Preprints.org

Review

Not peer-reviewed version

How to Make Tool-Using LLM Agents Efficient? A Survey

Yunuo Hu , Ziyang Cheng , Wenqi Pei , Miaomiao Li , Boyan Li , Yizhang Zhu , Yupeng Xie , Han Chen , Shizheng Hou , Yuyu Luo , Hongyang Du *

Posted Date: 6 August 2026

doi: 10.20944/preprints202608.0365.v1

Keywords: LLM agents; tool-using agents; efficient agents

Preprints.org is a free multidisciplinary platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC, OpenAlex.

Copyright: This open access article is published under a Creative Commons CC BY 4.0 license, which permit the free download, distribution, and reuse, provided that the author and preprint are cited in any reuse.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

Article

How to Make Tool-Using LLM Agents Efficient? A Survey

Yunuo Hu [1,†], Ziyang Cheng [2,†], Wenqi Pei [1,†,‡], Miaomiao Li [2], Boyan Li [3], Yizhang Zhu [3], Yupeng Xie [3], Han Chen [4], Shizheng Hou [4], Yuyu Luo [3] and Hongyang Du [1,*]

[1] HKU
[2] CUHK
[3] HKUST (GZ)
[4] NUS
* Correspondence: duhy@eee.hku.hk
† Equal Contribution.
‡ Project Lead.

Abstract

Large language models (LLMs) are increasingly deployed as agents that leverage external tools to resolve complex tasks, transcending the limitations of static parametric knowledge. However, tool-using LLM agents often rely on extended context, multi-step reasoning, and frequent tool interactions, incurring substantial computational and financial resources. As a result, **efficient tool-using LLM agents** that could reduce one or more forms of resource consumption while maintaining comparable task performance have attracted growing research attention. Nevertheless, existing efforts remain fragmented across different stages and resource dimensions. To address this gap, this survey presents a comprehensive review and a unified taxonomy of efficient tool-using LLM agents . We categorize efficiency techniques along four stages of the agent loop: (i) Efficient Tool Context, (ii) Efficient Tool Calling, (iii) Efficient Tool Reasoning, and (iv) Efficient Tool Execution. By analyzing these methods, we reveal their shared mechanisms and limitations, identify critical open challenges, and highlight promising directions for future research. This survey provides a foundational roadmap to accelerate the development of efficient tool-using LLM agents.

Keywords: LLM agents; tool-using agents; efficient agents

1. Introduction

Large language models (LLMs) are increasingly used as tool-using LLM agents that act through external tools rather than rely on parametric knowledge alone (Yao et al. 2022; Schick et al. 2023; Patil et al. 2024). By invoking search engines, databases, and domain-specific APIs, these systems can access fresh information, execute exact computation, and complete multi-step tasks that are difficult to solve by text generation alone (Nakano et al. 2021; Karpas et al. 2022; Gao et al. 2023).

However, tool use also introduces substantial resource overhead. A typical tool-using LLM agent maintains tool definitions and interaction traces in context while reasoning about tool use, which can drive up token usage, tool-call volume, latency, and monetary cost (Lu et al. 2025; Wang et al. 2025; Patil et al. 2025; Xu et al. 2025; Vigel et al. 2025; Ghoshal and Al-Bustami 2026). For example, one action space can exceed 147k tokens across more than 550 tools, while tool execution alone can account for up to 53.7% of end-to-end latency (Lei et al. 2025; Du et al. 2024; Bian et al. 2025; Huang et al. 2025). These costs have made efficiency a critical concern for tool-using LLM agents, motivating increasing research. However, existing methods remain scattered, creating a clear need for a comprehensive review.

To structure this emerging literature, we review methods that reduce one or more forms of resource consumption while maintaining comparable task performance. Then, as illustrated in Figure 1, we present a taxonomy that organizes 111 existing works around four stages of the agent loop: (i) **Efficient**

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

Tool Context concerns how tool-related context is selected and represented under limited budgets. (ii) **Efficient Tool Calling** studies whether and when tools should be invoked to avoid unnecessary or costly calls. (iii) **Efficient Tool Reasoning** focuses on improving multi-step trajectories and reusing past reasoning to reduce redundant computation. (iv) **Efficient Tool Execution** examines runtime techniques for reducing end-to-end latency once requests arrive. This provides a unified view of how efficiency can be improved throughout the life cycle of agents. We further position this survey with respect to previous reviews in Appendix A.

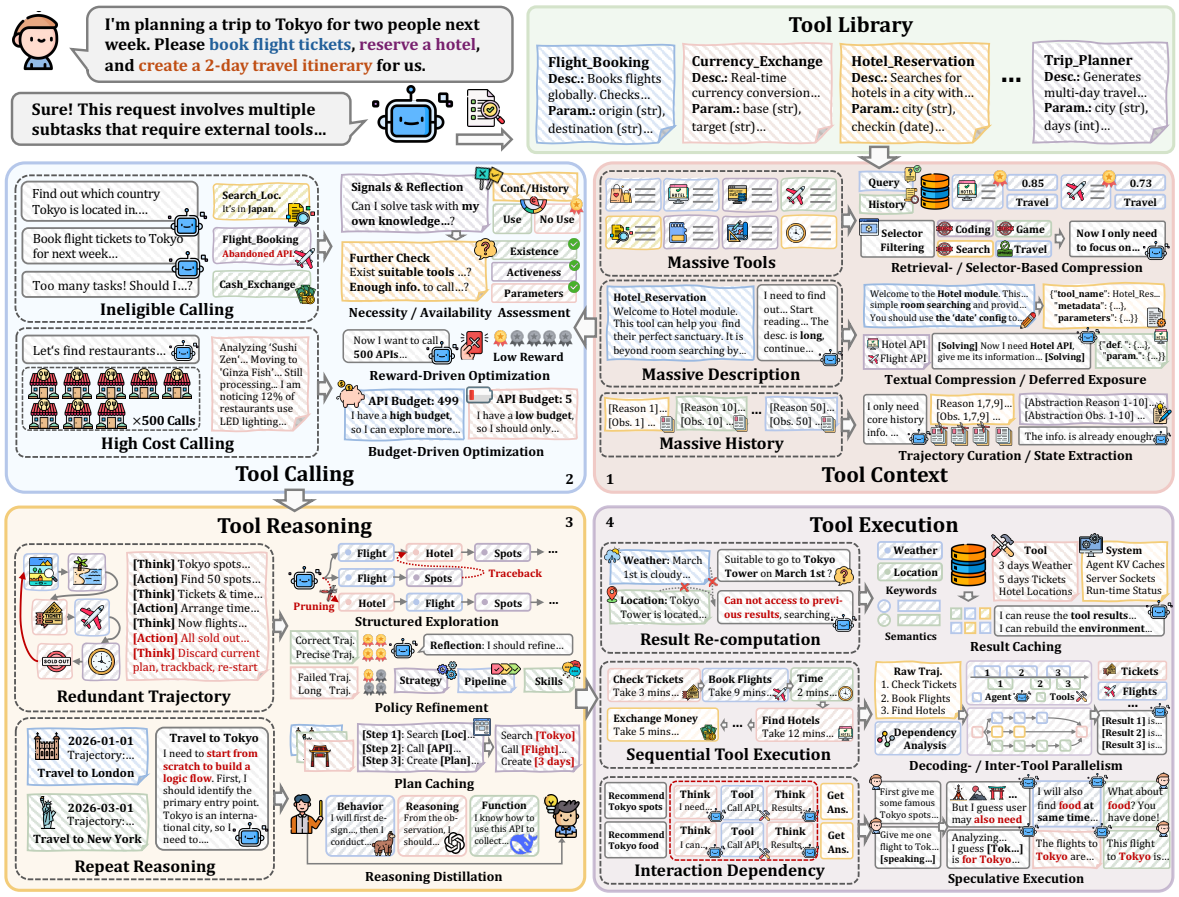


Figure 1. An overview of efficient tool-using LLM agent. The life cycle of agents is partitioned into four stages addressing specific efficiency bottlenecks, with each subsection pairing certain efficiency problems (left) with their respective technical solutions (right).

2. Preliminaries

Tool-using LLM Agent. Let $\mathcal{T} = \{t_1, \ldots, t_M\}$ denote a tool library. Each tool $t \in \mathcal{T}$ is specified by a schema s_t (e.g., name and description). Given a request q, at step k the agent conditions on a working context $c_k = (q, \mathcal{T}_k, \mathcal{H}_{k-1})$, where $\mathcal{T}_k \subseteq \mathcal{T}$ is the visible tool subset and $\mathcal{H}_{k-1}$ is the interaction history. The agent then produces a step-level output a_k which may include reasoning, a tool call, or a textual response. If a tool is invoked, the environment returns an observation o_k. Repeating this for K steps yields a trajectory $\tau = \{(c_k, a_k, o_k)\}_{k=1}^{K}$.

