A Joint Optimization Framework for Enhancing Efficiency of Tool Utilization in LLM Agents

Bin Wu [1] **Edgar Meij** [2] **Emine Yilmaz** [1]

[1] Centre for Artificial Intelligence, University College London
[2] Bloomberg
{bin.wu.23,emine.yilmaz}@ucl.ac.uk
emeij@bloomberg.net

Abstract

Large Language Models (LLMs) augmented with external tools have demonstrated remarkable capabilities in complex problem solving. Existing efforts for tool utilization typically involve an LLM agent that contains instructions on using the description of the available tools to determine and call the tools required to solve the problem. Inference Scaling techniques, such as chain-of-thought and tree-of-thought reasoning, are commonly used but require significant computational overhead and rendering such methods impractical in real-world applications. In this work, we recognize and formalize the critical role of instructions provided in agent prompts and tool descriptions—collectively referred to as *context*—and show that incomplete *context* is one of the reasons for this computational overhead. To fill this efficiency gap, we propose an optimization framework that jointly refines both the instructions provided in the agent prompt and tool description, enhancing their interaction. Experiments on StableToolBench and RestBench demonstrate that our optimized agents achieve superior efficiency while maintaining effectiveness. Our findings underscore the critical role of context optimization in improving LLM agents for tool utilization, paving the way for more responsive and cost-effective LLM agents. Our code is available at https://github.com/Bingo-W/ToolOptimization.

1 Introduction

Recent advancements in Large Language Models (LLMs) have significantly enhanced their ability to solve complex problems, particularly when integrated with external tools that provide access to private or specialized data and operations (Qin et al., 2024). These tool-augmented LLM agents are capable of executing multi-step tasks with increasing effectiveness (Gou et al., 2024).

To further improve LLM agent capabilities when utilizing tools, existing efforts aim to extend the inference ability by adding more steps. For example, ReAct (Yao et al., 2023) combines the chain-of-thought (CoT) reasoning and tool interaction, to maximize the LLM agent's capabilities (Wei et al., 2022). Further research extends the CoT approach to a tree-based structure (Qin et al., 2024; Zhuang et al., 2023), largely enhancing effectiveness. Despite the success of inference scaling methods in tool utilization, the significant computational overhead required by these methods makes them less useful in practical scenarios, leading to poor user engagement due to high latency in a real-time, end-user-facing system.

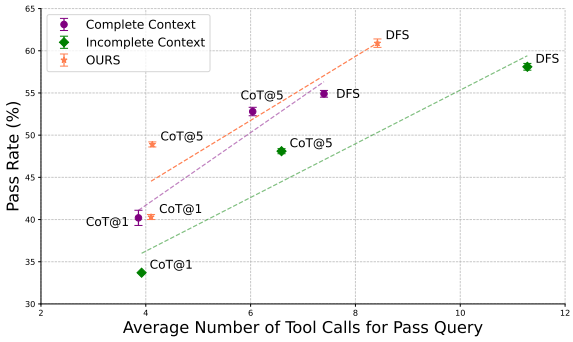


Figure 1: Pass Rate and Average Number of Tool Calls for Pass Query on *G1-instruction* between complete context and incomplete context, among three different Inference Scaling algorithms (CoT@1, CoT@5, and DFS). We show that (1) incomplete context easily leads to inefficiency issues and (2) our proposed optimization system can improve both efficiency and effectiveness of incomplete context.

Some recent works aim to further enhance efficiency by adding a backup mechanism to enable quick restart from the last successful tool call instead of beginning (Qin et al., 2024), or by grouping similar tools as a toolkit (Liu et al., 2024). However, they ignore the critical role of instructions provided in the agent prompt and tool descriptions, which together constitute what we refer to as *context*. Instructions provided in the agent prompt

Findings of the Association for Computational Linguistics: ACL 2025, pages 22361–22373

are usually designed by extensive human effort via many trials and errors, and tend to guide the interaction between the agent and the tools. Such human-designed instructions may result in being incomplete without covering all the necessary guidance (Wu et al., 2024). The tool description is expected to contain information about the functionality of the tool, so that LLM agents can identify which tool is needed for the current task at hand. However, tool descriptions may not fully reflect the exact functionalities of the tool, which is difficult to capture for tools with broad/complex functionalities.

In our work, we first recognize the need to formalize the critical role of instructions provided in agent prompts and tool descriptions. We posit that an incomplete context often degrades efficiency by increasing the number of tool calls required to yield correct responses, leading to the substantial computational overhead of inference scaling methods (shown in Figure 1).

To address the inefficiency issue caused by an incomplete context, we introduce an automated optimization system that optimizes context for efficient tool utilization. While previous work has explored improving instructions (Zhang et al., 2024; Wu et al., 2024) or refining tool descriptions (Hsieh et al., 2023; Yuan et al., 2024; Fang et al., 2024), these efforts primarily focus on effectiveness rather than efficiency. Furthermore, existing methods tend to optimize only one aspect—either the instructions provided in the prompt or the tool descriptions—failing to capture the interaction between the instructions and tool descriptions. Instruction optimization often lacks fine-grained tool-related knowledge, such as criteria that the input parameters should satisfy or tool availability. This omission can lead to wrong or unnecessary tool calls. On the other hand, tool description optimization typically focuses on tool-specific details but neglects task-dependent information, such as inter-tool dependencies (Liu et al., 2024), which are crucial for efficient multi-tool orchestration. Instead, our proposed optimization system systematically refines both instruction and tool descriptions incorporating the interaction between them, to meet the best of both worlds.

Specifically, our framework consists of three key components: (1) a feedback generator, (2) a suggestion coordinator, and (3) a context updater. The feedback evaluator is employed to generate a systematic evaluation regarding the effectiveness and efficiency of tool calling. The suggestion coordinator generates separate improvement suggestions for instruction and tool description from the feedback that contains a coupled evaluation. The context updater processes a batch of suggestions to refine the context, in order to ensure the stability and efficiency of updating. The whole optimization pipeline is operated in a verbalized way (Yuksekgonul et al., 2024), utilizing the power of text and reducing the updating cost by avoiding parameter weight changes.

