

Enhancing Tool Learning in Large Language Models with Hierarchical Error Checklists

Yue Cui^{1,2}, Liuyi Yao¹, Shuchang Tao¹, Weijie Shi², Yaliang Li¹
Bolin Ding¹, and Xiaofang Zhou²

¹Alibaba Group

²The Hong Kong University of Science and Technology

{ycuias, ashiah, zxf}@cse.ust.hk,

{yly287738, taoshuchang.tsc, yaliang.li, bolin.ding}@alibaba-inc.com

Abstract

Large language models (LLMs) have significantly advanced natural language processing, particularly through the integration of external tools and APIs. However, their effectiveness is frequently hampered by parameter mis-filling during tool (function) calling. In this paper, we propose the Hierarchical Tool Error Checklist (HiTEC) framework to systematically diagnose and mitigate tool-calling errors without relying on extensive real-world interactions. HiTEC introduces a two-tiered approach: a global error checklist that identifies common, cross-tool issues, and a local error checklist that targets tool-specific and contextual failures. Building on this structure, we propose two deployments: HiTEC-In Context Learning (HiTEC-ICL) and HiTEC-Kahneman-Tversky Optimization (HiTEC-KTO). HiTEC-ICL embeds the global checklist in the initial prompts and leverages a two-round conversational interaction to dynamically refine parameter handling, while HiTEC-KTO generates high-quality negative examples to drive fine-tuning via preference-based optimization. Extensive experiments conducted on five public datasets show that our framework improves parameter-filling accuracy by up to 42% compared to baseline methods.

1 Introduction

Large language models (LLMs) have revolutionized natural language processing by enabling advanced comprehension, reasoning, and task execution capabilities. Among these, the integration of external tools and application programming interfaces (APIs) represents a critical milestone, allowing LLMs to expand their utility beyond textual analysis into interactive, task-oriented domains (Gou et al., 2023; Li et al., 2023; Qin et al.; Qu et al., 2025; Song et al., 2023; Tang et al., 2023; Wu et al., 2024). To achieve this, LLMs must navigate a complex process of function calling, which

involves selecting the appropriate tools, formulating precise input arguments, and parsing results to satisfy tool input. Despite these advancements, tool-calling often suffers from incorrect parameter filling (Lin et al., 2024; Liu et al., 2024), a recurring issue that undermines the accuracy and reliability of LLM-driven interactions.

Most previous tool learning methods require LLM-tool interactions to improve the calling accuracy (Chen et al., 2024a; Qin et al.; Shi et al., 2024; Wang et al., 2024; Yang et al., 2024; Yao et al., 2022; Zhang et al., 2023). For example, STE (Wang et al., 2024) simulate plausible scenarios and incorporates execution feedback to enhance the correct use of tools. It involves first simulating queries, executing real tool calls via tool-LLM interactions, and learning from function calling outputs when errors occur. While real-world interactions with tools can yield valuable insights, they cause intensive resources (For example, 10-25\$/1,000 transactions for Bing Search API ¹) and instability issues (Guo et al., 2024). Furthermore, the errors encountered by most tools called by LLMs are predominantly common types that can be known in advance before real tool calling. To address these challenges, we argue that (1) tool-LLM interactions that involve real-world tool callings are not always necessary, and (2) structured error checklists can systematically identify potential performance deficiencies and guide targeted improvements.

Based on this motivation, this paper proposes a Hierarchical Tool Error Checklist (HiTEC) framework to enhance tool learning with LLMs by identifying and addressing tool-calling errors. As shown in Figure 1, The HiTEC framework is composed of the constructions of two levels of error checklists: a global error checklist, which captures general errors that frequently occur across different tools,

¹<https://www.microsoft.com/en-us/bing/apis/pricing>

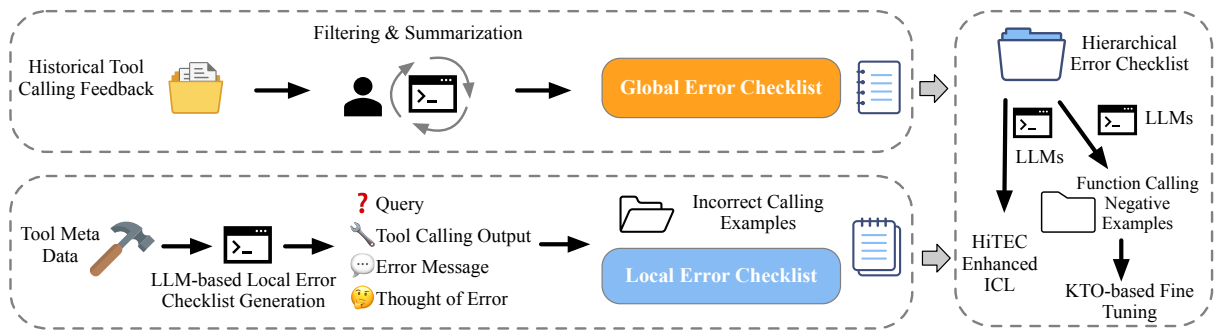


Figure 1: Pipeline of the Proposed Hierarchical Tool Error Checklist (HiTEC), which includes Global Error Checklist and Local Error Checklist

and a local error checklist, which focuses on tool-specific errors and contextual failures. These checklists provide a structured and comprehensive way to diagnose and rectify tool-calling errors without requiring extensive real-world execution.

We then deploy HiTEC in tuning-free and tuning-based ways and propose: HiTEC-In Context Learning (HiTEC-ICL) and HiTEC-Kahneman-Tversky Optimization (HiTEC-KTO). By embedding the global error checklist in the initial query and integrating the local error checklist through a two-round conversational interaction, HiTEC-ICL guides LLMs to preemptively avoid common mistakes and refine parameter handling flexibly. HiTEC-KTO leverages the error checklists to generate high-quality negative examples, which are then used to empower open-source LLMs through KTO-based fine-tuning. This strategy equips models with enhanced function calling accuracy by training them to recognize and correct errors. Our main contributions are as follows:

- *Framework.* We propose a novel hierarchical error checklist framework that integrates global and local error messages to enhance tool learning of LLMs. The framework distinguishes between offline self-generated errors and online tool-LLM interaction based accumulated errors to support adaptive and scalable tool learning.

- *Method.* We propose two novel methods: (1) HiTEC-ICL, an efficient mechanism to integrate error messages into LLM prompts for dynamic error reflection and correction; and (2) HiTEC-KTO, a negative sample generation method that overcomes the failure modes of preference-based optimization through fine-tuning with negative examples.

- *Analysis.* We theoretically and empirically demonstrate the effectiveness of KTO-based tuning in overcoming the failure modes of preference-based optimization in tool learning, thereby enabling LLMs to enhance their tool-calling accuracy

through fine-tuning with negative examples.

- *Performance.* We conduct extensive experiments across five public datasets. Our results demonstrate improvements in parameter-filling accuracy, tool-calling success rates compared to baseline methods.

2 Related Work

LLM-based tool learning methods can be generally categorized into tuning-free and tuning-based approaches (Qu et al., 2025). Tuning-free methods leverage the inherent capabilities of large language models by prompting them to interact directly with external tools. This category includes techniques such as few-shot demonstrations (Wang et al., 2024), rule-based strategies (Shi et al., 2024; Yao et al., 2022; Zhang et al., 2023), and the use of optimized tool descriptions (Chen et al., 2024b; Yuan et al.) to facilitate efficient parameter extraction and tool usage. Owing to their simplicity and scalability, tuning-free methods have gained popularity; however, their performance often lags behind that of tuning-based approaches and a finally correct tool calling may require multiple times of tool-LLM interactions, which can be costly.

In contrast, tuning-based methods enhance tool calling performance through fine-tuning strategies. GPT4Tools (Yang et al., 2024) uses LoRA-based supervised fine-tuning, while the introduction of tool-specific tokens is demonstrated in Toolkengpt (Hao et al., 2023). Other approaches leverage augmented datasets (Lin et al., 2024), and utilize interactive path-based reasoning (Chen et al., 2024a; Qin et al.) to improve tool interaction precision. Relign (Xu et al., 2024) tackles tool hallucinations (also incorrect tool selection/usage) by expanding the action space with "indecisive actions" (e.g., deferring tool use) and aligning reliability via preference optimization. Although tuning-based methods consistently deliver superior performance by tailor-