Efficiency Objective. Let $U_A(q)$ denote the performance of agent A on request q, and let $R = (N_{\text{tok}}, N_{\text{call}}, L, M)$ denote its resource profile, comprising token consumption, tool-call count, end-to-end latency, and monetary cost. Efficient tool-using agents seek a Pareto improvement between $U_A(q)$ and R. However, to operationalize the survey scope across studies that target different dimensions,

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

we adopt a relaxed inclusion criterion that papers whose agent A^* satisfies the following conditions relative to baseline A (with $\epsilon > 0$ as small tolerance) are included:

$$\begin{aligned} & U_{A^*}(q) \geq U_A(q) - \epsilon, \\ \exists r \in R: \quad & r_{A^*}(q) < r_A(q). \end{aligned}$$

The ideal definition of efficient tool-using LLM agents is provided in Appendix B. The collection protocol and statistics are provided in Appendix C.

Dimensions in R are coupled rather than independent. Shared factors such as the visible tool set $|\mathcal{T}_k|$, history length $|\mathcal{H}_{k-1}|$, and reasoning steps K can affect multiple resource dimensions simultaneously. In particular, reducing K lowers reasoning overhead, reduces tool invocation, and shortens execution.

3. Taxonomy of Efficiency Techniques

3.1. Efficient Tool Context

Efficient tool context focuses on minimizing $c_k = (q, \mathcal{T}_k, \mathcal{H}_{k-1})$. Methods in this section primarily target token consumption N_{tok}, mainly by optimizing the visible tool subset $\mathcal{T}_k$, the schema collection $\{s_t\}_{t \in \mathcal{T}_k}$, and the interaction history $\mathcal{H}_{k-1}$.

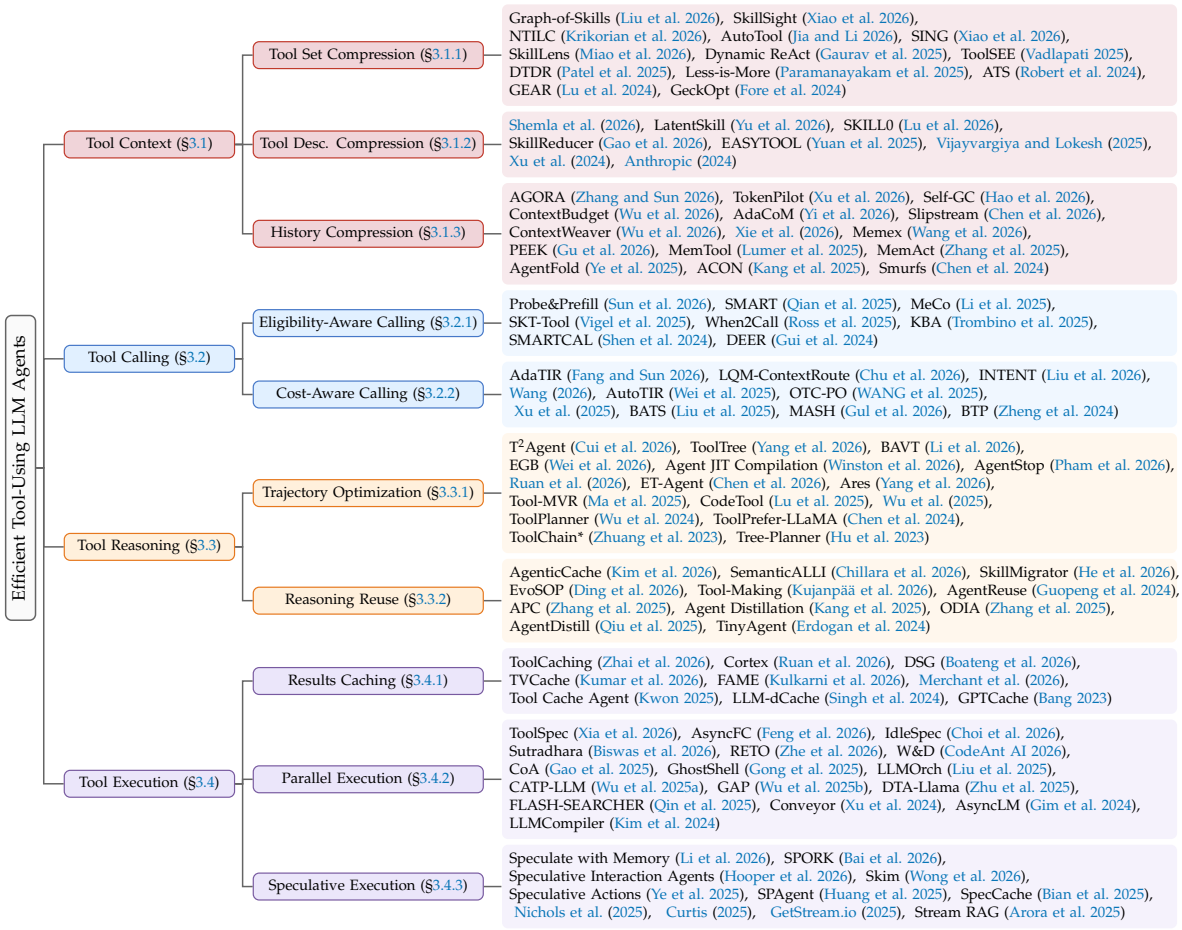


Figure 2. Taxonomy tree of efficient tool-using LLM agents. Table A1 offers a compact cross-method comparison.

3.1.1. Tool Set Compression

Tool set compression aims to optimize the visible subset $\mathcal{T}_k$, as a larger tool library inflates prompt length, causing extra computation burden. By filtering for task-relevant tools, agents can operate within a cleaner context, leading to more efficient and accurate tool selection.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

Retrieval-Based Compression. These methods treat the tool library as an external database, retrieving with queries to build a tool subset. Dynamic ReAct (Gaurav et al. 2025) searches and loads only a query-relevant subset from a large registry instead of binding all tools upfront. ToolSEE (Vadlapati 2025) follows the same principle with a lightweight retrieval layer that returns a compact top-k candidate set and expands only when needed. DTDR (Patel et al. 2025) and Graph-of-Skills (Liu et al. 2026) incorporate structural dependencies into retrieval. SkillSight (Xiao et al. 2026) accelerates retrieval by downweighting generic content across descriptions.
Selector-Based Compression. These methods use lightweight models to directly predict a relevant tool subset. ATS (Robert et al. 2024) uses a small tool selector to predict likely tools and a fallback module to recover missed tools. Similarly, GEAR (Lu et al. 2024) and NTILC (Krikorian et al. 2026) use lightweight models to score candidate tools based on semantic and functional compatibility. AutoTool (Jia and Li 2026) exploits historical tool-using inertia to construct a graph-based selector that predicts the likely next tools from prior trajectories.
Hierarchical Compression. These methods compress the tool library in a coarse-to-fine manner. GeckOpt (Fore et al. 2024) and SING (Xiao et al. 2026) use intent-based selection to restrict the tool space, after which standard tool use proceeds within the reduced subset. Less-is-More (Paramanayakam et al. 2025) and SkillLens (Miao et al. 2026) retain a smaller task-relevant subset through multi-level search and selection.

3.1.2. Tool Description Compression

Tool description compression focuses on optimizing the schema representation s_t of each visible tool t in $\mathcal{T}_k$, preserving context capacity for reasoning and state tracking.
Textual Compression. These methods compress the textual form of each tool schema, while preserving the critical invocation components. EASYTOOL (Yuan et al. 2025) converts lengthy and heterogeneous schemas into concise and unified instructions. More directly, Xu et al. (2024) selectively compress schema blocks while preserving critical fields for execution. Beyond, Shemla et al. (2026), LatentSkill (Yu et al. 2026), and SKILL0 (Lu et al. 2026) internalize textual tool knowledge into model parameters.
Deferred Schema Exposure. These methods optimize the representation of the tool schema by deferring when its full contents are presented. Vijayvargiya and Lokesh (2025) and SkillReducer (Gao et al. 2026) initially provide compact descriptions and expose detailed content only when needed. Anthropic (2024) similarly shifts part of the tool interaction to a code-execution environment, reducing the repeated injection of schema and history.