Our contributions are summarized as follows. (1) We highlight the critical role of context, which encompasses both the instructions provided in the prompt and the description of the tool, where an incomplete context would require more inference steps leading to inefficiency in inference. (2) We propose an automated optimization framework, refining the instruction and tool descriptions, to enhance the efficiency of tool utilization. The optimization is performed in a verbalized manner, benefiting from the expressivity and explainability of language. (3) We conduct experiments on two benchmarks, demonstrating that our optimized agents achieve up to 70% reduction of required tool calls on StableToolBench and avoid 47% redundant tool calls on RestBench while maintaining a comparable or even better performance.

2 Related Work

In what follows, we discuss two lines of related works, i.e., LLM for tool utilization and verbalized optimization.

2.1 LLMs for Tool Utilization

Recent studies have leveraged the remarkable language understanding and reasoning capabilities of LLMs to tackle complex problems. Integrating LLMs with external tools, allowing them to function as LLM agents, significantly enhances their problem-solving capacity. To benchmark LLM agents' ability to utilize tools, Qin et al. (2024) introduced ToolBench, which includes a comprehensive set of tools. A subsequent variant, StableToolBench (Guo et al., 2024), incorporates tool caching and an LLM simulator to improve benchmark stability.

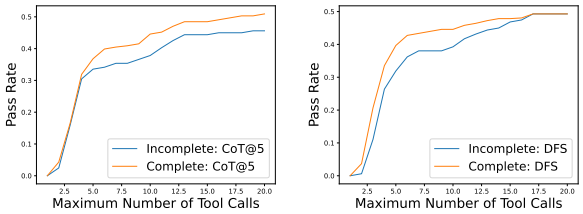
Existing approaches to optimizing LLM agents for tool utilization can be categorized into training-based and training-free methods. Training-based methods focus on fine-tuning LLMs for improved

tool use, employing solution-wise rewards (Patil et al., 2024; Qin et al., 2024) or process-level rewards (Nath et al., 2025). However, considering the prevalence of closed-source LLMs and resource-constrained environments, our work focuses on training-free methods, which optimize LLM agents without modifying their weights (Zhang et al., 2024). These methods enhance tool utilization by refining either the instructions given to the agent in a prompt (Wu et al., 2024), or in the tools' documentation (Yuan et al., 2024; Fang et al., 2024). Additionally, others employ inference scaling techniques to improve reasoning and decision-making regarding tool selection (Qin et al., 2024; Zhuang et al., 2023; Nath et al., 2025; Liu et al., 2024). Despite these advancements, most existing studies emphasize effectiveness while overlooking efficiency. The substantial computational overhead of LLM-based tool utilization poses challenges for more real-time applications. Our work addresses this gap by identifying inefficiencies caused by incomplete contextual information, including instructions in agent prompts and tool descriptions. We propose a verbalized optimization system designed to reduce unnecessary tool calls while maintaining effectiveness.

2.2 Verbalized Optimization

Prompts play a crucial role in LLM interactions (Brown et al., 2020), but designing effective prompts remains challenging, especially for non-experts. Early work on prompt tuning (Lester et al., 2021) established it as a standard technique across various applications. However, its high computational cost and reliance on gradient-based optimization make it impractical for large-scale models, particularly closed-source ones such as GPT.

Recent research has explored verbalized optimization, where LLMs serve as optimizers (Yang et al., 2024; Opsahl-Ong et al., 2024). In this paradigm, optimization tasks are described in natural language, and LLMs iteratively refine prompts based on performance feedback, resembling an evolutionary algorithm. Unlike these evolutionary-based approaches, more recent work introduce textual gradients (Pryzant et al., 2023; Yuksekgonul et al., 2024; Xiao et al., 2024; Wu et al., 2024), which relies on the power of verbalized feedback. However, most existing optimization techniques focus solely on instruction refinement (Zhang et al., 2024), neglecting external environment descriptions such as tool documentation. Furthermore, prior work primarily enhances effectiveness, leaving efficiency largely unexplored. Our work introduces a novel joint optimization framework that simultaneously optimizes both agent instructions and tool descriptions to improve the efficiency of LLM agents for tool utilization.



(a) CoT@5 (b) DFS

Figure 2: Pass Rate with different maximum number of tool calls for different inference scaling methods.

		PR% ↑	# Maximum Tool Calls ↓			
		-	30%	35%	40%	45%
CoT@1	Incomplete	33.7	5	-	-	-
	Complete	40.2	5	5	7	-
CoT@5	Incomplete	48.1	4	7	11	19
	Complete	52.8	4	5	7	11
DFS	Incomplete	58.1	5	6	11	15
	Complete	54.9	4	5	6	11

Table 1: Efficiency comparison between complete context and incomplete context across three inference scaling techniques.

3 Incomplete Context Leads to Inefficient Tool Utilization

Recent efforts have focused on inference scaling methods by adding additional inference steps to enhance the effectiveness of LLM agents in tool utilization (Zhuang et al., 2023; Liu et al., 2024). However, these approaches overlook the critical role of agent instructions and tool descriptions. In real-world scenarios, both elements are often incomplete. Agent instructions are typically crafted by engineers through iterative trial-and-error processes, making it difficult to provide comprehensive guidance (Wu et al., 2024). Similarly, tool descriptions—provided by developers—may not fully capture a tool's functionalities, especially when dealing with complex or multi-purpose tools. Thus, in this section, we empirically explore the role of context in tool utilization and analyze its effect when equipping the inference scaling methods.

