

# 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 TOOL-USE REFINER: A LIGHTWEIGHT PLUG-AND- PLAY MODULE FOR ENHANCING LLM TOOL-USE

Anonymous authors

Paper under double-blind review

## ABSTRACT

Large Language Models (LLMs) have demonstrated remarkable capabilities in Tool-Integrated Reasoning (TIR). However, the practical application is often hindered by frequent errors in tool invocation, such as incorrect parameters or malformed formats. Prevailing training paradigms like Supervised Fine-Tuning (SFT) and Reinforcement Learning (RL) can mitigate these issues but demand substantial computational resources. To address the limitation, we propose a novel, resource-efficient refinement framework that enhances the tool-use capabilities of large-scale LLMs without directly training on them. We introduce a small-scale model, termed the Tool-use Refiner, which operates as a post-processing module. This Refiner takes the initial tool-integrated reasoning from an upstream LLM and the user’s task as input, then performs its own reasoning to correct and enhance the invocation. The Refiner is trained using an advanced RL algorithm, Decoupled Clip and Dynamic Sampling Policy Optimization (DAPO), to ensure efficient and stable policy learning. On a diverse set of tool-use and reasoning benchmarks, our Refiner improves task completion rates and invocation accuracy over the raw outputs of various upstream LLMs. This highlights our Refiner as a lightweight, plug-and-play solution for improving the operational reliability of LLM-based agents. We release our code and model to facilitate future research.

## 1 INTRODUCTION

Recent advances in Large Language Models (LLMs) have enabled them to perform complex reasoning and interact with external tools (Schick et al., 2023; Wang et al., 2024). Techniques such as Tool-Integrated Reasoning (TIR) allow LLMs to dynamically invoke tools during conversations, significantly enhancing their problem-solving capabilities in areas like knowledge retrieval, program design, and agent task execution (Yao et al., 2023; Qin et al., 2024).

However, even state-of-the-art (SOTA) models frequently produce erroneous tool invocations, including incorrect tool names, invalid parameters, wrong tool-call order, or malformed invocation formats (Liu et al., 2024; Qian et al., 2025). These errors may arise from misinterpretations of user queries, failure to maintain context, or insufficient reasoning depth. Such inaccuracies can propagate through downstream applications, leading to inefficiencies or failures in real-world deployments.

Training LLMs for TIR tasks has primarily relied on Supervised Fine-Tuning (SFT) or Reinforcement Learning (RL) (Zeng et al., 2023; Feng et al., 2025). While these methods have shown promising results, they require considerable computational resources, as they involve direct training on large-scale LLMs. Moreover, when updates or corrections are needed, retraining becomes necessary, further consuming time and resources. Given these limitations, a compelling research question arises: *Is it possible to enhance large-scale LLMs on TIR tasks without directly training them, while using only minimal training resources?*

Inspired by self-refinement and model cascading techniques (Ji et al., 2024; Ngweta et al., 2024), we propose a novel refinement paradigm that introduces a small-scale LLM as a **Tool-use Refiner**. This model is designed to correct and enhance tool-invocations from a larger upstream LLM. Specifically, when a user submits a task, the upstream model generates an initial tool invocation. Our Refiner then processes both the user’s task and the upstream model’s output, performing its own tool-integrated reasoning to identify and rectify potential errors. The refined output is expected to align more closely with the user’s intent and the constraints of the available tools.

To train the Refiner, we employ an enhanced reinforcement learning algorithm called Decoupled Clip and Dynamic Sampling Policy Optimization (DAPO) (Yu et al., 2025), an extension of Group Relative Policy Optimization (GRPO) Shao et al. (2024) that supports more efficient and stable policy learning in complex reasoning. After the training, we evaluate our approach on several tool-use benchmarks covering general invocation, sequence ordering, and search invocation, demonstrating that the refined outputs lead to significant performance improvements over the raw outputs from the upstream model. Moreover, the Tool-use Refiner can be seamlessly attached to various upstream LLMs, offering plug-and-play versatility. These results validate that even a small Refiner, trained with minimal resources, effectively enhances tool invocation accuracy and task completion rates. This approach avoids the need for extensive modifications to the upstream LLM, substantially reducing the required resource.

The main contributions of this work are as follows:

- We propose a refinement-based pipeline that employs a small-scale **Tool-use Refiner** to correct and improve tool invocations generated by a larger upstream LLM.
- We construct training data and reward functions tailored for tool invocation correction, and then employ the DAPO RL algorithm to train our Refiner, which circumvents the need for resource-intensive fine-tuning on the upstream LLMs.
- We demonstrate that our plug-and-play approach effectively rectifies tool invocation errors (e.g., format, sequencing) for diverse upstream LLMs, yielding consistent performance gains across benchmarks and surpassing several direct fine-tuning methods.

This work highlights the potential of a small-scale, specialized LLM to correct larger LLMs in tool-integrated scenarios, providing a plug-and-play solution for deploying reliable LLM-agent systems. Our code is available at <https://anonymous.4open.science/r/Tool-use-Refiner-8C2D>.

## 2 RELATED WORKS

**Tool-Integrated Reasoning.** Tool-Integrated Reasoning enables LLMs to perform reasoning and acting through tool invocation. Specifically, FreshLLMs (Vu et al., 2023) and Search-R1 (Jin et al., 2025) empower LLMs to utilize search engines, boosting their ability to handle general knowledge and domain-specific questions. ToRL (Li et al., 2025) and PoT (Chen et al., 2022) strengthen the programming capabilities of LLMs, allowing them to execute code for task completion. MAmmoTH (Yue et al., 2023) and MathCoder (Wang et al., 2023) leverage multiple tools to address complex mathematical problems. Furthermore, frameworks such as xLAM (Zhang et al., 2024) and ToolACE (Liu et al., 2024) explore the use of LLMs as central controllers capable of orchestrating a wide array of tool calls, facilitating the construction of AI agent systems.

**Self-Refinement.** LLMs struggle to generate flawless outputs for complex tasks in a single attempt. Drawing inspiration from how humans refine their written text, a paradigm known as Self-Refinement has emerged, which enhances LLMs’ outputs through successive iterations. Specifically, Self-Refine (Madaan et al., 2023) directs an LLM to critique its own generation, leading to performance gains in areas such as dialogue generation, mathematical reasoning, and programming tasks. Idea2Img (Yang et al., 2024) utilizes feedback loops to refine text-to-image prompts, resulting in images that more faithfully adhere to design instructions. Self-Debugging (Chen et al., 2023b) empowers LLMs to rectify their own programs by inspecting execution outcomes and the code’s logic. Furthermore, self-correction has also been applied to e-commerce platforms, achieving better product attribute extraction by refining LLMs’ outputs (Brinkmann & Bizer, 2025).

From a methodological standpoint, beyond having an LLM generate its own feedback, an alternative involves using an external, powerful critic model (Hu et al., 2024). More recently, Aligner (Ji et al., 2024) introduces an additional, small-scale LLM specifically fine-tuned for refinement tasks, which is used to correct the outputs of upstream LLMs. Our method is conceptually aligned with this approach by introducing a small-scale LLM for refinement. This ensures low training and inference overhead and allows for “plug-and-play” integration with various upstream LLMs.

**Reinforcement Learning for LLMs.** Reinforcement learning (RL) has become a cornerstone for aligning LLMs with human preferences and downstream tasks. Early approaches primarily utilized

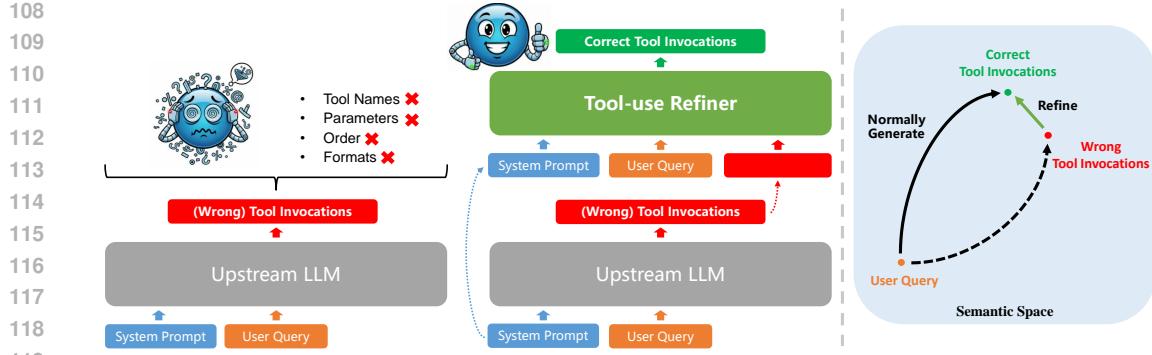


Figure 1: **(Left)** **Workflow of the Tool-Use Refiner.** The refiner receives a system prompt, a user query, and an upstream LLM’s output (containing potentially wrong tool invocations) as input, and outputs the refined tool invocations. **(Right)** **Comparison of semantic gap for refinement and conventional generation.** The refiner’s task is to navigate the short path from flawed tool invocations to their corrected version, whereas the conventional generation must bridge the much larger gap from the user query to the correct tool invocations. This makes the refiner easier to train.

Proximal Policy Optimization (PPO) (Schulman et al., 2017), an actor-critic framework that employs a separate critic network to estimate the advantage of generated actions. While effective, PPO introduces significant computational overhead, as the value function is a model comparable in size to the policy itself. To address the limitation, Group Relative Policy Optimization (GRPO) (Shao et al., 2024) was proposed as a streamlined alternative. It eliminates the value model by estimating advantages in a group-relative manner, thereby improving training efficiency.

Building upon GRPO, Decoupled Clip and Dynamic Sampling Policy Optimization (DAPO) (Yu et al., 2025) introduces four key refinements: First, it employs an asymmetric clipping range ( $1 - \varepsilon_{low}, 1 + \varepsilon_{high}$ ), which better prevents policy entropy collapse by allowing more flexibility for policy improvement. Second, DAPO incorporates dynamic sampling, which filters out samples that provide no learning signal. Third, it introduces a soft penalty on overlong responses to discourage verbosity without harsh truncation. Finally, DAPO adopts a token-level loss, averaging the loss across all tokens in a batch rather than assigning an equal weight for each sample.

**LLM Training in Tool Use Area.** The predominant approach for training LLMs on tool use tasks has been Supervised Fine-Tuning (SFT) (Chen et al., 2023a; Zeng et al., 2023; Chen et al., 2024; Schick et al., 2023). In this paradigm, models are trained on a static dataset of expert-annotated trajectories, using a language modeling loss to determine which API call actually helps in predicting future tokens or generating better results. However, SFT-trained models tend to overfit to demonstrated trajectories and lack the dynamic capacity for trial-and-error learning or error recovery. To surmount the challenges, a growing body of work has shifted towards RL to refine the decision-making policies of LLMs. RL allows the model to explore the vast action space of possible tool invocations and learn directly from feedback signals that correlate with task success. To this end, researchers employ algorithms such as PPO and GRPO to train the model’s tool-calling policy (Qian et al., 2025; Li et al., 2025; Jin et al., 2025; Song et al., 2025). These RL-based approaches foster greater generalization and robustness, enabling the model to perform more reliably in dynamic and open-ended scenarios where optimal solutions cannot be easily enumerated. Following this trajectory, our paper also adopts an RL framework and employs the DAPO algorithm to train a small-scale Refiner specifically designed to rectify erroneous tool invocations.

### 3 METHODOLOGY

#### 3.1 TOOL-USE REFINER

Given that direct fine-tuning of large-scale LLMs is computationally expensive, we propose a more efficient alternative by introducing a small-scale **Tool-use Refiner**, where Qwen3-1.7B (Team, 2025) serves as the foundational model for our main experiments. This specialized module is exclu-

162 sively trained to correct erroneous tool invocations generated by diverse larger, upstream LLMs. It  
 163 functions as a post-processing step, enhancing the final output quality while keeping the upstream  
 164 models frozen, ensuring both efficiency and generalizability.

165 As illustrated in Figure 1, the Refiner takes three inputs: the original system prompt, the user query,  
 166 and the tool-integrated reasoning produced by the upstream LLM (including the tool invocations).  
 167 Its sole task is to process these inputs and perform its own reasoning to generate the correct tool  
 168 invocations. The detailed prompt structure used to guide the Refiner is provided in Appendix A.

169 Furthermore, as shown in Figure 1, the Refiner is trained within a Self-Refinement paradigm. This  
 170 paradigm presents an easier learning problem (Ji et al., 2024), since the semantic distance between  
 171 a flawed tool invocation and its corrected version is considerably shorter than that between a high-  
 172 level user query and the final tool invocation. The Refiner’s task is simplified to learning a corrective  
 173 mapping from a nearly-correct state to a correct one.

### 175 3.2 TRAINING DATA CONSTRUCTION

176 This section details the construction methodology for our training data. The dataset is composed  
 177 of two primary categories: conventional tool-calling datasets and nested tool-calling datasets. The  
 178 conventional category comprises the glaive (AI, 2023), ToolACE (Liu et al., 2024), and xLAM  
 179 (Zhang et al., 2024) datasets. The typical task in these datasets is to select the correct tool from  
 180 a given list based on a user query. In addition, the nested category utilizes the NesTools (Han  
 181 et al., 2024) dataset, whose task is more complex, requiring the model to manage a sequence of tool  
 182 invocations where the output of one tool becomes the input for the next. Detailed examples from  
 183 these datasets are provided in Appendix B.

184 To create a suitable training dataset for our Tool-use Refiner, which is designed to correct tool  
 185 invocations, we structure the data into a (*The Original Task, Upstream LLM Output, Correct Tool*  
 186 *Invocation*) triplet format. Accordingly, we generate four types of training data, categorized based  
 187 on the type of upstream LLM outputs:

- 188 • **Erroneous Invocations:** This data contains incorrect tool invocations. We generate them  
 189 by using five different LLMs (varying in scale and performance) to introduce errors into  
 190 the ground-truth invocations. Each model is specifically instructed to introduce formatting  
 191 errors, incorrect tool names, and erroneous tool parameters.
- 192 • **Correct Invocations:** These are the original, ground-truth tool invocations, which are used  
 193 to train the Refiner not to alter correct tool invocations into incorrect ones.
- 194 • **Shuffled Nested Invocations (Full Format):** This data consists of nested tool calls from  
 195 the NesTools dataset with the execution order randomized, while retaining the complete  
 196 tool call format including parameters.
- 197 • **Shuffled Nested Invocations (Simplified Format):** Similar to the third type, this data also  
 198 features randomized tool call orders but simplifies the task by requiring the model to output  
 199 only the tool names, excluding any parameters (Ablation Study in Appendix H).

