ToolHallu: MITIGATING HALLUCINATION IN AGENTIC TOOL CALLS VIA REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM)-based agents have been applied across various tasks. A key factor underpinning their powerful capabilities is the use of external tools. While LLMs provide the basic understanding required for tool invocation, their susceptibility to hallucination continues to threaten the reliability of LLM-based agents, making hallucination mitigation imperative for reliable agent deployment. In this work, we propose a novel reinforcement learning (RL) framework named *ToolHallu* to tackle tool-call hallucination. We first systematically analyze hallucination patterns specific to agentic tool invocation. Based on this, we decompose tool-call hallucinations into distinct error types and assign differentiated rewards to provide interpretable training signals. To further address the varying difficulty of these errors, we introduce a curriculum learning schedule to adjust the optimization process, which first guides the model to learn accurate tool selection and then progressively shifts the optimization focus towards more fine-grained parameter-level hallucinations. Extensive experiments across multiple models and benchmark datasets demonstrate that our method consistently reduces hallucination rates and improves invocation reliability.

1 INTRODUCTION

Large language models (LLMs) have demonstrated remarkable capabilities in understanding user queries and generating coherent solutions. This has catalyzed their rapid deployment as autonomous agents capable of interacting with external environments through tool invocations, a paradigm that underpins a wide range of real-world applications, such as web navigation (Zhou et al., 2024) and scientific discovery (M. Bran et al., 2024). Despite these advances, hallucination remains a persistent and critical limitation in agentic tool calls: agents either invoke inappropriate tools, fabricate arguments, or produce plausible but non-executable calls (Xu et al., 2024). Such failures not only undermine task success but also erode user trust, particularly in high-stakes deployment scenarios.

Substantial efforts have been devoted to post-training LLMs for tool use (Qin et al., 2024a), with reinforcement learning (RL) emerging as a primary paradigm (Qian et al., 2026; Lin et al., 2026). However, existing agentic RL methods primarily aim to improve tool-use agents from the perspective of response format and task correctness, with limited attention to the critical issue of tool-use hallucination. To guide agents toward identifying and correcting such hallucinated behaviors, it is essential to explicitly characterize the severity of hallucination-related tool-calling errors. In particular, calling an unavailable tool should incur a substantially higher penalty than invoking an incorrect but valid tool. Likewise, among erroneous tool calls, a hallucinated parameter type should be penalized more heavily than a type-correct but semantically imperfect parameter value, as the former constitutes a more fundamental violation of the tool interface.

To address this gap, we propose a fine-grained, structure-aware reward supervision framework *ToolHallu*, which comprises two key components: a fine-grained reward design and a curriculum learning strategy. Beyond the widely accepted format and accuracy rewards, we incorporate a hallucination-aware reward targeting tool-call reliability, including tool name selection and parameter validity. Moreover, we observe that agents may repeatedly invoke the same tool with identical parameters within a single response, suggesting a failure to recognize and appropriately account for their previous actions. To mitigate this behavior, we also introduce a dedicated reward penalty for repetitive tool invocations. By explicitly disentangling tool-call hallucinations into distinct cat-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

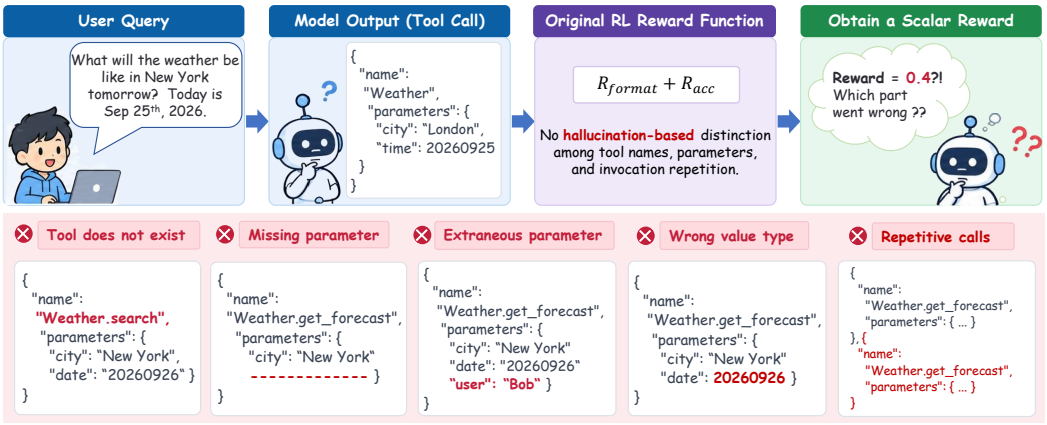

Figure 1: Existing reward designs fail to capture different types of tool-call hallucinations. We illustrate several hallucination types that cannot be explicitly identified by existing reward functions.

egories and assigning differentiated rewards or penalties to each hallucination type, *ToolHallu* delivers richer learning signals. Furthermore, we argue that effective hallucination mitigation should follow a natural progression from simple to complex. To realize this curriculum, we design a cosine-annealed reward weighting scheme that dynamically adjusts penalty strength for different hallucination types over training steps. We illustrate several types of tool-call hallucinations in Figure 1 to facilitate a better understanding of these errors.

Our contributions are summarized as follows:

1) We explicitly categorize agentic tool-call hallucinations into multiple types, and assign a structured, fine-grained reward supervision scheme. This provides dense, interpretable feedback that directly addresses the root causes of invocation hallucinations.

2) We incorporate a curriculum learning strategy into the reward framework, which enhances invocation fidelity from coarse-grained tool selection to fine-grained argument correctness, improving the model's tool calling capabilities without requiring additional model components.

3) Experiments across multiple models and benchmarks demonstrate that our proposed method consistently reduces hallucination rates while also delivering consistent gains in task success rates. Ablation studies validate that both our fine-grained reward decomposition and curriculum scheduling contribute to the final performance gains, confirming the effectiveness of each design.

2 Related Work

2.1 Tool-Use Agents

Tool use enables large language models (LLMs) to invoke external tools and APIs, thereby extending their capabilities in information retrieval, program execution, and domain-specific knowledge acquisition (Xu et al., 2025; Hao et al., 2025). Early studies primarily leveraged prompt engineering (Schick et al., 2023), in-context learning (Yao et al., 2022; Hsieh et al., 2023), and supervised fine-tuning (Chen et al., 2024) to teach models to understand tool descriptions, select tools, and generate arguments. Subsequently, researchers explored tool-integrated reasoning, enabling LLMs to perform multi-step planning and autonomously invoke multiple tools to accomplish complex tasks such as information retrieval, mathematical computation, and code generation (Qin et al., 2024b). With the development of reinforcement learning, tool-use agents have also been optimized with policy-gradient methods, among which GRPO (Shao et al., 2024) has become a representative choice, further improving tool invocation capabilities through task-level rewards, multi-trajectory sampling, and hierarchical reward schemes. Overall, existing research has primarily focused on capabilities including tool selection, argument generation, multi-step invocation, and task completion.

2.2 TOOL HALLUCINATION

LLM hallucination generally refers to the generation of content that is inconsistent with the input, context, or real-world knowledge (Huang et al., 2025). Existing studies have explored hallucination mitigation from various perspectives, including improving data quality (Abbas et al., 2023), supervised fine-tuning (Sharma et al., 2024), external knowledge augmentation (Izacard et al., 2023), and more effective decoding strategies (Shi et al., 2024). These approaches have achieved promising progress in open-ended generation tasks, yet they are primarily designed for free-form text. As LLMs are increasingly deployed as tool-use agents, hallucinations can also manifest as erroneous actions, such as invoking nonexistent tools, generating invalid parameters, or producing tool calls that are inconsistent with the task context, which may propagate through subsequent interactions and compromise the reliability of agent execution (Zhang et al., 2024; Peng et al., 2026). Researchers have begun to investigate erroneous tool-use behaviors and their impact on task execution reliability (Patil et al., 2024; Chen et al., 2023), categorizing tool hallucinations into tool selection hallucination and tool usage hallucination (Xu et al., 2024). However, most existing work tries to reduce tool-use errors in a general, outcome-oriented manner, primarily assigning rewards from the perspective of correctness, lacking fine-grained penalties for different hallucination behaviors rather than explicitly through the lens of hallucination(Qian et al., 2026; Lin et al., 2026).