Specifically, we evaluate different inference scaling methods on the StableToolBench dataset (Guo et al., 2024), including chain-of-thought-based (i.e.,

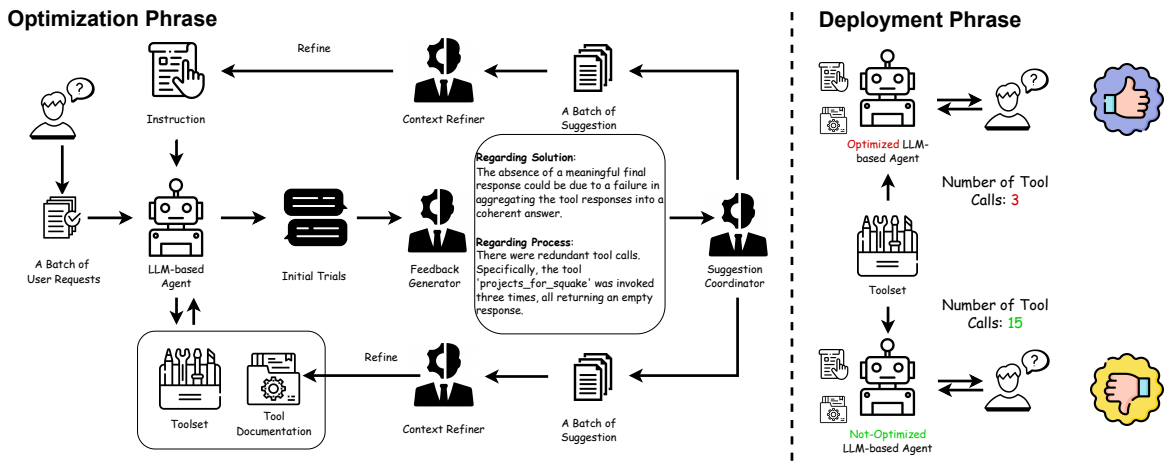


Figure 3: Overview of our optimization framework.

CoT@1 and CoT@5) and tree-of-thought-based methods (i.e., DFS). To simulate incomplete context, we treat instructions and tool descriptions from the original dataset as *complete context*, and extract only the first sentence as the *incomplete context*. We compare Pass Rate based on the fixed maximum number of tool calls in Figure 2 and the maximum number of tool calls to achieve a given Pass Rate in Table 1 (Qin et al., 2024) .

We observe that methods using complete context consistently achieve higher Pass Rate given a fixed maximum number of tool calls (seen in Figure 2). While methods using incomplete context may occasionally reach higher Pass Rate, they always require significantly more tool calls to do so. (seen in Table 1), and this gap widens as the required Pass Rate increases.

These findings demonstrate that inference scaling methods perform inefficiently in the presence of incomplete context. Incomplete instructions and tool descriptions increase computational overhead, leading to inefficiencies in tool utilization. This underscores the importance of optimizing both instructions and tool descriptions to improve efficiency without compromising effectiveness.

4 Optimizing LLM Agents for Efficient Tool Utilization

To address the inefficiency issue of LLM agents for tool utilization caused by an incomplete context, we introduce an automated optimization framework that jointly updates both instructions in the agent prompt and tool descriptions. Inspired by tool learning of humans through trial and error (Wang et al., 2024), our approach begins with a batch of requests and their initial trials on the system. The system then generates verbalized feedback based on the final response and tool call sequence, providing separate improvement suggestions for instructions and tool descriptions. These suggestions are aggregated to refine both instructions in agent prompt and tool descriptions. The whole pipeline is shown in Figure 3. Our optimization framework consists of three key components: (1) Stepwise and Pairwise Feedback Generator (Section 4.2) (2) Improvement Suggestion Coordinator (Section 4.3) (3) Batch Context Refiner (Section 4.4).

4.1 Problem Definition

Let q be a user request, and let $\mathcal{T} = \{t_1, ..., t_i, ...\}$ denote the available tools, each associated with corresponding tool description set $\mathcal{D} = \{d_{t_1}, ..., d_{t_i}, ...\}$, where d_{t_i} represents the documentation for tool t_i. An LLM agent that is guided by a verbalized instruction v is expected to generate a sequence of tool invocation to produce a final response:

$$\{a_1, ..., a_i, ...\} = f(q \mid v, \mathcal{T}, \mathcal{D}), \quad (1)$$

where $\{a_1, ..., a_i, ...\}$ represents the tool invocation sequence. The final response R is then generated based on the outputs $\{r_1, ..., r_i, ...\}$ obtained from executing these tool invocations.

4.2 Step 1: Generating Verbalized Feedback on Final Response and Tool Calls

Precise feedback is crucial for optimizing LLM agents. Inspired by recent work on verbalized optimization (Yuksekgonul et al., 2024), we leverage LLMs' natural language understanding to generate

verbalized feedback. Compared to numeric feedback (Yang et al., 2024), verbalized feedback offers greater flexibility and interpretability (Yuksekgonul et al., 2024; Xiao et al., 2024). Moreover, existing work focuses on the quality of the final response or the feedback on the whole tool call trajectory (Qin et al., 2024; Wu et al., 2024), ignoring the stepwise feedback that is critical to recognizing specific tool interactions, and essential for refining tool descriptions (Nath et al., 2025).

Thus, we employ a feedback generator G_{feedback} to generate the verbalized feedback v_f regarding the final response R and the tool-calling process $\{(a_i, r_i)\}$ based on the full interaction history, including tool invocations a_i and their responses r_i:

$$v_f = G_{\text{feedback}}(R \mid q, \{(a_i, r_i)\}^{N_q}, \mathcal{T}, \mathcal{D}), \quad (2)$$

where N_q represents the length of the tool invocation sequence for request q. The evaluation consists of two key aspects: effectiveness and efficiency. Effectiveness focuses on the quality of the final response, considering the user's task and the intermediate results generated by the tools. Efficiency assesses the quality of the tool-calling process, identifying any redundant tool calls.