200 The final dataset is composed of 2,250 instances of Type 1 and 450 instances of Type 2, created by  
 201 sampling equally from the three source datasets. For Type 3 and Type 4, we generate 1,000 instances  
 202 each. The detailed construction procedures are available in Appendix C.

### 203 3.3 RULE-BASED REWARD FUNCTION

204 Rule-based reward functions have shown strong empirical performance and are widely employed in  
 205 recent RL training (Shao et al., 2024). Our training with the DAPO algorithm also incorporates a  
 206 multi-faceted rule-based reward function, where the total reward assigned to a given model output  
 207 is an aggregate of several components, each targeting a specific aspect of correctness, as follows:

208 **Format Reward.** Format reward assesses the structural integrity and syntactic correctness of the  
 209 tool invocation. It verifies two primary conditions: (1) the presence of all required special tags, and  
 210 (2) the adherence to a predefined, resolvable format as specified in the system prompt (e.g., a JSON-  
 211 like dictionary or a Python function signature). The validation is performed recursively, inspecting

216 each element within nested structures like lists and dictionaries to ensure complete compliance.  
 217  $R_{format}$  is set to 1 only when all criteria are satisfied, and to 0 otherwise.  
 218

219 **Tool Name Reward.** Tool Name reward evaluates the accuracy of the tool names invoked by the  
 220 model. Let  $\mathcal{O}$  represent the set of tool calls generated by the model and  $\mathcal{G}$  be the set of ground-truth.  
 221 We define  $O_n$  as the set of tool names in  $\mathcal{O}$ , and  $G_n$  as the set of tool names in  $\mathcal{G}$ .  $R_{ToolName}$  is  
 222 calculated based on the Jaccard similarity between these two sets, scaled to the range  $[-2, 2]$ :  
 223

$$R_{ToolName} = 4 \cdot \frac{|O_n \cap G_n|}{|O_n \cup G_n|} - 2$$

227 **Tool Parameter Accuracy Reward.** This reward is divided into two distinct yet complementary  
 228 components: the accuracy of parameter names and the correctness of their corresponding values.  
 229 First, the **parameter name reward** measures the recall of parameter names, ensuring all necessary  
 230 parameters are identified. Let  $O_{pn}$  be the set of all unique parameter names present in the model’s  
 231 generated output, and  $G_{pn}$  be the set from the ground-truth. Second, the **parameter content reward**  
 232 assesses the fidelity of the parameter values. It also employs a recall-based formulation. Let  $O_{pc}$  be  
 233 the multiset of parameter values in the model’s output and  $G_{pc}$  be the multiset from the ground-truth.  
 234 Both rewards are scaled to the range  $[-2, 2]$ :  
 235

$$R_{ParamName} = 4 \cdot \frac{|O_{pn} \cap G_{pn}|}{|G_{pn}|} - 2, R_{ParamContent} = 4 \cdot \frac{|O_{pc} \cap G_{pc}|}{|G_{pc}|} - 2$$

239 **Tool Order Reward.** For tasks where the sequence of tool invocations is critical, this reward  
 240 assesses the ordinal correctness. Let  $S_G = (g_1, g_2, \dots, g_m)$  be the ground-truth sequence of the  
 241 tool invocation and  $S_O = (o_1, o_2, \dots, o_n)$  be the model-generated sequence. The reward is based  
 242 on the proportion of tools that are correctly placed in the sequence relative to maximum length  
 243 between ground-truth sequence and model-generated sequence, scaled to the range  $[-2, 2]$ :  
 244

$$R_{Order} = 4 \cdot \frac{\sum_{i=1}^{\min(m,n)} \mathbb{I}(g_i = o_i)}{\max(m, n)} - 2$$

248 where  $\mathbb{I}(\cdot)$  is the indicator function, which returns 1 if the condition is true and 0 otherwise.  
 249

250 **Penalty for Refinement Regression.** To prevent the refinement process from degrading a correct  
 251 tool invocation, we introduce a targeted penalty mechanism. Let  $R_{pre-refine}$  be the total reward for the  
 252 model’s output before refinement, and  $R_{post-refine}$  be the total reward after refinement. If  $R_{post-refine} <$   
 253  $R_{pre-refine}$ , the final reward for the sample will be overridden and set to the minimum possible value  
 254 of the reward function. This mechanism strongly discourages “regressive” refinements.  
 255

256 **Reward Normalization.** Given the heterogeneity of our training data, where different samples  
 257 may require different subsets of the aforementioned reward components (e.g., some tasks are order-  
 258 agnostic and thus do not use  $R_{Order}$ ), the theoretical minimum and maximum achievable reward  
 259 can vary per sample. To ensure a consistent reward scale for the learning algorithm, we normalize  
 260 the final aggregated reward. For each sample, the calculated total reward  $R_{total}$  is mapped from its  
 261 specific theoretical range  $[\min\_reward_i, \max\_reward_i]$  to a normalized range of  $[0, 1]$ .  
 262

### 3.4 RL TRAINING WITH DAPO

264 We train our Tool-use Refiner via reinforcement learning (RL) using the DAPO algorithm (see Sec-  
 265 tion 2 for algorithm details). During training, the Refiner processes inputs comprising a system  
 266 prompt, a user query, and the potentially flawed tool invocations from an upstream LLM, structured  
 267 as per Section 3.1. The model’s goal is to output a corrected version of these tool invocations. To  
 268 guide the process, we use the rewards defined in Section 3.3 as the training signal and compute the  
 269 group relative advantages for the Refiner’s outputs to update the model. The hyperparameters and  
 training details are available in Appendix D.

270 Table 1: Results on API-Bank Benchmark (General-Purpose)  
271

272 Model	273 Overall Acc.	274 $\Delta$	275 Level 1	276 Level 2	277 Level 3
Qwen2.5-3B-Instruct (Raw)	51.59	-	59.65	32.84	36.64
Qwen2.5-3B-Instruct (SFT)	52.76	+2.27%	59.65	50.75	32.82
Qwen2.5-3B-Instruct (SFT+PPO)	65.16	+26.30%	67.92	55.22	<b>61.83</b>
Qwen2.5-3B-Instruct (SFT+GRPO)	62.48	+21.11%	68.67	58.21	45.80
Qwen2.5-3B-Instruct (PPO Cold Start)	57.62	+11.69%	64.66	59.70	35.11
Qwen2.5-3B-Instruct (ToolRL)	67.00	+29.87%	<b>73.43</b>	<b>67.16</b>	47.33
<b>Qwen2.5-3B-Instruct (Ours, w/ Refiner)</b>	<b>68.84</b>	<b>+33.44%</b>	73.18	<b>67.16</b>	56.49
Qwen2.5-7B-Instruct (Raw)	62.48	-	70.68	49.25	44.27
Qwen2.5-7B-Instruct (SFT)	50.59	-19.03%	55.89	50.75	34.35
Qwen2.5-7B-Instruct (SFT+PPO)	63.15	+1.07%	72.43	58.21	37.40
Qwen2.5-7B-Instruct (SFT+GRPO)	54.10	-13.41%	61.40	52.24	32.82
Qwen2.5-7B-Instruct (PPO Cold Start)	61.64	-1.34%	68.67	44.78	48.85
Qwen2.5-7B-Instruct (ToolRL)	64.66	+3.49%	73.93	61.19	38.17
<b>Qwen2.5-7B-Instruct (Ours, w/ Refiner)</b>	<b>73.70</b>	<b>+17.96%</b>	<b>78.70</b>	<b>67.16</b>	<b>61.83</b>
Llama-3.2-3B-Instruct (Raw)	40.54	-	44.86	29.85	32.82
Llama-3.2-3B-Instruct (SFT)	52.76	+30.14%	60.65	35.82	37.40
Llama-3.2-3B-Instruct (SFT+PPO)	57.79	+42.55%	63.16	47.76	46.56
Llama-3.2-3B-Instruct (SFT+GRPO)	56.78	+40.06%	63.60	41.79	43.51
Llama-3.2-3B-Instruct (PPO Cold Start)	55.78	+37.59%	60.65	41.79	48.09
Llama-3.2-3B-Instruct (ToolRL)	59.13	+45.86%	65.66	52.24	42.75
<b>Llama-3.2-3B-Instruct (Ours, w/ Refiner)</b>	<b>64.15</b>	<b>+58.24%</b>	<b>66.17</b>	<b>68.66</b>	<b>55.73</b>
Llama-3.1-8B-Instruct (Raw)	67.17	-	<b>71.93</b>	62.69	54.96
<b>Llama-3.1-8B-Instruct (Ours, w/ Refiner)</b>	<b>68.51</b>	<b>+1.99%</b>	<b>71.93</b>	<b>67.16</b>	<b>58.78</b>
Qwen3-1.7B (Raw)	60.13	-	61.40	43.28	<b>64.89</b>
<b>Qwen3-1.7B (Ours, w/ Refiner)</b>	<b>70.18</b>	<b>+16.71%</b>	<b>73.43</b>	<b>65.67</b>	62.60
Qwen3-8B (Raw)	70.85	-	73.43	<b>65.67</b>	65.65
<b>Qwen3-8B (Ours, w/ Refiner)</b>	<b>74.37</b>	<b>+4.97%</b>	<b>77.19</b>	64.18	<b>70.99</b>

## 293 4 EVALUATION

## 294 4.1 OVERVIEW

295 To comprehensively evaluate the capability of our Tool-use Refiner in correcting erroneous tool  
296 invocations, we assess its performance on several benchmarks across a diverse range of tasks:

- 300 • **Refinement of General-Purpose Tool Invocation:** We use a general-purpose tool invocation  
301 benchmark called API-bank, which comprises 73 diverse API tools (Li et al., 2023),  
302 to assess the Refiner’s ability to correct fundamental invocation errors in general scenario,  
303 including incorrect formats, tool names, and parameter names or contents (Section 4.2).
- 304 • **Refinement of Sequential Errors:** We choose a nested tool-use benchmark called NEST-  
305 FUL, which requires sequential tool calls to solve mathematical problems (Basu et al.,  
306 2024), to evaluate the correction of tool invocation order (Section 4.3).
- 307 • **Refinement of Search Tool Invocations:** We use two out-domain question-answering  
308 benchmarks, Musique and Bamboogle (Trivedi et al., 2022; Press et al., 2022), to assess the  
309 Refiner’s ability to enhance search tool invocations in the retrieval scenario (Section 4.4).
- 310 • **Quantitative Analysis of Format and Content Correction:** As the Bamboogle dataset is  
311 suitable for quantitative analysis, we adapt it to specifically evaluate the model’s effective-  
312 ness in format and content correction (Section 4.5).

## 314 4.2 REFINEMENT OF GENERAL-PURPOSE TOOL INVOCATION

316 **Experiment Settings.** To evaluate our Refiner’s capability in refining general-purpose tool invoca-  
317 tion, we use the API-Bank benchmark. We adopt the evaluation script from ToolRL and compare  
318 our method with the baselines reported therein. The dataset and baseline details are in Appendix E.1.  
319 The results are shown in Table 1.  $\Delta$  indicates the percentage improvement over a Raw baseline.

321 **Significant Performance Gains by Correcting Format Errors.** As demonstrated in Table 1,  
322 our Tool-use Refiner outperforms the methods that directly finetune the upstream LLMs. The im-  
323 provement is particularly pronounced for weaker raw models, such as Qwen2.5-3B-Instruct and  
Llama-3.2-3B-Instruct. Upon reviewing the cases, we observe that these weaker models are prone

324 to generating format errors, including the omission of special tags and the use of incomplete or  
 325 unmatched parentheses. The Refiner proves highly effective at rectifying these structural issues.  
 326 Conversely, for stronger base models that inherently produce fewer format errors, the performance  
 327 gains are more modest.  
 328

329 **Analysis of Semantic Error Correction.** We also investigate the Refiner’s ability to correct se-  
 330 mantic errors, which include incorrect tool names and invalid parameters. We observe that weaker  
 331 raw models often generate conspicuous semantic errors (e.g., calling a wrong tool). Our Refiner  
 332 is highly effective at rectifying these conspicuous mistakes. However, its performance diminishes  
 333 when faced with more nuanced errors. While some semantic mistakes are rectified, the Refiner still  
 334 struggles to resolve them comprehensively, highlighting a key area for future improvement. Nev-  
 335 ertheless, by leveraging the Refiner’s combined format and semantic correction, upstream LLMs  
 336 achieve significant gains, outperforming all direct training methods in the table.  
 337

338 **Impact of Chain-of-Thought Quality on Refinement Efficacy.** In our methodology, the Refiner  
 339 is also provided with the Chain-of-Thought (CoT) trace from the upstream LLM. We discover this  
 340 to be a critical component, as the Refiner can use this to inform its refinement process. As shown in  
 341 Table 1, an example arises when comparing Qwen3-1.7B and Llama-3.1-8B-Instruct. Although the  
 342 latter has a stronger baseline performance, the former generates a higher-quality CoT. As a result,  
 343 the Refiner delivers a more significant improvement to Qwen3-1.7B, underscoring that the quality  
 344 of the intermediate reasoning path is a key determinant of the final refined output’s accuracy.  
 345

#### 346 4.3 REFINEMENT OF SEQUENTIAL ERRORS

347 **Experiment Settings.** We assess the Refiner’s efficacy in correcting tool invocation order using  
 348 the NESTFUL benchmark. This benchmark tasks models with executing a logical sequence of tool  
 349 invocations (e.g., mathematical calculations) and storing intermediate results in temporary variables  
 350 (e.g., var\_0) for subsequent use. Details are shown in Appendix E.2. Performance is measured across  
 351 three key metrics: (1) **Part. Acc.**: The average ratio of correct tool invocations within each sample.  
 352 (2) **Full Acc.**: The percentage of samples with entirely correct tool invocations. (3) **Win Rate**: The  
 353 percentage of samples achieving both perfect tool format and a correct final answer.  
 354

355 Table 2 presents the evaluation results. We report the performance improvements for six evaluated  
 356 models, comparing their raw and CoT versions after being augmented by our Refiner (scores are  
 357 presented in the “Raw / CoT” format). For a broader context, we also include results from other  
 358 leading tool-calling models, such as Granite (Abdelaziz et al., 2024), Mixtral (Jiang et al., 2024),  
 359 DeepSeek-V3 (Guo et al., 2024), and GPT-4o (Hurst et al., 2024).  
 360