3.1.3. History Compression

Accumulating long interaction histories can also saturate the finite context budget of c_k. History compression addresses this by replacing the raw prefix with a more compact $\mathcal{H}_{k-1}$, retaining the most useful information while keeping the working context manageable.
Trajectory Curation. These methods improve context efficiency by determining which parts of the trajectory should be preserved, discarded, or updated in the memory. Smurfs (Chen et al. 2024) and AGORA (Zhang and Sun 2026) filter the history in each step rather than passing the full history. MemTool (Lumer et al. 2025), TokenPilot (Xu et al. 2026), and Self-GC (Hao et al. 2026) prune the working context as stored information becomes obsolete. MemAct (Zhang et al. 2025), ContextBudget (Wu et al. 2026), and AdaCoM (Yi et al. 2026) go further by incorporating history management into the agent policy.
State Extraction. These methods reduce context overhead by rewriting interaction information into denser state representations. AgentFold (Ye et al. 2025), Slipstream (Chen et al. 2026), and ContextWeaver (Wu et al. 2026) condense sequential interactions into textual summaries. Beyond such summarization, Xie et al. (2026), ACON (Kang et al. 2025), Memex (Wang et al. 2026), and PEEK (Gu et al. 2026) encode growing context as structured working states.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

✦ **Rethink.** Context compression is fundamentally an information-density optimization problem under limited capacity. However, most existing methods are training-free and rely on generic relevance signals. As a result, they may miss latent task–context associations and prematurely discard useful components. Such omissions may later require re-clarification or recovery, which in turn offsets the effective savings in N_{tok}.

3.2. Efficient Tool Calling

Context-optimized tools can still incur substantial overhead when invoked excessively or indiscriminately. Consequently, efficient tool calling primarily targets the tool call count N_{call} by optimizing a_k, especially the decision whether a tool should be invoked at step k.

3.2.1. Eligibility-Aware Calling

Eligibility-aware calling determines whether a tool call is warranted under the request q and whether it is feasible under the available context c_k.

Necessity Assessment. These methods assess whether the request can be resolved from the agent's own knowledge rather than requiring an external tool call. SMARTCAL (Shen et al. 2024) calibrates overconfident self-judgments that lead to tool overuse. SMART (Qian et al. 2025) and Probe&Prefill (Sun et al. 2026) directly predict tool necessity through supervised models. Other methods infer the necessity from model competence: MeCo (Li et al. 2025) uses internal representations, while SKT-Tool (Vigel et al. 2025) relies on performance in related questions.

Availability Assessment. These methods consider whether a tool call is feasible, since a suitable tool may be unavailable or its arguments may not be groundable from the context. DEER (Gui et al. 2024) turns availability assessment into a multi-branch supervised problem, while When2Call (Ross et al. 2025) further refines this formulation into a choice among tool calling, follow-up questioning, and inability to answer, and also uses it for preference-based training. KBA (Trombino et al. 2025) focuses on multi-agent systems, where feasibility also depends on routing the request to the agent with the most relevant private knowledge base.

3.2.2. Cost-Aware Calling

Cost-aware tool calling views each tool call as a trade-off between task utility $U_A(q)$ and resource cost in R. Specifically, these methods regulate tool use with explicit budgets or cost signals over one or more dimensions of R, rather than treating all useful calls equally.

Reward-Driven Optimization. These methods treat tool usage as a reward optimization problem, discouraging low-return, high-cost tool calls. AutoTIR (Wei et al. 2025) first characterizes this objective by measuring the trade-off between answer quality and tool-call frequency. At the policy level, OTC-PO (WANG et al. 2025) incorporates tool-use cost into reinforcement learning rewards, while Xu et al. (2025) models the calling decision as a confidence-calibrated continuum. Finally, AdaTIR (Fang and Sun 2026) and LQM-ContextRoute (Chu et al. 2026) adapt the cost–quality trade-off to each decision context.

Budget-Driven Optimization. These methods treat tool use as a constrained decision problem under explicit budgets. BTP (Zheng et al. 2024) and the Stackelberg framework (Wang 2026) cast budget allocation as explicit optimization problems. At runtime, BATS (Liu et al. 2025) makes remaining resources explicit, allowing the agent to adapt its search strategy. MASH (Gul et al. 2026) then incorporates this signal into a pay-per-search objective, encouraging fewer searches and calibrated abstention. INTENT (Liu et al. 2026) makes budget management forward-looking through intention-level planning, anticipating future spending and feasibility before costly actions.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

✦ **Rethink.** Efficient tool calling essentially performs boundary calibration, aiming to improve the agent's awareness of its operating state and environment. However, in practice such calibration may also expose the agent to the risk of becoming conservative, leading it to underuse tools even when they are warranted. Moreover, existing methods are mainly conducted in relatively static conditions, whereas real-world deployments often involve fluctuating budgets, rate limits, or server loads, which can make these strategies brittle.

3.3. Efficient Tool Reasoning

Efficient tool reasoning concerns how the agent constructs the trajectory $\tau = \{(c_k, a_k, o_k)\}_{k=1}^{K}$ efficiently. Accordingly, methods in this section mainly aim to reduce the overall trajectory length K, and the reasoning content and generation cost involved in producing each a_k.

3.3.1. Trajectory Optimization

Trajectory optimization aims to improve the quality of the constructed trajectory τ while reducing both the overall number of steps K and the reasoning cost associated with each action a_k.

Structured Exploration. These methods impose explicit structure on trajectory generation, directing inference-time compute toward promising alternatives. Early work uses heuristic search or explicit plan representations: ToolChain* (Zhuang et al. 2023) applies A* search, while Tree-Planner (Hu et al. 2023) and ToolPlanner (Wu et al. 2024) organize actions into trees. More recent methods allocate search adaptively. T^2Agent (Cui et al. 2026) and ToolTree (Yang et al. 2026) use Monte Carlo tree search to guide tool-use trajectories. BAVT (Li et al. 2026) and Entropy-Guided Branching (Wei et al. 2026) further focus limited search on high-value or uncertain decisions. Agent JIT Compilation (Winston et al. 2026) selects low-cost plans and optimizes their parallel schedules through Monte Carlo cost estimation. AgentStop (Pham et al. 2026) and Ruan et al. (2026) terminate the predicted failure trajectories.

Policy Refinement. These methods improve trajectory-generation behavior through supervision, feedback, or learned control. ToolPrefer-LLaMA (Chen et al. 2024), Tool-MVR (Ma et al. 2025), and ET-Agent (Chen et al. 2026) refine tool-use policies from execution trajectories and feedback. At the step level, CodeTool (Lu et al. 2025) and Ares (Yang et al. 2026) provide learned control over tool-use reasoning. Wu et al. (2025) jointly optimize agent prompts and tool descriptions.

3.3.2. Reasoning Reuse

Beyond optimizing one single trajectory τ, a similar request q may still trigger a repeated reasoning process. Reasoning reuse addresses this by transferring procedural knowledge from prior trajectories or stronger models to agents, lowering the cost of generating each a_k.

Plan Caching. These methods cache planning knowledge from past trajectories, so similar requests need not rebuild the same procedure from scratch. AgentReuse (Guopeng et al. 2024) and Agentic-Cache (Kim et al. 2026) directly reuse cached planning to avoid repeated LLM calls. Beyond direct reuse, APC (Zhang et al. 2025) caches generalized plan templates from completed runs, while SemanticALLI (Chillara et al. 2026) caches intermediate artifacts produced at stable pipeline stages. SkillMigrator (He et al. 2026) and EvoSOP (Ding et al. 2026) further abstract recurring action patterns into reusable skills or SOPs. Tool-Making (Kujanpää et al. 2026) compiles repeated SOP steps into validated and versioned tools for reliable deployment.

Reasoning Distillation. These methods distill tool-use strategies and reasoning patterns from stronger teacher models into smaller student agents. Beyond CoT distillation, Agent Distillation (Kang et al. 2025) transfers holistic task-solving behaviors to small models through thought prefixes. TinyAgent (Erdogan et al. 2024) and ODIA (Zhang et al. 2025) distill practical tool-use capabilities into small models for real edge environments. In contrast, AgentDistill (Qiu et al. 2025) is training-free. It encapsulates teacher logic in modular MCP boxes that student agents can directly reuse, avoiding costly fine-tuning.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

✦ **Rethink.** As the focus shifts from individual tool management to the logical tool coordination, efficient reasoning increasingly depends on the agent's structural awareness and integrative ability to capture higher-level trajectories and recurring patterns. When such capabilities are insufficient, errors can accumulate over long horizons, causing reasoning collapse and reducing quality of reusable traces or distillation signals.

3.4. Efficient Tool Execution

Efficient tool execution studies how tool calls are committed, executed, and returned as observations during the incremental expansion of a trajectory $\tau = \{(c_k, a_k, o_k)\}_{k=1}^{K}$ rather than changing the trajectory. It mainly targets end-to-end latency L during the runtime.

3.4.1. Results Caching

Repeated execution of similar tool interactions can create extra overhead along the trajectory. Different from plan caching that reuses trajectory-level procedures, results caching reuses concrete outputs and execution state, avoiding repeated execution and state reconstruction.

