TacTool: Tactical Tool Usage in Agentic AI Systems

Manavjeet Singh*, Kunal Rao†, Giuseppe Coviello† and Srimat Chakradhar†

*Stony Brook University, manavsingh@cs.stonybrook.edu

†NEC Laboratories America, Inc., {kunal, giuseppe.coviello, chak}@nec-labs.com

***Abstract*—Large language models (LLMs) are becoming the centerpiece in the design and deployment of Agentic artificial intelligence (AI) systems. AI agents typically have (a) *reasoning* ability to analyze and think through the given task, (b) *context/memory* to remember things in the short-term and long-term, and (c) *tools* at their disposal to interact with the outside world. While solving the given task, it must decide whether tool use is required; if so, it must then select the appropriate tool and invoke it with the correct parameters. Although LLMs have advanced considerably in recent years, their tool-use capabilities remain limited. Even OpenAI's most capable model to date, GPT-5, continues to struggle with reliable tool usage. In this paper, we propose *TacTool*, which empowers AI agents with improved tool selection and tool call formulation using different LLMs. We conduct experiments using Nestful and Berkeley Function Calling Leaderboard version 3 (BFCLv3) benchmarks and show that *TacTool* achieves ∼27% and ∼3% improvement over GPT-4o on Nestful and BFCL v3 dataset, respectively.**

***Index Terms*—Generative AI, Large Language Models, Agentic AI Systems, Tool Usage**

I. Introduction

Large language models (LLMs) have rapidly become the foundation of modern agentic AI systems, powering diverse applications in healthcare, education, retail, and more. Fig. 1 shows a high-level view of an agentic AI system. Agents built on such systems integrate LLMs, tools, and memory to autonomously solve a wide range of user queries. A crucial factor in their effectiveness is the ability to invoke external tools, such as third-party REST APIs, database functions, and custom user-defined functions. Tool use allows agents to access real-time information and perform specialized computations. Given the central role of tools in agentic AI systems, Anthropic recently introduced the Model Context Protocol (MCP) [1], a standardized framework that streamlines connecting external tools to AI agents.

Despite tool calling being natively supported in most recent LLMs (e.g., GPT-4o), reliably deciding when to use a tool, which tool to call, and how to formulate the call, remains a significant challenge. Benchmarks such as Nestful [2], Berkeley Function-Calling Leaderboard (BFCL) [3], τ^2-Bench [4], ACEBench [5], and many others [6]–[10], highlight that even state-of-the-art models exhibit errors in tool usage, tool selection, sequencing, and argument construction.

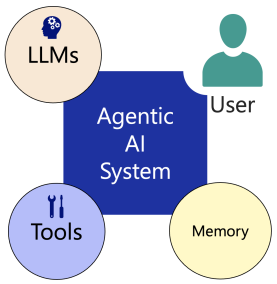


Figure 1: High-level view of Agentic AI System

Recent studies [11]–[13] have shown that accurate tool use critically depends on two intertwined abilities: reasoning about whether and how a tool is needed, and generating the corresponding function call correctly [1]. However, current LLMs rarely excel at both. For example, generic models such as GPT-4o achieve relatively strong tool-call formation but fall short in complex reasoning, often omitting fine-grained details necessary for accurate argument selection. In contrast, reasoning-oriented models such as o3, o4-mini, and DeepSeek-R1 exhibit strong deductive capabilities but struggle with reliable function-call construction. *In many cases, they default to answering from memory rather than invoking the appropriate tools to retrieve accurate, real-time information.* This mismatch suggests an opportunity: instead of relying on a single model, we can strategically combine complementary strengths across multiple LLMs.

In this paper, we introduce *TacTool*, a method for improving the accuracy of tool selection and invocation. *TacTool* decouples the reasoning and tool-calling stages, leveraging the strong reasoning capabilities of models such as o3 and DeepSeek-R1 while mitigating their weaknesses in function-call construction. Furthermore, *TacTool* exploits the complementary strengths of reasoning-oriented and general-purpose LLMs in making correct tool calls (see Section III). Our approach requires no model fine-tuning and instead relies on *tactical* coordination among multiple LLMs.

Our key contributions in this paper are as follows:

- We propose a novel method to tactically divide the tool usage problem into reasoning and tool-calling responsibilities, and then leverage different LLMs for effective

* Work done as an intern at NEC Laboratories America, Inc.

