-winning movies with high IMDb ratings? Also, provide the streaming platforms where these movies are available in the US.

✗ Baseline (ToolLlama):

[Step 1]
`Action: Streaming.Search(country="US", services="netflix,prime...")`
Error: `Invalid country code.`

[Step 2]
`Action: Streaming.Search(country="US", ...)` *(Retrying same args)*
Error: `Invalid country code.`

[Step 3-5]
(Repeatedly calls the same tool with minor variations, ignoring the error message.)

Outcome: Give Up. The model entered a repetitive loop and failed to adjust its strategy, eventually terminating with `give_up_and_restart`.

ToolOmni (Ours):

<reasoning> The user wants to watch critically acclaimed films with high IMDb ratings and wants to know the streaming platforms where these movies are available in the US. To fulfill this request, I should start by searching for movies that are both critically acclaimed and have high IMDb ratings. </reasoning>
<tool_call> {"tool": "Streaming Availability", "api": "Search Ultra", "input": {"country": "us", "min_imdb_rating": 80}} </tool_call>
<information> Empty </information>
<reasoning> Search returned empty... parameters might be too restrictive. I should try fetching "New Titles" via Watchmode or check a specific known movie. </reasoning>

<tool_call> {"tool": "Watchmode", "api": "New Titles", "input": {"types": "movie"}} </tool_call>
<information> Empty </information>
<reasoning> Still no results. Instead of relying on general search, I should directly verify a known critically acclaimed film like "The Shawshank Redemption". </reasoning>
<tool_call> {"tool": "OTT details", "api": "Title Details", "input": {"imdbid": "tt0111161"}} </tool_call>
<information> *Found "The Shawshank Redemption" (IMDb 9.3), Streaming: [Prime, Netflix]* </information>
<reasoning> The OTT details tool successfully returned information for "The Shawshank Redemption" </reasoning>
<answer> The Shawshank Redemption is a critically acclaimed film... It is available on Amazon Prime and Netflix. </answer>
Outcome: Solved.

Figure 6: Qualitative comparison (Case Study). **Case 2** highlights the robustness of **ToolOmni** against execution failures. While **ToolLlama** gets trapped in a repetitive error loop due to rigid parameter usage, **ToolOmni** demonstrates adaptive planning: after tool failures, it dynamically pivots its strategy—switching from general search to specific verification—to successfully fulfill the user request. *Note: The process is condensed for clarity.*

Retrieval Prompt

You are a tool search copilot for a generation model. Given the user question, your task is to understand the user's intents and search all relevant apis that could help solve the question.

To do this, you can call a tool search engine to issue a query using <search> query </search>.

Each query will return a list of apis between <information> and </information>, where each dictionary represents a api with its original metadata(e.g., api_id, category, tool_name, api_name, api_description).

For multi-hop user question, you can break it down into pieces and search one by one.

If the searched apis are not enough, you will go through a loop of <search> → <information> → <search> → <information> → <search> (if not complete) ..., to make sure you have retrieved all relevant apis.

When you have gathered enough apis, select the useful apis and output only their api_ids in a list inside <final_tools> and </final_tools>, sorted by descending relevance (most relevant first).

For example: <final_tools>[2,0,1]</final_tools>

Note:
1. Do not provide any explanations outside the tags.
2. The content inside <final_tools> and </final_tools> must be a list of useful api ids selected directly from earlier <information> blocks.
3. Remove duplicates if an api appears multiple times. Do not invent new apis.

Question: {question}

Execution Prompt

You are an AutoGPT for tool calling, capable of utilizing tools and functions to complete the given question.

Given the user question, your task is to understand the user's intents and call the most appropriate tool(s) in a logical sequence to address the user's needs.

You MUST conduct reasoning inside <reasoning> and </reasoning> first every time you get new information.

After reasoning, you can call a tool using <tool_call> content </tool_call>, where the content is a dictionary with the tool's category, tool_name, api_name, and required input arguments.

The result will be returned between <information> and </information>.

If additional tools are needed, you will go through a loop of <reasoning> → <tool_call> → <information> → <reasoning> → <tool_call> → <information> → <reasoning> (if not complete) ..., to make sure you have completed the task.

When you have gathered enough information and no further tool calls are needed, output the final answer directly between <answer> and </answer>.

Notes:
1. Do not provide any explanations outside the tags.
2. The content inside <tool_call> and </tool_call> must include the selected tool's category, tool_name, api_name, and the required input arguments.
3. You can only use the tools in available tools: {available_tools}

Question: {question}

Figure 7: System prompts used for **Retrieval** (Left) and **Execution** (Right) phases. The retrieval prompt guides the agent to proactively search and select tools, while the execution prompt instructs it to perform grounded reasoning and tool invocation.