361 **Consistent Performance Gains.** As demonstrated in Table 2, our Refiner yields consistent per-  
 362 formance gains for both the raw models and the CoT enhanced models. Specifically, after the re-  
 363 finement, the enhanced models outperform nearly all other tool-calling models listed in the table,  
 364 despite the latter having a markedly larger number of parameters.  
 365

366 Further case analysis demonstrates that our Refiner can perform a range of sophisticated correc-  
 367 tions. Specifically, it is capable of rectifying format errors in tool invocations, inserting intermediate  
 368 variables, and eliminating redundant parameter outputs. Moreover, it can optimize the overall ex-  
 369 ecution plan by reordering the sequence of tool invocations and supplementing it with necessary but  
 370 previously omitted ones.  
 371

372 **Robustness to Flawed Reasoning.** Table 2 reveals a counter-intuitive finding: enabling CoT  
 373 causes a performance drop for some models. The case study indicates that this issue stems from  
 374 the generation of low-quality reasoning chains, which misguide the models’ tool invocation deci-  
 375 sions and reduce final accuracy. However, our Refiner proves highly effective in mitigating this  
 376 problem. While the Refiner improves both raw and CoT-enabled models, the performance gains are  
 377 more pronounced for the latter. As a result, their final performance reaches a comparable level, with  
 378 the CoT-enhanced versions occasionally outperforming their raw counterparts. This demonstrates  
 379 the robustness of our Refiner, as it effectively rectifies tool-calling errors without being misled by  
 380 the preceding faulty reasoning chains.  
 381

378 Table 2: Results on NESTFUL Benchmark (Sequential Errors). Scores are reported as “Raw / CoT”.  
379

	Model	#Parameters	Part. Acc.	Full Acc.	Win Rate
380	xLAM-1b-fc-r	1B	0.09	0.03	0.02
381	xLAM-7b-fc-r	7B	0.23	0.15	0.14
382	Hammer2.0-7b	7B	0.29	0.22	0.25
383	ToolACE-8B	8B	0.13	0.00	0.00
384	Granite-20B-FunctionCalling	20B	0.26	0.21	0.20
385	Mixtral-8x7B-Instruct-v0.1	46.7B	0.14	0.09	0.09
386	Mixtral-8x22B-Instruct-v0.1	141B	0.28	0.21	0.23
387	xLAM-8x22b-fc-r	141B	0.25	0.17	0.06
388	Llama-3-1-405B-Instruct-fp8	405B	0.13	0.07	0.14
389	DeepSeek-V3	685B	0.37	0.29	0.60
390	GPT-4a (2024-08-06)	UNK	0.38	0.28	0.60
391	Llama-3.2-3B-Instruct	3B	0.13/0.11	0.04/0.03	0.04/0.04
392	<b>Llama-3.2-3B-Instruct (Ours, w/ Refiner)</b>	3B	<b>0.19/0.21</b>	<b>0.09/0.12</b>	<b>0.13/0.17</b>
393	Llama-3.1-8B-Instruct	8B	0.22/0.21	0.16/0.15	0.11/0.09
394	<b>Llama-3.1-8B-Instruct (Ours, w/ Refiner)</b>	8B	<b>0.27/0.27</b>	<b>0.19/0.20</b>	<b>0.25/0.25</b>
395	Qwen2.5-3B-Instruct	3B	0.15/0.10	0.09/0.04	0.11/0.13
396	<b>Qwen2.5-3B-Instruct (Ours, w/ Refiner)</b>	3B	<b>0.21/0.19</b>	<b>0.13/0.10</b>	<b>0.21/0.23</b>
397	Qwen2.5-7B-Instruct	7B	0.24/0.25	0.19/0.18	0.25/0.18
398	<b>Qwen2.5-7B-Instruct (Ours, w/ Refiner)</b>	7B	<b>0.27/0.28</b>	<b>0.21/0.22</b>	<b>0.29/0.31</b>
399	Qwen3-1.7B	1.7B	0.17/0.15	0.11/0.12	0.12/0.20
400	<b>Qwen3-1.7B (Ours, w/ Refiner)</b>	1.7B	<b>0.21/0.22</b>	<b>0.13/0.15</b>	<b>0.19/0.31</b>
401	Qwen3-8B	8B	0.29/0.25	0.23/0.21	0.25/0.33
402	<b>Qwen3-8B (Ours, w/ Refiner)</b>	8B	<b>0.30/0.29</b>	<b>0.23/0.22</b>	<b>0.29/0.41</b>

399 Table 3: Results on search benchmarks  
400

	Musique	$\Delta_{Musique}$	Bamboogle	$\Delta_{Bamboogle}$
<i>Qwen2.5-3B-Instruct</i>				
RAG	0.047/0.094	-	0.080/0.324	-
Search-R1	0.103	<b>+119.15%</b>	0.264	<b>+230.00%</b>
<b>Ours w/ Refiner</b>	0.103	+9.57%	0.352	+8.64%
<i>Qwen2.5-7B-Instruct</i>				
RAG	0.058/0.129	-	0.208/0.376	-
Search-R1	0.146	<b>+151.72%</b>	0.368	<b>+76.92%</b>
<b>Ours w/ Refiner</b>	0.144	+11.63%	0.408	+8.51%
<i>Llama-3.2-3B-Instruct</i>				
RAG	0.052	-	0.212	-
<b>Ours w/ Refiner</b>	0.060	<b>+15.38%</b>	0.232	<b>+9.43%</b>
<i>Llama-3.1-8B-Instruct</i>				
RAG	0.119	-	0.324	-
<b>Ours w/ Refiner</b>	0.130	<b>+9.24%</b>	0.328	<b>+1.23%</b>
<i>Qwen3-1.7B</i>				
RAG	0.144	-	0.216	-
<b>Ours w/ Refiner</b>	0.183	<b>+27.08%</b>	0.224	<b>+3.70%</b>
<i>Qwen3-8B</i>				
RAG	0.125	-	0.400	-
<b>Ours w/ Refiner</b>	0.150	<b>+20.00%</b>	0.408	<b>+2.00%</b>

399 Table 4: Quantitative evaluation of correction efficacy (with Bamboogle\*)  
400

Model	Acc <sub>Full</sub>	Acc <sub>Toolname</sub>	Acc <sub>Parameter</sub>
<i>Qwen2.5-3B-Instruct</i>			
CoT	0.00	54.27	52.27
<b>Ours, w/ Refiner</b>	<b>48.13</b>	<b>79.67</b>	<b>79.47</b>
<i>Qwen2.5-7B-Instruct</i>			
CoT	20.87	60.40	60.40
<b>Ours, w/ Refiner</b>	<b>49.13</b>	<b>79.47</b>	<b>79.47</b>
<i>Llama-3.2-3B-Instruct</i>			
CoT	1.60	54.20	45.70
<b>Ours, w/ Refiner</b>	<b>36.53</b>	<b>74.73</b>	<b>73.80</b>
<i>Llama-3.1-8B-Instruct</i>			
CoT	43.07	64.80	60.67
<b>Ours, w/ Refiner</b>	<b>60.27</b>	<b>72.20</b>	<b>72.00</b>
<i>Qwen3-1.7B</i>			
CoT	16.70	62.27	59.40
<b>Ours, w/ Refiner</b>	<b>39.67</b>	<b>68.60</b>	<b>66.80</b>
<i>Qwen3-8B</i>			
CoT	80.33	86.87	86.80
<b>Ours, w/ Refiner</b>	<b>86.60</b>	<b>87.13</b>	<b>87.13</b>

416  
417  
418 4.4 REFINEMENT OF SEARCH TOOL INVOCATIONS  
419420  
421 **Experiment Settings.** To assess whether our Refiner enhances the upstream LLM in retrieval-  
422 augmented scenarios, we conduct evaluations on the Musique and Bamboogle benchmarks. For a  
423 rigorous comparison, we adopt the evaluation protocol of Search-R1, where models generate an-  
424 swers using multi-turn retrieved content. Further details are provided in Appendix E.3.425 Nevertheless, we identify a limitation in the original evaluation script, which relies on exact match  
426 (EM) for assessing correctness. Upon manual inspection, we find that numerous model outputs,  
427 while not identical to the ground-truth, are semantically correct (i.e., one is a substring or superset  
428 of the other). Consequently, we modify the evaluation script to use a containment-based metric.429 The final results are presented in Table 3. Scores are reported in the format of “original / revised”,  
430 representing the performance under the EM and our modified metric, respectively.  $\Delta$  indicates  
431 the percentage improvement over a RAG baseline, with each comparison being made under its  
corresponding evaluation metric.

432 **Performance Enhancement Constrained by the Upstream Model.** As shown in Table 3, our  
 433 Refiner yields performance gains for upstream LLMs across various scales, which validates its ef-  
 434 ficacy in correcting the upstream LLM’s output to generate well-formatted  $\langle\text{search}\rangle$  call requests  
 435 and extract more effective search queries. However, the performance gains are less pronounced  
 436 compared to Search-R1, a method that fine-tunes the upstream LLM itself for search engine utilization.  
 437 We attribute this performance gap to their different training objectives. Search-R1 not only trains the  
 438 model to generate superior search queries but also to effectively utilize the retrieved information. In  
 439 contrast, our Refiner focuses solely on improving the quality of the search invocation, while the up-  
 440 stream LLM remains unoptimized. Consequently, even when provided with higher-quality retrieved  
 441 information, the upstream LLM can not leverage this information well.

#### 442 443 4.5 QUANTITATIVE ANALYSIS OF FORMAT AND CONTENT CORRECTION

444 **Experiment Settings.** Given the well-structured nature of the Bamboogle benchmark, which fea-  
 445 tures predefined retrieval categories and queries limited to celebrity birth information (dates or  
 446 places), we reformulate its search queries into a tool-calling format, yielding the Bamboogle\*  
 447 dataset. This modified dataset enables a quantitative assessment of the model’s tool invocation  
 448 accuracy, which we dissect by individually analyzing the correctness of the tool format, name, and  
 449 parameters. Detailed specifications for this evaluation are provided in Appendix E.4.

450 The evaluation results are presented in Table 4. We report three key metrics: (1)  $\text{Acc}_{\text{Full}}$ : The per-  
 451 centage of predictions where the tool names, parameters, and format are all correct. (2)  $\text{Acc}_{\text{Toolname}}$ :  
 452 The percentage of predictions that contain the correct tools, regardless of its parameters or format.  
 453 (3)  $\text{Acc}_{\text{Parameter}}$ : The percentage of predictions that contain the correct tools and the correct pa-  
 454 rameters, irrespective of the format.

455 **Validating the Refiner’s Contribution to Format and Content Accuracy.** As shown in Table  
 456 4, integrating the Refiner leads to a substantial improvement in  $\text{Acc}_{\text{Full}}$ , demonstrating its high  
 457 effectiveness in format correction. Moreover,  $\text{Acc}_{\text{Toolname}}$  and  $\text{Acc}_{\text{Parameter}}$  also show significant  
 458 gains, which validates that the Refiner can extract more effective information from the user query  
 459 and context, enhancing the overall tool-calling capability of the upstream LLM.

## 461 462 5 OVERVIEW OF TECHNICAL SUPPLEMENT

463 The appendices provide a comprehensive technical supplement to our work. We first detail the input  
 464 prompt design of our Refiner, the training data curation process, and the complete training and eval-  
 465 uation configurations (Appendix A, B, C, D, E). Subsequently, we present rigorous ablation studies  
 466 to demonstrate that the Refiner’s strong performance stems from our proposed training strategy and  
 467 data choices, specifically by: comparing performance against an untrained baseline, ablating the  
 468 overlong penalty, and ablating the use of simplified call sequence data (Appendix F, G, H). Finally,  
 469 we include a disclosure on the use of LLM and provide case studies to substantiate our findings  
 470 (Appendix I, J).

## 471 472 6 CONCLUSION

473 We introduce a novel refinement paradigm centered on a small-scale **Tool-use Refiner**, designed  
 474 to correct tool invocations generated by larger, upstream LLMs. This approach directly addresses  
 475 the pervasive issue of invocation errors in Tool-Integrated Reasoning while circumventing the high  
 476 computational costs and inflexibility associated with directly fine-tuning large-scale LLMs. We  
 477 demonstrate that by training this lightweight refiner using the DAPO algorithm, it can effectively  
 478 learn to identify and rectify a range of invocation errors, including incorrect formats, parameters,  
 479 and sequences. Our comprehensive evaluations across multiple tool-use benchmarks reveal that  
 480 this plug-and-play approach leads to consistent performance gains for various upstream models.  
 481 Notably, the refined outputs not only surpass the raw LLM generations but also outperform several  
 482 direct fine-tuning methods, validating the efficacy of our refinement strategy. This work highlights  
 483 the potential of using smaller, specialized models to augment the capabilities of larger ones, paving  
 484 the way for more robust, adaptable, and deployable AI systems in real-world applications.