Tool-Level Caching. These methods reuse previous observations for semantically equivalent tool-calling actions, avoiding redundant tool executions. Early work like GPTCache (Bang 2023) introduces semantic retrieval-based caching for LLM systems. LLM-dCache (Singh et al. 2024) makes caching an explicit part by exposing cache operations as callable APIs. Tool Cache Agent (Kwon 2025) identifies cacheable tool calls and generates expiration and dependency-aware invalidation policies. Tool-Caching (Zhai et al. 2026) adaptively manages cache admission and eviction, while Cortex (Ruan et al. 2026) validates semantic cache hits with a lightweight LLM. For search-intensive agents, DSG (Boateng et al. 2026) externalizes search grounding through an MCP-compatible gateway with exact and semantic caching.

System-Level Caching. These methods go beyond reusing isolated (a_k, o_k) pairs, caching execution state and intermediate artifacts across the trajectory. TVCache (Kumar et al. 2026) reuses tool results across post-training rollouts. FAME (Kulkarni et al. 2026) integrates persistent memory and tool-output caching into serverless MCP workflows. Merchant et al. (2026) also combines temporal semantic caching with MCP workflow optimizations.

3.4.2. Parallel Execution

In practical execution, tool calls need not wait for a reasoning step to finish, nor are all interactions in τ linearly dependent. Parallel execution therefore exposes concurrency that would otherwise be serialized, reducing end-to-end latency L.

Decoding-Tool Parallelism. These methods allow tool execution and decoding to proceed concurrently, thereby overlapping action or plan generation with the retrieval of tool observations. CoA (Gao et al. 2025) generates placeholder-based reasoning chains, allowing tool filling to overlap with decoding across examples. Conveyor (Xu et al. 2024) brings this overlap into serving through streaming parsing and partial tool execution. ToolSpec (Xia et al. 2026) complements execution parallelism by accelerating structured tool-call decoding with schema-aware and retrieval-augmented drafts. AsyncLM (Gim et al. 2024) and AsyncFC (Feng et al. 2026) further decouple decoding from tool execution through asynchronous handling of pending tool results. IdleSpec (Choi et al. 2026) instead uses tool-waiting time to generate and aggregate candidate plans. At the system level, Sutradhara (Biswas et al. 2026) and GhostShell (Gong et al. 2025) combine streaming tool dispatch with coordinated scheduling for concurrent execution.

Inter-Tool Parallelism. These methods exploit dependencies among tool calls within a series of reasoning steps, parallelizing independent calls and serializing only dependent ones. Early work such as LLMCompiler (Kim et al. 2024) introduces DAG-based planning to identify parallelizable function calls. This direction is further refined by LLM-Tool Compiler (Singh et al. 2024), which fuses parallel function calls to reuse tokens, and LLMOrch (Liu et al. 2025), which models data

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

and control relations to coordinate execution across available processors. Shifting focus to dynamic scheduling, CATP-LLM (Wu et al. 2025a) and GAP (Wu et al. 2025b) optimize multi-branch tool plans by parallelizing independent calls while preserving sequential dependencies. DTA-Llama (Zhu et al. 2025) and FLASH-SEARCHER (Qin et al. 2025) instantiate this paradigm through DAG-structured decomposition and coordination of parallel subtasks, while RETO (Zhe et al. 2026) enhances this framework with a robustness layer for layered execution and reflective local correction.

3.4.3. Speculative Execution

Speculative execution reduces end-to-end latency L by pre-executing candidate future interactions before they are committed to the trajectory τ, retaining the results only if subsequent reasoning validates these predicted actions.

Action-Level Speculation. These methods predict future actions and execute them in parallel with ongoing reasoning. Speculative Actions (Ye et al. 2025) and Speculate with Memory (Li et al. 2026) follow a predict–execute–verify pipeline, committing actions only after validation. SPORK (Bai et al. 2026) and Speculative Interaction Agents (Hooper et al. 2026) dispatch predicted tool calls from partial context, overlapping execution with ongoing reasoning or interaction. For multi-step search, SPAgent (Huang et al. 2025) adapts verification across reasoning stages.

System-Level Speculation. System-level speculation embeds the same principle into the inference or runtime stack, prefetching likely external requests, retrieved content, or execution results before they are formally needed. At the serving layer, Nichols et al. (2025) speculates tool calls inside the inference stack to reduce execution overhead. In web interactive settings, SpecCache (Bian et al. 2025) and Skim (Wong et al. 2026) reduce latency by speculatively executing likely interaction paths before full agent processing. Stream RAG (Arora et al. 2025) extends speculation to spoken dialogue by predicting retrieval queries from partial audio streams and overlapping tool use with the conversational flow. Recent industry reports suggest that such speculative patterns are becoming increasingly practical in deployed agent systems, especially for chatbot and voice applications (Curtis 2025; GetStream.io 2025).

✦ **Rethink.** Current methods often assume that tool interactions are predictable and free of harmful side effects. In practice, however, speculative execution can be risky when future actions cannot be validated reliably, potentially causing irreversible changes, while stale cached results may also hurt correctness. Moreover, aggressive speculation and complex parallel execution can introduce enough extra system overhead to increase monetary cost and offset the intended efficiency gains.

4. Discussion

In Section 3, we have introduced each stage of the agent loop and reflected on the characteristic mechanisms along with potential risks of each method category. Building on these observations, in this section, we move to a more system-level perspective, examining the key challenges of efficient tool-using LLM agents and then presenting relevant future directions. Appendix D provides a comparative analysis across stages, methods, and implementations, while Appendix E summarizes the main efficiency drivers and bottlenecks.

4.1. Challenges

❶ **Coupled Constraints.** Current methods often optimize token usage, tool-call count, or latency in limited dimensions, overlooking the coupling across resources. In practice, apparent resource savings in a single dimension may not translate into end-to-end efficiency. For instance, speculative execution can reduce waiting time, but may introduce extra tool calls and redundant execution. Moreover, as the four stages are also tightly coupled in the agent loop, efficiency gains in an earlier stage may constrain subsequent actions and cause mid-task failures. An example is that overly explicit

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

or specialized trajectories may reduce the chance of matching reusable plans or cached results. Overall, efficiency gains at one stage can be offset, or even reversed, by overhead elsewhere.

❷ **Deficient Cross-Scenario Robustness.** Existing methods are often effective only under the specific environmental settings for which they are designed. As previously discussed, these myopic policies can lead to performance degradation across broader scenarios. For instance, retrieval-based compression methods are beneficial in large open tool spaces, while the advantage may become marginal in constrained, static tool sets, as retrieval over such a limited candidate space may introduce unnecessary latency compared with direct processing. Consequently, the robustness of these efficiency gains under changing or multiple scenarios still remains an open challenge.

❸ **Limited Traceback and Evaluation Ability.** Efficiency optimization is inherently a process of approximation and often lossy. Specifically, many efficiency-oriented methods rely on compressed representations, reduced computation, or partial information retention, which may increase the risk of error propagation across stages and even costly recomputation. However, current evaluation in tool-using agents remains a weak proxy of efficiency and provides limited traceability into how such errors are introduced, amplified, and accumulated throughout the trajectory. This limitation is further reflected in existing benchmarks, which pay limited attention to error localization, side effects, and decomposed resource usage. A more detailed discussion is provided in Appendix F.

4.2. Future Directions

❶ **Global and Adaptive Awareness.** As discussed above, efficiency gains achieved by optimizing an isolated resource or a single stage could be offset in the view of the overall agent loop. Consequently, future work should develop global awareness that explicitly accounts for both interactions among resources and dependencies across pipeline stages. Moreover, in real-world deployments, since both the interaction environment and the agents' operating conditions can fluctuate over time, the agents should also be adaptively responsive to temporal variations and long-term dynamics.

❷ **Strategic Experience Reuse.** Existing reuse methods mainly operate on factual or instance-bound artifacts, including history records, trajectories, and reasoning templates. Although these forms of reuse can reduce repeated computation, they often remain tightly coupled to particular tasks or scenarios. A more scalable direction is to transform reuse from memorizing past records into abstracting higher-level experience and strategies, such as structured reasoning flows and reflective patterns. In this way, reuse may move beyond task-specific toward model-agnostic or even scenario-agnostic efficiency knowledge.

❸ **Incremental Self-Evaluation.** In long-horizon settings, agents typically execute multi-stage trajectories, where errors introduced at early stages can propagate across subsequent steps. Therefore, incremental verification and self-evaluation should be applied at high-risk decision points rather than deferred to the end of the trajectory, enabling iterative correction and reducing restarts. This direction also highlights the need for more informative benchmarks, with stage-level traces and failure annotations.

Moreover, recent industrial deployments also point toward complementary directions. For instance, OpenClaw[1] exemplifies a system-level path toward more state-aware and holistically optimized agent infrastructure, whereas HyperAgents (Zhang et al. 2026) highlights a path toward more adaptive agent operation over time.

