Proceedings of CONF-SPML 2026 Symposium: The 2nd Neural Computing and Applications Workshop 2025
DOI: 10.54254/2755-2721/2026.TJ28954

Empowering LLM-based Agents: Methods and Challenges in Tool Use

Xinyue Du

School of Information Science and Engineering, East China University of Science and Technology (ECUST), Shanghai, China
xiaoduxiaodu09@gmail.com

Abstract. The emergence of Large Language Model (LLM)-based agents marks a significant step towards more capable Artificial Intelligence. However, the effectiveness of these agents is fundamentally constrained by the static nature of their internal knowledge. Tool use has become a critical paradigm to overcome these limitations, enabling agents to interact with dynamic data, execute complex computations, and act upon the world. This paper provides a comprehensive survey of the methods, challenges, and future directions in empowering LLM-based agents with tool-use capabilities. Through a systematic literature review, we synthesized the current state of the art, charting the evolution from foundational agent architectures and core invocation mechanisms like function calling to advanced strategies such as dynamic tool retrieval and autonomous tool creation. Our analysis revealed several critical challenges that impede the deployment of robust agents, including knowledge conflicts between internal priors and external evidence, significant performance degradation in long-context scenarios, non-monotonic scaling behaviors in compound systems, and novel security vulnerabilities. By mapping the current research landscape and identifying these key obstacles, this survey proposes a research agenda to guide future efforts in building more capable, secure, and reliable AI agents.

Keywords: Large Language Models, AI Agents, Tool Use, Function Calling

1. Introduction

The rapid advancement of Large Language Models (LLMs) is increasingly seen as a potential catalyst for Artificial General Intelligence (AGI), providing a novel pathway for the development of general-purpose AI agents [1]. However, standalone LLMs face inherent limitations, such as possessing static knowledge, lacking expertise in specialized domains, and being unable to interact with the real-time world. These constraints have spurred the emergence of LLM-based agents, which aim to overcome these shortcomings [1-3]. Central to this endeavor is the concept of tool use, which serves as the cornerstone for empowering these agents by extending their capabilities beyond their native functionalities [1-3]. While significant progress has been made, the field lacks a systematic overview that consolidates the diverse methods, advanced strategies, and critical challenges associated with enabling agents to use tools effectively. This paper addresses this gap by providing a comprehensive survey of tool use in LLM-based agents. We aim to answer several key questions:

DOI: 10.54254/2755-2721/2026.TJ28954

What are the foundational frameworks for tool-augmented agents? How do agents learn, invoke, and optimize the use of tools? What advanced strategies, such as tool retrieval and creation, are being developed to enhance autonomy? Finally, what are the critical challenges and frontier problems, including knowledge conflicts, security risks, and performance in complex scenarios? To investigate these questions, this paper conducts a systematic literature review, synthesizing cutting-edge research to present a structured and holistic view of the field it is significant as it not only maps the current landscape but also highlights unresolved challenges and forecasts future research trajectories. By doing so, this study aims to provide valuable insights for researchers and practitioners, offering a guide for developing more autonomous, robust, and capable AI agents.

2. Foundational frameworks for tool-augmented agents

2.1. The general architecture of LLM-based agents

A general framework for LLM-based agents typically comprises four core modules: Profiling (defining the agent's role), Memory (managing information), Planning (decomposing tasks), and Action (executing sub-goals) [2]. The Action Module is where tool use becomes central, serving as the bridge between the agent's internal reasoning and the external world. It is the component responsible for orchestrating tool calls to execute planned tasks.

2.2. Classification and forms of tools

The tools available to an agent fall into two primary categories. The first, External Tools, consists of explicit, callable resources. These include making API calls to external services (e.g., search engines, computational tools), querying structured databases, and integrating other specialized models like code interpreters [2]. The second category involves leveraging the LLM's Internal Knowledge as implicit tools. This refers to the agent's ability to utilize the intrinsic capabilities of its underlying model—such as high-level planning, conversational reasoning, and commonsense understanding—to perform actions that do not require external calls [2].