486 ETHICS STATEMENT  
487488 This submission does not have any ethics issues to the best of our knowledge.  
489490 REPRODUCIBILITY STATEMENT  
491492 We are committed to the reproducibility of this research. The source code for our experiments is  
493 anonymously available in <https://anonymous.4open.science/r/Tool-use-Refiner-8C2D>. Descriptions  
494 of our methodology, the datasets, and the benchmarks, including all preprocessing procedures, can  
495 be found in the main text and the appendix. Further details on the experimental setup required to  
496 reproduce our results are also provided in the appendix.  
497498 REFERENCES  
499500 Ibrahim Abdelaziz, Kinjal Basu, Mayank Agarwal, Sadhana Kumaravel, Matthew Stallone,  
501 Rameswar Panda, Yara Rizk, GP Bhargav, Maxwell Crouse, Chulaka Gunasekara, et al. Granite-  
502 function calling model: Introducing function calling abilities via multi-task learning of granular  
503 tasks. *arXiv preprint arXiv:2407.00121*, 2024.504 Glaive AI. *glaive-function-calling-v2*, August 2023. URL <https://huggingface.co/datasets/glaiveai/glaive-function-calling-v2>.505 AI@Meta. Llama 3 model card. 2024. URL [https://github.com/meta-llama/llama3/blob/main/MODEL\\_CARD.md](https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md).506 Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey  
507 Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, et al. Nestful: A benchmark for  
508 evaluating llms on nested sequences of api calls. *arXiv preprint arXiv:2409.03797*, 2024.509 Alexander Brinkmann and Christian Bizer. Self-refinement strategies for llm-based product attribute  
510 value extraction. *arXiv preprint arXiv:2501.01237*, 2025.511 Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao.  
512 Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023a.513 Wenhui Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting:  
514 Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.515 Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models  
516 to self-debug. *arXiv preprint arXiv:2304.05128*, 2023b.517 Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and  
518 Feng Zhao. Agent-flan: Designing data and methods of effective agent tuning for large language  
519 models. *arXiv preprint arXiv:2403.12881*, 2024.520 Gheorghe Comanici, Eric Bieber, Mike Schaeckermann, Ice Pasupat, Noveen Sachdeva, Inderjit  
521 Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the  
522 frontier with advanced reasoning, multimodality, long context, and next generation agentic capa-  
523 bilities. *arXiv preprint arXiv:2507.06261*, 2025.524 Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu,  
525 Zhiyong Wu, Tianyu Liu, et al. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*,  
526 2022.527 Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang,  
528 Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms.  
529 *arXiv preprint arXiv:2504.11536*, 2025.530 Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao  
531 Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming—  
532 the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

- 540 Han Han, Tong Zhu, Xiang Zhang, Mengsong Wu, Hao Xiong, and Wenliang Chen. Nestools:  
 541 A dataset for evaluating nested tool learning abilities of large language models. *arXiv preprint*  
 542 *arXiv:2410.11805*, 2024.
- 543 Chi Hu, Yimin Hu, Hang Cao, Tong Xiao, and Jingbo Zhu. Teaching language models to self-  
 544 improve by learning from language feedback. *arXiv preprint arXiv:2406.07168*, 2024.
- 545 Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Os-  
 546 trow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint*  
 547 *arXiv:2410.21276*, 2024.
- 548 Jiaming Ji, Boyuan Chen, Hantao Lou, Donghai Hong, Borong Zhang, Xuehai Pan, Tianyi Alex Qiu,  
 549 Juntao Dai, and Yaodong Yang. Aligner: Efficient alignment by learning to correct. *Advances in*  
 550 *Neural Information Processing Systems*, 37:90853–90890, 2024.
- 551 Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bam-  
 552 ford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al.  
 553 Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.
- 554 Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and  
 555 Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement  
 556 learning. *arXiv preprint arXiv:2503.09516*, 2025.
- 557 Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi  
 558 Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *EMNLP*  
 559 (1), pp. 6769–6781, 2020.
- 560 Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei  
 561 Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv*  
 562 *preprint arXiv:2304.08244*, 2023.
- 563 Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated rl. *arXiv preprint*  
 564 *arXiv:2503.23383*, 2025.
- 565 Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu  
 566 Zhou, Cheng Cheng, Yin Zhao, et al. Hammer: Robust function-calling for on-device language  
 567 models via function masking. *arXiv preprint arXiv:2410.04587*, 2024.
- 568 Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan  
 569 Gan, Zhengying Liu, Yuanqing Yu, et al. Toolace: Winning the points of llm function calling.  
 570 *arXiv preprint arXiv:2409.00920*, 2024.
- 571 Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri  
 572 Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement  
 573 with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.
- 574 Lilian Ngweta, Mayank Agarwal, Subha Maity, Alex Gittens, Yuekai Sun, and Mikhail Yurochkin.  
 575 Aligners: Decoupling llms and alignment. *arXiv preprint arXiv:2403.04224*, 2024.
- 576 Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring  
 577 and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*,  
 578 2022.
- 579 Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan  
 580 Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*,  
 581 2025.
- 582 Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe  
 583 Zhou, Yufei Huang, Chaojun Xiao, et al. Tool learning with foundation models. *ACM Computing*  
 584 *Surveys*, 57(4):1–40, 2024.
- 585 Timo Schick, Jane Dwivedi-Yu, Roberto Dessim, Roberta Raileanu, Maria Lomeli, Eric Hambro,  
 586 Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can  
 587 teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–  
 588 68551, 2023.

- 594 John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy  
 595 optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.  
 596
- 597 Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang,  
 598 Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathemati-  
 599 cal reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- 600 Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang,  
 601 and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement  
 602 learning. *arXiv preprint arXiv:2503.05592*, 2025.
- 603
- 604 Qwen Team. Qwen3 technical report, 2025. URL <https://arxiv.org/abs/2505.09388>.
- 605
- 606 Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop  
 607 questions via single-hop question composition. *Transactions of the Association for Computational  
 608 Linguistics*, 10:539–554, 2022.
- 609
- 610 Tu Vu, Mohit Iyyer, Xuezhi Wang, Noah Constant, Jerry Wei, Jason Wei, Chris Tar, Yun-Hsuan  
 611 Sung, Denny Zhou, Quoc Le, et al. Freshllms: Refreshing large language models with search  
 612 engine augmentation. *arXiv preprint arXiv:2310.03214*, 2023.
- 613
- 614 Hongru Wang, Yujia Qin, Yankai Lin, Jeff Z Pan, and Kam-Fai Wong. Empowering large language  
 615 models: Tool learning for real-world interaction. In *Proceedings of the 47th International ACM  
 616 SIGIR Conference on Research and Development in Information Retrieval*, pp. 2983–2986, 2024.
- 617
- 618 Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi  
 619 Song, Mingjie Zhan, and Hongsheng Li. Mathcoder: Seamless code integration in llms for en-  
 620 hanced mathematical reasoning. *arXiv preprint arXiv:2310.03731*, 2023.
- 621
- 622 Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Dixin Jiang, Rangan Ma-  
 623 jumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv  
 624 preprint arXiv:2212.03533*, 2022.
- 625
- 626 Zhengyuan Yang, Jianfeng Wang, Linjie Li, Kevin Lin, Chung-Ching Lin, Zicheng Liu, and Lijuan  
 627 Wang. Idea2img: Iterative self-refinement with gpt-4v for automatic image design and generation.  
 628 In *European Conference on Computer Vision*, pp. 167–184. Springer, 2024.
- 629
- 630 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao.  
 631 React: Synergizing reasoning and acting in language models. In *International Conference on  
 632 Learning Representations (ICLR)*, 2023.
- 633
- 634 Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian  
 635 Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system  
 636 at scale. *arXiv preprint arXiv:2503.14476*, 2025.
- 637
- 638 Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhui Chen.  
 639 Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint  
 640 arXiv:2309.05653*, 2023.
- 641
- 642 Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttun-  
 643 ning: Enabling generalized agent abilities for llms. *arXiv preprint arXiv:2310.12823*, 2023.
- 644
- 645 Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao  
 646 Tan, Akshara Prabhakar, Haolin Chen, et al. xlam: A family of large action models to empower  
 647 ai agent systems. *arXiv preprint arXiv:2409.03215*, 2024.

648 APPENDIX  
649650 We present further details and experimental results in the appendix, organized as follows:  
651652 **1. Supplementary Model Details and Data Curation:**

- 653
- 654 • Appendix A. **Crafting the Input Prompt for Tool-use Refiner:** We present the template  
655 and format of the input prompt for our Tool-use Refiner.
  - 656 • Appendix B. **Examples from the Source Tool-calling Datasets:** We present examples  
657 from the source tool-calling datasets, including Glaive, ToolACE, xLAM, and NesTools.
  - 658 • Appendix C. **Crafting the Training Datasets for Tool-use Refiner:** We detail the process  
659 of creating the training datasets for our Tool-use Refiner.

660 **2. Experimental Setup and Training Details:**  
661

- 662
- 663 • Appendix D. **Training Details:** We present the training configurations for the Tool-use  
664 Refiner used in our main experiment, along with its reward and response length curves  
665 during the training process.
  - 666 • Appendix E. **Evaluation Setup:** We present our evaluation setup, detailing our assessments  
667 on refining general-purpose tool invocations, sequential errors, and search tool invocations,  
668 alongside a quantitative correction analysis.

669 **3. Model Performance and Ablation Studies:**  
670

- 671
- 672 • Appendix F. **Performance Comparison with an Untrained Refiner:** We compare our  
673 Tool-use Refiner against an untrained counterpart (Qwen3-1.7B), demonstrating that its  
674 strong performance is attributable to our RL training for tool invocation correction, rather  
675 than the inherent capabilities of the base model.
  - 676 • Appendix G. **An Ablation Study of the Overlong Penalty:** We conduct an ablation study  
677 on the overlong penalty, highlighting its necessity for guiding the Refiner to produce con-  
678 cise yet effective reasoning.
  - 679 • Appendix H. **An Ablation Study of the Simplified Call Sequence Data:** We conduct  
680 an ablation study on the simplified call sequence data, demonstrating that this data yields  
681 a more stable Refiner with improved performance on both general tool-use and specific  
682 tool-order correction tasks.

683 **4. Supplementary Materials and Disclosure:**

- 684
- 685 • Appendix I. **The Use of Large Language Models (LLMs):** We disclose the use of LLM  
686 within our writing process.
  - 687 • Appendix J. **Case Studies:** We present representative case studies to substantiate the find-  
688 ings from our comparative evaluation.

689 **A CRAFTING THE INPUT PROMPT FOR TOOL-USE REFINER**  
690691 This section outlines the structure of the input prompt for our Tool-use Refiner.  
692693 The input begins with a comprehensive system prompt designed to define the Refiner's role and  
694 operational parameters. First, the system prompt explicitly instructs the Tool-use Refiner that its  
695 primary function is to identify and rectify errors within tool invocations generated by other models.  
696 Furthermore, it details the expected input prompt format and enumerates the potential categories  
697 of errors the refiner may encounter. The complete System Prompt utilized in our experiments is  
698 presented below:699 *Your role is a highly professional tool-calling instruction corrector. Your primary task is to identify  
700 and fix errors in tool-calling instruction generated by other models. Note that the original tool-  
701 calling instruction from another model may already be entirely correct - in such cases, you do not  
need to make any modifications. You will be provided with: the list of available tools, the user's*

702 original query and the tool-calling instruction generated by another model. Your responsibility is  
 703 to carefully analyze this information and output only the corrected, properly formatted tool-calling  
 704 instruction and the special tags. When identifying format inaccuracies, character errors, special  
 705 token mistakes, or tool-calling order errors, confidently implement corrections. If you identify any  
 706 errors in the tool-calling instruction from another model, correct it and output only the fixed, correct  
 707 tool-calling instruction and the special tags.

708 Subsequently, for each call of the Refiner, we provide the system prompt for the current task, the  
 709 user query for the current sample, and the potentially erroneous tool invocations generated by the  
 710 upstream LLMs. The specific input is structured as follows:

711 *The original system prompt and the list of available tools are: “SYSTEM”. The user’s original query  
 712 is: “QUERY”. The tool-calling instruction generated by another model is: “UPSTREAM”. Please  
 713 correct the instruction from another model and output only the correct tool-calling instruction.*

714 Within this template, the placeholders are defined as follows: SYSTEM refers to the original system  
 715 prompt of the task, QUERY contains the user query for the current sample, and UPSTREAM holds  
 716 the tool invocations from the upstream LLM. It is important to note that while this structured format  
 717 is strictly adhered to during the training phase, we allow for minor modifications during inference.  
 718 Such adjustments, tailored to specific scenarios, can be implemented to obtain optimal performance.

## 720 B EXAMPLES FROM THE SOURCE TOOL-CALLING DATASETS

### 721 B.1 GLAIVE

722 **The tool invocation format of the glaive dataset is:** `<functioncall> {“name”: function_name,  
 723 “arguments”: {param_name: param_value, param_name2: param_value2, …}’} <|endoftext|>`.  
 724 Each invocation typically calls only one tool at a time.

725 **An example of a system prompt is:** *You are a helpful assistant with access to the following func-  
 726 tions. Use them if required - {“name”: “generate\_password”, “description”: “Generate a random  
 727 password”, “parameters”: {“type”: “object”, “properties”: {“length”: {“type”: “integer”,  
 728 “description”: “The length of the password”}, “include\_symbols”: {“type”: “boolean”, “de-  
 729 scription”: “Whether to include symbols in the password”}}, “required”: [“length”]} {“name”:  
 730 “create\_task”, “description”: “Create a new task in a task management system”, “parameters”:  
 731 {“type”: “object”, “properties”: {“title”: {“type”: “string”, “description”: “The title of the  
 732 task”}, “due\_date”: {“type”: “string”, “format”: “date”, “description”: “The due date of the  
 733 task”}, “priority”: {“type”: “string”, “enum”: [“low”, “medium”, “high”], “description”: “The  
 734 priority of the task”}}, “required”: [“title”, “due\_date”, “priority”]}.*

735 **An example of a user query (including conversations) is:** *USER: I need a new password. Can  
 736 you generate one for me? ASSISTANT: Of course. How long would you like your password to be?  
 737 And would you like it to include symbols? <|endoftext|> USER: I would like it to be 12 characters  
 738 long and yes, please include symbols.*

739 **An example of a correct tool invocation is:** `<functioncall> {“name”: “generate_password”,  
 740 “arguments”: {“length”: 12, “include_symbols”: true}’} <|endoftext|>`

### 741 B.2 TOOLACE

742 **The tool invocation format of the ToolACE dataset is:** `[func1(params_name=params_value,  
 743 params_name2=params_value2…), func2(params)]`. This is a list containing Python functions. It’s  
 744 worth noting that some function names in the dataset contain spaces, which could not be properly  
 745 parsed. Such samples are filtered out during preprocessing.

746 To train the Refiner to reconstruct missing special tags, we enclose the ToolACE format with the  
 747 tags `<function_list>` and `</function_list>` at the beginning and end, respectively.