4.3 Step 2: Improvement Suggestions

Given feedback v_f, our optimization system aims to refine both instructions and tool descriptions. However, the instruction provided in the agent prompt and the tool description serve different roles, where instruction is shared across all requests, while the tool description is considered more among requests requiring a similar demand. For example, in a flight booking task, the tool description might emphasize city-specific constraints, whereas a tool for scheduling online meetings does not require city information. Hence, our framework ensures task-agnostic suggestion for instructions and task-specific suggestion for tool descriptions. To achieve this, we introduce an improvement suggestion coordinator that generates separate improvement suggestions for instructions v_a and tool descriptions v_d:

$$v_a, v_d = G_{\text{sugg.}}(v_f \mid v, \{(a_i, r_i)\}^{N_q}, \mathcal{T}, \mathcal{D}). \quad (3)$$

By jointly generating v_a and v_d, we ensure that modifications maintain holistic consistency between instructions and tool descriptions.

4.4 Step 3: Batch Context Refining

Individually processing each improvement suggestion can lead to instability due to diverse refinements across requests. Additionally, separate updates for each instance result in computational inefficiency. To mitigate these issues, we introduce a batch-based context refiner, inspired by gradient accumulation in deep learning (Ott et al., 2018). The context refiner aggregates multiple improvement suggestions before updating instructions and tool descriptions:

$$v' = G_{\text{update}}(v, {v_a}^B), \; d_i' = G_{\text{update}}(d_i, {v_d}^B), \quad (4)$$

where B denotes the batch size of accumulated suggestions. This approach ensures stable, efficient, and consistent refinements.

5 Experimental Setup

Following prior work on benchmarking LLM agents for tool utilization, we employ StableToolBench and RestBench for evaluation. We employ the more stable version of Toolbench (Qin et al., 2024), i.e., StableToolBench (Guo et al., 2024), for evaluation. The ToolBench dataset comprises diverse user requests across a wide range of publicly available REST APIs from the RapidAPI Hub. However, ToolBench exhibits instability due to unsolvable requests, inconsistent tool availability, and lack of response caching. The subsequent version, StableToolBench (Guo et al., 2024), addresses these issues by removing the unsolvable request, introducing a tool response cache, and employing GPT4 as the tool simulator for unavailable tool. Thus, we use StableToolBench in our experiments. This benchmark includes 16,464 APIs spanning 49 categories, divided into three subsets: (G1) single-tool; (G2) multi-tool with the same category; and (G3) multi-tools with different categories. To evaluate our method, we partition the dataset into a training set for optimization, an *Agent Test* Set with unseen requests, and a *Tool Test* Set with unseen requests requiring tools overlapped by training set (details in Table 2).

We also employ another benchmark, RestBench (Song et al., 2023), for further comparison. RestBench is a human-annotated benchmark that includes two realistic scenarios: the TMDB movie database and the Spotify music player. The queries in this benchmark are drawn from diverse real-world user instructions, requiring the use of multiple APIs to complete. RestBench employs Cor-

	#Query			
	Total	G1	G2	G3
Train Set	38	24	11	3
Agent Test Set	258	100	100	58
Tool Test Set	300	100	100	100

Table 2: Number of queries for training and testing on StableToolBench.

rect Path Rate (CP%) to measure the proportion of correct paths and Δ Solution Len to assess the additional API calls relative to the length of the gold solutions. We focus on TMDB sub-dataset following the existing work (Yuan et al., 2024).

5.1 Dataset Split

In our work, we optimize the LLM agent using a training set. The dataset split is detailed for each benchmark as follows.

StableToolBench We split the dataset into a small training set for optimization, an *Agent Test* Set, and a *Tool Test* Set (details seen in Table 2). For the Training Set, we randomly sample 5% of the testing set in the original dataset as the training set. To ensure the training set distribution is same as the original dataset, we maintain the same sampling ratio among G1, G2, and G3. For *Agent Test* Set, we randomly select 100, 100, and 58 according to the limited budget from the original dataset, excluding the training set, for G1, G2, and G3. To further examine the quality of the optimized tool documentation, we select the request from the original dataset for the *Tool Test* Set. Considering the remaining requests in the original test set have fewer overlapping tools with our training data, we only keep requests for *Tool Test* Set that involve tools also present in the training set.

RestBench Following existing work (Yuan et al., 2024), we focus on one subset of RestBench, i.e., TMDB, consisting of 50 APIs and 100 user requests. We split 20% for the training set and the remainder for testing.

5.2 Evaluation

We assess the performance of our approach and baselines under a simulated incomplete context setting, where we truncate the first sentence of the instruction and tool description from the original dataset.

Following prior works in the area, we employ Pass Rate (evaluated by LLMs) as the primary metric (Qin et al., 2024; Guo et al., 2024). The evaluator model used is *gpt-4o-mini-2024-07-18*. To measure the efficiency of tool utilization, we observe that different requests require varying numbers of tool calls. Since the maximum number of tool calls can be influenced by requests that require more calls, we introduce a normalized cost metric: (1) **Cost**: The number of tool calls normalized by the number of relevant tools in the original dataset. (2) **Cost Threshold**: The maximum cost needed to achieve a given Pass Rate.

New Metrics: Cost-Aware Pass Rate (CAPR) Prior works primarily employ Pass Rate to evaluate whether the generated response meets the query's requirements (Qin et al., 2024; Guo et al., 2024). As shown in our preliminary study, Pass Rate cannot assess the comprehensive performance of LLM agents on tool utilization. This is caused by Pass Rate focusing solely on effectiveness while ignoring the number of tool calls, making it unable to reflect the comprehensive performance of LLM agents. Inspired by prior works (Ong et al., 2025), we introduce **Cost-Aware Pass Rate (CAPR)**, which measures Pass Rate under different cost constraints. Specifically, given a pass evaluation function $f(q, R)$, which returns 1 if the response passes and 0 otherwise, we define a cost-aware pass function as:

$$CP(\alpha) = \begin{cases} f(q,R), & \text{if } n < \alpha \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

where n represents the number of tool calls, and α is a hyperparameter specifying the maximum Cost Threshold. CAPR is computed as the integral of $CP(\alpha)$ over an allowed range of α:

$$CAPR = \int CP(\alpha) d\alpha, \tag{6}$$

where the integral spans the required computational cost, with the lower bound set to the necessary number of tool calls for a given query and the upper bound defined as k times the necessary tool calls, yielding CAPR@k. Since direct integration is intractable, we approximate CAPR using Monte Carlo estimation (Kroese et al., 2014):

$$CAPR(M) \approx \frac{1}{N} \sum CP(\alpha). \tag{7}$$

Intuitively, CAPR refers to the area below the curve of the cost-PR pair (shown in Figure 2). This new metric takes into account both effectiveness and efficiency, comprehensively reflecting the systematic performance of LLM agents on tool utilization.