3 METHOD

3.1 PROBLEM DEFINITION

We consider the task of training an LLM to reliably invoke external tools (e.g., APIs) in response to a user query. A typical multi-turn interaction involves multiple tool calls across several reasoning steps, where the final outcome depends on the cumulative correctness of these intermediate decisions. Formally, let $T = \{t_1, t_2, \ldots, t_n\}$ denote the set of n available tools, and let Q denote a user query. At each step k, the model generates a response that may include multiple tool calls based on the conversation history, each with a tool name and its associated parameters.

The model's policy is defined as π : history $\rightarrow$ (tool_name, tool_parameters), mapping the current context to a tool invocation decision. We assume the tool execution environment is stable, such that a correctly invoked tool with valid name and parameters will always return the expected result. Previously, the model is optimized to maximize cumulative rewards over the trajectory, where rewards are assigned based on the format and correctness of each tool call (Qian et al., 2026; Lin et al., 2026). The format reward focuses on whether the model's output adheres to the prescribed structural conventions, which is defined as:

$$R_{\text{format}} = 1 \quad \textit{if} \text{ all required fields appear and are in the correct order} \quad \textit{else } 0. \tag{1}$$

The correctness reward is defined as:

$$R_{\text{acc}} \propto \frac{\dfrac{|N_G \cap N_P|}{|N_G \cup N_P|} + \displaystyle\sum_{G_j \in G} \frac{|\text{keys}(G_j) \cap \text{keys}(P_i)|}{|\text{keys}(G_j) \cup \text{keys}(P_i)|} + \sum_{G_j \in G} \sum_{k \in \text{keys}(G_j)} \mathbb{1}[P_i[k] = G_j[k]]}{1 + |G| + \displaystyle\sum_{G_j \in G} |\text{keys}(G_j)|}. \tag{2}$$

Here, P and G denote the sets of predicted and ground-truth tool calls, respectively. N_P and N_G represent the sets of predicted and ground-truth tool names, respectively. P_i and G_j denote the i-th predicted and j-th ground-truth tool calls, respectively, while $\text{keys}(P_i)$ and $\text{keys}(G_j)$ denote their corresponding parameter-name sets. The indicator function $\mathbb{1}[\cdot]$ equals 1 when the predicted parameter value $P_i[k]$ exactly matches the ground-truth value $G_j[k]$, and 0 otherwise. The numerator measures the overall matching degree in terms of tool names, parameter names, and parameter values. The denominator $1 + |G| + \sum_{G_j \in G} |\text{keys}(G_j)|$ represents the maximum possible matching score. Finally, R_{acc} is proportional to the normalized matching score, where a higher value indicates a higher degree of agreement between the predicted and ground-truth tool calls.

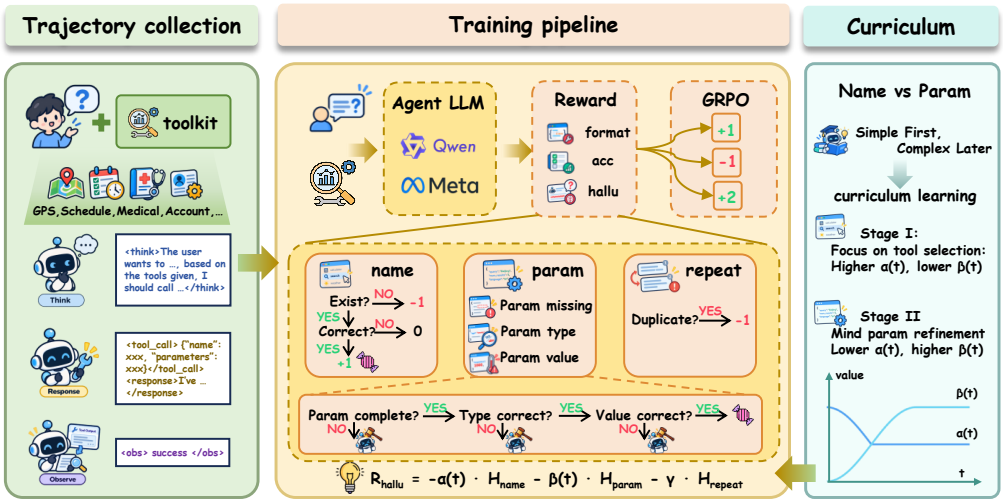

Figure 2: Overview of *ToolHallu* training pipeline. Trajectories from agent-environment interactions are used to train the Agent LLM with a hallucination-aware reward. The reward penalizes based on tool name, tool parameters, and invocation repetition, with weights scheduled via curriculum learning to progressively suppress hallucination.

3.2 Hallucination-Aware Reward Design

Rule-based reward mechanisms have demonstrated strong empirical effectiveness across diverse tasks (Guo et al., 2025; Mu et al., 2024), particularly when desired behaviors can be evaluated through objective and verifiable criteria. This suggests that programmatic, rule-based rewards can provide reliable supervision without requiring an additional learned reward model, motivating their use for evaluating and optimizing tool-use behaviors. However, existing reward designs typically collapse all errors into a single scalar, failing to distinguish whether a failure stems from the hallucination perspective (Qian et al., 2026; Lin et al., 2026). This coarse supervision provides weak incentives for the model to acquire the specific sub-skills required for reliable tool invocation and is particularly detrimental to mitigating tool-call hallucinations.

In this work, we address this limitation by formulating a hallucination-aware RL strategy that explicitly decomposes tool-call errors into distinct hallucination categories, enabling dense and interpretable feedback at each invocation step. The reward function explicitly decomposes tool-call hallucinations into three distinct error categories and assigns differentiated penalties accordingly. The overall architecture of the *ToolHallu* pipeline is illustrated in Figure 2.

Tool name hallucination. The first step is to judge whether the predicted tool name exists in the available tool set. If the tool is unavailable, we assign a name hallucination $h_{\text{name}} = 1$ and terminate the evaluation, as any subsequent parameters are meaningless. If the tool exists, we check its relevance by comparing it against the ground-truth tool list. When the predicted tool name matches one of the ground-truth tools, we assign a positive signal $h_{\text{name}} = -1$; otherwise, $h_{\text{name}} = 0$ and the process is terminated. This design incentivizes the model to prioritize selecting the correct tool before attending to parameter details.

Tool parameter hallucination. When the tool name is relevant, we further evaluate parameter-level hallucination across three dimensions:

- *Missing or extraneous arguments.* For each parameter required by the ground-truth tool, we check whether it appears in the predicted parameters. Missing required parameters are counted as hallucinations, whereas provided required parameters are counted as matches and should be encouraged. Additionally, using extraneous parameters, which are not defined in the tool schema, is also penalized to discourage the model from fabricating unsupported arguments. The combined count is normalized by the total number of predicted parameters: $e_{\text{force}} = (\#\text{missing} - \#\text{provided} + \#\text{extraneous})/\#\text{pd_params}$.

- *Argument type mismatch.* For each parameter provided by the model, we check whether its value conforms to the expected type defined in the available tool schema (e.g., str, int, float). The number of type violations is normalized by the total number of predicted parameters: $e_{\text{type}} = \#\text{type_mismatches}/\#\text{pd_params}$.
- *Argument value truthfulness.* For each correctly provided parameter, we check whether its values are truthful by matching them with the ground-truth value. Correct matches yield a negative penalty (i.e., a reward bonus), normalized by the number of ground-truth parameters for that tool: $e_{\text{value}} = -\#\text{correct_values}/\#\text{gt_params}$.