748 **An example of a system prompt is:** *You are an expert in composing functions. You are given  
 749 a question and a set of possible functions. Based on the question, you will need to make one or  
 750 more function/tool calls to achieve the purpose. If none of the function can be used, point it out.  
 751 If the given question lacks the parameters required by the function, also point it out. You should*

only return the function call in tools call sections. Here is a list of functions in JSON format that you can invoke: [ { "name": "Financial\_Fundamentals\_API", "description": "Retrieves the profitability (ROA ratio) for a specified financial year of a specific share.", "parameters": { "type": "dict", "properties": { "shareuid": { "description": "Unique identifier for the share searched", "type": "int" }, "from": { "description": "Start string of the searched period in American notation year-month-day with leading 0", "type": "string" }, "to": { "description": "End string of the searched period in American notation year-month-day with leading 0", "type": "string" } } }, "required": [ "shareuid", "from", "to" ], "required": null }, { "name": "Get\_Supported\_Currencies", "description": "Retrieve a list of supported currencies by Coinbase.", "parameters": { "type": "dict", "properties": {}, "required": [] }, "required": null }, { "name": "watchlists", "description": "Returns a list of private watchlists for the authenticating user.", "parameters": { "type": "dict", "properties": { "callback": { "description": "Define your own callback function name, add this parameter as the value.", "type": "string", "default": "" } } }, "required": [ "callback" ], "required": null } ]. Should you decide to return the function call(s). Put it in the format of [func1(params\_name=params\_value, params\_name2=params\_value2...), func2(params)]. NO other text MUST be included.

**An example of a user query is:** *Can you please provide the ROA ratio for the company with shareuid 6789 for the last financial year?*

774 An example of a correct tool invocation is: `[{"name": "Financial_Fundamentals_API", "re-`  
775 `sults": {"roa_ratio": 0.123, "financial_year": "2025"}}]`

777 B.3 xLAM

**The tool invocation format of the xLAM dataset is:** `[{"name": function_name, "arguments": {"params_name: params_value, params_name2: params_value2..."}}, ...]`. This is a list containing multiple Python dictionaries.

782 To train the Refiner to reconstruct missing special tags, we enclose the xLAM format with the tags  
783 <func\_call> and </func\_call> at the beginning and end, respectively.

784 **An example of a available tool list is:** [{"name": "live\_giveaways\_by\_type", "description":  
785 "Retrieve live giveaways from the GamerPower API based on the specified type.", "parameters":  
786 {"type": {"description": "The type of giveaways to retrieve (e.g., game, loot, beta).", "type":  
787 "str", "default": "game"} } }].

**An example of a user query is:** *Where can I find live giveaways for beta access and games?*

900 An example of a correct tool invocation is: `[{"name": "live_giveaways_by_type", "arguments": 901 {"type": "beta"}}, {"name": "live_giveaways_by_type", "arguments": {"type": "game"}}]`.  
91

793 B.4 NESTTOOLS

795 The tool invocation format of the NesTools dataset is:

796    [{"api\_name": "...", "parameters": {"arg0": "value0", "arg1": "value1", ...}, "responses":  
797    [{"API\_call\_0", ... , "API\_call\_n"}]}, {"api\_name": "...", "parameters": {"arg0": "value0", "arg1": "value1", ...}, "responses":  
798    [{"API\_call\_{n+1}", ...}], ...], ...].

799 This format captures sequential information, with the “step” field indicating the order in which the  
800 tool is called. Additionally, the output of each tool invocation is represented using API call\_x.

To train the Refiner to reconstruct missing special tags, we enclose the NesTools format with the tags `<nested function>` and `</nested function>` at the beginning and end, respectively.

804 **An example of a available tool list is:** [ {“api\_name”: “scan\_isbn”, “api\_description”: “Scan  
805 the ISBN of a book to retrieve information.”, “parameters”: {“isbn”: {“type”: “str”, “descrip-  
806 tion”: “the ISBN code of the book”}}, “required”: [“isbn”], “responses”: {“book\_details”:  
807 {“type”: “str”, “description”: “detailed information about the book”}, “availability”: {“type”:  
808 “bool”, “description”: “availability status of the book in the library”}}}, {“api\_name”:  
809 “locate\_book”, “api\_description”: “Locate the physical book in the library.”, “parameters”: {“book\_info”: {“type”: “str”, “description”: “detailed information about the book”}}}, “re-

810        `quired": [{"book_info": "responses": {"location_desc": {"type": "str", "description": "de-"},`  
 811        `scription of the exact location within the library"}]}, {"api_name": "engage_ar_experience",`  
 812        `"api_description": "Activate an augmented reality experience related to the book.", "par-"},`  
 813        `ameters": {"availability": {"type": "bool", "description": "availability status of the book"},`  
 814        `"exact_location": {"type": "str", "description": "exact location of the book"}}, "required":`  
 815        `["availability", "exact_location"], "responses": {"ar_message": {"type": "str", "description": "mes-"},`  
 816        `age to explain the AR experience"}, "ar_duration": {"type": "int", "description": "esti-"},`  
 817        `mated duration of the AR experience in minutes"}]}].`

818        **An example of a user query is:** Scan the ISBN “978-3-16-148410-0” of a book to extract comprehensive data and verify its presence in the library. Upon identifying the book situated, launch an augmented reality interaction correlating to the book.

819        **An example of a correct tool invocation is:** `[ {"api_name": "scan_isbn", "parameters":`  
 820        `{"isbn": "978-3-16-148410-0"}, "responses": ["API_call_0", "API_call_1"]}, {"api_name":`  
 821        `"locate_book", "parameters": {"book_info": "API_call_0"}, "responses": ["API_call_2"]},`  
 822        `{"api_name": "engage_ar_experience", "parameters": {"availability": "API_call_1", "ex-"},`  
 823        `act_location": "API_call_2"}, "responses": ["API_call_3", "API_call_4"]}]`.

## 824        C CRAFTING THE TRAINING DATASETS FOR TOOL-USE REFINER

### 825        C.1 GENERATION OF ERRONEOUS TOOL INVOCATIONS

826        To construct a training dataset of erroneous tool invocations for our Tool-use Refiner, we systematically generate flawed data by leveraging five LLMs with diverse scales and performance characteristics: Qwen3-1.7B, Qwen3-8B, Qwen3-14B (Team, 2025), Llama-3.1-8B-Instruct, and Llama-3.2-3B-Instruct (AI@Meta, 2024). The core strategy is to instruct these models to deliberately introduce errors into tool invocations, focusing on three specific types of corruption: incorrect formatting, incorrect tool or parameter names, and incorrect parameter content.

827        Our generation pipeline involves a two-step process. In the first step, we have each LLM generate an initial response to the original problem, yielding a set of tool invocations which we refer to as the “original answer.” These initial generations might be correct or faulty, reflecting the natural capabilities of each model. In the second step, we reframe the task. Instead of generating a correct answer, the models are prompted to intentionally modify the “original answer” to introduce a specific error. The instructions provided are as follows, wherein the term “refined answer” refers to correct tool invocations:

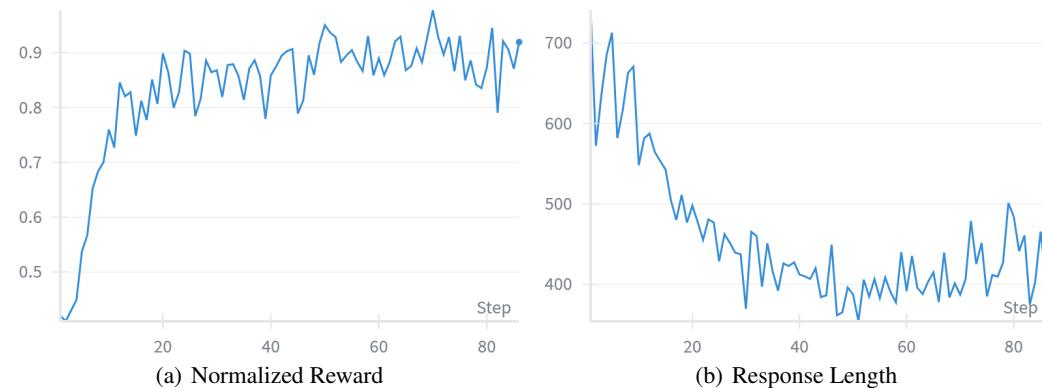
- 828        **• For Incorrect Format:** *Your task is to modify the ‘original answer’ below so that the tool*  
 829        *call format differs from the ‘refined answer’ and becomes incorrect. The ‘original answer’*  
 830        *is: “ORIGINAL”. You only need to output your modified ‘original answer’.*
- 831        **• For Incorrect Tool/Parameter Name:** *Your task is to modify the ‘original answer’ below*  
 832        *so that the tool names or tool call parameter names differ from the ‘refined answer’ and*  
 833        *become incorrect. The ‘original answer’ is: “ORIGINAL”. You only need to output your*  
 834        *modified ‘original answer’.*
- 835        **• For Incorrect Parameter Content:** *Your task is to modify the ‘original answer’ below so*  
 836        *that the tool call parameter content differs from the ‘refined answer’ and becomes incorrect.*  
 837        *The ‘original answer’ is: “ORIGINAL”. You only need to output your modified ‘original*  
 838        *answer’.*

839        This process yields a collection of tool invocations with intentionally injected errors from each of the five LLMs. From this collection, we curate our final dataset by randomly sampling 150 instances for each of the three error types from each model. To ensure diversity, each set of 150 samples is composed of 50 instances originating from the glaive dataset, 50 from ToolACE, and 50 from xLAM. This strategy produces a total of 2,250 erroneous tool invocation samples, which simulate the outputs of an upstream LLM in our training framework.

864 C.2 GENERATION OF RANDOMIZED CALL SEQUENCES  
865866 To train the Refiner’s capability in correcting tool call sequences, we leverage nested tool data  
867 (NesTools) and generate a training set of randomized call sequences by permuting the ground-truth  
868 solutions. We create two distinct batches of this data: one in a full format and another in a simplified  
869 format.870 The full-format batch adheres to the original NesTools data structure, requiring the model to output  
871 the tool name, its parameters, and a temporary placeholder for the output (e.g., API\_call\_x). Con-  
872 versely, the simplified-format batch only requires the model to output the name of the tool to be  
873 called at each step, omitting parameters and placeholders. The introduction of the simplified for-  
874 mat constitutes a form of curriculum learning, enabling the model to first master the simpler task of  
875 sequence correction before progressing to the more complex task.876 Specifically, to generate the full-format randomized call sequences, we employ Qwen3-14B to inject  
877 sequential errors into the original NesTools data using the following prompt:878 *Your task is to modify the “original answer” below so that the tool call order becomes incorrect.  
879 The ‘original answer’ is: “ORIGINAL”. You only need to output your modified ‘original answer’.*880 For the simplified-format randomized call sequences, we first convert the NesTools data into a  
881 streamlined structure: `<order_func>[{"step": 1, "tool_list": [tool_name1, tool_name2, ...]},`  
882 `{ "step": 2, "tool_list": [tool_name3, tool_name4, ...]}, ...]< /order_func>`. Subsequently, we  
883 utilize Qwen3-14B to introduce errors into this simplified data with the following prompt:884 *Your task is to modify the “original answer” below so that the tool call order and the number of  
885 steps becomes incorrect. This involves either splitting tools from a single step into multiple steps  
886 (e.g., converting [{"step": 1, "tool\_list": [tool\_name1, tool\_name2]}] into [{"step": 1, "tool\_list":  
887 [tool\_name1]}, {"step": 2, "tool\_list": [tool\_name2]}]) or merging/redistributing tool names from  
888 different steps into incorrect step groupings. The ‘original answer’ is: “ORIGINAL”. You only need  
889 to output your modified ‘original answer’.*890 A key outcome of this generation process is that the synthesized data inherently includes a mix of  
891 correct sequences, incorrect sequences, and malformed instances. This diverse composition per-  
892 fectly suits our requirement for a challenging training corpus. Each of the two resulting datasets  
893 contains a total of 1,000 samples.894 D TRAINING DETAILS  
895

## 896 900 D.1 TRAINING CONFIGURATIONS

901 For the training phase, we configure the following hyperparameters. The learning rate is set to  
902 1e-6, and the Refiner is trained for a single epoch. We use a maximum context window of 4096  
903 tokens. The clipping ratio for the optimization algorithm is configured with a lower clipping range  
904 (clip\_ratio\_low) of 0.2 and a higher clipping range (clip\_ratio\_high) of 0.28.905 The training batch size is set to 16. For each sample in a batch, 16 corresponding responses are  
906 generated. To ensure the quality of training instances, DAPO employ a dynamic sampling strategy  
907 that filters out samples providing no learning signal. Following this, to form a complete batch of 16  
908 qualified samples, the trainer could perform up to 48 sampling attempts.909 Furthermore, an overlong response punishment mechanism is implemented to discourage exces-  
910 sively long generations. Responses exceeding 1024 tokens are penalized, with the penalty value  
911 increasing linearly from 0 (at 1024 tokens) to a maximum of 1 (at 4096 tokens).912 914 D.2 TRAINING TIME AND RESOURCE CONSUMPTION  
915916 917 We trained the Tool-use Refiner (for our main experiments) on four NVIDIA RTX A5000 GPUs.  
918 The entire training process lasted for 6 hours and occupied 22GB of VRAM on each GPU.

918 D.3 TRAINING RESULTS  
919920 Figure 2 illustrates the training dynamics of the Tool-use Refiner used in our main experiments,  
921 plotting the changes in reward and average response length.  
922936  
937 Figure 2: Training dynamics of the Tool-use Refiner.  
938939 As shown in Figure 2(a), the reward score exhibits a consistent upward trend, rising from 0.4 and  
940 stabilizing at approximately 0.9. This serves as strong evidence that our proposed reward function  
941 successfully guides the Refiner to improve its proficiency in correcting tool invocations.942 Furthermore, Figure 2(b) highlights a substantial reduction in output verbosity. The average re-  
943 sponse length, inclusive of the reasoning tokens, drops from 741 to around 400. This trend towards  
944 more succinct outputs is crucial as it mitigates the risk of exceeding the context window limit.  
945946 E EVALUATION SETUP  
947948 E.1 EVALUATION SETUP FOR REFINING GENERAL-PURPOSE TOOL INVOCATION  
949950 E.1.1 API-BANK BENCHMARK SPECIFICATION  
951952 API-Bank is a tool invocation benchmark featuring three levels of difficulty: Level 1 (tool name is  
953 apparent in the dialogue), Level 2 (tool name requires retrieval from an available tool list), and Level  
954 3 (sustained tool calls are required).955 For our evaluation, we utilize the API-bank dataset provided by the official ToolRL repository. This  
956 dataset has been pre-processed into a multi-turn dialogue format. In this setup, the model's objective  
957 is to select the appropriate tools to address the user's final request at the conclusion of the dialogue.  
958 The list of available tools is provided to the model within the system prompt. The difficulty of  
959 each task is stratified based on two factors: (1) the number of tools available in the prompt, and (2)  
960 the presence of hints or cues provided to the model during the conversational turns. We measure  
961 performance using exact match accuracy, which requires the model's predicted tool invocations to  
962 be identical to the ground truth.  
963964 E.1.2 BASELINES FOR COMPARISON  
965966 In this subsection, we detail the training procedures for the various baselines compared in our eval-  
967 uation on API-bank, following the experimental setup proposed in the ToolRL (Qian et al., 2025).968 First, a balanced training dataset is curated to ensure exposure to a wide range of tasks. Specifically,  
969 2,000 samples are extracted from the ToolACE (Liu et al., 2024) dataset, and 1,000 samples are  
970 drawn from each of the Hammer (Lin et al., 2024) and xLAM (Zhang et al., 2024) datasets. This  
971 combination yields a total of 4,000 training instances, creating a balanced dataset that spans diverse  
levels of complexity and tool use.