978-8-3315-5634-1/25/$31.00 ©2025 IEEE

[1]Throughout this paper we use 'tool-call' and 'function-call' interchangeably.

tool handling by agents in an agentic AI system.
- We evaluate *TacTool* on Nestful and BFCLv3 benchmarks and demonstrate that *TacTool* is consistently better than GPT-4o (improvement of ∼27% and ∼3% on Nestful and BFCLv3, respectively), unlike GPT-5 which is better than *TacTool* on Nestful, but worse than GPT-4o on BFCLv3.

The rest of the paper is organized as follows. We discuss related work in Section II and show the motivation for *TacTool* in Section III. We present details of the design and implementation of *TacTool* in Section IV. Then, in Section V, we report our experimental results and conclude in Section VI.

II. Related Work

Early work on LLM tool calling focused on training models to invoke tools [14]. With rapid advances in LLMs and standardized tool interfaces [1], modern models can now operate unseen tools, although imperfectly. This section reviews approaches that enhance LLM tool-calling, grouped into: (a) hierarchical tool categorization, (b) improved documentation, (c) reasoning-based prompting and verification, and (d) separation of reasoning and execution.

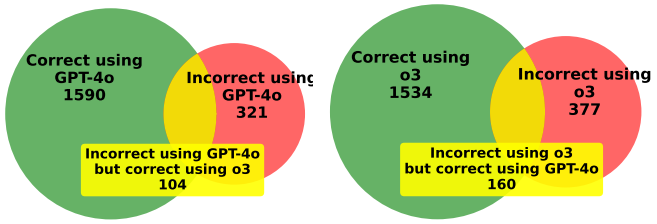
Hierarchical categorization. AnyTool [15] and ToolRerank [16] leverage predefined API categories from RapidAPI [17] for accurate tool selection. AnyTool uses hierarchical agents with self-reflection feedback to iteratively improve tool calls, while ToolRerank applies hierarchy-aware reranking for better selection. Such methods work well for large tool pools but rely on well-defined categories, which may not always exist.

Documentation enhancement. EasyTool [18] automatically refines tool descriptions, improving both accuracy and token efficiency. DRAFT [19] extends this idea through iterative self-reflection, achieving higher accuracy across models. Note that better tool documentation directly improves the tool calling accuracy.

Reasoning-based prompting. Recent studies combine reasoning-oriented prompts and fine-tuning to boost performance. Chain-of-Thought (CoT) [20] enables stepwise reasoning; ReAct [21] and DFSDT [22] extend this with action-based and search-based frameworks for higher accuracy but greater token cost. ToolLLM [22] fine-tunes on DFSDT paths for improved accuracy, while Divide-then-Aggregate (DTA) [23] reduces token usage via DAG-based parallelization. ARTIST [12] employs reinforcement learning for multi-turn improvement. Xu et al. [24] estimate model knowledge boundaries to reduce unnecessary tool calls.

Decoupled reasoning and execution. Zhang [25] separates reasoning and tool-calling agents, fine-tuning the reasoner via reinforcement learning for better coordination. ToolACE-R [26] uses self-refinement to enhance reasoning and fine-tune response generation.

Most existing approaches depend on fine-tuning, which demands labeled data and significant computation. In contrast, *TacTool* requires no fine-tuning and flexibly integrates outputs from different LLMs, adapting seamlessly as newer models emerge.



(a) (b)

Figure 2: (a) Venn diagram representing correct (green) and incorrect (red) responses for (a) GPT-4o and (b) o3 on selected categories of BFCLv3 dataset. Overlap (yellow) represent the cases answered incorrectly by (a) GPT-4o and (b) o3 but correctly answered by (a) o3 and (b) GPT-4o, respectively. Highlighting their complementary nature.

Table I: Comparing scores of reasoning and general-purpose models on four categories from BFCLv3.

Model	Category-wise accuracy (%)			
	simple	live_ simple	multiple	live_ multiple
o3-2025-04-16-FC	91	78.2	88.5	75.1
o4-mini-2025-04-16-FC	91.2	77.1	86	75.5
gpt-4o-2024-11-20-FC	92.7	81	92.5	78.5
gpt-4.1-2025-04-14-FC	93.5	79.4	90.5	77.7
gpt-4.1-mini-2025-04-14-FC	92.2	80.6	92.5	78.4

III. Motivation

In this section, we examine the limitations of general-purpose and reasoning LLMs for tool calling and highlight how tactical usage of both can be advantageous.