Redundant Tool Invocation. Another observed issue is the redundant invocations of the same tool with identical parameters as a form of tool-use hallucination, as such repetitions indicate that the agent fails to recognize previously executed actions. Note that invoking the same tool with identical parameters at different points in a multi-turn interaction can be legitimate, e.g., when new observations warrant re-querying the same state; we therefore only flag repetitions that occur within the same turn. Specifically, we track the $(\text{tool_name}, \text{tool_parameters})$ pair of every tool call, and a call is flagged as a repetition hallucination, i.e., $h_{\text{repeat}} = 1$, if an identical pair has already appeared earlier in the same response; otherwise, $h_{\text{repeat}} = 0$.

The hallucination reward is defined by aggregating the three hallucination components:

$$R_0 = -H_{\text{name}} - H_{\text{param}} - H_{\text{repeat}}. \tag{3}$$

Here, H_{name}, H_{param}, and H_{repeat} represent tool-name hallucinations, parameter-level hallucinations, and redundant tool invocations, respectively.

For each sample, the three hallucination components are averaged over all predicted tool calls:

$$H_{\text{name}} = \frac{\sum h_{\text{name}}}{\text{count}}, \quad H_{\text{param}} = \frac{\sum (e_{\text{force}} + e_{\text{type}} + e_{\text{value}})}{\text{count}}, \quad H_{\text{repeat}} = \frac{\sum h_{\text{repeat}}}{\text{count}}. \tag{4}$$

Here, count denotes the total number of tool calls predicted for the sample. e_{force}, e_{type}, and e_{value} represent errors arising from missing or extraneous arguments, incorrect parameter types, and untruthful parameter values, respectively.

3.3 Curriculum-Driven Reward Scheduling

Although the fine-grained reward in Section 3.2 explicitly distinguishes different types of tool-call hallucinations, treating these components equally throughout training overlooks their inherent differences in learning difficulty. Tool selection constitutes a relatively constrained classification problem over a finite set of available tools with only three possible scores (-1, 0, 1), where the error space is discrete and bounded. In contrast, parameter-level hallucinations, including missing arguments, extraneous parameters, type mismatches, and incorrect values, exhibit substantially more diverse and combinatorial error patterns. Consequently, tool-selection hallucinations are generally easier for the model to identify and correct, whereas parameter-level errors require finer-grained discrimination and are more effectively optimized after reliable tool-selection behavior has been established.

Motivated by this disparity, we introduce a curriculum-driven reward scheduling strategy that progressively shifts the optimization focus from tool selection to parameter precision. Specifically, the model is encouraged to first acquire reliable tool discrimination, then gradually devote more learning budgets to the more challenging parameter-level hallucinations. To incorporate this curriculum into the hallucination-aware reward, we extend the base reward defined in Eq. (3) by introducing two time-dependent coefficients, $\alpha(t)$ and $\beta(t)$, which dynamically control the contributions of tool-name and parameter-level hallucinations, respectively. We keep the contribution of H_{repeat} fixed throughout training, since redundant tool invocations should be consistently discouraged.

Let t denote the current training step, and let T_{decay} and T_{warmup} denote the decay horizon for the tool-name component and the warmup horizon for the parameter component, respectively. For the tool-name coefficient $\alpha(t)$, we adopt a cosine decay schedule:

$$\alpha(t) = \begin{cases} \max\left(0.5 \cdot \left(\cos\left(\frac{\pi \cdot t}{T_{\text{decay}}}\right) + 1\right), 0.5\right), & \text{if } t \le T_{\text{decay}} \\ 0.5, & \text{if } t > T_{\text{decay}} \end{cases} \tag{5}$$

Table 1: API-Bank Test Results. The evaluation dataset consists of 399 samples for Level 1, 67 for Level 2, and 131 for Level 3. Best in **bold**.

Series	Size	Method	Overall		Level 1		Level 2		Level 3	
			Acc	Hallu	Acc	Hallu	Acc	Hallu	Acc	Hallu
Qwen3	0.6B	/	0.57	0.33	0.61	0.31	0.41	0.50	0.55	0.28
		GRPO	0.61	0.30	0.62	0.32	0.61	0.37	0.57	0.18
		ResT	0.66	0.45	0.66	0.39	0.61	0.35	0.69	0.61
		Ours	**0.66**	**0.26**	0.67	0.27	0.57	0.34	0.66	0.17
	1.7B	/	0.74	0.33	0.75	0.32	0.70	0.49	0.73	0.27
		GRPO	0.77	0.21	0.77	0.22	0.73	0.34	0.77	0.13
		ResT	0.81	0.21	0.83	0.22	0.80	0.35	0.73	0.10
		Ours	**0.81**	**0.16**	0.84	0.16	0.84	0.24	0.71	0.12
	4B	/	0.82	0.26	0.84	0.28	0.68	0.34	0.85	0.13
		GRPO	0.82	0.18	0.84	0.18	0.82	0.26	0.74	0.14
		ResT	0.83	0.15	0.85	0.16	0.80	0.21	0.78	0.08
		Ours	**0.84**	**0.14**	0.86	0.15	0.79	0.23	0.80	0.09
	8B	/	0.82	0.18	0.86	0.19	0.82	0.32	0.70	0.08
		GRPO	0.82	0.19	0.85	0.17	0.64	0.37	0.84	0.14
		ResT	**0.85**	0.21	0.89	0.22	0.73	0.37	0.77	0.09
		Ours	0.83	**0.14**	0.88	0.15	0.71	0.28	0.73	0.07
Llama3.2 (Instruct)	1B	/	0.19	0.93	0.21	0.93	0.30	0.74	0.07	0.95
		GRPO	0.32	0.53	0.33	0.54	0.27	0.68	0.31	0.40
		Ours	**0.48**	**0.39**	0.42	0.45	0.39	0.56	0.66	0.19
	3B	/	0.34	0.62	0.38	0.61	0.34	0.70	0.25	0.61
		GRPO	0.60	0.39	0.71	0.32	0.64	0.44	0.25	0.61
		Ours	**0.77**	**0.29**	0.79	0.26	0.70	0.45	0.76	0.28

The coefficient starts from 1.0 and gradually decreases to a lower bound of 0.5. This lower bound preserves a persistent learning signal for tool selection, preventing the model from disregarding tool-name correctness after the optimization focus shifts toward parameter-level precision.

In contrast, for the parameter-level coefficient $\beta(t)$, we employ a cosine warmup schedule:

$$\beta(t) = \begin{cases} 0.5 \cdot \left(1 - \cos\left(\frac{\pi \cdot t}{T_{\text{warmup}}}\right)\right), & \text{if } t \leq T_{\text{warmup}} \\ 1.0, & \text{if } t > T_{\text{warmup}} \end{cases} \tag{6}$$

The coefficient starts from 0 and progressively increases to 1.0, allowing the model to initially tolerate parameter-level imperfections while gradually strengthening the supervision for argument correctness as training proceeds.

By incorporating these curriculum-driven coefficients into the base reward defined in Eq. (3), we obtain the final hallucination-aware reward at training step t:

$$R_{\text{hallu}}(t) = -\alpha(t) \cdot H_{\text{name}} - \beta(t) \cdot H_{\text{param}} - H_{\text{repeat}}. \tag{7}$$

During early training, the reward receives a high value of $\alpha(t)$ while $beta(t)$ is low, placing greater emphasis on suppressing tool-name hallucinations. As training progresses, $\alpha(t)$ gradually decreases while $\beta(t)$ increases, progressively shifting the optimization focus toward fine-grained parameter-level hallucinations. Note that the persistent penalty on H_{repeat} continuously discourages redundant and circular tool-use behaviors. This provides a smooth transition between hallucination types of different learning difficulties, enabling the model to progressively acquire the sub-skills required for reliable tool invocation. The final reward is computed as the sum of these components:

$$\mathcal{R}_{\text{final}} = R_{\text{format}} + R_{\text{acc}} + R_{\text{hallu}} \tag{8}$$

Table 2: StableToolBench Test Results. The evaluation dataset consists of 472 samples for G1, 230 for G2 and 61 for G3. Best in **bold**.