The methods listed in Table 1 represent different stages and strategies of model training. The **Raw** method serves as our baseline, representing the original model’s performance without any task-specific fine-tuning or reinforcement learning (RL). The **SFT** model is an instruction-tuned model that has been further fine-tuned on 400 data points randomly sampled from the aforementioned training data. Building upon this, the **SFT+PPO** and **SFT+GRPO** methods take the SFT model as a starting point and apply the PPO or GRPO RL algorithms, respectively, using the entire training dataset. In contrast, the **PPO Cold Start** method bypasses the SFT step, applying the PPO algorithm directly to the original model. Finally, the **ToolRL** method involves training the original model from scratch with the GRPO algorithm on all available training data.

### E.1.3 ADAPTING THE REFINER FOR THE API-BANK BENCHMARK

The API-Bank benchmark presents a unique formatting requirement where models must simultaneously generate reasoning (<think>), tool invocations (<tool\_call>), and a final response (<response>). This multi-part structure is inconsistent with our Refiner’s original training objective, creating ambiguity regarding the placement of the final response. To address this, we re-task the Refiner to focus exclusively on correcting tool calls. We provide it with a specific instructional prompt:

*As a tool-calling instruction corrector, you do not need to output the information related to “;response;”. Please correct the instructions from another model and output only the correct tool-calling instructions. Your output must strictly follow this format: ‘<tool\_call>{“name”: “Tool name”, “parameters”: {“Parameter name”: “Parameter content”, “...”: “...”}}{“name”: “...”, “parameters”: {“...”: “...”, “...”: “...”}}...</tool\_call>’. If the original instructions from another LLM are correct, do not make any modifications and directly output these instruction. If you determine that additional instructions are necessary, append them directly after the original instructions from another LLM.*

Furthermore, we identify a logic error in the benchmark’s ground truth: for certain samples, the task requires the model to generate parameters of int or float types, whereas the ground-truth parameters are formatted as string type. This inconsistency leads to an inaccurate assessment, so we have rectified this issue in our evaluation. We add an instruction to align the Refiner’s output with the ground truth format: “For any numeric parameters you generate (including int and float types), please enclose them in quotes to output them as strings.” These adjustments align the Refiner’s behavior with the benchmark’s specific requirements, ensuring a more accurate evaluation.

### E.2 EVALUATION SETUP FOR REFINING SEQUENTIAL ERRORS

Our evaluation is conducted on the NESTFUL benchmark (Basu et al., 2024), which is designed to assess the ability to execute a logical sequence of tool invocations for multi-step problem-solving. A core challenge within this benchmark is the management of intermediate results: models are required to store the output of a tool call in a temporary variable (e.g., var\_0) and subsequently use this variable as an argument in a later tool call. This process tests the model’s capacity for multi-step reasoning and planning, mirroring complex tasks that involve sequential manipulation, such as chained mathematical calculations.

Following the NESTFUL paper, we employ a standardized prompt format: “**SYSTEM**: You are a helpful assistant with access to the following function calls. ... <|function\_call\_library|>{**TOOL\_LIBRARY**} Here are some examples:{**ICL\_EXAMPLES**} **USER**: {**QUERY**} **ASSISTANT**:.”

Each prompt begins with a system message defining the model’s role as an assistant with access to a specific function library, which is provided under the {TOOL\_LIBRARY} placeholder. Consistent with the NESTFUL paper, we include three in-context learning examples, denoted by {ICL\_EXAMPLES}, to guide the model’s response generation (Dong et al., 2022). The prompt concludes with the specific user task, presented as the {QUERY}. The model is then expected to generate the response which contains the sequence of tool calls required to fulfill the user’s request.

The model’s output is a JSON-formatted list of tool calls, where each call specifies the tool name, its arguments, and a critical label parameter (e.g., \$var\_1) that assigns the tool’s return value to a variable. Our evaluation protocol rigorously verifies the correctness of this entire execution chain.

1026 We sequentially execute each tool call generated by the model, computing the value of each intermediate  
 1027 variable in the specified order. The final computed result is then compared against the ground  
 1028 truth answer for the sample. Performance is measured by the Win Rate, which is the percentage of  
 1029 tasks where the model’s final result exactly matches the ground truth.  
 1030

### 1031 E.3 EVALUATION SETUP FOR REFINING SEARCH TOOL INVOCATIONS

1033 Following the Search-R1 paper, we first deploy a search engine that utilizes the 2018 Wikipedia  
 1034 dump (Karpukhin et al., 2020) as the knowledge corpus and E5 (Wang et al., 2022) as the retriever.  
 1035 During inference, the model can iteratively issue a “<search>” tag to query the corpus when it  
 1036 deems more information is necessary. In response, the search engine returns the top-3 most relevant  
 1037 documents. This process repeats until the model has sufficient information to generate a final  
 1038 response, which is then enclosed within an “<answer>” tag.  
 1039

1040 All tasks are structured using the following prompt format: *Answer the given question. You must*  
*1041 conduct reasoning inside <think> and </think> first every time you get new information. After*  
*1042 reasoning, if you find you lack some knowledge, you can call a search engine by <search> query </search>, and it will return the top searched results between <information> and </information>.*  
*1043 You can search as many times as you want. If you find no further external knowledge needed, you*  
*1044 can directly provide the answer inside <answer> and </answer> without detailed illustrations.*  
*1045 For example, <answer> xxx </answer>. Question: question.*

1046 Building on the paradigm of the Search-R1 paper, we implement an evaluation framework that  
 1047 facilitates multi-turn retrieval, with a maximum of three turns allowed per sample. Our Tool-use  
 1048 Refiner is activated when the upstream LLM fails to produce a conclusive “<answer>” tag, either  
 1049 by generating a new “<search>” tag or by exceeding the context window limit: (1) If the upstream  
 1050 LLM generates a “<search>” tag, the Refiner processes the full generation history and the current  
 1051 output to produce a revised search query. (2) If the upstream LLM exceeds the context limit, the  
 1052 Refiner is given the history concatenated with the first 500 tokens of the current output to generate  
 1053 an appropriate search query.  
 1054

### 1055 E.4 EVALUATION SETUP FOR QUANTITATIVE CORRECTION ANALYSIS

1056 This evaluation is conducted on the Bamboolle benchmark, which comprises 17 distinct retrieval  
 1057 tasks. A key characteristic of this benchmark is that all queries revolve around retrieving temporal  
 1058 or geographical information based on the birth dates or birthplaces of celebrities. Leveraging this  
 1059 structure, we reformulate the retrieval tasks into a tool-calling framework. This involves converting  
 1060 each of the 17 retrieval tasks into a distinct API. Additionally, we create foundational APIs to first  
 1061 retrieve a celebrity’s birth date or birthplace. Consequently, the model’s task is transformed from  
 1062 direct question-answering to generating the correct sequence of API calls to solve the problem.  
 1063

1064 For instance, a foundational API to retrieve a celebrity’s birthplace is defined as: {“name”:  
 1065 “find\_birthplace”, “description”: “Returns the birthplace information of a celebrity.”, “parameters”: {“celebrity\_name”: {“type”: “string”, “description”: “One of the input parameters: the  
 1066 name of the celebrity.”}}, “output\_name”: “output\_birthplace”}.

1067 An example of a secondary tool that requires the output of the above api is find\_capital: {“name”:  
 1068 “find\_capital”, “description”: “Returns a country’s capital.”, “parameters”: {“information\_type”:  
 1069 {“type”: “string”, “description”: “One of the input parameters, applicable information for  
 1070 search, limited to either ‘birthplace’, ‘birthdate’ or ‘birthyear’.”}, “information\_content”: {“type”:  
 1071 “string”, “description”: “Stores the specific content provided to the tool (can use codes starting  
 1072 with ‘output\_’.”}}, “output\_name”: “output\_capital”}.

1073 Furthermore, to assess the model’s ability to adhere to specific formatting instructions, we introduce  
 1074 three distinct tool invocation formats. For each sample, the model is randomly prompted to generate  
 1075 its response in one of the following formats:  
 1076

- 1077 • <tool\_call> [{“name”: “tool\_name1”, “arguments”: {“parameter\_name1”: “pa-  
 1078 rameter\_content1”, “parameter\_name2”: “parameter\_content2”, ...}}, {“name”:  
 1079 “tool\_name2”, “arguments”: {“parameter\_name3”: “parameter\_content3”, ...}}, ...]</tool\_call>.

- 1080     •  $[func1(params\_name=params\_value, params\_name2=params\_value2...), func2(params)]$ .  
 1081  
 1082     •  $[ \{ "type": "tool\_use", "name": "tool\_name1", "input": \{ "parameter\_name1": "parameter\_content1", "parameter\_name2": "parameter\_content2", ... \}, \{ "type": "tool\_use", "name": "tool\_name2", "input": \{ "parameter\_name3": "parameter\_content3", ... \}, ... \} ]$ .
- 1083  
 1084  
 1085  
 1086

1087     This controlled setup, where the required format, tool names, and parameter contents are all pro-  
 1088     grammatically parsable from the model’s output, enables a quantitative, multi-faceted assessment  
 1089     of a model’s ability to generate syntactically valid and semantically correct tool invocations under  
 1090     diverse constraints. Moreover, this framework provides an ideal testbed for evaluating the correction  
 1091     capabilities of our Tool-use Refiner.

1092  
 1093     F PERFORMANCE COMPARISON WITH AN UNTRAINED REFINER  
 1094  
 1095

1096     In this section, we present an ablation study to isolate the contribution of our RL training process.  
 1097     We evaluate the performance of upstream models when using the original Qwen3-1.7B as a tool-use  
 1098     corrector. This allows us to disentangle the improvements stemming from our RL training from the  
 1099     pre-existing capabilities of Qwen3-1.7B itself.

1100     Following the protocol of our main evaluations, we quantitatively evaluate the corrected tool in-  
 1101     vocations. Specifically, we use the modified Bamboogle benchmark to assess accuracy across three  
 1102     metrics:  $Acc_{Full}$ ,  $Acc_{Toolname}$ ,  $Acc_{Parameter}$ . Furthermore, we employ the NESTFUL benchmark  
 1103     to evaluate the sequential accuracy (with CoT). The detailed results are presented in Table 5.

1104

1105     Table 5: Performance comparison with an untrained Refiner (w/ Raw Refiner)  
 1106

1107  
 1108     \begin{table}[h]  
 1109         \begin{tblr}{c c c c c c c}  
 1110         \hline  
 1111         Model & & \multicolumn{2}{c}{Bamboogle\*} & & \multicolumn{2}{c}{NESTFUL} & \\  
 1112         & Acc. &  $Acc_{Toolname}$  &  $Acc_{Parameter}$  & Part. Acc. & Full Acc. & Win Rate & \\  
 1113         \hline  
 1114         \textit{Qwen2.5-3B-Instruct} & & & & & & & \\  
 1115         CoT & 0.00 & 54.27 & 52.27 & 0.10 & 0.04 & 0.13 & \\  
 1116         w/ Raw Refiner & 14.13 & 56.33 & 55.53 & 0.15 & 0.08 & 0.22 & \\  
 1117         \textbf{Ours, w/ Refiner} & \textbf{48.13} & \textbf{79.67} & \textbf{79.47} & \textbf{0.19} & \textbf{0.10} & \textbf{0.23} & \\  
 1118         \hline  
 1119         \textit{Qwen2.5-7B-Instruct} & & & & & & & \\  
 1120         CoT & 20.87 & 60.40 & 60.40 & 0.25 & 0.18 & 0.18 & \\  
 1121         w/ Raw Refiner & 8.67 & 57.53 & 57.47 & 0.24 & 0.19 & 0.27 & \\  
 1122         \textbf{Ours, w/ Refiner} & \textbf{49.13} & \textbf{79.47} & \textbf{79.47} & \textbf{0.28} & \textbf{0.22} & \textbf{0.31} & \\  
 1123         \hline  
 1124         \textit{Llama-3.2-3B-Instruct} & & & & & & & \\  
 1125         CoT & 1.60 & 54.20 & 45.70 & 0.11 & 0.03 & 0.04 & \\  
 1126         w/ Raw Refiner & 22.13 & 53.13 & 50.40 & 0.17 & 0.09 & 0.12 & \\  
 1127         \textbf{Ours, w/ Refiner} & \textbf{36.53} & \textbf{74.73} & \textbf{73.80} & \textbf{0.21} & \textbf{0.12} & \textbf{0.17} & \\  
 1128         \hline  
 1129         \textit{Llama-3.1-8B-Instruct} & & & & & & & \\  
 1130         CoT & 43.07 & 64.80 & 60.67 & 0.21 & 0.15 & 0.09 & \\  
 1131         w/ Raw Refiner & 31.40 & 59.47 & 58.67 & 0.22 & 0.16 & 0.21 & \\  
 1132         \textbf{Ours, w/ Refiner} & \textbf{60.27} & \textbf{72.20} & \textbf{72.00} & \textbf{0.27} & \textbf{0.20} & \textbf{0.25} & \\  
 1133         \hline  
 1134         \textit{Qwen3-1.7B} & & & & & & & \\  
 1135         CoT & 16.70 & 62.27 & 59.40 & 0.15 & 0.12 & 0.20 & \\  
 1136         w/ Raw Refiner & 22.47 & 64.40 & 62.73 & 0.15 & 0.11 & 0.24 & \\  
 1137         \textbf{Ours, w/ Refiner} & \textbf{39.67} & \textbf{68.60} & \textbf{66.80} & \textbf{0.22} & \textbf{0.15} & \textbf{0.31} & \\  
 1138         \hline  
 1139         \textit{Qwen3-8B} & & & & & & & \\  
 1140         CoT & 80.33 & 86.87 & 86.80 & 0.25 & 0.21 & 0.33 & \\  
 1141         w/ Raw Refiner & 56.53 & 81.80 & 81.67 & 0.25 & 0.20 & 0.36 & \\  
 1142         \textbf{Ours, w/ Refiner} & \textbf{86.60} & \textbf{87.13} & \textbf{87.13} & \textbf{0.29} & \textbf{0.22} & \textbf{0.41} & \\  
 1143         \hline  
 1144     \end{tblr}

1145     As demonstrated in Table 5, our trained Tool-use Refiner comprehensively outperforms the original  
 1146     Qwen3-1.7B on both the Bamboogle\* and NESTFUL benchmarks. The superior performance is  
 1147     evident across all evaluated aspects, including the correction of tool invocation format, selection,  
 1148     parameters, and order. On certain metrics, the original Qwen3-1.7B even leads to a decrease in  
 1149     scores. This result indicates that the strong performance of our refiner in the main evaluation stems  
 1150     from our specialized RL training for the tool invocation correction task, rather than the inherent  
 1151     capabilities of the base model, validating the effectiveness of our proposed training methodology.