	Pass Rate (%) ↑				Cost Threshold↓					
	G1	G2	G3	Ave.	25%	30%	35%	40%	45%	50%
CoT@5	$47.3_{\pm 0.5}$	$34.3_{\pm 0.5}$	$4.0_{\pm 0.8}$	32.52	1.67	2.40	-	-	-	-
+Reflexion	$44.7_{\pm 0.5}$	$22.0_{\pm 0.0}$	$9.2_{\pm 0.8}$	27.92	1.67	-	-	-	-	-
+EasyTool	$40.0_{\pm 0.0}$	$28.2_{\pm 0.8}$	$8.6_{\pm 0.0}$	25.60	2.00	-	-	-	-	-
+OURS	$45.0_{\pm 0.0}$	$35.0_{\pm 0.4}$	$6.9_{\pm 0.0}$	32.55	$1.50_{\downarrow 10\%}$	$1.67_{\downarrow 30\%}$	-	-	-	-
DFS	$61.2_{\pm 0.2}$	$47.7_{\pm 0.5}$	$26.1_{\pm 1.5}$	48.07	1.40	2.00	3.00	5.34	9.00	-
+Reflexion	$60.0_{\pm 0.0}$	$\mathbf{48.0_{\pm 0.8}}$	$\mathbf{37.1_{\pm 0.0}}$	50.20	1.50	2.00	4.50	6.67	10.34	18.67
+EasyTool	$57.0_{\pm 0.8}$	$44.0_{\pm 0.8}$	$28.2_{\pm 0.8}$	43.07	1.67	2.67	4.75	7.75	17.67	-
+OURS	$\mathbf{67.3_{\pm 0.5}}$	$47.7_{\pm 0.5}$	$35.1_{\pm 0.8}$	**52.46**	$\mathbf{1.34}_{\downarrow 4\%}$	$\mathbf{1.50}_{\downarrow 25\%}$	$\mathbf{1.67}_{\downarrow 44\%}$	$\mathbf{3.67}_{\downarrow 31\%}$	$\mathbf{7.67}_{\downarrow 15\%}$	$\mathbf{16.67}_{-}$

Table 3: Effectiveness (i.e., Pass Rate) and efficiency (i.e., Cost Threshold) on the *Agent Test* Set.

5.3 Baseline and Models

We compare our method against the following baselines: (1) **CoT@5**: It follows ReAct (Yao et al., 2023) that combines CoT and tool interaction. For a fair comparison, we allow at most 5 retries when suffering from failure following the original paper (Guo et al., 2024). (2) **Deep-first Tree Search (DFS)**: This method extends ReAct into tree-of-thought reasoning (Guo et al., 2024). It employs a depth-first tree search algorithm, which utilizes a backup mechanism to quickly restart from the last state when suffering from failure. Compared to CoT@5 restarting from scratch, DFS is more efficient and effective. (3) **Reflexion**: Based on (Shinn et al., 2024), this method employs verbalized feedback from trial-and-error learning to enhance tool utilization efficiency. We generate reflexion on the training set and evaluate it on the test set. (4) **EasyTool**: This method introduces a framework for transforming raw tool documentation into a more concise format (Yuan et al., 2024).

We employ *gpt-3.5-turbo-16k-0613* as our base model following previous works (Qin et al., 2024; Guo et al., 2024), and *gpt-4o-mini-2024-07-18* as the base model for optimization. Regarding the prompt, we follow the prompt of CoT@5 and DFS from the original paper (Guo et al., 2024). For Reflexion, we design a dedicated prompt. The prompts we used can be found in Appendix A.1.

6 Results and Discussion

In this section, we present and analyze the results to show the effectiveness of our approach and metrics.

6.1 Enhancing LLM Agents' Efficiency Through Context Optimization

We compare our proposed optimization framework against baseline methods to demonstrate its effectiveness in improving efficiency.

Optimized Instructions Improve Efficiency and Generalization. As shown in Table 3, optimized instructions noticeably enhance efficiency, even for user requests unseen in the training set. Our method achieves a comparable Pass Rate with CoT@5 and a substantially higher performance with DFS, while requiring up to 30% fewer tool calls to reach a given Pass Rate. This demonstrates that optimized instructions reduce the Cost Threshold while maintaining or improving task success. Additionally, these results confirm that optimized instructions generalize well to new user requests.

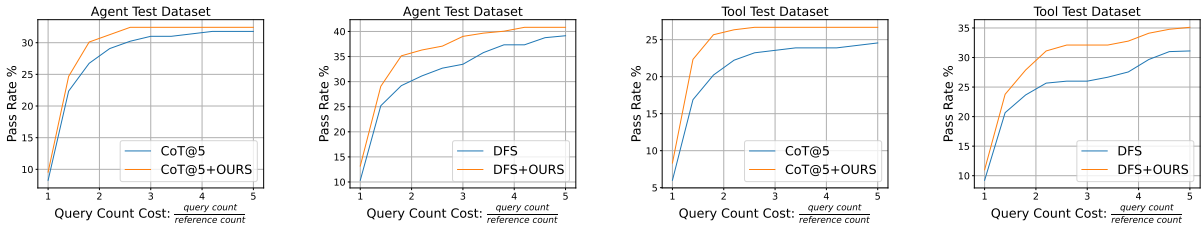
Optimized Tool Descriptions Further Improve Efficiency. As seen in Table 4, when tested on unseen requests that require overlapping tools, optimized tool descriptions further reduce tool calls while preserving a comparable Pass Rate. The reduction in Cost Threshold is noteworthy, with decreases of 71% for CoT@5 and 59% for DFS. Notably, this efficiency gain is greater than that when evaluating on unseen requests without overlapping tools (Table 3 and Figure 4). This confirms that optimized tool descriptions contribute more to efficiency improvements than optimized instructions.