2.3. Mechanisms for learning tool use

An agent’s ability to use tools is not innate but learned through several mechanisms. Initially, an agent learns to understand a tool's function and syntax through zero-shot or few-shot prompting, where it is provided with descriptions or examples, much like reading a manual [1]. Subsequently, it learns to master the tool through experience, either by imitating expert demonstrations or by refining its strategies based on environmental and human feedback [1]. Ultimately, a truly capable agent must adapt by generalizing its skills to novel tools and situations, a process that can be accelerated through advanced methods like meta-tool learning and curriculum learning [1].

3. Core mechanisms and optimizations for tool invocation

Once an agent understands what tools are and how to learn them, the next critical step is the invocation—the process of planning, executing, and optimizing tool calls. This has evolved from simple chained commands to sophisticated, efficient, and robust systems.

DOI: 10.54254/2755-2721/2026.TJ28954

3.1. Planning and execution frameworks

The strategy for how an agent plans and executes tool calls significantly impacts its performance. One foundational approach is to interleave reasoning and acting, as exemplified by the ReAct framework. In this model, the agent generates a reasoning trace to analyze the problem, decides on an action (a tool call), executes it, observes the result, and repeats this cycle until the task is complete. This tight loop allows for dynamic adjustments but can increase latency due to multiple sequential model interactions [4]. Variants like ChatCoT adapt this interactive process for multi-turn conversational settings [4]. In contrast, other frameworks decouple planning from execution. The ReWOO model, for instance, employs a Planner-Worker-Solver architecture. The Planner first generates a complete, static plan of all required tool calls. Then, multiple Workers execute these calls, often concurrently. Finally, a Solver integrates all the evidence gathered by the Workers to produce the final answer [5]. This approach can improve efficiency and reduce the number of required LLM calls. More advanced systems like LangGraph utilize graph-based structures to define agent workflows, enabling more flexible and complex cyclical processes that go beyond simple linear chains [4].

3.2. Function calling: the De facto paradigm

Modern tool invocation is predominantly implemented through Function Calling, where the LLM outputs a structured representation (typically a JSON object) specifying the tool's name and the arguments for its invocation. This paradigm has several implementation pathways and optimization strategies.

3.2.1. Implementation paths

The most direct path is through prompt engineering, where in-context examples and detailed instructions guide a general-purpose LLM to produce the correct structured output without any model modification [6]. While flexible, this approach can lack robustness. A more reliable method is fine-tuning, where the model is trained on a dataset of domain-specific tool-calling examples. This injects specialized knowledge directly into the model, significantly improving accuracy and reliability [7, 8]. A third path involves using modular frameworks like eidos, which externalize the process by allowing declarative registration and validation of tools. This decouples the tool ecosystem from the LLM, enhancing modularity and security [9].

3.2.2. Enhancing efficiency and performance

A major bottleneck in tool use is latency. Asynchronous calling, as proposed by AsyncLM, addresses this by breaking the synchronous "call-wait-respond" loop. It allows the LLM to continue generating while a tool executes in the background, with the result inserted later via an interrupt, boosting concurrency and speed [10]. For tasks requiring multiple sequential steps, frameworks like the Simple Action Model aim to reduce the number of LLM interactions by allowing a single model output to define a complete chain of actions to be executed sequentially [11]. Furthermore, techniques like distillation have given rise to dual-model architectures where a small, fast "router" model handles simple, frequent requests, delegating more complex tasks to a larger model, thereby optimizing overall system latency [12].

DOI: 10.54254/2755-2721/2026.TJ28954

3.2.3. Improving robustness and accuracy

Ensuring the correctness of tool calls is paramount. This can be improved through carefully designed prompt and data strategies, such as optimizing prompt templates, fusing tool-use data with general instruction-following data, and introducing special "decision tokens" that force the model to explicitly decide whether to use a tool [13]. For greater reliability, closed-loop verification and rectification offer a powerful solution. The ThorV2 architecture, for example, uses an Agent-Validator model where a programmatic validator checks the agent's proposed tool call. If an error is detected, feedback is provided to the agent, which then attempts to correct its call, creating an iterative self-correction loop that significantly boosts accuracy [14].