Series	Size	Method	Overall		G1		G2		G3	
			Acc	Hallu	Acc	Hallu	Acc	Hallu	Acc	Hallu
Qwen3	0.6B	/	0.39	0.64	0.40	0.60	0.39	0.70	0.26	0.70
		GRPO	0.66	0.42	0.72	0.38	0.58	0.44	0.54	0.64
		ResT	0.62	0.67	0.70	0.57	0.50	0.74	0.38	0.85
		Ours	**0.67**	**0.40**	0.75	0.36	0.36	0.57	0.42	0.52
	1.7B	/	0.34	0.48	0.35	0.46	0.32	0.53	0.33	0.50
		GRPO	**0.61**	0.45	0.71	0.41	0.45	0.47	0.43	0.69
		ResT	0.16	0.76	0.19	0.76	0.12	0.67	0.11	0.89
		Ours	0.60	**0.37**	0.62	0.34	0.57	0.39	0.56	0.50
	4B	/	0.64	0.33	0.67	0.30	0.57	0.34	0.62	0.42
		GRPO	**0.74**	0.34	0.79	0.31	0.66	0.37	0.62	0.42
		ResT	0.74	0.35	0.81	0.31	0.65	0.38	0.48	0.52
		Ours	0.73	**0.31**	0.78	0.30	0.65	0.33	0.66	0.38
	8B	/	0.75	0.30	0.79	0.27	0.67	0.33	0.72	0.34
		GRPO	0.67	0.32	0.72	0.29	0.64	0.36	0.44	0.40
		ResT	0.79	0.38	0.85	0.30	0.70	0.14	0.66	0.20
		Ours	**0.89**	**0.27**	0.92	0.25	0.83	0.30	0.90	0.30
Llama3.2 (Instruct)	1B	/	0.03	0.97	0.03	0.97	0.02	0.96	0.05	0.98
		GRPO	0.03	0.80	0.04	0.74	0.02	0.83	0.02	0.91
		Ours	**0.07**	**0.61**	0.09	0.65	0.04	0.44	0.03	0.58
	3B	/	0.28	0.72	0.30	0.74	0.27	0.68	0.21	0.72
		GRPO	0.64	0.43	0.71	0.40	0.55	0.44	0.38	0.58
		Ours	**0.66**	**0.39**	0.74	0.37	0.55	0.42	0.41	0.45

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models and datasets. We evaluate our method on two model families: the Qwen3 series (including 0.6B, 1.7B, 4B, and 8B variants) (Yang et al., 2025) and the Llama 3.2 series (including 1B-Instruct and 3B-Instruct) (Grattafiori et al., 2024). For training, we adopt the rlla-4k dataset (Qian et al., 2026; Lin et al., 2026), a mixed corpus designed to facilitate robust tool learning under reinforcement learning. It comprises three complementary sources: ToolACE (Liu et al., 2025) (2k samples), which targets the invoke-versus-answer decision in multi-step interactions; Hammer (masked) (Lin et al., 2025) (1k samples), which randomizes tool and argument names to enforce description-grounded generalization; and XLAM (Zhang et al., 2025) (1k samples), which supplies compositional tasks requiring one or multiple tool calls per turn. For evaluation, we employ API-Bank (Li et al., 2023), a three-tiered benchmark testing tool invocation in multi-turn dialogues, and StableToolBench (Guo et al., 2024), a stable evaluation subset of ToolBench comprising solvable samples. We do not execute or simulate actual API calls. Instead, we parse the tool calls generated by the model and assess their correctness based on the predicted tool names and parameters.

Baselines and training setup. Following prior work (Qian et al., 2026; Lin et al., 2026), we convert multi-turn dialogue samples into single-step instances to provide denser and more fine-grained reward signals. Specifically, a dialogue with K interaction steps is decomposed into K sub-instances, each conditioned on the full preceding conversation as input and supervised with the model's current action (e.g., tool invocation or response) as the target. For a fair comparison, we use the original GRPO method with the reward design from ToolRL (Qian et al., 2026) as our primary baseline. We also compare against ResT (Lin et al., 2026), a state-of-the-art method that assigns token-level importance weights. However, due to implementation mismatches when transferring ResT to the Llama series, we report ResT results only on the Qwen series.

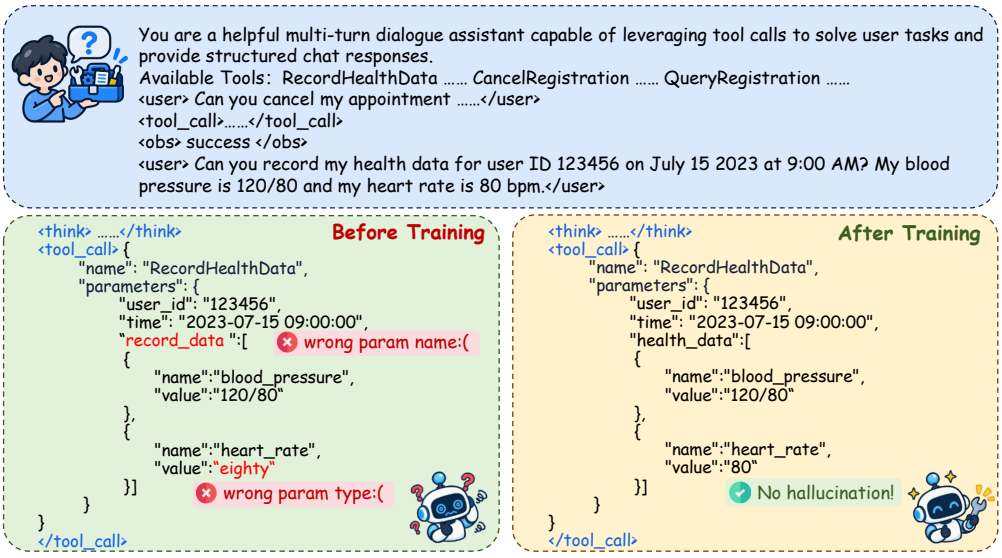

Figure 3: Qualitative comparison of model behavior before and after training. Given the same dialogue context and user request, the trained model produces the correct tool call with valid parameters, while the baseline exhibits both parameter type errors and hallucinated tool calls.

4.2 Performance Evaluation

We evaluate our method using two primary metrics: task-level accuracy and tool-call-level hallucination rate, which capture performance at different granularities. Task-level accuracy measures whether the model successfully completes a given user query, computed as the proportion of correctly answered tasks over the total number of evaluation instances. In contrast, the hallucination rate is assessed at the tool-call level, calculated as the ratio of hallucinated tool calls to the total number of tool invocations made by the model. For API-Bank, which provides ground-truth labels, we employ regex-based exact matching to determine answer correctness (Qian et al., 2026; Lin et al., 2026). For StableToolBench, we adopt LLM-as-a-Judge (Manakul et al., 2023; Min et al., 2023) to assess response accuracy, following established evaluation protocols(Xu et al., 2024). The hallucination rate is measured by inspecting each tool call against a comprehensive set of criteria. A call is classified as hallucinated if it invokes a nonexistent tool, selects an irrelevant tool, uses undefined parameters, omits required arguments, fabricates parameter values, uses mismatched parameter types, or repeats an identical $(\text{tool_name}, \text{tool_parameters})$ pair in the same turn. Both regex matching and LLM-as-a-Judge are applied to identify these hallucination patterns.