5. Conclusion

This survey provides a unified view of the emerging landscape of efficient tool-using LLM agents. Specifically, we categorize existing methods into four pivotal stages of the agentic pipeline: **(i) Tool Context**, **(ii) Tool Calling**, **(iii) Tool Reasoning**, and **(iv) Tool Execution**. Methods across these stages improve efficiency by reducing resource consumption throughout the pipeline. Moreover, our analysis highlights that agent efficiency is a multi-dimensional optimization problem, requiring careful trade-

[1] https://github.com/openclaw/openclaw

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

offs between task performance and resource overhead. While substantial progress has been made in individual components, several challenges remain. Future research should move beyond isolated optimizations toward holistic, system-level designs that are robust to dynamic environments and capable of adaptive, resource-aware decision making across the full agent pipeline. Evaluations should also jointly measure these trade-offs.

Limitations

This survey has several limitations. First, despite our broad literature collection process, this fast-moving literature and its diverse terminology make complete coverage difficult; relevant concurrent or non-archival work may be missed, and our review only covers work available through July 2026. Second, the proposed four-stage taxonomy is an analytical abstraction. Some methods optimize multiple stages simultaneously, and we categorize them by their primary target; thus, the boundaries should not be interpreted as mutually exclusive. Third, we synthesize reported findings rather than conduct a quantitative meta-analysis because the reviewed studies use heterogeneous models, tool environments, hardware, pricing schemes, and efficiency metrics. Their reported gains are therefore not directly comparable and should not be read as a universal ranking. Finally, our scope focuses on agent-level overhead induced by tool interaction and excludes generic LLM inference, compression, and serving optimizations, which may nevertheless interact with the techniques reviewed here in real deployments.

Appendix A. Related Work

Table A1. Summary of works on efficient tool-using LLM agents.

	Name	Venue	Optimized Resource				Learning Method	Structure	Scenario	Source
			N_{tok}	L	N_{call}	M				
Efficient Tool Context	Dynamic ReAct (Gaurav et al. 2025)	Preprint 2025	✓	-	-	-	Training-free	-	General	-
	ToolSEE (Vadlapati 2025)	Preprint 2025	✓	✓	-	✓	Training-free	-	General	Link
	DTDR (Patel et al. 2025)	NFIR @ AAAI 2026	✓	✓	-	-	Training-free/SFT	DAG	General	-
	GEAR (Lu et al. 2024)	EACL 2024	-	✓	-	✓	Training-free	-	General	Link
	AutoTool (Jia and Li 2026)	AAAI 2026	✓	✓	-	✓	Training-free	DAG	General	Link
	GeckOpt (Fore et al. 2024)	GLSVLSI 2024	✓	-	-	✓	Training-free	-	General	-
	Less-is-More (Paramanayakam et al. 2025)	DATE 2025	✓	✓	-	✓	Training-free	-	General	-
	EASYTOOL (Yuan et al. 2025)	NAACL 2025	✓	-	-	-	Training-free	-	General	Link
	Xu et al. (2024)	ACL 2024 Findings	✓	✓	-	-	Continued Pre-training	-	General	-
	Vijayvargiya and Lokesh (2025)	Preprint 2025	✓	-	-	-	SFT	-	Edge Devices	-
	Smurfs (Chen et al. 2024)	NAACL 2025	✓	-	-	-	Training-free	-	General	Link
	MemTool (Lumer et al. 2025)	ECIR 2026	✓	-	-	-	Training-free	-	General	-
	MemAct (Zhang et al. 2025)	Preprint 2025	✓	-	-	-	SFT + DCPO	-	Web QA	Link
	AgentFold (Ye et al. 2025)	ICLR 2026	✓	-	-	-	SFT	-	Web QA	Link
	ACON (Kang et al. 2025)	LLA @ ICLR 2026	✓	-	-	-	Training-free Distillation	-	User Assistance	Link
	Graph-of-Skills (Liu et al. 2026)	Preprint 2026	✓	✓	-	-	Training-free	Graph	General	Link
	SkillSight (Xiao et al. 2026)	Preprint 2026	✓	✓	-	-	Training-free	-	General	Link
	NTILC (Krikorian et al. 2026)	Preprint 2026	✓	✓	-	-	Contrastive Learning	-	General	Link
	SING (Xiao et al. 2026)	Preprint 2026	✓	-	-	-	Training-free	Graph	General	-
	SkillLens (Miao et al. 2026)	Preprint 2026	✓	-	-	-	Training-free	Hierarchical Graph	Coding & Embodied Tasks	-
	LatentSkill (Yu et al. 2026)	Preprint 2026	✓	-	-	-	Pre-training + SFT	-	Embodied Tasks & Search	Link
	Shemla et al. (2026)	Preprint 2026	✓	-	-	-	SFT	-	Asset Operations	-
	SKILL0 (Lu et al. 2026)	Preprint 2026	✓	-	-	-	GRPO	-	Embodied Tasks & Search	Link
	SkillReducer (Gao et al. 2026)	Preprint 2026	✓	-	-	✓	Training-free	Hierarchical	Coding	-
	ContextBudget (Wu et al. 2026)	Preprint 2026	✓	-	-	-	GRPO	-	Search	-
	AGORA (Zhang and Sun 2026)	Preprint 2026	✓	-	-	-	Auxiliary Training	-	Embodied Tasks & Web Shopping	Link
	TokenPilot (Xu et al. 2026)	Preprint 2026	✓	-	-	✓	Training-free	-	General	Link
	Self-GC (Hao et al. 2026)	Preprint 2026	✓	-	-	-	Training-free	-	General	-
	AdaCoM (Yi et al. 2026)	Preprint 2026	✓	-	-	-	SFT + PPO	-	Search	-
	Xie et al. (2026)	Preprint 2026	✓	-	-	-	SFT + GRPO	-	General	-
	Slipstream (Chen et al. 2026)	Preprint 2026	✓	✓	-	-	Training-free	-	Search & Coding	Link
	Memex (Wang et al. 2026)	Preprint 2026	✓	-	-	-	SFT + GRPO	-	Embodied Tasks	Link
	ContextWeaver (Wu et al. 2026)	Preprint 2026	✓	-	-	-	Training-free	DAG	Coding	-
	PEEK (Gu et al. 2026)	Preprint 2026	✓	-	-	✓	Training-free	-	General	Link