3.3. Data-driven capability acquisition via data synthesis

The effectiveness of fine-tuned tool-using agents is heavily dependent on the quality and diversity of their training data, which is often a major bottleneck. To overcome this, automated data generation pipelines like ToolACE have been developed [15, 16]. Such frameworks systematically create high-quality, large-scale datasets through a multi-stage process. First, a diverse set of tools is generated via API self-evolution (TSS) to ensure broad coverage [15]. Next, a self-guided dialogue generation (SDG) process, often using a multi-agent setup, creates realistic tool-use scenarios. Crucially, this stage often includes dynamic complexity adjustment, tailoring the difficulty of generated tasks to the target model's current abilities [15, 16]. Finally, a dual-layer verification (DLV) step uses both rule-based checkers and powerful LLMs to validate the syntactic and semantic correctness of the generated data, ensuring its quality and executability [15, 16].

4. Advanced tool management: from retrieval to creation

As agents become more sophisticated, their interaction with tools evolves beyond simply using a predefined set. Advanced tool management encompasses the ability to navigate vast tool libraries and even to create novel tools, marking a significant leap towards greater autonomy. This section explores these advanced capabilities, focusing on tool retrieval in open-world settings and the paradigm of automated tool creation and evolution.

4.1. Tool retrieval and selection in the open world

In realistic scenarios, an agent may have access to hundreds or thousands of potential tools, making the decision of "whether to use a tool" and "which tool to use" a critical challenge [17]. This "open-world" problem requires mechanisms for dynamic tool retrieval rather than relying on a fixed set of in-context tools. The Meta-Tool framework addresses this by proposing a "hypothesize-retrieve-invoke" paradigm. Instead of having all tools available in its context, the agent first hypothesizes the kind of functionality it needs, then retrieves the most relevant tool from a large external library, and finally, invokes it [18]. This approach is highly scalable and mirrors how humans find and use new software or APIs. To systematically evaluate these capabilities, dedicated benchmarks like the Meta-Tool Benchmark have been developed. They are designed to assess an agent's "tool use awareness" (the ability to correctly decide if a tool is necessary) and its "tool selection" accuracy from a large pool of options, providing a standardized way to measure progress in this crucial area [17].

DOI: 10.54254/2755-2721/2026.TJ28954

4.2. Automated tool creation and evolution

The pinnacle of tool management is the ability for an agent to create its own tools, transitioning from a "tool user" to a "tool maker." This capability allows an agent to generate bespoke, reusable solutions for novel problems, embodying a higher level of autonomous problem-solving. Several frameworks have been proposed to realize this vision. LATM (Large Language Models as Tool Makers) introduces a division of labor: a powerful "Tool Maker" model (e.g., GPT-4) generates a general-purpose tool from a few examples, while a more efficient "Tool User" model (e.g., GPT-3.5) repeatedly uses this tool. This architecture achieves a remarkable balance between high performance and significantly reduced computational cost [19]. The CREATOR framework further refines this by decoupling abstract reasoning (creating the tool) from concrete reasoning (deciding how to use it). It also introduces a crucial execution-rectification loop, enabling the agent to test its created tools and self-correct based on execution feedback [20]. CRAFT likewise follows a disciplined pipeline—generate, abstract, verify, deduplicate—to yield a lean, non-redundant tool library that is accessed through a multi-faceted retrieval engine [21]. Beyond one-time creation, the TROVE framework explores the continuous evolution of a toolset. It operates on a dynamic "Use-Grow-Trim" loop: the agent deploys existing tools, spawning new ones on demand, and periodically prunes those that are inefficient or seldom used. This enables the agent's toolbox to adapt and optimize itself over time, ensuring its continued relevance and efficiency [22, 23].