Tables 1 and 2 demonstrate the effectiveness of our method in improving the reliability of LLM-based tool use across different model families, model sizes, and evaluation settings. Figure 3 presents a qualitative comparison before and after training. Notably, our method consistently achieves the lowest hallucination rate in every reported evaluation setting. At the same time, it maintains competitive and often superior tool-call accuracy. The improvement is particularly pronounced for smaller models: on API-Bank, our method improves the accuracy of Llama3.2-1B-Instruct from 0.19 to 0.48 while reducing hallucination from 0.93 to 0.39, and improves Llama3.2-3B-Instruct from 0.34 to 0.77 while reducing hallucination from 0.62 to 0.29. Similar gains are observed on StableToolBench. These results demonstrate that *ToolHallu* manages to substantially improve tool-use reliability while preserving, or even enhancing, task accuracy.

In contrast, although GRPO and ResT can achieve competitive or even higher accuracy in some settings, their hallucination rates remain relatively high. This observation highlights that high tool-use accuracy does not necessarily imply a low hallucination rate, possibly because existing methods primarily optimize for tool-use correctness without explicitly considering hallucinated tool invocations. Moreover, by analyzing the outputs of ResT on Qwen3-1.7B under StableToolBench, we observe a large number of repetitive responses, a typical form of generation degeneration that can disrupt tool-use trajectories and lead to unstable interactions. Conversely, our method exhibits substantially fewer such behaviors, further demonstrating stronger stability and more reliable tool interactions.

4.3 Ablation Study

Effectiveness of curriculum scheduling. To validate the contribution of our curriculum-driven reward weighting, we compare our full method against a variant that disables the curriculum by fixing both coefficients at $\alpha = 1.0$ and $\beta = 1.0$ throughout training, applying uniform penalty strength to tool-name and parameter hallucinations without dynamic scheduling. As shown in Figure 4, the curriculum-enhanced variant consistently outperforms the uniform counterpart across different model sizes, demonstrating that progressive shifting from tool-selection emphasis to fine-grained parameter-level precision effectively guides the model toward more reliable tool invocation.

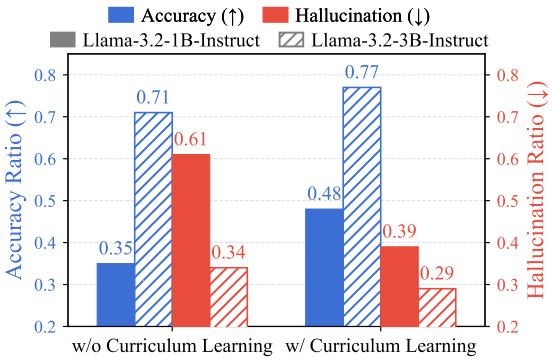

Figure 4: The comparison of curriculum learning with Llama 3.2 series on API-Bank.

Effectiveness of hallucination components. To disentangle the contribution of each hallucination category in our reward design, we conduct component-wise ablation by selectively removing penalties for tool-name hallucination, tool-parameter hallucination, and repeated invocations, respectively. Specifically, we train three variants, gradually canceling tool-name punishment, tool-parameter punishment, and repetition punishment. The results, reported in Table 3, show that removing the components leads to a noticeable performance drop, confirming that each error type plays a distinct and non-redundant role in the overall reward supervision. Among the three, removing tool-name punishment leads to the largest drop. This pattern is consistent with our curriculum intuition that tool-name selection is a foundational prerequisite, laying the groundwork for learning more complex parameter-level hallucination penalties. These findings collectively verify that multi-component hallucination rewards are essential for comprehensive hallucination mitigation.

Table 3: API-Bank Test Results of Llama-3.2-3B-Instruct with gradually canceled hallucination punishment. Each ablation is conducted cumulatively based on the preceding variant. Best in **bold**.

Method	Overall		Level 1		Level 2		Level 3	
	Acc	Hallu	Acc	Hallu	Acc	Hallu	Acc	Hallu
Ours	**0.77**	**0.29**	0.79	0.26	0.70	0.45	0.76	0.28
w/o H_{name}	0.70	0.34	0.72	0.32	0.62	0.47	0.65	0.34
w/o $H_{name} + H_{param}$	0.68	0.36	0.69	0.34	0.59	0.51	0.69	0.35
w/o $H_{name} + H_{param} + H_{repeat}$	0.60	0.39	0.71	0.32	0.64	0.44	0.25	0.61

5 Conclusion

In this work, we address the critical challenge of tool-call hallucination in LLM-based agents by formulating it as a reward misalignment problem and proposing a fine-grained, multi-tiered reward supervision scheme, *ToolHallu*. We also incorporate a curriculum-driven training mechanism that progressively shifts supervision from tool names to tool parameters, enabling the model to acquire tool-use capabilities stably and incrementally. Extensive experiments across different model families, model sizes, and benchmarks demonstrate the effectiveness and generalizability of our approach. Notably, our method achieves the lowest hallucination rate in every reported evaluation setting on both API-Bank and StableToolBench. Ablation studies validate the effectiveness of our hallucination-aware reward design and curriculum learning.

Future work will explore more flexible curriculum learning settings, moving beyond fixed scheduling rules toward dynamically adjusting the curriculum based on real-time reward feedback or other metrics. We also plan to extend *ToolHallu* to more open-ended scenarios, such as online environments with real-world API feedback and broader language-agent tasks, adopting scalar signals produced by environment execution feedback or LLM-as-a-judge evaluators.

AI USE STATEMENT

In this work, we used generative AI tools for editing and polishing the writing to improve readability. We employed LLM-as-a-Judge for evaluation. To verify the reliability of this evaluation protocol, we additionally conducted a human-LLM consistency study, detailed in the appendix. We have not used generative AI tools for generating synthetic datasets, developing theoretical models or conceptual frameworks, formulating or proving mathematical claims, proposing or refining hypotheses, designing experiments, implementing methods, analyzing or interpreting results, or any other research-related tasks. All AI-assisted text was reviewed and revised by the authors to ensure accuracy and alignment with our technical claims. We take responsibility for the final content of this work, including text produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects, personal data, or sensitive information. All datasets used (rlla-4k, API-Bank, StableToolBench) are publicly available and used in accordance with their original licenses. Our method operates on synthetically constructed tool schemas and does not access private user data. We acknowledge the dual-use nature of improving tool-calling agents: while our work aims to enhance deployment safety by reducing hallucination, we recognize that more capable agents could be misused. Our contribution is intended solely for beneficial purposes, and we release no artifacts that could be directly repurposed for harm.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure the reproducibility of our method and experimental results. The complete description of our fine-grained reward design, including the formalization of each hallucination category and the corresponding penalty computation, is provided in Section 3.2, with the curriculum scheduling scheme detailed in Section 3.3. All hyperparameters, including the cosine decay and warmup horizons, are explicitly specified in Appendix A. The training data composition (rlla-4k) and evaluation benchmarks (API-Bank and StableToolBench) are described in Section 4.1 and Appendix C, along with the model families and sizes used in our experiments in Section 4.1. The judge prompts for training and evaluation are shown in Appendix D. We will release our code, data used, reward implementation, and training scripts as anonymous supplementary materials upon acceptance. All experimental results reported in Section 4.2 are obtained under an identical data budget to ensure fair comparison, and the corresponding evaluation scripts will be made publicly available. We believe these efforts provide sufficient information for independent verification and reproduction of our results.

REFERENCES

Amro Abbas, Kushal Tirumala, Dániel Simig, Surya Ganguli, and Ari S Morcos. Semdedup: Data-efficient learning at web-scale through semantic deduplication. *arXiv preprint arXiv:2303.09540*, 2023.

Nuo Chen, Hongguang Li, Baoyuan Wang, and Jia Li. From good to great: Improving math reasoning with tool-augmented interleaf prompting. In *Proceedings of the 2nd Workshop on Natural Language Reasoning and Structured Explanations (@ ACL 2024)*, pp. 64–79, 2024.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, et al. T-eval: Evaluating the tool utilization capability step by step. *arXiv preprint arXiv:2312.14033*, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11143–11156, 2024.