We compare general-purpose models (e.g., GPT-4 series) with reasoning models (e.g., OpenAI's "o" series) on tool selection and execution. As shown in Table I, general-purpose models (GPT-4o, GPT-4.1, GPT-4.1-mini) consistently outperform reasoning models across BFCLv3's *simple*, *live_simple*, *multiple*, and *live_multiple* categories (1911 samples). This trend persists across providers (e.g., DeepSeek) and benchmarks [5], suggesting reasoning models are often less effective for straightforward tool calls. For instance, when asked for a lesser-known person's birthday, o3 relied on memory, while GPT-4o correctly queried the database.

However, this gap is not absolute. Figure 2a shows that some GPT-4o failures are correctly handled by o3, and vice versa (Figure 2b), revealing complementary error patterns. In detail-oriented tasks such as retrieving an employee's professional experiences with the multi-purpose tool *experiences_and_education*, which requires boolean arguments *include_experiences* and *include_education*, generic LLMs often overlook finer details. For example, o3 correctly sets *include_experiences* to true and *include_education* to false, whereas GPT-4o fails to do so. This advantage can be attributed to o3's stronger reasoning and planning abilities.

When a reasoning model first outlines a problem-solving approach and its explanation is then provided to a general-purpose model along with tool options, the latter produces results that surpass both standalone models. *TacTool* leverages

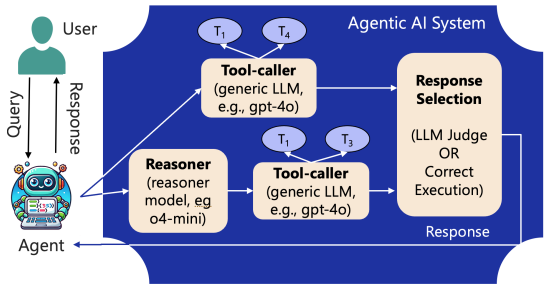


Figure 3: *TacTool* System Architecture

this insight to construct an enhanced, hybrid tool-calling pipeline for AI agents.

IV. Design and Implementation

In this section, we first present details of the design and implementation of *TacTool* and then we also discuss another voting-based strategy to improve accuracy of tool usage in agentic AI systems.

A. TacTool

Our observations that (i) providing a reasoning model's (e.g., o4-mini) plan to a generic LLM (e.g., GPT-4o) during tool use improves accuracy, and (ii) reasoning and generic models offer complementary strengths, motivated a system design that leverages both while mitigating their weaknesses.

Figure 3 shows *TacTool*'s architecture. Upon receiving a user query, two execution paths run in parallel: (1) a generic LLM (e.g., GPT-4o) selects and executes tool calls to produce the answer, and (2) a reasoning LLM (e.g., o4-mini) analyzes the query and generates a logical plan using available tools, without executing the calls. The prompt used in this reasoning step is as follows:

```
Concisely explain the process for solving the
problem using the given tools. Problem:{question}.
Given tools: {tools}
```

Next, the reasoning model's plan is passed to a generic LLM, which uses it to execute the tool call with the appropriate arguments. The LLM prompt used is as follows:

```
Here is the process to solve the problem:
{output − of − reasoner}
Return a tool call string WITHOUT any
explanation.
```

Splitting the process in this manner increases the likelihood of correct tool usage. Because LLM outputs are non-deterministic, *TacTool* incorporates a response selection module to identify the most plausible answer from the two parallel paths. If one path fails due to a tool-call error, its output is discarded; if both fail, *TacTool* returns a reasoned fallback response. When both paths succeed, the module uses an LLM-as-a-judge approach to select the final answer. The prompt for this step is as follows:

```
You are very good at reasoning. You will be
given a question and two possible answers for
it. Select the best answer among the two.
```

Table II: Nestful results ('↓' = lower is better, '↑' = higher is better).

Method/Model	Logic (%) ↓	Format (%) ↓	Correct (%) ↑
gpt-4o	40.2	7.2	52.6
Voting	36.5	3.8	59.7
o4-mini	21.5	7.8	70.7
ReAct	30.6	11.1	58.3
DFSDT	27.5	10.3	62.2
TacTool	**20.8**	**0.2**	**79.0**

```
Problem: {question}
First answer (A): {answer − 1}
Second Answer (B): {answer − 2}
Return only the correct selection with an upper
case A or B without any reasoning content.
```

The most plausible correct answer selected by the response selection module goes back to the agent and finally to the user.

B. Voting

Different LLMs have complementary strengths and biases, producing varied outputs. The voting method aggregates $toolcalls^{n\times t}$ from n LLMs, where $toolcalls[i] = [toolcall_1, \ldots, toolcall_t]$ and each $toolcall$ has keys $'name'$ and $'args'$. For each call, the most-voted tool and arguments are selected, repeating t times.