5. Key challenges and frontier problems

Despite rapid advancements, deploying robust and reliable tool-augmented agents in real-world scenarios is fraught with challenges. These range from fundamental conflicts in knowledge sources to performance degradation in complex situations and emerging security vulnerabilities. This section details these key challenges and frontier problems that the field currently faces.

5.1. Knowledge conflict: internal prior vs. external evidence

A fundamental challenge arises when the information retrieved from an external tool conflict with the LLM's internal, parametric knowledge. Research using benchmarks like ClashEval reveals that models exhibit a strong context bias; they are more than 60% likely to be misled by incorrect external evidence, even when their internal knowledge is correct [24]. This preference for external context makes agents vulnerable to misinformation from unreliable tools or poisoned data sources. The problem is exacerbated when agents use their own generated context, as they show an even stronger preference for self-generated information over retrieved evidence, a phenomenon described as being "Blinded by Generated Contexts" [25]. This can create a dangerous feedback loop, where an initial error is reinforced and amplified.

5.2. Robustness in complex and long-context scenarios

The effectiveness of tool use often degrades as task complexity and context length increase. The ComplexFuncBench benchmark defines complexity through characteristics like the need for multi-step tool calls, adherence to strict user constraints, and the inference of implicit parameters [26]. Furthermore, research with the LongFuncEval benchmark has identified three critical long-context challenges that severely impact performance: navigating large tool catalogs with hundreds of potential functions, parsing long tool responses such as verbose JSON outputs from APIs, and maintaining context across long multi-turn conversations [27]. A universal finding is that nearly all

DOI: 10.54254/2755-2721/2026.TJ28954

models, regardless of their stated context window size, suffer significant performance degradation—in some cases by as much as 85-91%—when faced with these long-context challenges. They also exhibit strong positional biases, such as failing to find information in the middle of the context ("lost in the middle") or overly favoring the most recent information [27].

5.3. The scaling paradox of compound systems

It is a common misconception that in compound AI systems, such as those that aggregate results from multiple tool calls, increasing the number of calls invariably leads to better performance. Recent studies uncover a non-monotonic scaling paradox: accuracy traces a U- or inverted-U curve as more LLM calls are issued [28]. For "easy" queries, more calls can help correct minor errors through mechanisms like majority voting. However, for "difficult" queries, additional calls are more likely to introduce further errors or reinforce incorrect initial assumptions, leading to a decline in performance. This finding implies that blindly increasing computational budget is not an effective strategy. Instead, future systems must develop adaptive strategies that dynamically adjust the number of tool calls based on the perceived difficulty of the query [28].

5.4. Security risks: new attack surfaces from function calling

The introduction of function-calling capabilities creates new, previously unexamined attack surfaces. A significant threat is the "Jailbreak Function Attack," where an adversary crafts a seemingly benign function call to induce the LLM to generate harmful or restricted content within the function's parameters [29]. This attack is highly effective, with success rates exceeding 90% against major LLMs. The vulnerability stems from several root causes: the function-calling modality often lacks the same rigorous safety alignment applied to standard chat interactions, system parameters can be used to force the model to use a specific tool, and there is a general lack of content filtering on the data flowing through the tool-calling interface [29]. This highlights an urgent need to rethink security and alignment for the entire tool-use pipeline.

5.5. Challenges in specific domains

It is a common misconception that in compound AI systems, such as those that Beyond these general issues, significant challenges arise when applying tool use to specific domains. When querying databases, models struggle to differentiate between similar operations like text filters and semantic search, and smaller models perform poorly on complex queries with multiple constraints [30]. In graph reasoning, LLMs are prone to hallucination and require multi-turn, closed-loop interactions with graph computation libraries to achieve accuracy, often needing self-correction mechanisms to recover from errors [31]. For small language model (SLM) deployment on edge devices, while fine-tuning can significantly boost tool-calling performance, it comes at the cost of high latency, creating a trade-off between capability and responsiveness [8]. Conversely, in areas like dialogue state tracking, reframing the problem as a function-calling task has proven highly effective, enabling zero-shot performance where previous methods required extensive training data [32].