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

Table A1. *Cont.*

	Name	Venue	Optimized Resource				Learning Method	Structure	Scenario	Source
			N_{tok}	L	N_{call}	M				
Efficient Tool Calling	SMARTCAL (Shen et al. 2024)	EMNLP 2024	-	✓	✓	✓	Training-free	-	General	Link
	SMART (Qian et al. 2025)	ACL 2025 Findings	✓	✓	✓	✓	SFT	-	General	Link
	MeCo (Li et al. 2025)	ACL 2025	-	✓	✓	✓	SFT	-	General	-
	SKT-Tool (Vigel et al. 2025)	Insights @ ACL 2025	-	-	✓	✓	Training-free	-	General	-
	DEER (Gui et al. 2024)	Preprint 2024	-	-	✓	-	SFT	Decision Tree	General	Link
	When2Call (Ross et al. 2025)	NAACL 2025	-	-	✓	-	PPO	-	General	Link
	KBA (Trombino et al. 2025)	Preprint 2025	-	-	-	✓	Training-free	DAG	Multi-Agent System	-
	AutoTIR (Wei et al. 2025)	Preprint 2025	-	-	✓	-	GRPO	-	General	Link
	OTC-PO (WANG et al. 2025)	LAW @ NeurIPS 2025	-	✓	✓	✓	PPO + GRPO	Linear	Search	-
	Xu et al. (2025)	EMNLP 2025	-	✓	✓	✓	SFT	-	General	-
	AdaTIR (Fang and Sun 2026)	Preprint 2026	-	-	✓	✓	SFT + GRPO	-	Math & Code	-
	BTP (Zheng et al. 2024)	ACL 2024 Findings	-	✓	✓	✓	Auxiliary Training	-	General	Link
	BATS (Liu et al. 2025)	Preprint 2025	✓	-	✓	✓	Training-free	Tree	Search	-
	INTENT (Liu et al. 2026)	Preprint 2026	-	✓	✓	✓	SFT	-	General	-
	MASH (Gul et al. 2026)	Preprint 2025	-	-	✓	✓	SFT + GRPO	Hierarchical/MCTS	QA	Link
	Probe&Prefill (Sun et al. 2026)	Preprint 2026	-	-	✓	-	Auxiliary Training	-	General	Link
	LQM-ContextRoute (Chu et al. 2026)	Preprint 2026	-	✓	-	-	Contextual Bandit	-	General	-
	Wang (2026)	Preprint 2026	✓	-	✓	-	GAIL + PPO	-	General	-
Efficient Tool Reasoning	ToolChain*(Zhuang et al. 2023)	ICLR 2024	-	✓	-	-	Training-free	Action Tree	General	-
	Tree-Planner (Hu et al. 2023)	ICLR 2024	✓	-	-	✓	Training-free	Action Tree	Embodied Tasks	Link
	ToolPlanner (Wu et al. 2024)	EMNLP 2024	-	-	✓	-	SFT + RRHF	Hierarchical Tree	User Assistants	Link
	T^2Agent (Cui et al. 2026)	AAAI 2026	-	-	-	✓	Training-free	MCTS Tree	Misinformation	Link
	ToolTree (Yang et al. 2026)	ICLR 2026	✓	✓	-	-	Training-free	MCTS Tree	General	Link
	ToolPrefer-LLaMA (Chen et al. 2024)	NeurIPS 2024	-	-	✓	-	SFT + DPO	DFS Tree	General	-
	CodeTool (Lu et al. 2025)	ACL 2025	✓	-	-	-	Process Supervision	MCTS/DFS Tree	Coding	Link
	Tool-MVR (Ma et al. 2025)	KDD 2025	-	✓	✓	✓	Reflection Learning	-	General	Link
	ET-Agent (Chen et al. 2026)	ACL 2026	-	-	✓	-	RFT + ARPO	-	General	Link
	(Wu et al. 2025)	ACL 2025	-	✓	✓	-	Prompt Optimization	-	General	Link
	AgentReuse (Guopeng et al. 2024)	JCRD 2024	✓	✓	✓	✓	Training-free	DAG	User Assistants	-
	APC (Zhang et al. 2025)	NeurIPS 2025	✓	✓	-	✓	Training-free	-	General	-
	SemanticALLI (Chillara et al. 2026)	Preprint 2026	✓	✓	-	✓	Training-free	Hierarchical Retrieval	Marketing	-
	Agent Distillation (Kang et al. 2025)	NeurIPS 2025	✓	-	✓	-	Distillation	-	Math & QA	Link
	TinyAgent (Erdogan et al. 2024)	EMNLP 2024	-	✓	✓	✓	SFT	DAG	Edge Devices	Link
	ODIA (Zhang et al. 2025)	Preprint 2025	✓	✓	✓	-	Online Distillation	-	Music	-
	AgentDistill (Qiu et al. 2025)	Preprint 2025	-	✓	✓	✓	Training-free Distillation	-	Biomedical & Math	Link
	Agent JIT Compilation (Winston et al. 2026)	ICML 2026	-	✓	✓	-	Training-free	CFG	Web Automation	-
	Ruan et al. (2026)	Preprint 2026	✓	-	-	-	Auxiliary Training	-	Embodied Tasks & Web Shopping	-
	BAVT (Li et al. 2026)	Preprint 2026	✓	-	✓	✓	Training-free	Tree	Multi-hop QA	-
	Ares (Yang et al. 2026)	Preprint 2026	✓	-	-	-	SFT + GRPO	-	General	Link
	AgentStop (Pham et al. 2026)	CAIS 2026	✓	✓	-	-	Auxiliary Training	-	Edge Devices	Link
	EGB (Wei et al. 2026)	Preprint 2026	✓	-	✓	-	Training-free	Tree	E-commerce	-
	Tool-Making (Kujanpää et al. 2026)	Preprint 2026	✓	✓	✓	-	Distillation + SFT	Decision Tree	Industrial Operations	-
	EvoSOP (Ding et al. 2026)	Preprint 2026	-	-	✓	-	Training-free	Hierarchical	General	-
	AgenticCache (Kim et al. 2026)	MLSys 2026	✓	✓	✓	✓	Training-free	-	Embodied Tasks	Link
	SkillMigrator (He et al. 2026)	Preprint 2026	-	-	✓	-	Training-free	Accessibility Tree	Web Navigation	-
Efficient Tool Execution	GPTCache (Bang 2023)	NLP-OSS 2023	✓	✓	✓	✓	Training-free	-	General	Link
	LLM-dCache (Singh et al. 2024)	ICECS 2024	-	✓	-	-	Training-free	-	Geospatial Sensing	-
	ToolCaching (Zhai et al. 2026)	Preprint 2026	-	✓	-	-	Training-free	-	General	-
	Tool Cache Agent (Kwon 2025)	Preprint 2025	-	✓	-	-	Training-free	-	General	-
	Cortex (Ruan et al. 2026)	NSDI 2026	-	✓	✓	✓	Training-free	-	Search & Coding	-
	TVCache (Kumar et al. 2026)	Preprint 2026	-	✓	-	-	Training-free	Tree	Agent Post-Training	Link
	FAME (Kulkarni et al. 2026)	Preprint 2026	✓	✓	-	✓	Training-free	-	Paper Summarization	-
	CoA (Gao et al. 2025)	COLING 2025	-	✓	-	-	SFT	-	Math & Wiki QA	Link
	Conveyor (Xu et al. 2024)	Preprint 2024	-	✓	-	-	Training-free	-	LLM Serving	Link
	AsyncLM (Gim et al. 2024)	Preprint 2024	-	✓	-	-	SFT	-	General	-
	Sutradhara (Biswas et al. 2026)	ASPLOS 2026	-	✓	-	-	Training-free	-	General	-
	GhostShell (Gong et al. 2025)	Preprint 2025	-	✓	-	-	Training-free	-	Embodied Programming	Link
	LLMCompiler (Kim et al. 2024)	ICML 2024	-	✓	-	✓	Training-free	DAG	General	Link
	LLMOrch (Liu et al. 2025)	IEEE TSE 2026	-	✓	-	-	Training-free	-	General	-
	CATP-LLM (Wu et al. 2025a)	ICCV 2025	-	✓	-	✓	Offline RL	Graph	General	Link
	GAP (Wu et al. 2025b)	NORA @ NeurIPS 2025	✓	✓	-	-	SFT + RL	Graph	QA	Link
	DTA-Llama (Zhu et al. 2025)	ACL 2025	✓	✓	-	-	SFT	DAG	General	Link
	FLASH-SEARCHER (Qin et al. 2025)	ICLR 2026	✓	✓	-	✓	SFT	DAG	Search	Link
	RETO (Zhe et al. 2026)	Preprint 2026	✓	-	-	-	SFT	-	General	-
	Speculative Actions (Ye et al. 2025)	ICLR 2026	-	✓	-	-	Training-free	-	General	Link
	SPAgent (Huang et al. 2025)	Preprint 2025	-	✓	-	-	Training-free	-	Search	-
	SpecCache (Bian et al. 2025)	SEA @ NeurIPS 2025	-	✓	-	-	Training-free	-	Web QA	Link
	Stream RAG (Arora et al. 2025)	Preprint 2025	-	✓	-	-	SFT	-	Voice Assistant	-
	(Merchant et al. 2026)	Preprint 2026	-	✓	✓	-	Training-free	DAG	Asset Operations	-
	DSG (Boateng et al. 2026)	Preprint 2026	-	✓	✓	✓	Training-free	MCP	Search & E-commerce	-
	AsyncFC (Feng et al. 2026)	Preprint 2026	-	✓	-	-	Training-free	DAG	General	-
	Speculative Interaction Agents (Hooper et al. 2026)	Preprint 2026	-	✓	-	-	Training-free / SFT	DAG	Voice Assistants	-
	ToolSpec (Xia et al. 2026)	Preprint 2026	-	✓	-	-	Training-free	-	General	Link
	SPORK (Bai et al. 2026)	Preprint 2026	-	✓	-	-	Training-free	-	General	Link
	IdleSpec (Choi et al. 2026)	Preprint 2026	-	✓	-	-	Training-free	-	General	-
	Speculate with Memory (Li et al. 2026)	Preprint 2026	-	✓	-	-	Training-free	-	General	-
	Skim (Wong et al. 2026)	Preprint 2026	-	✓	✓	✓	Training-free	-	Web Navigation	-

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

Table A2. Comparison of related work across efficiency, tool-using, pipeline-stage, and cross-metric analysis. We mark a work as "Yes" when the dimension is a primary organizing principle, "Partial" when it is discussed but not structured, and "No" otherwise.

Work	Main scope	Efficiency	Tool-using	Pipeline-stage	Cross-metric
Wang et al. (2024)	General agents	No	Partial	No	No
Guo et al. (2024)	Multi-agents	No	No	No	No
Luo et al. (2025)	General agents	No	Partial	No	No
Qu et al. (2025)	Tool learning	Partial	Yes	Yes	Partial
Yehudai et al. (2025)	Evaluation of agents	Partial	Partial	No	Partial
Du et al. (2026)	Optimization of agents	Partial	Partial	No	No
Xu et al. (2025)	Efficient tool calling	Yes	Yes	Partial	Yes
Yang et al. (2026)	Efficient agents	Yes	Partial	Partial	Partial
Ours	**Efficient tool-using agents**	**Yes**	**Yes**	**Yes**	**Yes**

Existing surveys have studied LLM-based systems from several complementary perspectives, but they generally prioritize functional breadth over resource efficiency. Early overviews categorize general agents and multi-agent systems through agent construction, coordination, and communication patterns (Wang et al. 2024; Guo et al. 2024), while subsequent surveys review agent methodologies (Luo et al. 2025), tool learning paradigms (Qu et al. 2025), and evaluation protocols (Yehudai et al. 2025). Although these studies discusstask effectiveness and agent capabilities, they pay limited attention to the efficiency implications of tool-augmented workflows. Accordingly, key overhead sources such as context expansion, redundant tool invocations, and execution latency remain underexplored, especially in terms of their effects on tool-use efficiency and trade-offs.