## 1134 G AN ABLATION STUDY OF THE OVERLONG PENALTY

1135  
 1136 To validate the effectiveness of the overlong penalty, we conduct an ablation study by removing it  
 1137 from the training objective. Our hypothesis is that without this penalty, the Refiner would be more  
 1138 prone to generating outputs that exceed the context window, impairing its performance. We evaluate  
 1139 both our full Refiner (w/ Refiner) and the ablated Refiner (w/o Overlong Penalty) on the modified  
 1140 Bamboogle dataset (Bamboogle\*) and the API-Bank benchmark.

1141  
 1142 Table 6: Ablation study on the overlong penalty (w/o Overlong Penalty)

1144 Model	Bamboogle*			API-bank
	1145 Acc.	Acc <sub>Toolname</sub>	Acc <sub>Parameter</sub>	
<i>Llama-3.2-3B-Instruct</i>				
CoT	1.60	54.20	45.70	39.03
w/o Overlong Penalty	35.13	73.60	70.67	55.28
<b>Ours, w/ Refiner</b>	<b>36.53</b>	<b>74.73</b>	<b>73.8</b>	<b>64.15</b>
<i>Llama-3.1-8B-Instruct</i>				
CoT	43.07	64.80	60.67	67.17
w/o Overlong Penalty	<b>63.80</b>	68.13	68.00	57.45
<b>Ours, w/ Refiner</b>	60.27	<b>72.20</b>	<b>72.00</b>	<b>68.51</b>
<i>Qwen3-1.7B</i>				
CoT	16.70	62.27	59.40	60.13
w/o Overlong Penalty	38.67	<b>70.66</b>	<b>69.00</b>	60.30
<b>Ours, w/ Refiner</b>	<b>39.67</b>	68.60	66.80	<b>70.18</b>
<i>Qwen3-8B</i>				
CoT	80.33	86.87	86.80	70.85
w/o Overlong Penalty	<b>86.60</b>	<b>87.33</b>	<b>87.33</b>	60.47
<b>Ours, w/ Refiner</b>	<b>86.60</b>	87.13	87.13	<b>74.37</b>

1159 The results, summarized in Table 6, confirm our hypothesis. On the API-Bank benchmark, the full  
 1160 Refiner significantly outperforms the ablated version. Error analysis further reveals that the ablated  
 1161 model exhibits a high incidence of context window violations, with overlong outputs occurring in  
 1162 13.5% of cases on average. This finding underscores the crucial role of the overlong penalty. In con-  
 1163 trast, on the Bamboogle\* dataset, both models deliver comparable performance, with no instances  
 1164 of overlong generation observed.

1165 This performance difference can be attributed to the complexity and diversity of the tasks in each  
 1166 benchmark. API-Bank, as a general-purpose tool-use dataset, features a wide array of distinct tools,  
 1167 often demanding intricate and lengthy reasoning chains. The ablated model, lacking the length  
 1168 constraint, tends to over-analyze, leading to context window breaches. Bamboogle\*, however, is  
 1169 constrained to 17 similar sub-tasks (see Section E.4) focused on generating search queries from  
 1170 celebrity data. The similarity of these tasks necessitates a less complex reasoning process, which  
 1171 remains well within the context limits for both models.

1172 Ultimately, this study demonstrates that the overlong penalty is a critical component for ensuring that  
 1173 the Refiner produces concise yet effective reasoning, particularly in complex scenarios. It effectively  
 1174 mitigates the risk of generating excessively long outputs that break the context window, confirming  
 1175 its value in our training methodology.

## 1177 H AN ABLATION STUDY OF THE SIMPLIFIED CALL SEQUENCE DATA

1179 The introduction of the simplified call sequence data serves three primary strategic purposes in our  
 1180 training methodology. First, it functions as a form of curriculum learning, reducing the initial task  
 1181 complexity by decoupling sequence ordering from parameter generation. This allows the model to  
 1182 master the foundational logic of tool dependency first, ensuring a more stable and efficient training  
 1183 convergence. Second, it enhances model generalization. By introducing a novel structure (i.e.,  
 1184 a sequence of steps, each potentially containing concurrent calls), we intentionally diverge from  
 1185 standard flat-list formats. This structural variance compels the model to learn the abstract principles  
 1186 of tool coordination rather than overfitting to a specific data schema. Finally, and most critically,  
 1187 this format enables targeted, high-quality negative sampling. By deliberately placing tools with  
 1188 sequential dependencies into a single concurrent step, we create explicit, instructive error cases. This

1188  
 1189 efficiently teaches the model the crucial distinction between parallelizable and sequential operations,  
 1190 a fundamental aspect of complex tool use.

1191  
 1192 In this section, we conduct an ablation study on the simplified call sequence data, investigating  
 1193 the performance change of the Refiner when this data is excluded from the training process. We  
 1194 perform evaluations on the modified Bamboogle dataset (Bamboogle\*), API-bank, and NESTFUL  
 1195 benchmarks. The results are presented in Table 7.

1196  
 1197 Table 7: Ablation study on the simplified call sequence data (w/o Simplified Order Data)

Model	NESTFUL			API-bank	Bamboogle*		
	Part. Acc.	Full Acc.	Win Rate		Acc.	Acc <sub>Toolname</sub>	Acc <sub>Parameter</sub>
<i>Qwen2.5-3B-Instruct</i>							
CoT	0.10	0.04	0.13	40.54	0.00	54.27	52.27
w/ Refiner (No simplified order data)	0.18	0.09	0.22	65.83	40.86	72.93	72.60
<b>Ours, w/ Refiner</b>	<b>0.19</b>	<b>0.10</b>	<b>0.23</b>	<b>68.84</b>	<b>48.13</b>	<b>79.67</b>	<b>79.47</b>
<i>Qwen2.5-7B-Instruct</i>							
CoT	0.25	0.18	0.18	68.34	20.87	60.40	60.40
w/ Refiner (No simplified order data)	0.26	0.20	0.28	71.36	21.6	59.80	59.80
<b>Ours, w/ Refiner</b>	<b>0.28</b>	<b>0.22</b>	<b>0.31</b>	<b>73.70</b>	<b>49.13</b>	<b>79.47</b>	<b>79.47</b>
<i>Llama-3.2-3B-Instruct</i>							
CoT	0.11	0.03	0.04	39.03	1.60	54.20	45.70
w/ Refiner (No simplified order data)	0.19	0.11	0.14	63.17	26.73	59.67	58.00
<b>Ours, w/ Refiner</b>	<b>0.21</b>	<b>0.12</b>	<b>0.17</b>	<b>64.15</b>	<b>36.53</b>	<b>74.73</b>	<b>73.80</b>
<i>Llama-3.1-8B-Instruct</i>							
CoT	0.21	0.15	0.09	67.17	43.07	64.80	60.67
w/ Refiner (No simplified order data)	0.28	0.20	0.28	70.18	58.73	67.00	66.73
<b>Ours, w/ Refiner</b>	<b>0.27</b>	<b>0.20</b>	<b>0.25</b>	<b>68.51</b>	<b>60.27</b>	<b>72.20</b>	<b>72.00</b>
<i>Qwen3-1.7B</i>							
CoT	0.15	0.12	0.20	60.13	16.70	62.27	59.40
w/ Refiner (No simplified order data)	0.19	0.15	0.25	73.70	42.53	72.27	70.33
<b>Ours, w/ Refiner</b>	<b>0.22</b>	<b>0.15</b>	<b>0.31</b>	<b>70.18</b>	<b>39.67</b>	<b>68.60</b>	<b>66.80</b>
<i>Qwen3-8B</i>							
CoT	0.25	0.21	0.33	70.85	80.33	86.87	86.80
w/ Refiner (No simplified order data)	0.26	0.21	0.35	73.87	83.93	86.40	86.40
<b>Ours, w/ Refiner</b>	<b>0.29</b>	<b>0.22</b>	<b>0.41</b>	<b>74.37</b>	<b>86.60</b>	<b>87.13</b>	<b>87.13</b>

1217 As shown in Table 7, on the NESTFUL benchmark, which specifically evaluates tool sequencing,  
 1218 our full Refiner slightly outperforms the version trained without simplified call sequence data. This  
 1219 confirms that the introduction of simplified call sequence data effectively enhances the Refiner’s  
 1220 ability to correct tool invocation sequences.

1221 Furthermore, the performance gap between the two versions becomes more pronounced on bench-  
 1222 marks that use conventional tool-calling formats (Bamboogle\*, API-bank). On these tasks, our  
 1223 full model demonstrates significantly better performance. In contrast, the ablated Refiner not only  
 1224 yields lower scores across the board but, when paired with certain upstream LLMs, its performance  
 1225 is merely on par with the CoT baseline, indicating a minimal improvement. This degradation is  
 1226 particularly evident in scenarios where the upstream LLM produces convoluted or ambiguous rea-  
 1227 soning chains. The ablated model’s inability to identify these failures suggests a lack of stability and  
 1228 poorer generalization.

1229 In conclusion, training with simplified call sequence data leads to a more stable Refiner, enhancing  
 1230 its proficiency in both general tool-use correction and specific tool order correction.

## I THE USE OF LARGE LANGUAGE MODELS (LLMs)

1234 During the preparation of this manuscript, we utilize the Large Language Model (LLM) Gemini 2.5  
 1235 Pro (Comanici et al., 2025) for language refinement and polishing.

1242  
12431244 J. CASE STUDIES  
12451246 J.1 CORRECTION OF FORMATTING ERRORS  
12471248 The following case demonstrates our Refiner's proficiency in fixing formatting errors. Here, the  
1249 Refiner first validates the semantic correctness of the tool invocation but then pinpoint a formatting  
1250 issue. The error is then resolved through a precise modification targeted solely at the structure.  
1251

**The Original System Prompt:** You are a helpful multi-turn dialogue assistant capable of leveraging tool calls to solve user tasks and provide structured chat responses.\n\n\*\*Available Tools\*\*\nIn your response, you can use the following tools:\n1. GetOccupationSalary: API for querying the salary of a given occupation.\nParameters: {\"occupation\": {\"type\": 'str', \"description\": 'The occupation to query.'}}\n\n\*\*Steps for Each Turn\*\*\n1. \*\*Think:\*\* Recall relevant context and analyze the current user goal.\n2. \*\*Decide on Tool Usage:\*\* If a tool is needed, specify the tool and its parameters.\n3. \*\*Respond Appropriately:\*\* If a response is needed, generate one while maintaining consistency across user queries.\n\n\*\*Output Format\*\*\nplaintext\n Your thoughts and reasoning \n<tool\_call>\n{\\"name\\": \"Tool name\", \\"parameters\\": {\\"Parameter name\\\": \"Parameter content\", \\"... ...\\\": \"... ...\"}}\n{\\"name\\": \"... ...\", \\"parameters\\\": {\\"... ...\\\", \\"... ...\\\", \\"... ...\\\"}}\n<response> AI's final response\n\n\*\*Important Notes\*\*\n1. 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.\n2. 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.\n3. 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).

**The User Query:** \*\*Dialogue Records History\*\*\n<user>Calculate the monthly salary after taxes for a Data Scientist</user>\n<user> Based on our conversation above, please only make one tool call to solve my need.</user>\n

**Ground Truth:** : <tool\_call>\n{\\"name\\": \"GetOccupationSalary\", \\"parameters\\\": {\\"occupation\\\": \"Data Scientist\"}}\n</tool\_call>

**Upstream LLM's Thought:** The user wants to know the monthly salary of a Data Scientist after taxes. We need to use the GetOccupationSalary tool to fetch the salary information for a Data Scientist and then calculate the salary after taxes. However, the tool does not provide tax information, so we will assume a standard tax rate for the calculation.

**Upstream LLM's Answer:** <tool\_call>\n{\\"name\\": \"GetOccupationSalary\", \\"parameters\\\": {\\"occupation\\\": \"Data Scientist\"}}\n<response>AI will fetch the salary information for a Data Scientist and then calculate the salary after taxes. Please wait.</response>\n</tool\_call>

**Refiner's Thought:** Okay, let's see. The user wants to calculate the monthly salary after taxes for a Data Scientist. The available tool is GetOccupationSalary, which queries the salary of a given occupation. The user's query is about calculating the monthly salary after taxes, so the tool call should be to GetOccupationSalary with the occupation \"Data Scientist\". The parameters are correct as the occupation is provided. The user also mentioned that only one tool call is needed. The original instruction from the model is correct, so no corrections are needed. The output should include the tool call with the parameters. The response is not needed here because the user wants only the tool call. So the correct instruction is the one provided by the model.

**Refiner's Answer:** <tool\_call>\n{\\"name\\": \"GetOccupationSalary\", \\"parameters\\\": {\\"occupation\\\": \"Data Scientist\"}}\n</tool\_call>

1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295

Figure 3: Correction of formatting errors.

1296  
1297

## J.2 PRESERVING A CORRECT TOOL INVOCATION

1298  
1299  
1300

The following case shows our Refiner's response to a correct tool invocation from the upstream LLM. The output shows that our Refiner correctly recognizes the tool invocation as valid and preserves it without modification.