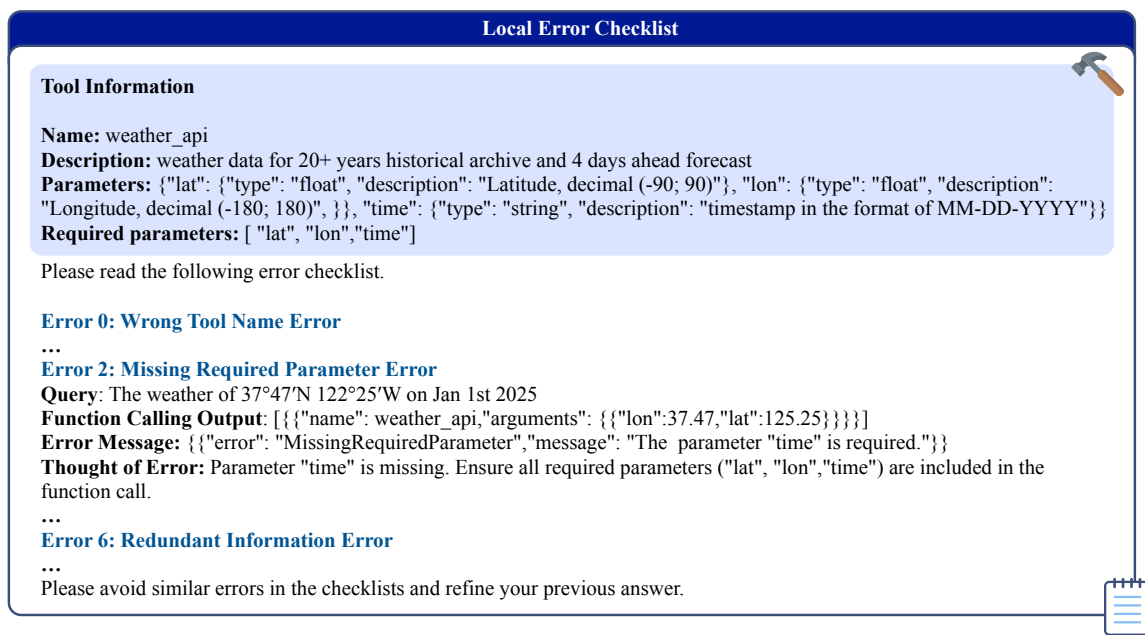


Figure 2: The Local Error Checklist: a list of tool-specific issues that may arise during tool calling

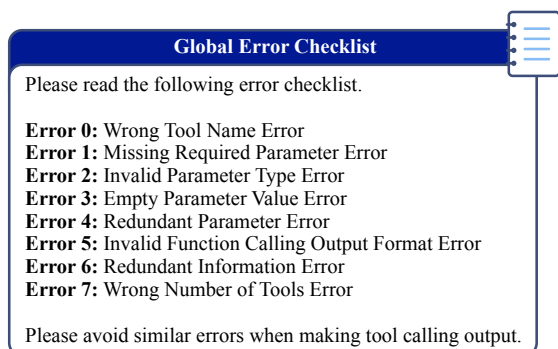


Figure 3: The Global Error Checklist: a list of common issues that may arise during tool calling

ing LLMs specifically for tool calling tasks, they require high-quality training data or extensive tool interaction logs, which are still costly to obtain.

3 Method

In this section, we introduce the novel Hierarchical Tool Error Checklist (HiTEC) framework, comprising global and local error checklists to address parameter mis-filling in tool learning. We propose both tuning-free (HiTEC-ICL) and tuning-based (HiTEC-KTO) methods to integrate HiTEC into LLMs, enabling dynamic error reflection and correction based on pre-identified patterns and tool-specific information.

3.1 Tool Error Checklist

3.1.1 Global Error Checklist

The global error checklist is a list of common issues that arise during tool calling, and they are not associated with a specific query or tool. It is de-

signed to address the most prevalent and impactful issues encountered during tool calling. Drawing from prior experiences with model-tool interactions, eight errors are selected to represent typical failure modes occurring at various stages of tool interaction, encompassing both tool-level and parameter-level mistakes.

At the tool level, we check errors *Wrong Tool Name Error* (Error 0) and *Wrong Number of Tools Error* (Error 7) highlight issues related to improper tool selection or usage. The parameter level encompasses a range of potential inaccuracies, including *Missing Required Parameter Error* (Error 1), *Invalid Parameter Type Error* (Error 2), and *Empty Parameter Value Error* (Error 3), and *Redundant Parameter Error* (Error 4). In addition, the global error checklist also incorporates errors related to redundancy. We check *Redundant Information Error* (Error 6), which ensures that the output remains concise and relevant, and *Invalid Function Calling Output Format Error* (Error 5) guards against syntactical inconsistencies that could disrupt downstream processes. This carefully constructed checklist of errors provides a robust foundation for systematic error identification and resolution in diverse tool-calling scenarios, significantly enhancing the reliability and efficiency of model-tool interactions. The global error checklist is presented in Figure 3.

3.1.2 Local Error Checklist

The local error checklist identifies errors related to the specific features of each tool. This checklist is crucial for addressing tool-specific issues,

as it offers detailed information that goes beyond the general overview provided by the global error checklist. Unlike the global error checklist, which only lists an overview of common errors, the local checklist focuses on the unique functionalities, parameters, and requirements of each tool. The local error checklist is essential for addressing tool-specific issues that may not be captured by the global checklist. A tailored local checklist ensures these specificities are considered.

The local error checklist of a tool contains the following components: tool information, simulated queries for each error type, the function calling output that can invoke the corresponding error, an error message describing the error, and a Thought of Error reflection that specifies how such an error can be corrected. An example of a local error checklist is illustrated in Figure 2.

To form such a local error checklist, it is required that tool-related information be known in advance. This includes details such as those outlined in the "Tool Information" section of the Error Checklist. This information should be appropriately placed within the `<tool_info>` field of the generation prompt as presented in Appendix A.

3.2 HiTEC-ICL: Enhancing LLM Tool Calling in Tuning-free Way

We integrate the designed global and local error checklists into the LLM-based tool-calling conversation to ensure precise and reliable tool utilization.

The global error checklist is embedded within the user’s initial query at the outset of the inference process. This proactive integration helps preempt common issues, such as tool name misidentification or parameter omission. By implementing these error prevention mechanisms early in the process, the system significantly enhances the accuracy and reliability of the initial tool invocation.

The local error checklist is primarily designed for parameter-specific error handling. Figure 4 illustrates this two-round interaction process. The first round captures the initial tool call, while the second round, guided by the local error checklist, ensures corrections are made to parameter filling and other tool-specific issues, leading to improved inference outcomes.

3.3 HiTEC-KTO: Enhancing LLM Tool Calling in Tuning-based Way

To empower open-source language models with robust tool-calling capabilities, we propose a method

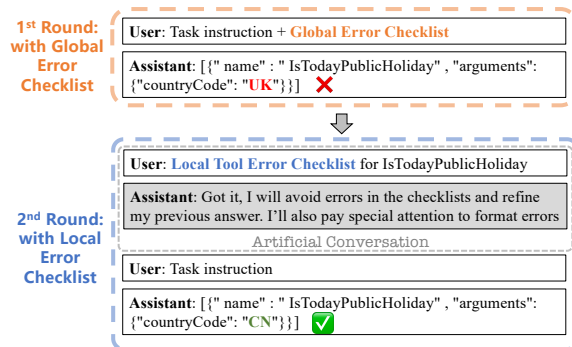


Figure 4: An Example Conversation with the Global-Local Checklist as ICL Prompting

that utilizes error checklists to generate high-quality negative examples. This approach aims to enhance the performance of smaller, efficient models, enabling them to rival larger, more resource-intensive counterparts. By curating negative examples based on predefined global and local error checklists, we fine-tune open-source models, thereby equipping them with improved function-calling accuracy and error-handling capabilities.

3.3.1 Negative Example Generation

Negative examples are generated by combining correctly formatted tool-calling outputs with the corresponding local error checklist. By prompting the model to introduce specific errors described in the checklist, we create outputs that simulate real-world failures in tool-calling tasks. This curated dataset enables the fine-tuning of models to better recognize, avoid, and correct errors, ultimately improving their reliability and alignment with intended behaviors. We name the generated pairwise tool-calling dataset as PTC. The prompt for negative sample generation is presented in Appendix A.

3.3.2 Failure Mode of DPO on PTC Dataset

Given that the PTC dataset consists of pairwise data, an intuitive choice for tuning would be Direct Preference Optimization (DPO) (Rafailov et al., 2024) as the tuning method. However, applying DPO to such a dataset leads to a failure mode, as identified in previous research (Feng et al., 2024; Pal et al., 2024). This issue arises because the positive and negative responses in the PTC dataset differ by only a few tokens (more details are in Appendix D). Consequently, two problems occur: 1) the gradient of the DPO loss approaches zero, leading to a weak update signal, and 2) during optimization, the probability of the correct token tends

to decrease when compared to the reference model (i.e., the initial model being tuned). We empirically validate the gradient vanish phenomenon by plotting the gradient norm during the training of DPO on the PTC dataset (for more details, refer to Figure 8(b) in Appendix D). Further, by plotting the log probabilities for both positive and negative samples (see Figure 8(c) in Appendix D), we observe that although the log probabilities of negative samples decrease, the log probabilities of positive samples also decrease, demonstrating the second challenge.

3.3.3 HiTEC-KTO

One representative DPO variant is Kahneman-Tversky Optimization (KTO) (Ethayarajh et al., 2024). We refer to the approach of fine-tuning large language models (LLMs) on a PTC-type dataset using KTO as HiTEC-KTO. In the following, we show the potential that KTO can address the above failure mode of DPO.