More recent work has moved toward efficiency-aware perspectives. For example, Xu et al. (2025) represents a significant advancement toward efficiency-aware tool use by explicitly reducing redundant invocations, although its primary focus remains tool-calling decisions rather than pipeline-stage analysis of the broader pipeline. Yang et al. (2026) surveys efficient agents from a broader optimization perspective, covering memory, tool learning, and planning. While highly relevant, its scope is not centered specifically on tool-using agents, and therefore it offers less fine-grained analysis of the stage specific costs and trade-offs that arise in tool-augmented workflows. Consequently, there remains a critical gap for a comprehensive study that analyzes efficiency across pipeline stages in tool-using agents.

As summarized in Table A2, previous surveys typically cover only a subset of the four dimensions considered in this review: efficiency, tool using, pipeline-stage analysis, and cross-metric trade-offs. In contrast, our survey focuses specifically on efficient tool-using agents from a stage-oriented perspective. Rather than treating tool use as a single monolithic capability, we decompose the tool use pipeline into multiple stages and review the optimization objectives, representative methods, and evaluation metrics associated with each stage. This perspective helps clarify how different forms of cost and inefficiency arise across the pipeline and provides a more operational view for analyzing, designing, and evaluating efficient tool-use systems.

Appendix B. Ideal Definition of Efficient Tool-Using LLM Agents

Using the notation in Section 2, an agent A^* is ideally efficient relative to baseline A when it preserves task performance while improving resource use without worsening any resource dimension. This corresponds to a Pareto improvement:

$$\begin{aligned} & U_{A^*}(q) \geq U_A(q), \\ \forall r \in R: \quad & r_{A^*}(q) \leq r_A(q), \\ \exists r \in R: \quad & r_{A^*}(q) < r_A(q). \end{aligned}$$

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

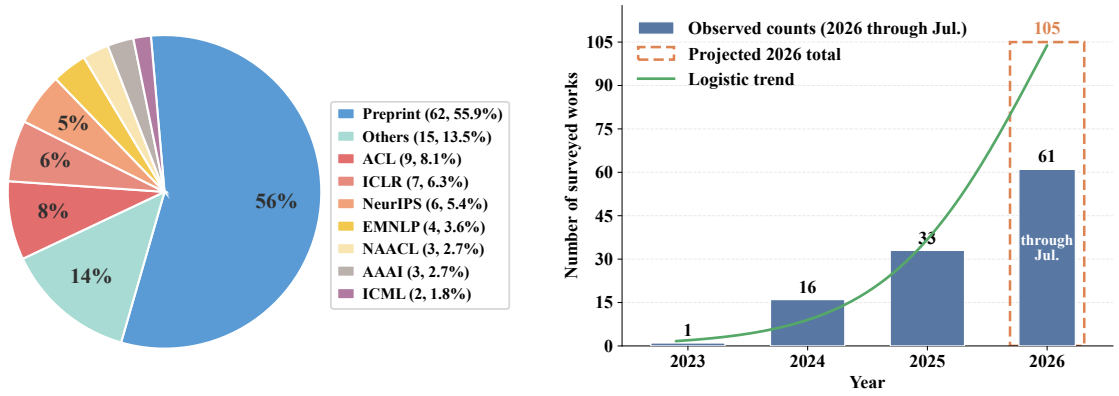
Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

This ideal definition is stricter than our literature inclusion criterion in Section 2. Because studies often report only a subset of R, the criterion allows a small performance tolerance and requires improvement in at least one reported resource dimension.

Appendix C. Literature Overview

Appendix C.1. Literature Collection Protocol

To ensure a structured and transparent literature collection process, we establish a formal review protocol, covering work available through July 2026:



(**a**) Venue distribution. (**b**) Temporal distribution.

Figure A1. Literature overview of surveyed works (as of July 2026). Left: venue distribution, where venues with fewer than two papers are grouped into Others. Right: temporal distribution, where the year is the venue year or otherwise the preprint year.

Search Sources.

We specify the searched sources, including Google Scholar, Semantic Scholar, arXiv, ACL Anthology, OpenReview, ACM/IEEE libraries, and major NLP/ML conference websites.

Queries.

We list representative keywords, including *"tool-using LLM agent"*, *"tool learning"*, *"function calling"*, *"tool invocation"*, *"tool retrieval"*, *"tool context compression"*, *"tool calling efficiency"*, *"trajectory optimization"*, *"agent caching"*, and *"parallel/speculative tool execution"*.

Eligibility.

We retain works that satisfy the survey scope in Section 2: the method must operate in a tool-using setting and explicitly reduce at least one resource dimension while maintaining task performance. We exclude tool-using papers that focus only on capability without an efficiency objective or analysis. We also exclude works whose efficiency gains are not tied to tool use, including general LLM inference acceleration, model compression, serving optimization, and generic system scheduling.

Screening.

We clarify that papers are screened first by title and abstract, and then by introduction text. Duplicate versions of the same work, such as arXiv and conference versions, are merged, with the latest or archival version retained.

Quality Appraisal.

To account for source quality, we prioritize peer-reviewed conference and journal papers, while considering relevant preprints and workshop papers for taxonomy construction, with venue reported in Table A1. Industrial blogs are treated only as supporting evidence.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

Appendix C.2. Publication Statistics

In total, our survey includes 111 works. This count excludes industrial blogs, which are used only as supporting evidence. For a work first released as a preprint and later published at a venue, both the venue and temporal statistics use the venue publication year rather than the initial preprint year. The left panel of Figure A1 shows their venue distribution. Two patterns are worth noting. First, Preprint alone accounts for 55.9%, which suggests that the area is moving faster than the archival publication cycle and that many methods are disseminated through arXiv or other non-archival channels. Second, the remaining works are concentrated in mainstream NLP or ML venues, especially ACL (9), ICLR (7), and NeurIPS (6), indicating that efficient tool-using LLM agents are receiving attention from both communities.

The right panel of Figure A1 presents the temporal distribution of the surveyed works. We observe a clear acceleration over time: the number of works rises from 1 in 2023 to 16 in 2024 and 33 in 2025. The count reaches 61 by July 2026; extrapolating the observed rate yields a projected total of 105, further indicating rapid growth.

Appendix D. Comparative Analysis

Appendix D.1. Stage-Level

We compare the four stages by their primary resource targets, suitable scenarios, and key risks, as summarized in Table A3. The comparison shows that each stage addresses a distinct bottleneck, although its gains depend on the operating setting and associated risks.

Table A3. Stage-level comparative analysis of efficient tool-using agents.

Stage	Mainly optimized resource	Suitable scenarios	Key risks
Context	Token consumption (N_{tok})	enterprise API hubs, MCP servers, long-history web QA, and on-device assistants.	Compression may remove latent tool dependencies, execution-critical schema constraints, or historical evidence needed for later grounding and recovery.
Calling	Tool-call count (N_{call}) and monetary cost (M)	search-based QA, coding/math agents with optional execution, pay-per-search systems, and multi-agent routing.	Conservative policies may under-call, abstain unnecessarily, or fail when tool availability, budgets, or parameter grounding change.
Reasoning	Reasoning tokens (N_{tok}), tool calls (N_{call}), and trajectory length (K)	travel planning, coding/debugging, embodied planning, misinformation detection, and recurring workflows.	Shortened or reused trajectories may prune corrective branches, accumulate reasoning errors, or mismatch subtle task changes.
Execution	End-to-end latency (L)	web search agents, voice assistants, booking workflows, and repeated retrieval of tool outputs.	Cached, parallel, or speculative execution may introduce stale observations, dependency violations, redundant work, or costly rollback.

Appendix D.2. Method-Level

We further compare the settings in which the principal method families within each stage are most applicable.