Bingguang Hao, ZengZhuang Xu, Maolin Wang, Yuntao Wen, Yicheng Chen, Cunyin Peng, Long Chen, Dong Wang, Xiangyu Zhao, Jinjie Gu, et al. Funreason: Enhancing large language models' function calling via self-refinement multiscale loss and automated data refinement. *arXiv preprint arXiv:2505.20192*, 2025.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*, 2023.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM transactions on information systems*, 43(2):1–55, 2025.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251): 1–43, 2023.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 3102–3116, 2023.

Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, et al. Robust function-calling for on-device language model via function masking. In *The Thirteenth International Conference on Learning Representations*, 2025.

Zihan Lin, Xiaohan Wang, Jie Cao, Jiajun Chai, Guojun Yin, Wei Lin, and Ran He. Rest: Reshaping token-level policy gradients for tool-use large language models. In *International Conference on Learning Representations*, volume 2026, pp. 83901–83945, 2026.

Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, et al. Toolace: Winning the points of llm function calling. In *International conference on learning representations*, volume 2025, pp. 41359–41381, 2025.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature machine intelligence*, 6(5):525–535, 2024.

Potsawee Manakul, Adian Liusie, and Mark Gales. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 9004–9017, 2023.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. Factscore: Fine-grained atomic evaluation of factual precision in long form text generation. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 12076–12100, 2023.

Tong Mu, Alec Helyar, Johannes Heidecke, Joshua Achiam, Andrea Vallone, Ian Kivlichan, Molly Lin, Alex Beutel, John Schulman, and Lilian Weng. Rule based rewards for language model safety. *Advances in Neural Information Processing Systems*, 37:108877–108901, 2024.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024.

Hanli Peng, Yongsen Zheng, Ziyao Liu, and Kwok-Yan Lam. Tool execution hallucination in llm-based agents: A unified taxonomy with detection, mitigation, and future directions. 2026.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *Advances in Neural Information Processing Systems*, 38:105523–105553, 2026.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40, 2024a.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024b.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551, 2023.

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

Mrinank Sharma, Meg Tong, Tomek Korbak, David Duvenaud, Amanda Askell, Sam Bowman, Esin Durmus, Zac Hatfield-Dodds, Scott Johnston, Shauna Kravec, et al. Towards understanding sycophancy in language models. In *International Conference on Learning Representations*, volume 2024, pp. 110–144, 2024.

Weijia Shi, Xiaochuang Han, Mike Lewis, Yulia Tsvetkov, Luke Zettlemoyer, and Wen-tau Yih. Trusting your evidence: Hallucinate less with context-aware decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pp. 783–791, 2024.

Hongshen Xu, Zichen Zhu, Lei Pan, Zihan Wang, Su Zhu, Da Ma, Ruisheng Cao, Lu Chen, and Kai Yu. Reducing tool hallucination via reliability alignment. *arXiv preprint arXiv:2412.04141*, 2024.

Weikai Xu, Chengrui Huang, Shen Gao, and Shuo Shang. Llm-based agents for tool learning: A survey: W. xu et al. *Data Science and Engineering*, 10(4):533–563, 2025.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Zhiwei Liu, Yihao Feng, et al. xlam: A family of large action models to empower ai agent systems. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11583–11597, 2025.

Yuxiang Zhang, Jing Chen, Junjie Wang, Yaxin Liu, Cheng Yang, Chufan Shi, Xinyu Zhu, Zihao Lin, Hanwen Wan, Yujiu Yang, et al. Toolbehonest: A multi-level hallucination diagnostic benchmark for tool-augmented large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 11388–11422, 2024.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

A TRAINING DETAILS

Table A.1: Configuration for training.

Data Configuration	
Train Batch Size	512
Validation Batch Size	128
Max Prompt Length	2048
Max Response Length	2048
Optimization	
Learning Rate	1×10^{-6}
PPO Mini Batch Size	64
KL Loss Used	False
Rollout Configuration	
Rollout Name	vllm 0.11.0
GPUs	8×80G A800, 4×48G L20, 8×80G H800
GPU Memory Utilization	0.6
Number of Rollouts	4
Training & Logging	
Save Frequency (Steps)	10
Test Frequency (Steps)	10
Total Epochs	15
Curriculum Learning Configuration	
T_{warmup}	100
T_{decay}	100

B TRAINING ALGORITHM

Our reward design primarily consists of three components: format, accuracy, and hallucination-related rewards. The format and accuracy rewards follow the designs of prior work, while the hallucination-related reward is designed along three dimensions: tool name, tool parameters, and invocation repetition. We further introduce a curriculum learning mechanism that modulates the penalty coefficients for tool-name and tool-parameter hallucinations, facilitating a progressive learning process from simple to complex.

Algorithm 1 Rule-Based Reward Computation

Require: Model response y; ground-truth tool calls $\mathcal{G}$; available tools $\mathcal{A}$; training step t.
Ensure: Final reward $\mathcal{R}$.

$R_{\text{format}} \leftarrow \text{FormatReward}(y, \mathcal{G})$ ▷ Following ToolRL and ResT
$R_{\text{acc}} \leftarrow \text{AccuracyReward}(y, \mathcal{G})$ ▷ Following ToolRL and ResT
$(H_{\text{name}}, H_{\text{param}}, H_{\text{repeat}}) \leftarrow \text{Hallucination}(\mathcal{P}, \mathcal{G}, \mathcal{A})$

$$\alpha(t) \leftarrow \begin{cases} \max\left(0.5 \cdot \left(\cos\left(\frac{\pi t}{T_{\text{decay}}}\right) + 1\right), 0.5\right), & \text{if } t \le T_{\text{decay}} \\ 0.5, & \text{if } t > T_{\text{decay}} \end{cases}$$

$$\beta(t) \leftarrow \begin{cases} 0.5\left(1 - \cos\left(\frac{\pi t}{T_{\text{warmup}}}\right)\right), & t \le T_{\text{warmup}}, \\ 1.0, & t > T_{\text{warmup}}. \end{cases}$$

$R_{\text{hallu}} \leftarrow -\alpha(t) H_{\text{name}} - \beta(t) H_{\text{param}} - H_{\text{repeat}}$
$\mathcal{R} \leftarrow R_{\text{format}} + R_{\text{acc}} + R_{\text{hallu}}$
return $\mathcal{R}$

Algorithm 2 Hallucination Score Computation

Require: Predicted tool calls $\mathcal{P}$; ground-truth tool calls $\mathcal{G}$; available tools $\mathcal{A}$.
Ensure: $(H_{\text{name}}, H_{\text{param}}, H_{\text{repeat}})$.