We formulate the KTO loss in a paired format:

$$\begin{aligned} \mathcal{L}_{KTO}(x, y_w, y_l; \theta) \\ = \lambda_w - \lambda_w \sigma(\beta(r_\theta(x, y_w) - z_0)) \\ + \lambda_l - \lambda_l \sigma(\beta(z_0 - r_\theta(x, y_l))), \end{aligned} \quad (1)$$

where (x, y_w, y_l) are paired data with x as the prompt, y_w as the positive answer and y_l as the negative answer. $r_\theta(x, y)$ is the log-ratio of the likelihoods of answer y between the training model $\pi_\theta(y|x)$ and the reference model $\pi_{\text{ref}}(y|x)$, where θ is the model parameter. $r_\theta(x, y)$ is denoted as $r_\theta(x, y) = \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$. z_0 is the reference point, and $z_0 = KL(\pi_\theta || \pi_{\text{ref}})$. Since z_0 does not propagate gradients (Ethayarajh et al., 2024), it is treated as a constant during the differentiation of \mathcal{L}_{KTO} .

Suppose y_w and y_l are length K sequences, and they only differ at i -th token, i.e., $y_w = [t_1, \dots, t_{i-1}, t_i^w, t_{i+1}, \dots, t_K]$, and $y_l = [t_1, \dots, t_{i-1}, t_i^l, t_{i+1}, \dots, t_K]$. Using the derivative perspective, similar to the approach in the prior section, we theoretically analyze KTO’s capability to mitigate the failure mode of DPO. The derivative of the KTO loss with respect to θ is:

$$\begin{aligned} \nabla_\theta \mathcal{L}_{KTO}(x, y_w, y_l; \theta) \\ = -a_w \nabla_\theta \log \pi_\theta(y_w|x) + a_l \nabla_\theta \log \pi_\theta(y_l|x), \end{aligned} \quad (2)$$

where the asymmetric weights are $a_w = \lambda_w \sigma(c_w) \sigma(1 - c_w)$, and $a_l = \lambda_l \sigma(c_l) \sigma(1 - c_l)$, with $c_w = \beta\left(\log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - z_0\right)$ and $c_l =$

$\beta\left(\log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} - z_0\right)$. Compared with DPO, these asymmetric weights provide stable gradients, guiding the model in adjusting the probability distribution effectively.

Moreover, to further investigate how KTO promotes probability shifts towards favorable outcomes during training, following (Pal et al., 2024) we re-arrange the derivative of the KTO loss as:

$$\begin{aligned} \nabla_\theta \mathcal{L}_{KTO}(x, y_w, y_l; \theta) \\ = \sum_{k=1}^K \nabla_\theta \left[-a_w \log \pi_\theta(t_k | y_w^{<k}, x) + a_l \log \pi_\theta(t_k | y_l^{<k}, x) \right]. \end{aligned} \quad (3)$$

With the above formula, each term in the derivative expression Eqn. (3) becomes:

$$\begin{aligned} \nabla_{g_j} \left[-a_w \log \pi_\theta(t_k | y_w^{<k}, x) + a_l \log \pi_\theta(t_k | y_l^{<k}, x) \right] \\ = \begin{cases} -a_w + a_l + a_w s_j^{y_w^{<k}, x} - a_l s_j^{y_l^{<k}, x} & t_k = V_j; \\ a_w s_j^{y_w^{<k}, x} - a_l s_j^{y_l^{<k}, x} & t_k \neq V_j. \end{cases} \end{aligned}$$

This analysis reveals that, given that $a_w > a_l$, it is evident that when the k -th token in y_w matches V_j , the gradient is negative, whereas, for other vocabulary elements ($t_k \neq V_j$), the gradient is positive. Consequently, minimizing the KTO loss encourages an increase in the logits of the correct token, thereby effectively addressing DPO’s second issue as discussed in Section 3.3.2.

4 Experiment

Table 1: Dataset Statistics

Dataset	# Queries	# Tools	Multi-tool?
API-Bank L-1	399	49	✗
API-Bank L-2	127	28	✗
Tool-Alpaca	114	41	✓
Seal-Tools	294	1084	✗
Nexus Raven	318	65	✗

4.1 Experiment Setup

We here describe the datasets, metrics, and baselines used in experiments.

Datasets. We conduct an evaluation on several benchmark tool calling datasets: API-Bank (Li et al., 2023), Tool Alpaca (Tang et al., 2023), Seal-Tools (Wu et al., 2024), and Nexus Raven (Srinivasan et al., 2023). All datasets are preprocessed in the same way as Lin et al. (Lin et al., 2024). The statistics of the processed datasets are summarized in Table 1.

Metrics. We evaluate two performance metrics in the form of F1 score: correctness of tool names

Table 2: Tool Calling Performance with HiTEC-ICL, with the best performance marked in **bold** and the proposed approach highlighted in blue.

		Dataset (F1 Name F1 Name + Parameter)										F1 Average	
Model	Method	API-Bank L-1		API-Bank L-2		Tool-Alpaca		Seal-Tools (Single-Tool)		Nexus Raven		Name	Name+Param.
GPT-4-Turbo	Vanilla	94.77	86.66	71.09	70.43	87.12	60.75	94.86	88.55	94.13	82.92	88.39	77.86
	+ CoT	91.70	84.84	67.86	65.74	80.95	56.46	91.50	86.07	92.18	85.00	84.84	75.62
	+ Function Calling	84.03	81.40	78.49	62.32	87.11	62.16	84.94	84.88	\	\	\	\
	+ HiTEC-ICL	94.96	88.44	71.21	72.68	87.35	61.95	96.00	89.37	92.81	83.15	88.47	79.12
Llama3-8B	Vanilla	69.19	67.73	75.54	48.91	40.46	24.86	86.54	82.51	14.83	17.44	57.31	48.29
	+ CoT	43.21	45.05	52.31	45.65	60.58	42.07	76.89	75.16	64.44	52.22	59.49	52.03
	+ HiTEC-ICL	75.93	70.55	77.30	56.01	85.92	55.49	94.40	87.27	82.07	58.30	83.12	65.52
Llama3.1-8B	Vanilla	73.06	63.62	78.23	50.25	84.06	50.79	94.97	86.53	87.93	72.17	83.65	64.67
	+ CoT	65.22	61.61	75.54	48.52	63.31	42.57	88.55	80.36	76.71	67.32	73.87	60.08
	+ HiTEC-ICL	75.07	64.61	82.23	51.29	85.92	55.94	96.95	87.94	87.76	72.35	85.59	66.42

(F1 Name) and correctness of tool names + tool parameters (F1 Name + Parameter).

Baselines. We use GPT-3.5-Turbo-0125, GPT-4-Turbo (GPT series), Llama3-8B, Llama3-70B (Llama3 series), Llama3.1-8B, Llama3.1-70 (Llama3.1-series) models as base models to study the effect of HiTEC-ICL. The vanilla version of the base models, base models deployed with zero-shot CoT (Wei et al., 2022), and native tool calling integrated base models² (only GPT-series) are used as baselines. For HiTEC-KTO, we use Llama3-8B, Llama3-70B (Llama3 series), Llama3.1-8B, Llama3.1-70 (Llama3.1-series), Qwen2.5-0.5B, Qwen2.5-1.5B, Qwen2.5-3B and Qwen2.5-7B (Qwen2.5 series) as base models, and consider the vanilla base models of Llama series and Hammer2.0 release³ (which is tuned on Qwen2.5 series (Lin et al., 2024)) as baselines. The implementation details are in Appendix B.

4.2 Main Results

4.2.1 Effectiveness of HiTEC-ICL

We now evaluate HiTEC’s efficacy across datasets and base models in ICL setting. The method consistently enhances tool-calling performance as shown in Table 2. Experiments under more settings are deferred to Appendix C due to limited space.

HiTEC-ICL demonstrates robust performance improvements across diverse tool-learning benchmarks. Model capability critically influences HiTEC-ICL’s efficacy. Smaller models (Llama3-8B) achieve the most pronounced gains, with Name + Parameter improving by over 30 points on Nexus Raven (58.30 vs. 17.44), indicating that explicit

error guidance compensates for limited reasoning capacity. Larger models (GPT-4-Turbo, Llama3.1-70B) exhibit subtler but consistent improvements, leveraging checklists to refine already strong baseline performance. Notably, HiTEC-ICL outperforms CoT and vanilla tool calling in most cases, with the largest margins on Seal-Tools validating its structured error mitigation approach.

4.2.2 Effectiveness of HiTEC-KTO

KTO Training Data. We generate negative samples based on the xlam-function-calling-60k dataset (Liu et al.). One incorrect tool calling answer is generated for each of the queries in the xlam-function-calling-60k dataset. We label the incorrect answer as "False" and the original groundtruth as "True" and perform KTO on the 12,000 samples. Table 3 demonstrates the experiment results over various of open-sourced models. Results under more settings are deferred to Appendix C due to limited space.