Efficiency Gains Persist Even for Difficult Requests. Across both datasets, the Cost Threshold gap narrows for higher Pass Rate requirements, as more tool calls are needed to resolve complex requests. However, our optimized context maintains an efficiency advantage, demonstrating its robustness even under stringent performance constraints.

Context Optimization Enhances Both Efficiency and Effectiveness. These findings confirm that context optimization not only improves effectiveness, i.e., Pass Rate, but also largely enhances efficiency by reducing unnecessary tool calls. More results on RestBench (Table 5) further confirm this enhancement. This efficiency aspect has been largely overlooked in prior studies, highlighting the novelty and importance of our approach.

	Pass Rate (%) ↑				Cost Threshold ↓					
	G1	G2	G3	Ave.	20%	25%	30%	35%	40%	45%
CoT@5	$37.7_{\pm 0.5}$	$22.0_{\pm 0.0}$	$19.2_{\pm 0.2}$	26.30	1.67	5.75	-	-	-	-
+Reflexion	$42.0_{\pm 0.0}$	$18.7_{\pm 0.5}$	$14.7_{\pm 0.5}$	25.13	1.67	2.34	-	-	-	-
+EasyTool	$38.2_{\pm 0.5}$	$22.7_{\pm 0.5}$	$14.0_{\pm 0.0}$	24.97	1.67	-	-	-	-	-
+OURS	$41.0_{\pm 0.0}$	$20.0_{\pm 0.0}$	$19.3_{\pm 0.2}$	26.77	$1.34_{\downarrow 20\%}$	$1.67_{\downarrow 71\%}$	-	-	-	-
DFS	$\mathbf{62.2_{\pm 0.5}}$	$\mathbf{33.5_{\pm 0.4}}$	$39.0_{\pm 0.8}$	44.90	1.34	2.00	4.34	7.25	11.67	-
+Reflexion	$53.5_{\pm 1.1}$	$32.0_{\pm 0.0}$	$38.7_{\pm 0.5}$	41.40	1.34	3.25	6.34	10.00	17.50	-
+EasyTool	$50.8_{\pm 0.2}$	$34.7_{\pm 1.2}$	$39.2_{\pm 0.8}$	41.57	1.67	2.34	5.34	9.00	20.34	-
+OURS	$60.3_{\pm 0.5}$	$33.3_{\pm 0.2}$	$\mathbf{43.5_{\pm 0.4}}$	**45.70**	$\mathbf{1.34}_{\downarrow 0\%}$	$\mathbf{1.67}_{\downarrow 17\%}$	$\mathbf{2.00}_{\downarrow 59\%}$	$\mathbf{5.00}_{\downarrow 17\%}$	$\mathbf{8.34}_{\downarrow 29\%}$	**29.00**–

Table 4: Effectiveness (i.e., Pass Rate) and efficiency (i.e., Cost Threshold) on the *Tool Test* Set.



(a) Agent: *CoT@5* (b) Agent: *DFS* (c) Tool: *CoT@5* (d) Tool: *DFS*

Figure 4: Relationship between Cost Threshold (x-axis) and Pass Rate (y-axis) on *Agent Test* Set and *Tool Test* Set.

	CP (%) ↑	Δ Solution Len ↓
Incomplete	73.37	+0.38
+ Reflexion	71.20	+0.26
+ EasyTool	73.91	+0.30
+ OURS	**76.21**	**+0.20**

Table 5: Effectiveness (i.e., Correct Path Rate (CP%)) and efficiency (i.e., Δ Solution Len) on RestBench.

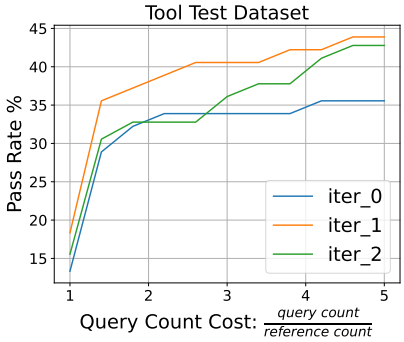


Figure 5: Performance over different iterations.

6.2 The Effect of Contextual Optimization

We conduct an ablation study to assess the contribution of each component, along with a detailed analysis to better understand the effectiveness.

Feedback generation, suggestion coordination, and batch refinement contribute to large efficiency gains. We individually remove the Feedback Generator, Suggestion Coordinator, and Context Refiner components. The results in Table 6 indicate that omitting any of these components not only leads to a substantial performance drop but also increases tool calls required to generate an acceptable response. These findings underscore the importance of feedback generation over the entire trajectory, decoupled suggestion coordination, and batch-based refinement for optimizing efficiency.

More and diverse trials enhance verbalized optimization. To examine the impact of training data on verbalized optimization, we consider two factors: request scale and request difficulty. We sample 25% and 50% subsets of easy and hard user requests to construct varying training scenarios. As shown in Table 6, smaller and simpler datasets result in reduced efficiency gains. This highlights that both the quantity and diversity of training trials play a critical role in improving verbalized optimization.

Overfitting Risk occurs in Iterative Optimization. Similar to traditional machine learning, verbalized optimization supports iterative refinements. However, as shown in Figure 5, the second iteration, while improving Pass Rate over the non-optimized baseline, exhibits increased tool calls, suggesting that over-optimization can lead to overfitting and reduced generalizability. Thus, exploring regularization techniques for verbalized optimization is necessary.