For tasks with multiple valid sequences (eg. Nestful benchmark [2]), we instead use an LLM-judge, prompted with the user query, tool calls from all LLMs, and function descriptions.

```
Evaluate which function call(s) best match the
given prompt.
Given the prompt: {user_question}, and the returned
function calls:
Model 1 function calls: {model1_toolcall}
...
Model n function calls: {modeln_toolcall}
Identify the correct call(s) and parameters. You
may mix and match parameters from the provided
calls, but do not create any new information.
Base your decision on weighted voting of function
names and parameters, ensuring syntactic and
logical correctness.
Function descriptions: {function_descriptions}.
Return only the correct function call(s) and
parameters in valid JSON format. No explanations,
no extra characters, and do not modify any
function or parameter names.
```

V. Experimental Results

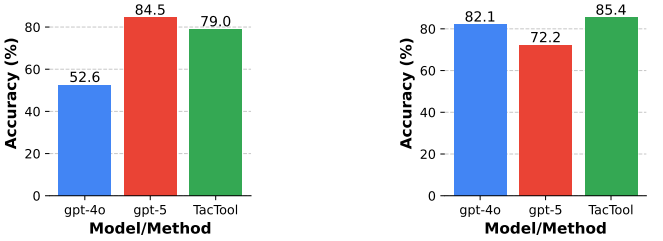
We evaluate *TacTool* on two tool-calling benchmarks: *Nestful* [2] and Berkeley Function Calling Leaderboard v3 (BFCLv3) [3]. *Nestful* measures nested tool use in math and programming (1861 problems), while BFCLv3 includes ten tool-use categories totaling 2501 problems.

Baselines. We compare *TacTool* with ReAct [21], DFSDT [22], GPT-4o, o4-mini (Nestful only[2]), and a voting ensemble (GPT-4o, GPT-4.1, GPT-4.1-mini; see Section IV-B).

[2]BFCLv3 does not support parallel tool calling with o4-mini, hence skipped.

Table III: BFCLv3 results ('↓' = lower is better, '↑' = higher is better).

Method/Model	Logic (%) ↓	Format (%) ↓	Accuracy (%) ↑ across BFCLv3 categories										Avg. ↑
gpt-4o	16.9	1.0	92.7	81.0	92.5	81.2	85.0	79.1	92.5	78.5	65.0	74.0	82.1
Voting	17.9	**0.0**	94.2	84.1	91.5	75.0	89.0	75.0	93.0	**80.2**	**67.0**	72.0	82.1
ReAct	18.7	0.7	93.5	83.3	**94.5**	81.2	86.5	79.1	91.5	73.3	65.0	58.0	80.6
DFSDT	62.0	1.0	52.0	36.8	37.5	37.5	22.0	33.3	48.0	38.5	27.0	26.0	37.0
TacTool	**14.2**	0.4	**97.2**	**89.5**	91.0	**87.5**	**89.5**	**83.3**	**94.5**	79.6	64.0	**78.0**	**85.4**



(a) Nestful (b) BFCLv3

Figure 4: Performance comparison of GPT-4o, GPT-5, and *TacTool* on Nestful and BFCLv3.

Setup. *TacTool* uses GPT-4o as the tool caller and o4-mini as the reasoner. Tables II and III report results. "Incorrect Logic" denotes wrong tool or argument choices (main error source), and "Incorrect Format" reflects syntax issues, more common in Nestful due to its unique nested call syntax.

On *Nestful*, *TacTool* achieves 79% accuracy, which is 26.4% and 8.3% higher than GPT-4o and o4-mini, and 20.7% and 16.8% higher than ReAct and DFSDT, respectively. On *BFCLv3*, *TacTool* reaches 85.4% average accuracy, surpassing GPT-4o, Voting, and ReAct by 3.2–4.8%. DFSDT performs poorly due to repeated calls and lack of context reuse.

Comparison with GPT-5. We compare *TacTool* with the recently released GPT-5 in Figure 4, which uses an internal router to decide the need for deeper reasoning. GPT-5 slightly outperforms *TacTool* on Nestful but underperforms even GPT-4o on BFCLv3. A key issue is its tendency to answer directly rather than invoke tools. While this may suffice for simple math or general knowledge, it fails on tasks requiring specific APIs. For example, when generating normally distributed random variables with a designated tool, GPT-5 instead used a Python library function, highlighting its difficulty in reliably using given tools over familiar but inappropriate alternatives.