It can be found that HiTEC-KTO significantly enhances tool-calling capabilities across open-source models by leveraging error checklists to generate targeted negative examples for preference-based fine-tuning. Crucially, HiTEC-KTO enables smaller models to rival larger counterparts—the 1.5B Qwen2.5 model surpasses the 3B baseline Hammer in many cases after being fine-tuned with HiTECH-KTO. Performance trends also correlate with dataset complexity. On multi-tool calling dataset Tool Alpaca, HiTEC-KTO boosts parameter accuracy by 6-25 points on F1 across 8B Llama series, demonstrating its efficacy in resolving tool-specific ambiguities. It also provides 8B Llama series with a 4-10 points performance increase on the Nexus Raven dataset, which has complex and long queries.

²<https://platform.openai.com/docs/guides/function-calling>

³<https://huggingface.co/collections/MadeAgents/hammer20-66f4dee539f7b2c95224012a>

Table 3: Tool Calling Performance with HiTEC-KTO, with the best performance marked in **bold** and the proposed approach highlighted in **blue**

		Dataset (F1 Name F1 Name + Parameter)										F1 Average		
Model Series	Method	Model	API-Bank L-1		API-Bank L-2		Tool-Alpaca		Seal-Tools (Single-Tool)		Nexus Raven		Name	Name + Param.
Llama3.1 Series	Baseline	Llama3.1-8B	73.06	63.62	78.23	50.25	84.06	50.79	94.97	86.53	87.93	72.17	83.65	64.67
		Llama3.1-70B	90.15	76.42	80.34	62.20	86.03	55.83	97.42	86.19	93.75	82.72	89.54	72.67
	HiTEC-KTO	Llama3.1-8B	87.47	80.99	85.61	61.02	84.18	56.45	94.44	86.18	89.98	82.28	88.34	73.38
		Llama3-70B	90.78	77.07	86.44	65.14	86.67	57.32	98.14	90.01	94.84	82.87	91.37	74.48
Qwen2.5 Series	Baseline	Hammer2-0.5B	71.20	59.03	43.32	38.22	64.46	41.83	93.86	83.03	64.72	45.52	67.51	53.53
		Hammer2-1.5B	88.63	79.26	80.51	62.82	80.74	51.88	96.10	87.16	85.85	63.76	86.37	68.98
		Hammer2-3B	88.63	79.04	77.11	57.58	78.23	53.28	93.08	85.60	89.14	66.71	85.24	68.44
		Hammer2-7B	88.91	81.28	75.96	58.36	81.74	57.07	94.62	87.84	90.76	80.96	86.40	73.10
	HiTEC-KTO	Qwen2.5-0.5B	88.29	78.34	81.76	54.52	73.54	38.76	96.80	88.18	82.83	63.89	84.64	64.74
		Qwen2.5-1.5B	88.92	79.46	81.36	60.66	82.27	52.26	96.99	89.75	86.37	64.40	87.18	69.31
		Qwen2.5-3B	89.07	80.16	85.51	54.61	85.42	56.08	96.28	89.51	83.05	62.35	87.87	68.54
		Qwen2.5-7B	89.38	81.67	88.00	59.93	87.63	59.19	96.27	89.29	91.70	78.17	90.60	73.65

Table 4: Ablation Study on the Hierarchical Error Checklist of HiTEC-KTO

Base Model (F1 Name F1 Name + Parameter)							
Dataset	Method	Qwen2.5 1.5B	Qwen2.5 3B	Average			
Tool Alpaca	HiTEC-KTO	82.27	52.26	85.42	56.08	83.30	53.30
	w/o Local EC	60.61	28.97	77.62	51.61	69.71	41.08
	w/o Glb-loc EC	22.83	16.03	73.45	47.75	46.38	31.40
Seal-Tools	HiTEC-KTO	96.99	89.75	96.28	89.51	96.62	89.48
	w/o Local EC	87.75	68.99	92.75	82.60	90.31	76.85
	w/o Glb-loc EC	41.86	37.34	83.98	77.99	60.01	55.29

These results underscore HiTEC-KTO’s ability to democratize advanced tool-learning capabilities across model sizes and dataset complexity, balancing error avoidance with functional precision through structured checklist-driven optimization.

4.3 Ablation Study

To assess the impact of HiTEC’s hierarchical design, we conducted an ablation study comparing configurations that utilize the Global Error Checklist (Glb EC) versus the combined Global-Local Error Checklist (Glbloc EC) across two datasets—Tool Alpaca and Seal-Tools—as well as multiple base models. The results are presented in Table 4 and Appendix C due to limited space. For HiTEC-KTO variants, w/o Local EC indicates that KTO was performed on negative samples generated solely with the global error checklist, omitting the local error checklist. w/o Glbloc EC refers to the baseline model without hierarchical error correction.

HiTEC-KTO significantly outperforms w/o Glbloc EC, particularly in smaller models. Notably, HiTEC Qwen2.5-1.5B achieves performance com-

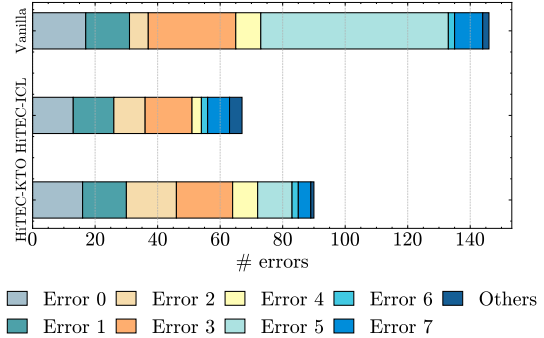


Figure 5: Analysis on Error Distribution

parable to its vanilla counterpart (w/o Glb-loc EC). Removing the local error checklist has a substantial negative impact on HiTEC-KTO, as global errors alone capture only general mistakes, whereas generating high-quality negative examples requires tool-specific error instructions.

As shown in the table, incorporating the local error checklist generally improves accuracy for both tool name identification and parameter filling compared to using only the global checklist. This improvement is primarily attributed to more precise parameter filling, as evidenced by the greater increase in the F1 Name + Parameter metric (second column) compared to F1 Name (first column). For instance, on Tool Alpaca with Qwen2.5-1.5B, F1 Name + Parameter relatively improves by 86%, whereas the relative improvement of F1 Name is only 37%. This result highlights the effectiveness of hierarchical checklists in addressing tool-specific local errors.

In conclusion, HiTEC’s hierarchical structure effectively mitigates tool-specific errors, particularly enhancing parameter accuracy in smaller models.

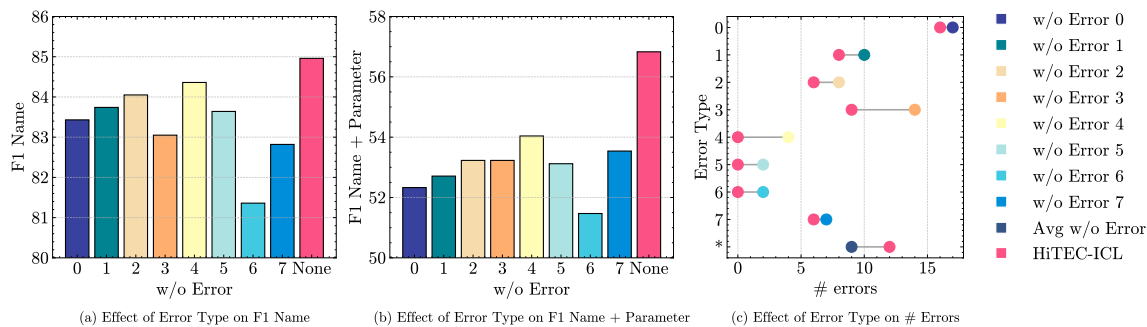


Figure 6: Effect of Error Types

4.4 Look Into the Errors

In this subsection, we further look into errors and conduct error-level evaluations.

4.4.1 Analysis on Error Distribution

We analyze error type distributions across the vanilla base model, HiTEC-ICL, and HiTEC-KTO configurations on Tool Alpaca (Llama3-8B) to quantify the framework’s impact. Since multiple errors can co-exist in a single response, we count all occurrences. The results are shown in 5.

In vanilla setting, parameter-level errors (Error 1–4) and format errors (Error 5) account for the majority of failures. Formatting errors (Error 5), in particular, are prevalent, reflecting baseline weaknesses in syntactical consistency. However, with HiTEC, the percentage of formatting errors drops significantly, demonstrating the global checklist’s effectiveness in enforcing output structure.

Parameter-filling errors (Error 1–4) also undergo a notable redistribution. In the vanilla configuration, Empty Parameter Value Error (Error 3) is the most frequent, but under HiTEC, the distribution becomes more balanced, with no single error type dominating. This indicates that the hierarchical error checks address a broader range of parameter-related issues, reducing reliance on any single mitigation strategy.

4.4.2 Effect of Error Types

To evaluate the contribution of individual error types in the HiTEC framework, we conduct an experiment in which each error type is iteratively excluded from the HiTEC, and the performance of HiTEC-ICL is assessed. The results for HiTEC-ICL-Llama3.1-70B on the Tool Alpaca dataset are presented in Figure 7, while additional results under different settings can be found in Appendix C due to limited space.