6.3 The Effectiveness of CAPR

We analyze the effectiveness of our proposed evaluation metric in providing a more comprehensive

		Pass Rate (%) ↑				Cost Threshold ↓			
		G1	G2	G3	Ave.	15%	25%	35%	45%
	DFS	$61.7_{\pm 2.4}$	$46.7_{\pm 2.4}$	$41.7_{\pm 2.4}$	50.03	1.34	1.34	4.00	12.67
	OURS	$70.0_{\pm 0.0}$	$45.0_{\pm 0.6}$	$55.8_{\pm 1.2}$	56.93	1.00	1.34	1.34	5.00
Training Set	50% hard	$61.7_{\pm 2.4}$	$50.0_{\pm 0.0}$	$41.7_{\pm 6.2}$	51.13	1.00	1.50	3.00	6.67
	25% hard	$55.8_{\pm 3.1}$	$45.0_{\pm 0.0}$	$40.0_{\pm 0.0}$	46.93	1.00	1.34	4.67	8.34
	50% easy	$60.0_{\pm 0.0}$	$41.7_{\pm 2.4}$	$40.0_{\pm 0.0}$	47.23	1.34	1.34	5.34	14.67
	25% easy	$61.7_{\pm 2.4}$	$36.7_{\pm 2.4}$	$45.8_{\pm 1.2}$	48.07	1.34	1.67	2.00	20.00
Ablation	w/o Coordinator	$58.3_{\pm 2.4}$	$54.2_{\pm 1.2}$	$45.0_{\pm 0.0}$	52.50	1.34	1.34	2.00	6.25
	w/o Feedback	$60.0_{\pm 0.0}$	$47.5_{\pm 2.0}$	$51.7_{\pm 2.4}$	53.07	1.34	1.34	3.00	6.34
	w/o Refiner	$50.0_{\pm 0.0}$	$53.3_{\pm 2.4}$	$55.0_{\pm 0.0}$	52.77	1.00	1.34	4.00	6.67

Table 6: Effectiveness (i.e., Pass Rate) and efficiency (i.e., Cost Threshold) on a subset (20%) of *Tool Test* Set.

		Pass Rate (%)	CAPR
CoT@1	Incomplete	$33.7_{\pm 0.0}$	31.34
	Complete	$40.2_{\pm 0.9}$	37.57 ↑
CoT@5	Incomplete	$48.1_{\pm 0.3}$	37.42
	Complete	$52.8_{\pm 0.5}$	41.42 ↑
DFS	Incomplete	$58.1_{\pm 0.4}$	37.14
	Complete	$54.9_{\pm 0.4}$	43.17↑

Table 7: CAPR@5 for long context and short context among different inference scaling algorithms.

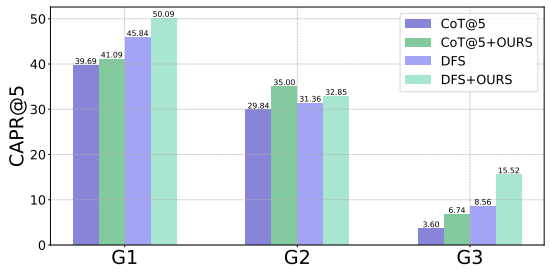


Figure 6: CAPR@5 comparison on *Agent Test* Set.

assessment of LLM agent performance.

CAPR Reflects Efficiency More Accurately. As shown in Table 7, the CAPR metric aligns with efficiency assessments from Table 1. For DFS, while incomplete context yields a higher Pass Rate, it does so inefficiently, requiring significantly more tool calls, as reflected by our CAPR metric. Further evaluations as visualized in Figure 6 show that our optimized context consistently outperforms baselines across various inference scaling methods and datasets. Notably, even when using a simple retry strategy (CoT@5), our method outperforms backup-based methods (DFS) on the G1 of *Tool Test* Set. These results not only validate the efficiency-focused nature of CAPR but also further confirm the superiority of our method.

CAPR Captures Task Difficulty. Request difficulty increases from G1 to G3 based on tool count and diversity. However, Table 4 shows that for DFS, Pass Rate on G2 is lower than on G3, contradicting the design. This is because PR ignores efficiency. In contrast, CAPR aligns with increasing difficulty, showing a performance decline from G1 to G3. This indicates CAPR better evaluates both effectiveness and efficiency, making it a superior metric for LLM agent performance.

7 Conclusion

In this work, we recognize and formalize the critical role of instructions provided in agent prompts and tool descriptions in efficient LLM agents for tool utilization. Our findings reveal that incomplete context, involving instructions and tool descriptions, often degrades the efficiency of inference scaling methods on tool utilization, by increasing the number of tool calls required to yield a correct response. To fill this efficiency gap, we propose an automated optimization system that jointly refines both instructions and tool descriptions, enabling more efficient LLM agents for tool utilization. This optimization process leverages verbalized feedback, capitalizing on the power of language. The experiments show that our approach leads to a remarkable reduction of tool calls by up to 70% on the StableToolBench dataset and avoids 47% redundant tool calls on RestBench, demonstrating the effectiveness of our optimization system in improving the efficiency of tool utilization.

Acknowledgments

Bin Wu is supported by a Bloomberg Data Science Ph.D. Fellowship. We thank the ARR reviewers for their feedback.

Limitations

Due to budget constraints, our experiments were conducted using the StableToolBench and RestBench. Since ToolBench and RestBench offer a diverse set of tools and requests, our findings are both interesting and valuable to the community. Future work will explore the effectiveness of our approach by testing it on other datasets, which could offer broader insights and further validate the effectiveness of our optimization system across different domains.

References

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pages 1877–1901.

Wei Fang, Yang Zhang, Kaizhi Qian, James R Glass, and Yada Zhu. 2024. Play2prompt: Zero-shot tool instruction optimization for llm agents via tool play. *openreview*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. 2024. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations*.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

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

Dirk P Kroese, Tim Brereton, Thomas Taimre, and Zdravko I Botev. 2014. Why the monte carlo method is so important today. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(6):386–392.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059.

Yanming Liu, Xinyue Peng, Yuwei Zhang, Jiannan Cao, Xuhong Zhang, Sheng Cheng, Xun Wang, Jianwei Yin, and Tianyu Du. 2024. Tool-planner: Dynamic solution tree planning for large language model with tool clustering. *arXiv preprint arXiv:2406.03807*.