VI. Conclusion

LLM agents are typically equipped with set of "tools", which they can use to arrive at the final solution. In this paper, we *show* that even the most capable LLMs, e.g. GPT-5 are not adept at using tools properly in a consistent manner. To overcome this problem, in this paper, we *propose TacTool*, which tactically combines responses from different LLMs to arrive at the final answer. Our experimental results demonstrate that *TacTool* improves accuracy of tool usage over vanilla GPT-4o by ∼27% and ∼3% on Nestful and BFCL v3 dataset, respectively.

References

[1] Anthropic, "Model Context Protocol (MCP)," Online Standard, 2024, accessed on August 26, 2025. [Online]. Available: https://modelcontextprotocol.io/docs/getting-started/intro

[2] K. Basu *et al.*, "NESTFUL: A benchmark for evaluating LLMs on nested sequences of API calls," *arXiv [cs.AI]*, Sep. 2024.

[3] "Berkeley function calling leaderboard V3 (aka berkeley tool calling leaderboard V3)," https://gorilla.cs.berkeley.edu/leaderboard.html, accessed: 2025-8-7.

[4] V. Barres *et al.*, "τ^2-bench: Evaluating conversational agents in a dual-control environment," 2025. [Online]. Available: https://arxiv.org/abs/2506.07982

[5] C. Chen *et al.*, "Acebench: Who wins the match point in tool usage?" 2025. [Online]. Available: https://arxiv.org/abs/2501.12851

[6] Q. Tang *et al.*, "ToolAlpaca: Generalized tool learning for language models with 3000 simulated cases," *arXiv [cs.CL]*, Jun. 2023.

[7] Z. Guo *et al.*, "StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models," *arXiv [cs.CL]*, Mar. 2024.

[8] M. Li *et al.*, "API-bank: A comprehensive benchmark for tool-augmented LLMs," *arXiv [cs.CL]*, Mar. 2024.

[9] J. Lu *et al.*, "ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities," *arXiv [cs.CL]*, Aug. 2024.

[10] L. Zhong *et al.*, "ComplexFuncBench: Exploring multi-step and constrained function calling under long-context scenario," *arXiv [cs.CL]*, Jan. 2025.

[11] W. Li *et al.*, "Adaptive tool use in large language models with meta-cognition trigger," 2025. [Online]. Available: https://arxiv.org/abs/2502.12961

[12] J. Singh *et al.*, "Agentic reasoning and tool integration for LLMs via reinforcement learning," *arXiv [cs.AI]*, Apr. 2025.

[13] S. Qiao *et al.*, "Making language models better tool learners with execution feedback," 2024. [Online]. Available: https://arxiv.org/abs/2305.13068

[14] T. Schick *et al.*, "Toolformer: Language models can teach themselves to use tools," *arXiv [cs.CL]*, Feb. 2023.

[15] Y. Du *et al.*, "AnyTool: Self-reflective, hierarchical agents for large-scale API calls," *arXiv [cs.CL]*, Feb. 2024.

[16] Y. Zheng *et al.*, "ToolRerank: Adaptive and hierarchy-aware reranking for tool retrieval," *arXiv [cs.IR]*, Mar. 2024.

[17] RapidAPI, "Rapid technology," https://rapidapi.com/hub, accessed: 2025-8-4.

[18] S. Yuan *et al.*, "EASYTOOL: Enhancing LLM-based agents with concise tool instruction," *arXiv [cs.CL]*, Jan. 2024.

[19] C. Qu *et al.*, "From exploration to mastery: Enabling LLMs to master tools via self-driven interactions," *arXiv [cs.CL]*, Oct. 2024.

[20] J. Wei *et al.*, "Chain of thought prompting elicits reasoning in large language models," *Neural Inf Process Syst*, vol. abs/2201.11903, Jan. 2022.

[21] S. Yao *et al.*, "ReAct: Synergizing reasoning and acting in language models," *arXiv [cs.CL]*, Oct. 2022.

[22] Y. Qin *et al.*, "ToolLLM: Facilitating large language models to master 16000+ real-world APIs," *arXiv [cs.AI]*, Jul. 2023.

[23] D. Zhu *et al.*, "Divide-then-aggregate: An efficient tool learning method via parallel tool invocation," *arXiv [cs.LG]*, Jan. 2025.

[24] H. Xu *et al.*, "Alignment for efficient tool calling of large language models," *arXiv [cs.CL]*, Mar. 2025.

[25] Y. Zhang, "Agent-as-tool: A study on the hierarchical decision making with reinforcement learning," *arXiv [cs.AI]*, Jul. 2025.

[26] X. Zeng *et al.*, "ToolACE-R: Tool learning with adaptive self-refinement," *arXiv [cs.CL]*, Apr. 2025.