DOI: 10.54254/2755-2721/2026.TJ28954

6. Conclusion and future directions

6.1. Conclusion

The ability to use tools represents a pivotal and transformative leap in the evolution of Large Language Models. It is the critical catalyst that elevates them from being passive generators of text to becoming active, goal-oriented agents capable of interacting with, reasoning about, and acting upon the world. This paper has provided a systematic overview of this rapidly advancing field, charting a course from foundational agent architectures to the sophisticated mechanisms of tool invocation, management, and autonomous creation. We have seen a clear trajectory from simple tool users to sophisticated tool makers, reflecting a steady march towards greater autonomy. However, our review also illuminates a set of profound challenges that mark the current research frontier, including fundamental knowledge conflicts, performance degradation in complex scenarios, counter-intuitive scaling paradoxes, and urgent security vulnerabilities.

6.2. An agenda for future research

Addressing these formidable challenges requires a multi-faceted research agenda that points toward several key future directions. First, the development of tool ecosystem standardization and interoperability is essential to move beyond the current fragmented landscape. Second, agents must achieve stronger autonomy through self-correction and lifelong learning, enabling them to diagnose failures and continuously adapt their strategies [3, 6, 30]. Third, we need more advanced tool orchestration and workflow optimization to handle complex, multi-step tasks efficiently [3, 5, 10]. A particularly promising path is the creation of pre-trained tool-augmented foundation models, which would make tool use a more native and reliable faculty [3]. Finally, and most critically, security and reliability must be addressed by design, incorporating robust validation layers and specialized safety alignment to fortify agents against manipulation [9, 14, 29]. Successfully navigating these research avenues will be the key to unlocking the full, transformative potential of LLM-based agents. Doing so will move them from promising prototypes in controlled environments to reliable, trustworthy, and indispensable partners in science, industry, and our daily digital lives, ultimately paving the way for the next generation of general-purpose artificial intelligence.

References

[1] Chen, H., Liu, Q., Sun, Z. (2023) The rise and potential of large language model-based agents: a survey. Science China – Information Sciences, 66: 1–20.
[2] Wang, Y., Zhao, M., Li, K. (2023) A survey on large language model based autonomous agents. Frontiers of Computer Science, 17: 1–18.
[3] Zhang, H., Lin, J., Wang, X. (2024) LLM With Tools: A Survey. arXiv preprint.
[4] Sun, Y., He, Z., Li, J. (2024) LLMs Working in Harmony: A Survey on the Technological Aspects of Building Effective LLM-Based Multi Agent Systems. arXiv preprint.
[5] Zhang, X., Li, Y., Wang, J. (2024) A study on classification based concurrent API calls and optimal model combination for tool augmented LLMs for AI agent. Scientific Reports, 14: 1–14.
[6] Li, J., Zhao, K., Hu, F. (2024) Achieving Tool Calling Functionality in LLMs Using Only Prompt Engineering Without Fine-Tuning. arXiv preprint.
[7] Gao, L., Chen, H., Xu, W. (2024) Granite-Function Calling Model: Introducing Function Calling Abilities via Multi-task Learning of Granular Tasks. arXiv preprint.
[8] Wang, Y., Zhang, Z., Xu, J. (2025) Small Models, Big Tasks: An Exploratory Empirical Study on Small Language Models for Function Calling. arXiv preprint.