1301

**The Original System Prompt:** You are a helpful multi-turn dialogue assistant capable of leveraging tool calls to solve user tasks and provide structured chat responses.\n\n\*\*Available Tools\*\*\nIn your response, you can use the following tools:\n1. Name: QueryHealthData\nDescription: This API queries the recorded health data in database of a given user and time span.\nParameters: {`user\_id`: {`type`: `str`, `description`: 'The user id of the given user. Cases are ignored.'}, `start\_time`: {`type`: `str`, `description`: 'The start time of the time span. Format: %Y-%m-%d %H:%M:%S'}, `end\_time`: {`type`: `str`, `description`: 'The end time of the time span. Format: %Y-%m-%d %H:%M:%S'}}\n2. Name: CancelRegistration\nDescription: This API cancels the registration of a patient given appointment ID.\nParameters: {`appointment\_id`: {`type`: `str`, `description`: 'The ID of appointment.'}}\n3. Name: ModifyRegistration\nDescription: This API modifies the registration of a patient given appointment ID.\nParameters: {`appointment\_id`: {`type`: `str`, `description`: 'The ID of appointment.'}, `new\_appointment\_date`: {`type`: `str`, `description`: 'The new appointment date. Format: %Y-%m-%d.'}, `new\_appointment\_doctor`: {`type`: `str`, `description`: 'The new appointment doctor.'}}\n\n\*\*Steps for Each Turn\*\*\n1. \*\*Think:\*\* Recall relevant context and analyze the current user goal.\n2. \*\*Decide on Tool Usage:\*\* If a tool is needed, specify the tool and its parameters.\n3. \*\*Respond Appropriately:\*\* If a response is needed, generate one while maintaining consistency across user queries.\n\n\*\*Output Format\*\*\n``plaintext\n<think> Your thoughts and reasoning\n</think>\n<tool\_call>\n {"name": "Tool name", "parameters": {"Parameter name": "Parameter content", "...": "..."}\n</tool\_call>\n<response> AI's final response\n</response>\n``\n\n\*\*Important Notes\*\*\n1. 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.\n2. 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.\n3. 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).

1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317

**The User Query:** \*\*Dialogue Records History\*\*\n<user>Can you please modify my appointment scheduled for March 25th with Dr. Kim to March 26th with Dr. Lee?</user>\n<response>Sure, I can help you with that. Please provide me with the appointment ID and the new appointment date and doctor's name.</response>\n<user>The appointment ID is 34567890 and the new date is March 26th with Dr. Lee.</user>\n<response>Alright. I'll modify your appointment now.</response>\n<user> Based on our conversation above, please only make one tool call to solve my need.</user>

1318  
1319  
1320  
1321

**Ground Truth:** : <tool\_call>\n {"name": "ModifyRegistration", "parameters": {"appointment\_id": "34567890", "new\_appointment\_date": "2023-03-26", "new\_appointment\_doctor": "Dr. Lee"} }\n</tool\_call>

1322  
1323  
1324  
1325

**Upstream LLM's Thought:** The user needs their appointment modified. They have provided the appointment ID, the new appointment date, and the new doctor's name. I need to use the ModifyRegistration API to perform this action.

1326  
1327  
1328

**Upstream LLM's Answer:** <tool\_call>\n {"name": "ModifyRegistration", "parameters": {"appointment\_id": "34567890", "new\_appointment\_date": "2023-03-26", "new\_appointment\_doctor": "Dr. Lee"} }\n</tool\_call>

1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338

**Refiner's Thought:** Okay, let's see. The user wants to modify their appointment from March 25th with Dr. Kim to March 26th with Dr. Lee. The original tool-calling instruction uses "ModifyRegistration" with parameters "appointment\_id": "34567890", "new\_appointment\_date": "2023-03-26", "new\_appointment\_doctor": "Dr. Lee". \n\nWait, the user's original query says "March 26th with Dr. Lee". The parameter for the new appointment date is "new\_appointment\_date" and the doctor's name is "new\_appointment\_doctor". The user provided the appointment ID as 34567890, which is correct. The date format is "2023-03-26" which is in the correct format. The doctor's name is "Dr. Lee" which is correct. \n\nBut wait, the user wrote "March 26th with Dr. Lee" which is a bit ambiguous. The date is March 26th, so the format should be "2023-03-26" and the doctor's name is "Dr. Lee". The parameters in the tool-calling instruction are correct. The only thing I notice is that the user might have intended the date to be in the format "2023-03-26" instead of "2023-03-26" (without the hyphen). But the user's input is already in the correct format. So the tool-calling instruction is correct. The user might have made a typo, but the parameters are correct. Therefore, the tool-calling instruction is correct and no corrections are needed.

1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349

Figure 4: Preserving a correct tool invocation.

1350

## J.3 CORRECTION OF CONTENT ERRORS.

1351

1352 The following case demonstrates how our Refiner addresses erroneous content in tool calls generated  
 1353 by an upstream LLM. As shown, our Refiner correctly identifies an error in the tool invocation (i.e.,  
 1354 in this case, the presence of an extraneous parameter), and accurately rectifies the call.

1355

1356

**The Original System Prompt:** You are a helpful multi-turn dialogue assistant capable of leveraging tool calls to solve user tasks and provide structured chat responses.\n\n\*\*Available Tools\*\*\nIn your response, you can use the following tools:\n1. AccountInfo: API for retrieving and updating user account information.\nParameters: {'username': {'type': 'str', 'description': 'Name of the user.'}, 'password': {'type': 'str', 'description': 'Password of the user.'}}\n\n\*\*Steps for Each Turn\*\*\n1. \*\*Think:\*\* Recall relevant context and analyze the current user goal.\n2. \*\*Decide on Tool Usage:\*\* If a tool is needed, specify the tool and its parameters.\n3. \*\*Respond Appropriately:\*\* If a response is needed, generate one while maintaining consistency across user queries.\n\n\*\*Output Format\*\*\n```\nplaintext\n<think> Your thoughts and reasoning\n</think>\n<tool\_call>\n {\n "name": "Tool name",\n "parameters": {\n "Parameter name": "Parameter content",\n "...": "..."\n }\n }\n</tool\_call>\n<response> AI's final response\n</response>\n```\n\n\*\*Important Notes\*\*\n1. 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.\n2. 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.\n3. 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).

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

**The User Query:** \*\*Dialogue Records History\*\*\n<user>Update John's personal information and address. Password is 123456. Address is 123 Main St.</user>\n<user> Based on our conversation above, please only make one tool call to solve my need.</user>\n

**Ground Truth:** : <tool\_call>\n {\n "name": "AccountInfo",\n "parameters": {\n "username": "John",\n "password": "123456"\n }\n }\n</tool\_call>

**Upstream LLM's Thought:** Okay, let's see. The user wants to update John's personal information and address. They provided the password as 123456 and the new address as 123 Main St.\nFirst, I need to check which tool to use. The available tool is AccountInfo, which is for retrieving and updating user account info. The parameters required are username, password, and possibly other fields like address.\nWait, the parameters listed for AccountInfo are username and password. But the user is mentioning an address. Hmm, maybe the tool allows updating address through the AccountInfo API. Maybe the address is part of the account information. But according to the tool's parameters, the only parameters are username and password. That's confusing.\nWait, maybe the tool's parameters are just the username and password for authentication, but the actual update fields (like address) are part of the account info that can be modified. So, the final answer is to call the AccountInfo tool with username John, password 123456, and address 123 Main St. Even though the tool's parameters are listed as only username and password, the user's request requires including the address. Therefore, the tool call should include those parameters.

**Upstream LLM's Answer:** <tool\_call>\n {\n "name": "AccountInfo",\n "parameters": {\n "username": "John",\n "password": "123456",\n "address": "123 Main St"\n }\n }\n</tool\_call>

**Refiner's Thought:** Okay, let's see. The user wants to update John's personal information and address. The password is 123456, and the address is 123 Main St. The tool-calling instruction provided is using the AccountInfo tool with username John, password 123456, and address 123 Main St.\nWait, the AccountInfo tool's parameters are username and password. The address is supposed to be part of the parameters. Let me check the tool's parameters again. The AccountInfo tool's parameters are username and password. The address is not part of the parameters. So the user is trying to include the address in the parameters, but according to the tool's definition, it's not allowed.\nSo the correct approach would be to only use the username and password. The address is not part of the parameters for AccountInfo. Therefore, the tool-calling instruction is incorrect because it includes the address in the parameters. The address should be part of the user's request, but the tool doesn't accept it. So the correct tool call would only include username and password.\nTherefore, the corrected tool-calling instruction should only include username and password, and the address should be handled elsewhere, maybe in a separate tool or by the user. So the correct instruction would be to call AccountInfo with username John and password 123456, and the address is not included in the parameters. So the original instruction is incorrect because it includes the address in the parameters. The correct one would exclude the address.

**Refiner's Answer:** <tool\_call>\n {\n "name": "AccountInfo",\n "parameters": {\n "username": "John",\n "password": "123456"\n }\n }\n</tool\_call>

Figure 5: Correction of content errors.

1404 J.4 CORRECTION OF SEQUENTIAL ERRORS.  
14051406 The following case demonstrates our Refiner's ability to rectify flawed tool invocation sequences.  
1407 Specifically, it identifies and corrects both parameter and sequential errors in the output from the  
1408 upstream LLM.  
14091410 **The Original System Prompt:** You are an expert in composing functions. You are given a question and a set of possible  
1411 functions.\nBased on the question, you will need to make one or more function/tool calls to achieve the purpose.\nIf none of  
1412 the function can be used, point it out. If the given question lacks the parameters required by the function, also point it out.  
1413 You should only return the function call in tools call sections.\n\nIf you decide to invoke any of the function(s), you MUST put  
1414 it in the format of '<think> Your thoughts and reasoning </think>\n [{"name": "func\_name1", "arguments":  
1415 "[{"argument1": "value1", "argument2": "value2"}, {"label": "\$var\_1"}], ... (more tool calls as required)}]\n\nYou  
1416 SHOULD NOT include any other text in the response.\n\nHere is a list of functions in JSON format that you can  
1417 invoke.\n\n{FUNCTION\_STR}\n\nHere are some examples:\n\n{ICL\_EXAMPLES}.1418 **The User Query:** The average age of 15 students of a class is 15 years. Out of these, the average age of 4 students is 14 years  
1419 and that of the other 10 students is 16 years. The age of the 15th student is  
14201421 **Ground Truth:** [{"name": "multiply", "arguments": {"arg\_0": 4, "arg\_1": 14}, "label": "\$var\_1"}, {"name": "multiply",  
1422 "arguments": {"arg\_0": 10, "arg\_1": 16}, "label": "\$var\_2"}, {"name": "add", "arguments": {"arg\_0": "\$var\_1.result\$", "arg\_1":  
1423 "\$var\_2.result\$"}, "label": "\$var\_3"}, {"name": "multiply", "arguments": {"arg\_0": 15, "arg\_1": 15}, "label": "\$var\_4"}, {"name": "subtract",  
1424 "arguments": {"arg\_0": "\$var\_4.result\$", "arg\_1": "\$var\_3.result\$"}, "label": "\$var\_5"}]1425 **Upstream LLM's Thought:** To find the age of the 15th student, we need to first find the total age of all 15 students, then  
1426 subtract the total age of the first 14 students. We can find the total age of the first 14 students by finding the total age of the  
1427 4 students with an average age of 14 and the total age of the 10 students with an average age of 16. We can use the 'multiply'  
1428 function to find the total age of each group and then add them together. We can use the 'add' function to find the total age of  
1429 all 15 students. We can then use the 'subtract' function to find the age of the 15th student.  
14301431 **Upstream LLM's Answer:** [{"name": "multiply", "arguments": {"arg\_0": 15, "arg\_1": 15}, "label": "\$var\_1"}, {"name": "multiply",  
1432 "arguments": {"arg\_0": 4, "arg\_1": 14}, "label": "\$var\_2"}, {"name": "subtract", "arguments": {"arg\_0": "\$var\_1.result\$", "arg\_1":  
1433 "\$var\_2.result\$"}, "label": "\$var\_3"}, {"name": "multiply", "arguments": {"arg\_0": 4, "arg\_1": 16}, "label": "\$var\_4"}, {"name": "add",  
1434 "arguments": {"arg\_0": "\$var\_4.result\$", "arg\_1": "\$var\_5.result\$"}, "label": "\$var\_6"}, {"name": "subtract",  
1435 "arguments": {"arg\_0": "\$var\_1.result\$", "arg\_1": "\$var\_6.result\$"}, "label": "\$var\_7"}]1436 **Refiner's Thought:** Okay, let's see. The user is asking about the age of the 15th student in a class. The average age of 15  
1437 students is 15 years. Then, the average age of 4 students is 14, and the other 10 students is 16. So, the total age for the 4  
1438 students is  $4 \times 14 = 56$ , and for the 10 students is  $10 \times 16 = 160$ . The total age for all 15 students is  $15 \times 15 = 225$ . The 15th  
1439 student's age is the total age minus the sum of the other 14 students. Wait, but the user says the average age of 15 students is  
1440 15, so total age is  $15 \times 15 = 225$ . Then, the 4 students have 56 and 10 students have 160. So, the total is  $56 + 160 = 216$ . Then,  
1441 the 15th student's age is  $225 - 216 = 9$ . So, the answer should be 9. The tool-calling instruction has some errors. The multiply  
1442 functions are called with arg\_0 and arg\_1, but the parameters are not correct. The user might have intended to call the add  
1443 function with the sum of the 4 and 10 students' ages. So, the correct approach is to call multiply for 4 and 14, multiply for 10  
1444 and 16, add them, then subtract from the total. The original instruction has the multiply function called with the result of the  
1445 multiply functions, but the parameters are incorrect. The correct instruction should call multiply, add, and subtract in the right  
1446 order.  
14471448 **Refiner's Answer:** [{"name": "multiply", "arguments": {"arg\_0": 4, "arg\_1": 14}, "label": "\$var\_1"}, {"name": "multiply",  
1449 "arguments": {"arg\_0": 10, "arg\_1": 16}, "label": "\$var\_2"}, {"name": "add", "arguments": {"arg\_0": "\$var\_1.result\$", "arg\_1":  
1450 "\$var\_2.result\$"}, "label": "\$var\_3"}, {"name": "multiply", "arguments": {"arg\_0": 15, "arg\_1": 15}, "label": "\$var\_4"}, {"name": "subtract",  
1451 "arguments": {"arg\_0": "\$var\_4.result\$", "arg\_1": "\$var\_3.result\$"}, "label": "\$var\_5"}]1452 Figure 6: Correction of sequential errors.  
1453  
1454  
1455  
1456  
1457