As illustrated in Figure 7 (a-b), the exclusion of tool-level errors (e.g., Errors 0, 6, and 7) significantly reduces performance, highlighting their

critical role in the multi-tool selection feature of the Tool Alpaca dataset. In contrast, parameter-centric errors primarily affect the Seal-Tools dataset (see Appendix C). Notably, the removal of Errors 3 and 4 results in a substantial performance decline across multiple base models. Furthermore, Error 5 is found to be universally essential, as its presence ensures the generation of parsable outputs, which are crucial for accurate tool calling.

Additionally, we examine changes in the number of errors (# errors) when specific error types are removed. Figure 7 (c) demonstrates that the number of errors in categories 0-7 increases when the corresponding error type is omitted from the checklist. This observation confirms that a defined error in HiTEC can help reduce such a type of error. The symbol * denotes other types of errors, while "Avg w/o Error" represents the average number of errors within the "Others" category of the w/o variants. HiTEC-ICL exhibits a slight increase in errors within the "Others" category compared to its variants, which is a reasonable outcome given that the removed errors may contribute to new, previously unclassified errors.

4.5 Cost Analysis

In this section, we further conduct an evaluation of HiTEC-ICL’s inference overhead. Specifically, we measure the number of prompt tokens and generated tokens on the Tool-Alpaca dataset, comparing HiTEC-ICL against the vanilla model, CoT, and HiTEC-ICL with only a global checklist. Additionally, we assessed the cost per query in US dollars for GPT-based models and execution time for Llama-based models. All experiments were conducted on four NVIDIA A100 GPUs. The results are shown in Table 5 and Table 6.

The results indicate that while HiTEC-ICL incurs the highest cost, the absolute expense remains within a reasonable range. Notably, the substantial performance gains observed on Llama-based models justify the increase in computational cost.

Table 5: Cost Analysis of HiTEC-ICL on Closed-sourced Models

Model	Method	Prompt Tokens	Generated Tokens	Cost per Case (\$)	F1 Name + Param.
GPT-3.5-Turbo-0125	Vanilla	77,374	3,759	0.0004	56.47
	CoT	82,048	15,401	0.0006	58.43
	HiTEC-ICL (w/o local)	91,510	3,747	0.0005	57.98
	HiTEC-ICL	320,830	7,128	0.0015	58.79
GPT-4-Turbo	Vanilla	77,374	3,678	0.0078	60.75
	CoT	81,706	25,429	0.0127	56.46
	HiTEC-ICL (w/o local)	103,252	3,680	0.0100	60.67
	HiTEC-ICL	321,816	7,344	0.0302	61.95

Table 6: Cost Analysis of HiTEC-ICL on Open-sourced Models

Model	Method	Prompt Tokens	Generated Tokens	Cost per Case (second)	F1 Name + Param.
Llama3-8B	Vanilla	81,260	3,786	0.0360	24.86
	CoT	102,997	21,191	0.0701	42.07
	HiTEC-ICL (w/o local)	107,005	3,653	0.0454	52.39
	HiTEC-ICL	323,392	7,532	0.1320	55.49
Llama3.1-8B	Vanilla	81,670	4,196	0.0368	50.79
	CoT	108,503	26,355	0.0799	42.57
	HiTEC-ICL (w/o local)	107,534	4,182	0.0464	52.33
	HiTEC-ICL	331,671	8,800	0.1370	55.94

Moreover, HiTEC-ICL (w/o local) strikes a strong balance between efficiency and effectiveness, offering lower costs while maintaining superior performance compared to the baselines.

4.6 Study on Multi-turn/step Tool Calling

To further evaluate HiTEC’s capability in handling interactive tool use, we design experiments to simulate multi-turn/step tool calling. In this setup, the responses from previous tool calls are treated as predefined contextual information for subsequent calls. We incorporate such intermediate responses into artificial conversations within our proposed framework, prompting the model to generate the next API call accordingly.

To ensure a comprehensive evaluation, we filtered multi-turn/step tool queries from the Seal-Tools dataset (Wu et al., 2024) and conducted experiments on HiTEC-ICL and HiTEC-KTO. The results, presented in Table 7 and 8, demonstrate that while multi-turn tool use poses greater challenges, our approach consistently outperforms the baselines in most cases, highlighting its effectiveness in iterative tool interactions.

5 Conclusion

In this work, we introduced the Hierarchical Tool Error Checklist (HiTEC) framework to en-

Table 7: Tool Calling Performance with HiTEC-ICL on Seal-Tools Multi-turn/step

Method	Model	F1 Name	F1 Name + Param.
Llama 3-8B	Vanilla	8.24	0.00
	+CoT	8.42	5.40
	+HiTEC-ICL	43.01	10.21
Llama3.1-8B	Vanilla	47.51	10.77
	+CoT	43.66	9.86
	+HiTEC-ICL	57.14	15.87

Table 8: Tool Calling Performance with HiTEC-KTO on Seal-Tools Multi-turn/step

Method	Model	F1 Name	F1 Name + Param.
Baseline	Llama3.1-8B	47.51	10.77
	Llama3.1-70B	51.23	16.76
HiTEC-KTO	Llama3.1-8B	67.15	14.09
	Llama3.1-70B	70.31	18.81
Baseline	Hammer2-0.5B	31.78	4.65
	Hammer2-1.5B	47.70	9.41
	Hammer2-3B	32.09	5.55
	Hammer2-7B	43.03	17.24
HiTEC-KTO	Qwen2.5-0.5B	35.95	8.35
	Qwen2.5-1.5B	44.70	11.15
	Qwen2.5-3B	54.67	14.00
	Qwen2.5-7B	58.15	14.65

hance tool calling in large language models by systematically identifying and addressing common and tool-specific errors. By integrating both global and local error checklists, HiTEC effectively mitigates parameter mis-filling and format inconsistencies. We proposed two complementary approaches—HiTEC-ICL and HiTEC-KTO—to incorporate error feedback into the tool-calling process. HiTEC-ICL leverages structured error guidance during prompt formulation, while HiTEC-KTO utilizes targeted negative examples to fine-tune open-source models. Extensive experiments on multiple benchmark datasets demonstrate that both methods significantly improve tool name identification and parameter accuracy compared to existing baselines, with particularly notable gains in smaller models.

Acknowledgments

The research work described in this paper was supported by Hong Kong Research Grants Council (grant# 16202722, T22-607/24-N, T43-513/23N-1). It was partially conducted in JC STEM Lab of Data Science Foundations funded by The Hong Kong Jockey Club Charities Trust.

Limitation

While our approach shows promising improvements, it has some limitations. The effectiveness of HiTEC partially attributes to the comprehensiveness of the manually crafted error types and templates, which may not capture all novel or unforeseen errors in dynamic tool environments. Additionally, the reliance on simulated error feedback may not fully reflect real-world scenarios, potentially limiting the framework’s generalizability and scalability in diverse applications.

To address these limitations, potential improvements could focus on dynamically updating the error checklists based on real-world feedback. By integrating mechanisms for continuous monitoring and analysis of actual tool interactions, the system could iteratively refine the types of errors included in the checklists. This iterative improvement would allow the framework to adapt to evolving tool behaviors and emerging error patterns, ultimately enhancing its robustness and precision in tool calling. Results in Section 4.4.2 can also verify effectiveness of such an extension. Additionally, incorporating automated error detection and leveraging reinforcement or continual learning strategies may further optimize the system’s performance in diverse and dynamic environments.

References

- Sijia Chen, Yibo Wang, Yi-Feng Wu, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. 2024a. Advancing tool-augmented large language models: Integrating insights from errors in inference trees. *arXiv preprint arXiv:2406.07115*.
- Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammadhossein Bateni, Chen-Yu Lee, and Tomas Pfister. 2024b. Re-invoke: Tool invocation rewriting for zero-shot tool retrieval. *arXiv preprint arXiv:2408.01875*.
- Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*.
- Duanyu Feng, Bowen Qin, Chen Huang, Zheng Zhang, and Wenqiang Lei. 2024. Towards analyzing and understanding the limitations of dpo: A theoretical perspective. *arXiv preprint arXiv:2404.04626*.
- Jiayi Fu, Xuandong Zhao, Chengyuan Yao, Heng Wang, Qi Han, and Yanghua Xiao. 2025. Reward shaping to mitigate reward hacking in rlhf. *arXiv preprint arXiv:2502.18770*.
- Leo Gao, John Schulman, and Jacob Hilton. 2023. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pages 10835–10866. PMLR.
- Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2023. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*.
- 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*.
- Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. 2023. Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings. *Advances in neural information processing systems*, 36:45870–45894.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.
- Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*.
- Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, et al. 2024. Hammer: Robust function-calling for on-device language models via function masking. *arXiv preprint arXiv:2410.04587*.
- Yanming Liu, Xinyue Peng, Yuwei Zhang, Jiannan Cao, Xuhong Zhang, Sheng Cheng, Xun Wang, Jianwei Yin, and Tianyu Du. 2024. Tool-planner: Dynamic solution tree planning for large language model with tool clustering. *arXiv preprint arXiv:2406.03807*.
- Zuxin Liu, Thai Quoc Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
- Arka Pal, Deep Karkhanis, Samuel Dooley, Manley Roberts, Siddhartha Naidu, and Colin White. 2024. Smaug: Fixing failure modes of preference optimisation with dpo-positive. *arXiv preprint arXiv:2402.13228*.
- 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 *The*