$s_{\text{name}} \leftarrow 0, \quad s_{\text{param}} \leftarrow 0, \quad s_{\text{repeat}} \leftarrow 0, \quad e_{\text{force}} \leftarrow 0, \quad e_{\text{type}} \leftarrow 0, \quad e_{\text{value}} \leftarrow 0$
$U \leftarrow \emptyset$ ▷ Previously observed tool calls
for each predicted tool call $p \in \mathcal{P}$ **do**
  $n_p \leftarrow p.\text{name}, \quad \theta_p \leftarrow p.\text{parameters}, \quad \kappa_p \leftarrow (n_p, \text{Canonicalize}(\theta_p))$
  if $\kappa_p \in U$ **then**
    $s_{\text{repeat}} \leftarrow s_{\text{repeat}} + 1$ ▷ Repeated invocation
  end if
  $U \leftarrow U \cup \{\kappa_p\}$
  if $n_p \notin \mathcal{A}$ **then**
    $s_{\text{name}} \leftarrow s_{\text{name}} + 1$ ▷ Non-existent tool
    continue
  end if
  $\theta_g \leftarrow \text{GTParameters}(n_p, \mathcal{G}), \quad s_{\text{force}} \leftarrow 0, \quad s_{\text{type}} \leftarrow 0, \quad s_{\text{value}} \leftarrow 0$
  if $n_p \in \text{Names}(\mathcal{G})$ **then** ▷ Relevant tool
    $s_{\text{name}} \leftarrow s_{\text{name}} - 1$
    for each $k \in \text{Keys}(\theta_g)$ **do**
      if $k \notin \theta_p$ **then**
        $s_{\text{force}} \leftarrow s_{\text{force}} + 1$ ▷ Missing parameter
      else
        $s_{\text{force}} \leftarrow s_{\text{force}} - 1$
        if $\theta_p[k] = \theta_g[k]$ **then**
          $s_{\text{value}} \leftarrow s_{\text{value}} - 1$ ▷ Truthful parameter value
        end if
      end if
    end for
    for each $k \in \text{Keys}(\theta_p)$ **do**
      if $k \notin \mathcal{A}[n_p].\text{parameters}$ **then**
        $s_{\text{force}} \leftarrow s_{\text{force}} + 1$ ▷ Extraneous parameter
      else if $\text{Type}(\theta_p[k]) \neq \text{ExpectedType}(n_p, k)$ **then**
        $s_{\text{type}} \leftarrow s_{\text{type}} + 1$ ▷ Invalid parameter type
      end if
    end for
    $e_{\text{force}} \leftarrow s_{\text{force}}/|\text{Keys}(\theta_p)|, \quad e_{\text{type}} \leftarrow s_{\text{type}}/|\text{Keys}(\theta_p)|$
    $e_{\text{value}} \leftarrow s_{\text{value}}/|\text{Keys}(\theta_g)|, \quad s_{\text{param}} \leftarrow s_{\text{param}} + e_{\text{force}} + e_{\text{type}} + e_{\text{value}}$
  end if
end for
$H_{\text{name}} \leftarrow s_{\text{name}}/|\mathcal{P}|, \quad H_{\text{param}} \leftarrow s_{\text{param}}/|\mathcal{P}|, \quad H_{\text{repeat}} \leftarrow s_{\text{repeat}}/|\mathcal{P}|$
return $(H_{\text{name}}, H_{\text{param}}, H_{\text{repeat}})$

C DATASET PROCESSING

For API-Bank, we first filter out samples that cannot be reliably evaluated against the ground truth. Specifically, we remove samples where the tool invoked in the ground-truth answer is not included in the provided tool list, as well as samples where the information required to fill the tool-call parameters is missing from the user query. This filtering is important because our accuracy evaluation determines whether a predicted tool call is correct by performing regular-expression matching against the ground-truth tool call. Samples with an unavailable ground-truth tool or insufficient information for constructing the reference call cannot provide a valid basis for such matching and may therefore lead to inaccurate accuracy estimates. After filtering, we evaluate 351 samples for Level 1, 56 samples for Level 2, and 124 samples for Level 3. In contrast, we retain all API-Bank samples for hallucination evaluation, since hallucination is assessed by an LLM-based evaluator rather than by direct matching with the ground truth. We also verify the parameter types specified in the questions and corresponding answers to ensure that the resulting evaluation is well-defined and consistent.

Since stableToolbench contains heterogeneous task instructions and tool schemas, we organize its prompts into a structure similar to those of API-Bank and rlla-4k to facilitate consistent evaluation.

D PROMPT DETAILS

We provide the prompt templates for training and evaluation, referring to the settings reported in previous work.

Training Data Prompt:

You are a helpful multi-turn dialogue assistant capable of leveraging tool calls to solve user tasks and provide structured chat responses.
Available Tools
In your response, you can use the following tools:
1. Name: {tool_name}
Description: {tool_description}
Parameters:{"param": {"description": "...", "type": "...", "default": "..."}, ...}
Steps for Each Turn
1. **Think:** Recall relevant context and analyze the current user goal.
2. **Decide on Tool Usage:** If a tool is needed, specify the tool and its parameters.
3. **Respond Appropriately:** If a response is needed, generate one while maintaining consistency across user queries.
Output Format

```
<think> Your thoughts and reasoning </think>
<tool_call>
{"name": "Tool name", "parameters":
{"Parameter name": "Parameter content", "... ...": "... ..."}}
{"name": "... ...", "parameters": {"... ...": "... ...",
"... ...": "... ..."}}
...
</tool_call>
<response> AI's final response </response>
```

Important Notes
1. You must always include the `<think>` field to outline your reasoning. Provide at least one of `<tool_call>` or `<response>`. Decide whether to use `<tool_call>` (possibly multiple times), `<response>`, or both.
2. You can invoke multiple tool calls simultaneously in the `<tool_call>` fields. Each tool call should be a JSON object with a "name" field and an "parameters" field containing a dictionary of parameters. If no parameters are needed, leave the "parameters" field an empty dictionary.
3. Refer to the previous dialogue records in the history, including the user's queries, previous `<tool_call>`, `<response>`, and any tool feedback noted as `<obs>` (if exists).
Dialogue Records History

```
<user> ... </user>
<think> ... </think>
<tool_call> {"name": "...", "parameters": {"...": ..."}}
</tool_call>
<obs> {"name": "...", "results": {"...": ...}} <obs>
<think> ... </think>
<response> ... </response>
<user> ... </user>
```

Figure D.1: The template of rlla-4k dataset as the training data; API-Bank and StableToolBench organize evaluation prompts similarly.

Hallucination Judge Prompt (Tool Relevance):

Query:
{query}

Tool Description:
{tool_description}

Tool Parameter:
{tool_parameter}

Given the above query, along with the description and parameter information of a certain tool, you need to infer the tool's purpose and determine whether it might be relevant to completing a specific task within the query. You need to provide `tool_relevance` according to the following rules:
1. If the tool's purpose is completely irrelevant to the query, return "Irrelevant".
2. If the tool's purpose can be used to solve the user's query, return "Relevant".
3. If the tool's purpose might be relevant to the query, or if the tool's description does not contain enough information to determine its use, return "Unsure".

Now give your reasoning in "content" and `tool_relevance` in JSON format to `check_tool_suitability`.

Figure D.2: The hallucination judge prompt to evaluate tool relevance.

Hallucination Judge Prompt (Parameter Truthfulness):

Interaction History:
{user_history}

Tool Description:
{tool_parameter}

Specific Tool Calling:
{tool_calling}

Given the interaction history and tool description, you need to determine whether the value of each parameter in a specific tool call is hallucinated. If there are hallucinated parameters, then the entire tool call is deemed untruthful. You need to provide `calling_trufulness` based on the following rules:

1. If the tool parameters are explicitly mentioned in the interaction history, then return "Truthful".
2. If the tool parameters are explicitly mentioned in the tool description, then return "Truthful".
3. If the tool parameters are values that can be inferred from the interaction history or common knowledge (such as query keywords, pages, number of limits, etc.), then return "Truthful".
4. If the tool parameters require specific values from the user (such as user or product IDs, specific flight numbers, etc.), and the parameter value in the tool call does not appear in the interaction history or the example value of tool description, then return "Untruthful".
5. If you cannot determine whether the tool call is truthful or not, return "Unsure".

Now give your reason in "content" and `calling_trufulness` of JSON to `evaluate_calling_truthfulness`.

Figure D.3: The hallucination judge prompt to evaluate parameter truthfulness.

Accuracy Judge Prompt (StableToolBench):

You are an expert evaluator of AI assistant behavior in multi-turn dialogues. Your task is to judge whether the agent's response is **reasonable and appropriate**, not whether it completed the entire task in one turn.

Key Principles for Your Evaluation:

1. **Prefer Safety over Speculation**: An agent that makes a single correct, safe tool call and appropriately waits for feedback is **more reasonable** than one that makes multiple tool calls based on unverified assumptions, especially when later calls depend on earlier results.

2. **Tool Dependency Awareness**: If the user's request requires information that can only be obtained from a previous tool call, the agent **must** wait. Calling dependent tools in the same turn is unreasonable.