DOI: 10.54254/2755-2721/2026.TJ28954

[9] Johnson, M., Lee, D., Parker, S. (2025) eidos: A modular approach to external function integration in LLMs. SoftwareX, 24: 101–110.
[10] Li, D., Chen, J., Yang, Q. (2024) Asynchronous LLM Function Calling. arXiv preprint.
[11] Kumar, S., Patel, R., Singh, A. (2024) Simple Action Model: Enabling LLM to Sequential Function Calling Tool Chain. Proceedings of the International Conference on Advancement in Renewable Energy and Intelligent Systems (AREIS): 1–8.
[12] Huang, P., Zhou, J., Wang, T. (2025) ODIA: Oriented Distillation for Inline Acceleration of LLM-based Function Calling. arXiv preprint.
[13] Zhao, R., Chen, Q., Fang, L. (2024) Enhancing Function-Calling Capabilities in LLMs: Strategies for Prompt Formats, Data Integration, and Multilingual Translation. arXiv preprint.
[14] Lee, C., Park, J., Smith, A. (2024) Benchmarking Floworks against OpenAI & Anthropic: A Novel Framework for Enhanced LLM Function Calling. arXiv preprint.
[15] Tan, R., Luo, M., Wei, H. (2024) ToolACE: Winning the Points of LLM Function Calling. arXiv preprint.
[16] Wang, F., Zhang, H., Zhao, K. (2024) Improving Small-Scale Large Language Models Function Calling for Reasoning Tasks. arXiv preprint.
[17] Zhou, T., Wang, Y., Liu, Q. (2023) MetaTool Benchmark for Large Language Models: Deciding Whether to Use Tools and Which to Use. arXiv preprint.
[18] Fang, Z., Liu, H., Chen, J. (2025) Meta-Tool: Unleash Open-World Function Calling Capabilities of General-Purpose Large Language Models. Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL, Long Papers): 1481–1495.
[19] Schick, T., Dwivedi-Yu, J., Zellers, R. (2023) Large Language Models as Tool Makers. arXiv preprint.
[20] Li, X., Zhou, P., Tang, J. (2023) CREATOR: Tool Creation for Disentangling Abstract and Concrete Reasoning of Large Language Models. arXiv preprint.
[21] Yang, F., Chen, R., Xu, H. (2023) CRAFT: Customizing LLMs by Creating and Retrieving from Specialized Toolsets. arXiv preprint.
[22] Anonymous. (2024) From RAG to Multi-Agent Systems: A Survey of Modern Approaches in LLM Development. arXiv preprint.
[23] Gupta, R., Feng, Y., Wang, L. (2024) TroVE: Inducing Verifiable and Efficient Toolboxes for Solving Programmatic Tasks. arXiv preprint.
[24] Xu, M., Zhang, Y., Chen, H. (2024) ClashEval: Quantifying the tug-of-war between an LLM's internal prior and external evidence. arXiv preprint.
[25] Zhao, L., Lin, H., Wu, P. (2024) Blinded by Generated Contexts: How Language Models Merge Generated and Retrieved Contexts When Knowledge Conflicts? arXiv preprint.
[26] Sun, Z., Gao, Y., Li, P. (2025) ComplexFuncBench: Exploring Multi-Step and Constrained Function Calling under Long-Context Scenario. arXiv preprint.
[27] Wu, T., Lin, X., Huang, Y. (2025) LongFuncEval: Measuring the effectiveness of long context models for function calling. arXiv preprint.
[28] Nguyen, H., Tran, P., Li, X. (2024) Are More LLM Calls All You Need? Towards the Scaling Properties of Compound AI Systems. Advances in Neural Information Processing Systems, 37: 51173–51180.
[29] Patel, A., Singh, R., Kumar, V. (2024) The Dark Side of Function Calling: Pathways to Jailbreaking Large Language Models. arXiv preprint.
[30] He, Y., Ma, C., Zhang, L. (2025) Querying Databases with Function Calling. arXiv preprint.
[31] Chen, M., Wang, L., Huang, S. (2025) Graph-Grounded LLMs: Leveraging Graphical Function Calling to Minimize LLM Hallucinations. arXiv preprint.
[32] Ma, R., Zhou, Y., Xu, L. (2024) Large Language Models as Zero-shot Dialogue State Tracker through Function Calling. arXiv preprint.