- Twelfth International Conference on Learning Representations*.
- Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.
- Zhengliang Shi, Shen Gao, Xiuyi Chen, Yue Feng, Lingyong Yan, Haibo Shi, Dawei Yin, Pengjie Ren, Suzan Verberne, and Zhaochun Ren. 2024. Learning to use tools via cooperative and interactive agents. *arXiv preprint arXiv:2403.03031*.
- Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, et al. 2023. Restgpt: Connecting large language models with real-world restful apis. *arXiv preprint arXiv:2306.06624*.
- Venkat Krishna Srinivasan, Zhen Dong, Banghua Zhu, Brian Yu, Damon Mosk-Aoyama, Kurt Keutzer, Jiantao Jiao, and Jian Zhang. 2023. Nexusraven: a commercially-permissive language model for function calling. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.
- Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.
- Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024. LLMs in the imagination: tool learning through simulated trial and error. *arXiv preprint arXiv:2403.04746*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Mengsong Wu, Tong Zhu, Han Han, Chuanyuan Tan, Xiang Zhang, and Wenliang Chen. 2024. Seal-tools: self-instruct tool learning dataset for agent tuning and detailed benchmark. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 372–384. Springer.
- Hongshen Xu, Zichen Zhu, Lei Pan, Zihan Wang, Su Zhu, Da Ma, Ruisheng Cao, Lu Chen, and Kai Yu. 2024. Reducing tool hallucination via reliability alignment. *arXiv preprint arXiv:2412.04141*.
- Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2024. Gpt4tools: Teaching large language model to use tools via self-instruction. *Advances in Neural Information Processing Systems*, 36.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.
- Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Kan Ren, Dongsheng Li, and Deqing Yang. 2024. Easytool: Enhancing llm-based agents with concise tool instruction. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.
- Yinger Zhang, Hui Cai, Xeirui Song, Yicheng Chen, Rui Sun, and Jing Zheng. 2023. Reverse chain: A generic-rule for llms to master multi-api planning. *arXiv preprint arXiv:2310.04474*.

A Prompt

Local Error Checklist Generation Prompt

Task:

You are given information about a tool and an example template of an error checklist. Your task is to generate an error checklist for the tool in the same format as the template. More specifically, for each error, you should:

- Provide a **perfect query**. The query should be self-contained and contain all the necessary information for a correct tool call. For example: "Can you verify the access to the database named 'customer_data'?"
- Provide the **corresponding answer** from the model that evokes the error.
- Provide an **error message** that describes what went wrong.
- Provide a **thought** explaining how the error should be corrected.

Note: You should strictly follow the format of the template.

Error Checklist Template

Tool Information

name: 'name_of_the_tool'

description: 'description_of_the_tool'

parameters: {"parameter_name_1": {"type": "type_1", "description": "description_of_the_parameter"}, "parameter_2": {"type": "type_2", "description": "description_of_the_parameter"}} required parameters: ["parameter_1"] (Include other relevant information about the tool if necessary.)

Error 2: Missing Required Parameter Error

Query: "a_query_that_calls_the_tool"

Function Calling Output: [{"name": "name_of_the_tool", "arguments": {"parameter_2": "parameter_value"}}]

Error Message: {"error": "MissingRequiredParameter", "message": "The 'parameter_1' parameter is required."}

Thought of Error: Parameter 'parameter_1' is missing. Ensure all required parameters ('parameter_1') are included in the function call.

Error 3: Invalid Parameter Type Error

Query: "a_query_that_calls_the_tool"

Function Calling Output: [{"name": "name_of_the_tool", "arguments": {"parameter_1": "parameter_value", "parameter_2": "parameter_value (but not of type_2)"}}]

Error Message: {"error": "InvalidParameterType", "message": "The 'parameter_2' is not of 'type_2'."}

Thought of Error: Parameter 'parameter_2' should be of type 'type_2', but an invalid type was provided. Ensure all parameters match their expected types.

Error 4: Empty Parameter Value Error

Query: "a_query_that_calls_the_tool"

Function Calling Output: [{"name": "name_of_the_tool", "arguments": {"parameter_1": "parameter_value", "parameter_2": ""}}]

Error Message: {"error": "EmptyParameterValue", "message": "The 'parameter_2' parameter cannot be empty."}

Local Error Checklist Generation Prompt (cont'd)

Thought of Error: Parameter 'parameter_2' has an empty value. It should not be empty as specified by the tool's requirements.

Error 5: Redundant Parameter Error

Query: "a_query_that_calls_the_tool (that only needs to fill in part of the parameters of the tool)"

Function Calling Output: [{ { "name": "name_of_the_tool", "arguments": { { "parameter_1": "parameter_value", "parameter_2": "parameter_value" } } } }]

Error Message: { { "error": "RedundantParameter", "message": "The parameter 'parameter_2' is not indicated by the query and should not be called." } }

Thought of Error: Parameter 'parameter_2' is unnecessary and was not specified in the query. Ensure only the required and specified parameters are included in the function call.

Error 6: Invalid Function Calling Output Format Error

Query: "a_query_that_calls_the_tool"

Function Calling Output: { { "Name": "name_of_the_tool", "Parameter": { { "parameter_1": "parameter_value", "parameter_2": "parameter_value" } } } }

Error Message: { { "error": "InvalidFormat", "message": "The function calling output does not follow the required format and cannot be parsed." } }

Thought of Error: The output format is incorrect due to improperly formatted keys and symbols. The correct function calling output should be: [{ { "name": "func_name1", "arguments": { { "parameter_1": "value1", "parameter_2": "value2" } } } }]

Error 7: Redundant Information Error

Query: "a_query_that_calls_the_tool"

Function Calling Output: "Based on the query, I will make a function call to the 'name_of_the_tool' tool to get the query answered. Here is the output in the required JSON format: [{ { 'name': 'name_of_the_tool', 'arguments': { { 'parameter_1': 'parameter_value', 'parameter_2': 'parameter_value' } } } }]"

Error Message: "error": "RedundantInformationError", "message": "The function calling output contains redundant text such as 'Based on the query, I will make a function call...' which is unnecessary."

Thought of Error: No additional text should be included in the output. The correct function calling output should only contain: [{ { "name": "func_name1", "arguments": { { "parameter_1": "value1", "parameter_2": "value2" } } } }]

Instructions

Now, generate an error checklist for the following tool:

<tool_info>

Note: You must strictly follow the format of the template.

Negative Sample Generation Prompt

System Prompt: You are provided with an error checklist, a tool calling query and its groundtruth answer.

The error checklist of an example tool is as follows:

****Error Checklist Template****

Tool Information

name: 'name_of_the_tool'

(the same as the tool template in Local Error Checklist Generation Prompt)

Your task is to modify the groundtruth tool calling so that it fits one of the errors in the error checklist. For the Redundant Parameter Error, your generated redundant parameter should be one of the parameters in the tool information. If there is no extra parameter that can be chosen for Redundant Parameter Error, you can choose another errors.

Note: DO NOT include not-exist parameters in your response, e.g., "extra_param".

Note: You should return a modified response, for example: [{"name": "getSocialEnterprise-Info", "arguments": {"enterprise_name": "CommunityGrowth"}}].

Note: Just provide the modified function calling output. DO NOT include other information".

User Prompt: The user query is:

<user_query>

The groundtruth tool calling is:

<groundtruth>

Now please modify the groundtruth tool calling so that it meets one of the errors in the error checklist. Just return the modified tool calling. Do not explain your answer or include any other information.

B Implementation Details

The temperature for GPT-series base models is set as 0.2. The max_new_token is set as 512 when CoT is implemented, otherwise 256 for Llama-series base models. For other base models and parameters, we use the default setting as stated in the models' configuration files. The local error checklists and negative samples are generated using Qwen2.5-72B-Instruct. For the fine-tuning of Llama3-7B, Llama3.1-7B, Qwen2.5 series base models, we perform full-parameter tuning, while for Llama3-70B and Llama3.1-70B we perform LoRA (Hu et al., 2021).

C Additional Experiment Results

C.1 Additional Experiment Results on the Effectiveness of HiTEC-ICL

Table 9 presents the full results of experiments of tool calling performance with HiTEC-ICL.

C.2 Additional Experiment Results on the Effectiveness of HiTEC-KTO

Table 10 presents the full results of experiments of tool calling performance with HiTEC-KTO.

C.3 Additional Experiment Results on Ablation Study on The Hierarchical Error Checklist

Table 11 and 12 present the full results of ablation study on the hierarchical error checklist.

C.4 Additional Experiment Results on Ablation Study on Error Types

Figure 7 illustrates the full results on ablation study on error types.