3. **The `<response>` Field is Optional but Encouraged**: The agent is **not required** to include a `<response>` section if it is only making a tool call. However, if the agent is waiting for more information or has only partially completed the task, it **should** communicate this clearly to the user in the `<response>` section.

4. **A Good First Step is Valuable**: A response that takes a correct first step toward solving the user's request, even if incomplete, is a **good response**, provided the step is logically sound and does not make incorrect assumptions.

Evaluation Rules:

- **"Reasonable"**: The agent's action is logical, safe, and contextually appropriate. It may be incomplete, but it does not make incorrect assumptions or hallucinate. It takes a correct step toward solving the user's request or appropriately asks for missing information.

- **"Unreasonable"**: The agent's action is illogical, unsafe, or makes incorrect assumptions. It calls tools with invalid or guessed parameters, ignores clear safety protocols, or contradicts its own reasoning.

- **"Unsure"**: You cannot confidently determine if the action is appropriate due to ambiguity or lack of context.

Query:
{query}

Agent's Response:
{response}

Instructions for Output:
Now give your reason in "content" and `response_reasonableness` as JSON to `check_response_reasonableness`.

Figure D.4: The accuracy judge prompt used to evaluate accuracy in StableToolBench.

E EVALUATION DETAILS AND HUMAN EVALUATION

We employ Qwen3.5-27B as the judge model, applying the criteria defined in Figure D.2, Figure D.3, and Figure D.4. Parameter values judged as "Untruthful" and tool names judged as "Irrelevant" are counted as hallucinations. For accuracy evaluation on StableToolBench, responses judged

as "Reasonable" receive a score of 1, and "Unsure" receive 0.5. Both generation and evaluation use greedy sampling. To validate the reliability of our LLM-based evaluation, we conduct a human evaluation targeting three aspects: accuracy judgment, tool relevance hallucination detection, and parameter value hallucination. For each aspect, we randomly sample 50 positive and 50 negative cases, and human annotators assess whether the classification is correct. The detailed results are presented in Table E.1. The evaluation achieves an overall accuracy of 95.7%. These results demonstrate the high reliability of our evaluation framework in distinguishing correct model answers from hallucinated tool calls, and further confirm the alignment between our LLM evaluator and human judgment in this intricate domain.

Table E.1: Human evaluation results on the reliability of the LLM-based evaluator. "Human Agr." denotes the percentage of cases where human annotators agree with the LLM evaluator's judgment. The overall agreement across all categories is reported in the last row.

Evaluation Aspect	**# Pos.**	**# Neg.**	**Human Agr. (%)**
Accuracy Judgment	47/50	47/50	94.0
Tool Relevance Hallucination	48/50	47/50	95.0
Parameter Value Hallucination	48/50	50/50	98.0
Overall	143/150	144/150	**95.7**

F MORE RESULTS

Table F.1: Statistical Validation with Multiple Seed Runs on Llama-3.2-3B-Instruct trained with ToolHallu and GRPO tested on API-Bank

Method	**Overall**		**Level 1**		**Level 2**		**Level 3**	
	Acc	**Hallu**	**Acc**	**Hallu**	**Acc**	**Hallu**	**Acc**	**Hallu**
Ours-seed1	0.77	0.30	0.79	0.27	0.70	0.45	0.76	0.31
Ours-seed2	0.77	0.29	0.79	0.26	0.70	0.45	0.76	0.31
Ours-seed3	0.77	0.29	0.79	0.26	0.70	0.45	0.76	0.28
GRPO-seed1	0.60	0.40	0.71	0.33	0.64	0.48	0.25	0.61
GRPO-seed2	0.59	0.39	0.71	0.33	0.61	0.44	0.26	0.59
GRPO-seed3	0.60	0.39	0.71	0.32	0.64	0.44	0.25	0.61

Table F.2: Statistical Validation with Multiple Seed Runs on Llama-3.2-3B-Instruct trained with ToolHallu and GRPO tested on StableToolBench

Method	**Overall**		**G1**		**G2**		**G3**	
	Acc	**Hallu**	**Acc**	**Hallu**	**Acc**	**Hallu**	**Acc**	**Hallu**
Ours-seed1	0.66	0.40	0.74	0.38	0.55	0.41	0.44	0.44
Ours-seed2	0.66	0.39	0.74	0.38	0.55	0.41	0.41	0.44
Ours-seed3	0.66	0.39	0.74	0.37	0.55	0.42	0.41	0.45
GRPO-seed1	0.63	0.43	0.70	0.40	0.55	0.43	0.36	0.60
GRPO-seed2	0.62	0.43	0.71	0.40	0.52	0.44	0.33	0.59
GRPO-seed3	0.64	0.43	0.71	0.40	0.55	0.44	0.38	0.58

G OUTPUT CASES

We provide an illustrative example tested on API-Bank in Figure G.1. The base model hallucinates an unavailable tool argument by directly invoking `QueryBalance` with a token that has

not yet been obtained. GRPO learns the underlying tool dependency and generates the complete `GetUserToken` to `QueryBalance` sequence. However, our method provides finer-grained supervision at both the tool-call and parameter levels, encouraging the model to make only the currently justified call and to defer subsequent actions until the required tool output becomes available.

A Case of Correcting Hallucination Tool Use

User Query:
Can you tell me my account balance? Based on our conversation above, please give the tool call to address my request. You shouldn't make any assumption and please only call the tool once you are sure which tool to call and what arguments to use.

Tool Call (Base):

```
{
  "name": "QueryBalance",
  "parameters": {
    "token":
    "<token obtained from GetUserToken tool call>"
  }
}
```

Tool Call (GRPO):

```
{
  "name": "GetUserToken",
  "parameters": {
    "username": "JohnDoe", "password": "pass123"
  }
},
{
  "name": "QueryBalance",
  "parameters": {
  "token": "the token received from the GetUserToken tool call"
  }
}
```

Tool Call (Ours):

```
  {
    "name": "GetUserToken",
    "parameters": {
      "username": "JohnDoe", "password": "pass123"
    }
  }
```

Figure G.1: The case study of Llama3.2-3B-Instruct's performance on API-Bank.

We also provide an example of Llama3.2-3B-Instruct tested on StableToolBench in Figure G.2. These three outputs exhibit progressively different levels of tool-use ability. The original model confuses parameter schemas with parameter values, embedding type and description metadata inside the arguments and thereby producing multiple parameter-type errors. The model trained using GRPO selects the appropriate tools and recovers the correct parameter semantics, but introduces a type-level error by representing the integer length as a string. However, our method helps the model to correctly ground both tool selection and parameter values. These examples illustrate why tool-use alignment benefits from our hallucination-aware reward signals.

A Case of Correcting Hallucination Tool Use

User Query:
I'm looking for a random string of 70 characters. Could you also generate 150 UUIDs with dashes? Thank you!

Tool Call (Base):

```
{
  "name": "Helper Function_Random",
  "parameters": { "length": { "type": "int",
      "description": "Default: 20", "content": "70" }
  }
},
{
  "name": "Helper Function_Generate UUID",
  "parameters": {
    "timestamp_first": { "type": "bool",
      "description": "Default: false", "content": "true"
    },
    "remove_dash": { "type": "bool",
      "description": "Default: true", "content": "false"
    },
    "count": { "type": "str",
      "description": "Default: 5", "content": "150"
    }
  }
}
```

Tool Call (GRPO):

```
{
  "name": "Helper Function_Random",
  "parameters": { "length": "70"
  }
},
{
  "name": "Helper Function_Generate UUID",
  "parameters": { "count": "150", "remove_dash": false
  }
}
```

Tool Call (Ours):

```
{
  "name": "Helper Function_Random",
  "parameters": { "length": 70 }
},
{
  "name": "Helper Function_Generate UUID",
  "parameters": { "timestamp_first": false,
    "remove_dash": false, "count": "150"
  }
}
```

Figure G.2: The case study of Llama3.2-3B-Instruct's performance on StableToolBench.