Vaskar Nath, Pranav Raja, Claire Yoon, and Sean Hendryx. 2025. Toolcomp: A multi-tool reasoning & process supervision benchmark. *arXiv preprint arXiv:2501.01290*.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. 2025. Routellm: Learning to route llms from preference data. In *The Thirteenth International Conference on Learning Representations*.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. *arXiv preprint arXiv:2406.11695*.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. *arXiv preprint arXiv:1806.00187*.

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

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with" gradient descent" and beam search. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, et al. 2023. Restgpt: Connecting large language models with real-world restful apis. *arXiv preprint arXiv:2306.06624*.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024. Llms in the imaginarium: tool learning through simulated trial and error. *arXiv preprint arXiv:2403.04746*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N Ioannidis, Karthik Subbian, Jure Leskovec, and James Zou. 2024. Avatar: Optimizing llm agents for tool-assisted knowledge retrieval. *Advances in Neural Information Processing Systems*.

Tim Z Xiao, Robert Bamler, Bernhard Schölkopf, and Weiyang Liu. 2024. Verbalized machine learning: Revisiting machine learning with language models. *arXiv preprint arXiv:2406.04344*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng Li, and Deqing Yang. 2024. Easytool: Enhancing llm-based agents with concise tool instruction. *arXiv preprint arXiv:2401.06201*.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*.

Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun Wu. 2024. Offline training of language model agents with functions as learnable weights. In *Forty-first International Conference on Machine Learning*.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A Rossi, Somdeb Sarkhel, and Chao Zhang. 2023. Toolchain*: Efficient action space navigation in large language models with a* search. In *The Twelfth International Conference on Learning Representations*.

A Appendix

The appendix is structured as follows. We first show the prompts used in our experiments, followed by reports on additional experimental results.

A.1 Prompts

We show the prompts used for our proposed optimization system as follows:

Prompt for Feedback Generator

You are part of an optimization system that improves the given agent prompt and tool documentation. You are the feedback engine. Your only responsibility is:
Given the following user query, and the answer consisting of the final response, and tool invocation and its response, to generate the detailed feedback based on the following consideration:

1. Regarding the effectiveness of the solution,

   a. Did the response correctly and fully answer the query?
   b. If no response was generated, analyze the possible reasons why.
   c. If a final response is provided, evaluate whether it effectively utilizes the tool responses to construct a complete and accurate answer.

2. Regarding the efficiency of the solution:

   a. Were the tool calls necessary, or could the same result have been achieved with fewer tool invocations?
   b. Were there redundant or inefficient tool calls that could have been optimized? Please point the specific name and the reason.

Query:
{QUERY}
Answer:
{ANSWER}

Prompt for Suggestion Coordinator

You are part of an optimization system that improves the given text (i.e., task description and tool descriptions). Your only responsibility is:
Given the feedback on effectiveness and efficiency, the conversation history, and the current task desciption as well as each tool documentation, please explain how to improve the text one by one.

Feedback:
{FEEDBACK}

Converstaion History:
{CONVERSATION_HISTORY}

Task Description:
{TASK_DESCRIPTION}

Tool Descriptions:
{TOOL_DESCRIPTION}

Prompt for Context Refiner.

You are part of an optimization system that improves the given text (i.e., task description and tool descriptions). Your only responsibility is:
Given the text (i.e., agent prompt or tool documentation) and its associated improvement suggestion, update the context by adding new words or rewriting it. Please focuses more on the shared suggestion. Note that the optimized text should be in English even if the original text is not in English.

Text:
{TEXT}

Improvement Suggestion:
{IMPROVEMENT_SUGGESTION}

A.2 More Experimental Results

We include additional results comparing performance under scenarios with incomplete agent instructions and tool descriptions, as well as CAPR@5 results on *Tool Test* Set. As shown in

		Pass Rate (%) ↑				Cost Threshold ↓			
		G1	G2	G3	Ave.	15%	25%	35%	45%
Incomplete Agent Instruction	DFS	$55.0_{\pm 0.0}$	$40.0_{\pm 0.0}$	$45.0_{\pm 0.0}$	46.67	1.34	2.00	6.00	16.34
	OURS	$60.0_{\pm 0.0}$	$40.0_{\pm 0.0}$	$53.3_{\pm 4.7}$	51.10	1.34	1.67	6.00	11.34
Incomplete Tool Description	DFS	$55.0_{\pm 0.0}$	$40.0_{\pm 0.0}$	$50.0_{\pm 0.0}$	48.33	1.34	1.34	2.00	5.75
	OURS	$60.0_{\pm 0.0}$	$53.3_{\pm 2.4}$	$50.0_{\pm 0.0}$	54.43	1.00	1.34	2.00	4.50

Table 8: Effectiveness (i.e., Pass Rate) and efficiency (i.e., Cost Threshold) on a subset (20%) of *Tool Test* Set, with only optimizing on incomplete agent instructions / tool descriptions.

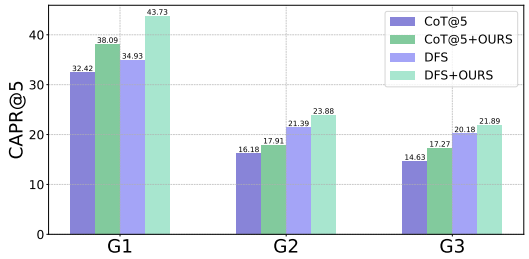


Figure 7: The CAPR@5 comparison on *Tool Test* Set.

Table 8, our proposed optimization system consistently outperforms baselines, achieving better performance with fewer tool calls, even under incomplete agent instructions or tool descriptions. These findings further demonstrate the robustness and effectiveness of our optimization framework. Similarly, the CAPR@5 results in Figure 7 indicate that our framework maintains superior performance on *Tool Test* Set, reinforcing the advantages of our approach.