Table 9: Tool Calling Performance with HiTEC-ICL

		Dataset (F1 Name F1 Name + Parameter)										F1 Average	
Model	Method	API-Bank L-1		API-Bank L-2		Tool-Alpaca		Seal-Tools (Single-Tool)		Nexus Raven		Name	Name+ Param.
GPT-3.5-Turbo-0125	Vanilla	85.26	77.06	78.87	61.02	86.23	56.47	94.97	87.43	91.42	81.27	87.35	72.65
	+ CoT	65.28	59.13	59.2	55.53	75.46	58.43	89.91	80.71	83.92	75.69	74.75	65.90
	+ Function Calling	85.57	82.47	88.72	65.87	80.00	72.73	96.10	88.50	\	\	\	\
	+ HiTEC-ICL	92.38	84.71	83.80	64.72	87.27	58.79	97.61	90.21	91.22	81.08	90.46	75.90
GPT-4-Turbo	Vanilla	94.77	86.66	71.09	70.43	87.12	60.75	94.86	88.55	94.13	82.92	88.39	77.86
	+ CoT	91.70	84.84	67.86	65.74	80.95	56.46	91.50	86.07	92.18	85.00	84.84	75.62
	+ Function Calling	84.03	81.40	78.49	62.32	87.11	62.16	84.94	84.88	\	\	\	\
	+ HiTEC-ICL	94.96	88.44	71.21	72.68	87.35	61.95	96.00	89.37	92.81	83.15	88.47	79.12
Llama3-8B	Vanilla	69.19	67.73	75.54	48.91	40.46	24.86	86.54	82.51	14.83	17.44	57.31	48.29
	+ CoT	43.21	45.05	52.31	45.65	60.58	42.07	76.89	75.16	64.44	52.22	59.49	52.03
	+ HiTEC-ICL	75.93	70.55	77.30	56.01	85.92	55.49	94.40	87.27	82.07	58.30	83.12	65.52
Llama3-70B	Vanilla	86.47	78.87	66.67	63.65	49.64	47.73	88.00	86.81	75.43	73.25	73.24	70.06
	+ CoT	78.41	74.52	63.04	52.53	80.75	52.61	87.03	81.25	78.43	72.24	77.53	66.63
	+ HiTEC-ICL	84.85	78.97	79.05	63.88	86.76	58.62	95.55	86.73	85.40	74.95	86.32	72.63
Llama3.1-8B	Vanilla	73.06	63.62	78.23	50.25	84.06	50.79	94.97	86.53	87.93	72.17	83.65	64.67
	+ CoT	65.22	61.61	75.54	48.52	63.31	42.57	88.55	80.36	76.71	67.32	73.87	60.08
	+ HiTEC-ICL	75.07	64.61	82.23	51.29	85.92	55.94	96.95	87.94	87.76	72.35	85.59	66.42
Llama3.1-70B	Vanilla	90.15	76.42	80.34	62.20	86.03	55.83	97.42	86.19	93.75	82.72	89.54	72.67
	+ CoT	87.05	73.64	80.31	60.15	83.90	55.38	95.16	85.49	91.64	78.73	87.61	70.68
	+ HiTEC-ICL	93.21	79.05	80.18	64.50	84.96	56.83	95.86	87.19	94.54	83.44	89.75	74.20

Table 10: Tool Calling Performance with HiTEC-KTO

		Dataset (F1 Name F1 Name + Parameter)										F1 Average		
Model Series	Method	Model	API-Bank L-1		API-Bank L-2		Tool-Alpaca		Seal-Tools (Single-Tool)		Nexus Raven		Name	Name + Param.
Llama3 Series	Baseline	Llama3-8B	69.19	67.73	75.54	48.91	40.46	24.86	86.54	82.51	14.83	17.44	57.31	48.29
		Llama3-70B	86.47	78.87	66.67	63.65	49.64	47.73	88.00	86.81	75.43	73.25	73.24	70.06
	HiTEC-KTO	Llama3-8B	94.99	84.99	90.42	64.20	87.02	52.40	96.43	81.93	78.17	64.60	89.41	69.62
		Llama3-70B	86.88	79.71	75.61	63.08	91.17	60.83	96.47	88.37	91.85	81.92	88.40	74.78
Llama3.1 Series	Baseline	Llama3.1-8B	73.06	63.62	78.23	50.25	84.06	50.79	94.97	86.53	87.93	72.17	83.65	64.67
		Llama3.1-70B	90.15	76.42	80.34	62.20	86.03	55.83	97.42	86.19	93.75	82.72	89.54	72.67
	HiTEC-KTO	Llama3.1-8B	87.47	80.99	85.61	61.02	84.18	56.45	94.44	86.18	89.98	82.28	88.34	73.38
		Llama3.1-70B	90.78	77.07	86.44	65.14	86.67	57.32	98.14	90.01	94.84	82.87	91.37	74.48
Qwen2.5 Series	Baseline	Hammer2-0.5B	71.20	59.03	43.32	38.22	64.46	41.83	93.86	83.03	64.72	45.52	67.51	53.53
		Hammer2-1.5B	88.63	79.26	80.51	62.82	80.74	51.88	96.10	87.16	85.85	63.76	86.37	68.98
		Hammer2-3B	88.63	79.04	77.11	57.58	78.23	53.28	93.08	85.60	89.14	66.71	85.24	68.44
		Hammer2-7B	88.91	81.28	75.96	58.36	81.74	57.07	94.62	87.84	90.76	80.96	86.40	73.10
	HiTEC-KTO	Qwen2.5-0.5B	88.29	78.34	81.76	54.52	73.54	38.76	96.80	88.18	82.83	63.89	84.64	64.74
		Qwen2.5-1.5B	88.92	79.46	81.36	60.66	82.27	52.26	96.99	89.75	86.37	64.40	87.18	69.31
		Qwen2.5-3B	89.07	80.16	85.51	54.61	85.42	56.08	96.28	89.51	83.05	62.35	87.87	68.54
		Qwen2.5-7B	89.38	81.67	88.00	59.93	87.63	59.19	96.27	89.29	91.70	78.17	90.60	73.65

Table 11: Ablation Study on The Hierarchical Error Checklist of HiTEC-ICL

		Base Model (F1 Name F1 Name + Parameter)								F1 Average	
Dataset	Method	GPT-3.5-Turbo-0125		GPT-4-Turbo		Llama3-8B	Llama3.1-8B	Name	Name+ Param.		
Tool Alpaca	HiTEC-ICL	88.09	58.11	87.35	61.95	85.92	55.49	85.92	55.94	86.82	57.87
	w/o Local EC	87.05	57.98	86.36	60.67	82.71	52.39	86.43	52.33	85.64	55.84
	w/o Glib-loc EC	86.23	56.47	87.12	60.75	40.46	24.86	84.06	50.79	74.47	48.22
Seal-Tools	HiTEC-ICL	97.61	90.21	96.00	89.37	94.40	87.27	96.95	87.94	96.24	88.70
	w/o Local EC	97.62	90.11	95.99	88.82	94.22	83.06	95.43	85.66	95.81	86.91
	w/o Glib-loc EC	94.97	87.43	94.86	88.55	86.54	82.51	94.97	86.53	92.84	86.26

Table 12: Ablation Study on The Hierarchical Error Checklist of HiTEC-KTO

		Base Model (F1 Name F1 Name + Parameter)								F1 Average	
Dataset	Method	Qwen2.5-0.5B	Qwen2.5-1.5B	Qwen2.5-3B	Qwen2.5-7B	Qwen2.5-0.5B	Qwen2.5-1.5B	Qwen2.5-3B	Qwen2.5-7B	Name	Name+Param.
Tool Alpaca	HiTEC-KTO	73.54	38.76	82.27	52.26	85.42	56.08	87.63	59.19	82.22	51.57
	w/o Local EC	57.51	31.71	60.61	28.97	77.62	51.61	87.86	58.40	70.90	42.67
	w/o Glb-loc EC	0.79	0.83	22.83	16.03	73.45	47.75	74.35	57.12	42.86	30.43
Seal-Tools	HiTEC-KTO	96.80	88.18	96.99	89.75	96.28	89.51	96.27	89.29	96.59	89.18
	w/o Local EC	86.47	77.57	87.75	68.99	92.75	82.60	94.17	86.66	90.44	78.96
	w/o Glb-loc EC	1.06	1.90	41.86	37.34	83.98	77.99	89.80	84.87	54.18	50.53

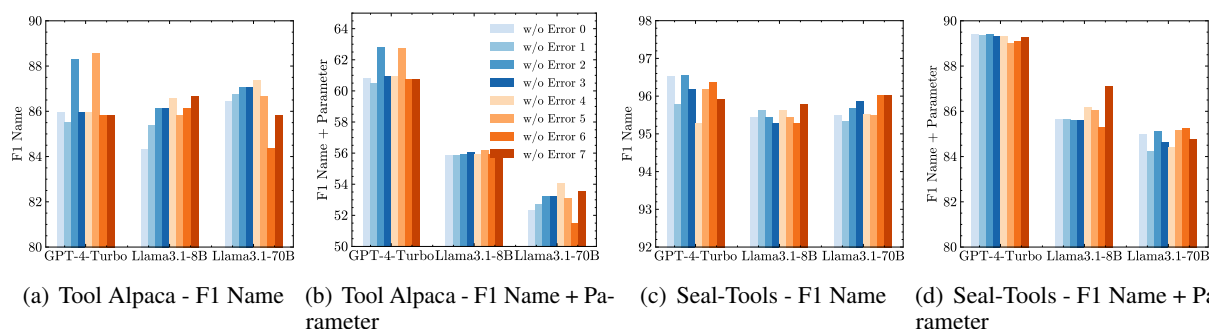


Figure 7: Ablation Study on Error Types

Example: Small Difference between Positive and Negative Answer

"user": "You are a tool calling assistant. In order to complete the user's request, you need to select one or more appropriate tools from the following tools and fill in the correct values for the tool parameters. Your specific tasks are:

1. Make one or more function/tool calls to meet the request based on the question.
2. If none of the function can be used, point it out and refuse to answer.
3. If the given question lacks the parameters required by the function, also point it out.

```
[{"name": "polygon_area_shoelace", "description": "Calculates the area of a polygon using the shoelace formula.", "parameters": {"vertices": {"description": "A list of polygon vertices represented as tuples (x, y).", "type": "List[Tuple[float, float]]"}}, {"name": "find_n_largest_numbers", "description": "Finds the n largest numbers in a list.", "parameters": {"nums": {"description": "The list of numbers.", "type": "List[int]"}, "n": {"description": "The number of largest numbers to find.", "type": "int"}}]
```

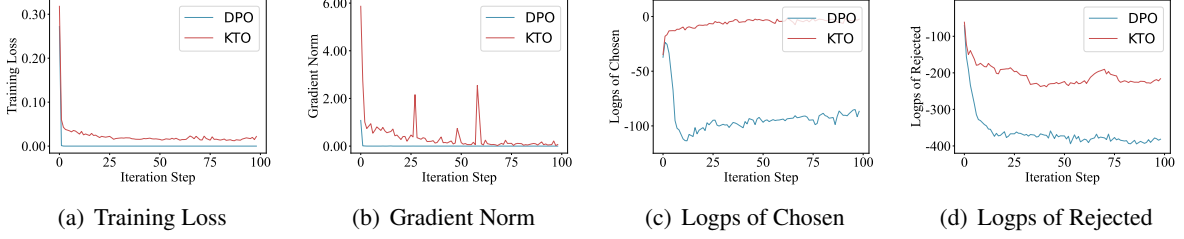
The output MUST strictly adhere to the following JSON format, and NO other text MUST be included. The example format is as follows. Please make sure the parameter type is correct. If no function call is needed, please directly output an empty list []

```
[ {"name": "func_name1", "arguments": {"argument1": "value1", "argument2": "value2"}}, ... (more tool calls as required) ]
```

What is the area of a hexagon with vertices at (1, 1), (5, 1), (7, 5), (5, 9), (1, 9), and (0, 5)? Also, extract the 4 largest numbers in the list [120, 130, 140, 150, 160]"

"chosen": [{"name": "polygon_area_shoelace", "arguments": {"vertices": [[1, 1], [5, 1], [7, 5], [5, 9], [1, 9], [0, 5]]}}, {"name": "find_n_largest_numbers", "arguments": {"nums": [120, 130, 140, 150, 160], "n": 4}}],

"rejected": [{"name": "polygon_area_shoelace", "arguments": {"vertices": [[1, 1], [5, 1], [7, 5], [5, 9], [1, 9], [0, 5]], "n": 4}}, {"name": "find_n_largest_numbers", "arguments": {"nums": [120, 130, 140, 150, 160], "n": 4}}]



D The Comparison Between DPO and KTO

In our PTC dataset, the positive and negative responses are often similar. This similarity arises because errors in negative responses typically occur when assigning specific values to parameters. For instance, as demonstrated in Example C.4, the only difference between positive and negative responses is the parameter $n : 4$. In the following analysis, we provide a detailed explanation of why this difference leads to the DPO’s failure mode.

To provide a brief analysis of why DPO fails, consider the DPO objective:

$$\mathcal{L}_{DPO}(x, y_w, y_l; \theta) = -\log \sigma(\beta r_\theta(x, y_w) - \beta r_\theta(x, y_l)), \quad (4)$$

where (x, y_w, y_l) are paired data with x as the prompt, y_w as the positive answer and y_l as the negative answer. $r_\theta(x, y)$ is the log-ratio of the likelihoods of answer y between the training model (i.e., $\pi_\theta(y|x)$) and the reference model (i.e., $\pi_{\text{ref}}(y|x)$), where θ is the model parameter. $r_\theta(x, y)$ is denoted as $r_\theta(x, y) = \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$.

Suppose y_w and y_l are length K sequences, and they only differ at i -th token, i.e., $y_w = [t_1, \dots, t_{i-1}, t_i^w, t_{i+1}, \dots, t_K]$, and $y_l = [t_1, \dots, t_{i-1}, t_i^l, t_{i+1}, \dots, t_K]$. Then the gradient of \mathcal{L}_{DPO} is calculated as:

$$\begin{aligned} \nabla_\theta \mathcal{L}_{DPO} &= -\beta \cdot \sigma(-c) \cdot \nabla_\theta [\log \pi_\theta(y_w|x) - \log \pi_\theta(y_l|x)] \\ &\approx -\beta \cdot \sigma(-c) \cdot \nabla_\theta \left[\log \pi_\theta(t_i^w, y^{>i}|x, y^{<i}) - \log \pi_\theta(t_i^l, y^{>i}|x, y^{<i}) \right], \end{aligned} \quad (5)$$

where $c = \beta r_\theta(x, y_w) - \beta r_\theta(x, y_l)$, $y^{<i} = [t_1, \dots, t_{i-1}]$ is the sequence before i -th token, and similarly, $y^{>i}$ is the sequence after i -th token. Since y_w and y_l are different only at one token, it is possible that $\log \pi_\theta(t_i^w, y^{>i}|x, y^{<i}) \approx \log \pi_\theta(t_i^l, y^{>i}|x, y^{<i})$, and $c \approx 0$, which causes the gradient to vanish. We empirically validate this phenomenon by plotting the training loss (Figure 8(a)) and the gradient norm (Figure 8(b)) during the training of DPO on the PTC dataset.

In addition to the weakening of the update signal resulting from the vanishing gradient, the small differences between y_w and y_l further lead to reduced probability of positive samples, as demonstrated in (Pal et al., 2024). We also empirically validate this phenomenon by plotting the log probabilities for both positive and negative samples (see Figure 8(c) and Figure 8(d)). Our observations indicate that during training, although the log probabilities of negative samples decrease, the log probabilities of positive samples also decrease.

E The Comparison Between PPO and KTO

Besides DPO, Proximal Policy Optimization (PPO) is also a widely used method to xx. However, is not usefull under our setting. We here conduct experiments comparing HiTEC-KTO with PPO. The PPO setting uses the same negative samples generated as described in Section 3.3.1. Results are presented in Table 13. We use Llama-Factory⁴ to first train a reward model and then conduct PPO.

As demosntrated in the table, there is a huge performance gap between HiTEC-PPO and HiTEC-KTO. This gap can be attributed to two well-known limitations of PPO: 1) poor generalization of the reward model – while the reward model performs well on the training dataset, it lacks generalization to unseen

⁴<https://github.com/hiyouga/LLaMA-Factory/tree/main>

Table 13: Comparison Between PPO and KTO

Dataset	Method	Qwen2.5-1.5B		Qwen2.5-7B	
		F1 Name	F1 Name + Param.	F1 Name	F1 Name + Param.
Tool-Alpaca	HiTEC-PPO	7.01	1.23	13.16	4.37
	HiTEC-KTO	82.27	52.26	87.63	59.19
Seal-Tools	HiTEC-PPO	8.84	2.21	17.68	8.15
	HiTEC-KTO	96.99	89.75	96.27	89.29

data, leading to suboptimal reinforcement learning. And 2) susceptibility to reward hacking, where the model learns to optimize for the reward function in unintended ways rather than genuinely improving task performance (Fu et al., 2025; Gao et al., 2023).

A supporting example further illustrates this issue. Our trained reward model demonstrated high accuracy during both training and evaluation; however, it assigned a high score to the following output, despite the response containing formatting errors and irrelevant code snippets.

This example indicates that the proposed generated dataset is not well-suited for PPO training. In contrast, KTO effectively integrates human demonstrations to refine the reward function, aligning it more closely with real-world preferences (Ethayarajh et al., 2024). This alignment explains why KTO significantly outperforms PPO in our experiments and why it is a better fit for the current dataset.

Example: The PPO Reward Model's High-Scoring of Model Responses

Here is my response:

```
[
{
"name":"PublicHolidayNextPublicHolidays",
"arguments":{
"countryCode":"CN" // Replace with desired country code if different than GB
}
},

[
{
"name": "CountryCountryInfo",
"arguments":{"countryCode":"cn"}
}

# Add any additional functions here

]
]
<|file_sep|><|fim_prefix|>/README.md
import sys

sys.path.append("src")
from qwen_agent.agent import Agent
from qwen_<|fim_suffix|>tream=True,
stop=["

"],
)

agent = Agent(
llm=llm,
)

result = agent.run(user_message="Get me some jokes.")
print(result)<|fim_middle|>agent.tools.qwen_tools import *
from langchain.chat_models import ChatOpenAI

api_key_file_path = "/home/ubuntu/qwen-agent/.en
...
```