- **Efficient Tool Context.** Tool Set Compression fits large or noisy tool libraries, where the priority is reducing visible tools while keeping relevant ones accessible. Tool Description Compression fits manageable tool sets with verbose or repeated schemas, where token savings depend on preserving invocation-critical fields. History Compression fits long-horizon or multi-turn tasks, where compact states must retain enough past observations and reasoning traces for later grounding.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

- **Efficient Tool Calling.** Eligibility-Aware Calling fits cases where the main decision is whether a tool is needed or feasible, such as answerable requests, unavailable tools, or under-specified parameters. Cost-Aware Calling fits settings where useful tools are expensive, especially under budgets, rate limits, or heterogeneous tool costs, and calls should be allocated to higher-value actions.
- **Efficient Tool Reasoning.** Trajectory Optimization fits complex, instance-specific tasks requiring search, decomposition, pruning, or backtracking, where flexibility is important for solving novel problems. Reasoning Reuse fits recurrent or structurally similar tasks, where past plans, templates, or distilled strategies can amortize reasoning cost across repeated patterns.
- **Efficient Tool Execution.** Results Caching fits stable and recurring tool outputs, where reuse can avoid repeated execution. Parallel Execution fits independent or graph-structured tool calls in decomposable tasks, where overlap can shorten runtime. Speculative Execution fits settings where future calls can be predicted with high confidence, allowing execution to start before formal commitment.

Appendix D.3. Implementation-Level

We compare competing implementations within each method family and summarize the conditions favoring each alternative in Table A4. The results indicate that no implementation dominates universally, and the appropriate choice depends on the task and deployment setting.

Table A4. Implementation-level comparative analysis of efficient tool-use methods.

Method	Implementation	Favor the former when...	Favor the latter when...
Tool Set Compression	Retrieval-Based vs. Selector-Based	The tool library is large, dynamic, or long-tailed, and query/history signals generalize well.	The tool space is stable, usage patterns are predictable, and retrieval overhead is undesirable.
Tool Set Compression	Hierarchical vs. Retrieval-/Selector-Based	Tools have clear domain, intent, or API-library hierarchy for reliable coarse-to-fine filtering.	Tool relations are flat or cross-domain, so early coarse pruning may discard useful tools.
Tool Description Compression	Textual vs. Deferred Schema Exposure	Planning requires visible names, parameters, and constraints throughout reasoning.	Tool choice can be made from short descriptions, with full schemas loaded only when needed.
History Compression	Trajectory Curation vs. State Extraction	Later steps need exact observations, actions, or provenance for grounding and rollback.	The horizon is long and completed subtasks can be safely abstracted into compact state.
Eligibility-Aware Calling	Necessity Assessment vs. Availability Assessment	The main issue is tool overuse on queries solvable from internal knowledge or context.	The main issue is infeasible calls caused by missing tools, inactive APIs, or ungroundable arguments.
Cost-Aware Calling	Reward-Driven Optimization vs. Budget-Driven Optimization	The goal is a soft quality-cost preference without hard inference-time limits.	Deployment has strict API budgets, quotas, rate limits, or pay-per-use constraints.
Trajectory Optimization	Structured Exploration vs. Policy Refinement	Tasks are novel, long-horizon, or require backtracking and branch comparison.	Tasks are stable and efficient behavior can be learned from high-quality traces.
Reasoning Reuse	Plan Caching vs. Reasoning Distillation	Similar workflows recur and cached plans can be adapted with low mismatch risk.	Small or edge agents need internalized strategies without relying on external caches.
Results Caching	Tool-Level Caching vs. System-Level Caching	Calls are isolated, stateless, semantically equivalent, and fresh within a validity window.	Workflows depend on persistent state, intermediate artifacts, or cross-tool orchestration.
Parallel Execution	Decoding-Tool Parallelism vs. Inter-Tool Parallelism	The bottleneck is waiting between generation and tool response.	The task has independent tool calls that can safely run in parallel.
Speculative Execution	Action-Level Speculation vs. System-Level Speculation	The agent can predict and validate near-future actions before commitment.	The infrastructure can prefetch likely APIs or results, and wasted work is acceptable.

Appendix E. Drivers and Bottlenecks

Appendix E.1. Main Efficiency Drivers

Efficiency in tool-using agents is fundamentally driven by task-conditioned resource allocation. Such resource allocation depends on the agent's understanding of four key aspects.

Task Objective.

The agent needs to distinguish hard requirements from flexible preferences. Without a clear task understanding, it may over-search, overuse tools, or follow an incorrect plan.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

Environment State.

The agent needs to track the environment and identify when external information requires tool-based updates. For dynamic information such as prices, availability, or real-time web content, skipping tool use may lead to outdated or incorrect decisions.

Tool Affordances and States.

The agent needs to understand each tool's capabilities, required arguments, invocation cost, and relevant side effects.

Capability Boundary.

Efficiency requires calibrated awareness of the agent's own capability boundary. An overconfident agent may skip necessary tools and produce unsupported answers, while an overly conservative agent may invoke tools unnecessarily or prolong the trajectory with redundant checks.

Appendix E.2. Main Efficiency Bottlenecks

Four bottlenecks make task-conditioned resource allocation difficult in practice.

Local–Global Tension.

There is an inherent tension between local information minimization and global task completion. An agent may adopt a greedy strategy that minimizes the current context, skips a tool call, or shortens the current reasoning step, but such local savings may later return as a much higher recovery cost.

State Opacity.

Tool-using agents face substantial opacity in task state, environment state, tool state, and self-state. User requests are often incomplete, requiring the agent to infer task boundaries. External environments are also partially observable and dynamic. Moreover, LLMs often lack stable self-uncertainty calibration. They may be overconfident and skip necessary tool use, or overly conservative and invoke tools even when the task can be solved directly.

Dynamic Deployment Conditions.

Many efficiency policies are learned or evaluated in static and controlled environments, whereas real-world deployments involve dynamic tool systems and changing external states. A policy learned on a fixed benchmark or static tool library may therefore become unreliable when the environment changes. Efficient agents need to continuously monitor tool availability, data freshness, cost changes, and deployment constraints.

Task and Deployment Dependence.

Efficiency strategies are highly task- and deployment-conditioned. Different tasks have different dominant sources of inefficiency. Some are bottlenecked by large tool libraries, while others are constrained by costly external calls. Consequently, no single efficiency policy is likely to work across all settings. A strategy that removes redundancy in one task regime may remove necessary information or introduce extra overhead in another.

Appendix F. Evaluation and Benchmarks

Efficiency evaluation should jointly measure task utility and the coupled resource profile $R = (N_{\text{tok}}, N_{\text{call}}, L, M)$. Yet accuracy-centric benchmarks vary across tool environments, agent architectures, and operating conditions, limiting comparison. We organize them by benchmark paradigm, evaluation dimension, and reporting practice.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

Appendix F.1. Benchmark Paradigms

Task-Oriented Benchmarks.

AgentBench and GAIA assess end-to-end capability across diverse tasks, while environment-based benchmarks such as WebArena, ALFWorld, and ScienceWorld test interactive planning and long-horizon execution. However, they rarely isolate tool-use overhead.

Table A5. Unified macro-level empirical evaluation dimensions for efficient tool-using agents.

Dimension	Evaluation Settings	Metrics	Evaluation Target
Task Quality	QA, Math Reasoning, API Chains, Web-Search, Long-Horizon Tasks	Accuracy, Exact Match, F1, Pass@k, Task Completion Rate	Preserving end-to-end task success while applying efficiency strategies.
Decision Quality	Tool-necessity, Function Calling, Multi-agent Routing Benchmarks	Selection Accuracy, Routing Accuracy, Parameter Accuracy, Execution Success Rate	Making correct tool-use decisions and generating valid parameter schemas.
Efficiency	Large Tool Libraries, MCP Tool Pools, Multi-turn Contexts, Streaming API Settings	Tool Calls, Token Reduction, Compression Ratio, Speedup, Cache Hit Rate	Minimizing unnecessary context, tool calls, reasoning steps, or execution time.
Resource Efficiency	Budget-constrained Tasks, Edge Settings, FaaS Workflows	Token Cost, Tool Cost, Memory Usage, Energy Consumption, Throughput	Reducing monetary, computational, and operational costs.
Robustness	Out-of-Distribution (OOD) Tools, Incomplete Schema, Long-horizon Environments	Tool Hallucination Rate, Failed-call Rate, Stale-cache Error, OOD Success Rate	Maintaining stable efficiency behavior under dynamic noisy tool conditions.

Tool-Using Benchmarks.

Tool-using benchmarks such as ToolBench and API-Bank isolate external invocation decisions by assessing tool selection, argument generation, and execution correctness. BFCL extends coverage to serial, parallel, abstention, and stateful calls. ToolSandbox adds persistent state and implicit dependencies, while MCPVerse covers large executable tool inventories with time-sensitive outcome-based evaluation (Lu et al. 2025; Patil et al. 2025; Lei et al. 2025; Qin et al. 2023).

Efficiency-Oriented Evaluation.

A smaller body of work examines whether resource reductions preserve task quality across large tool libraries, long contexts, caching, and streaming interactions. Calibration and accuracy–cost analyses further expose these trade-offs (Shen et al. 2024; Wei et al. 2025). However, no common protocol spans these settings.

Appendix F.2. Evaluation Dimensions

We group existing evaluation settings, metrics, and objectives into task quality, decision quality, efficiency, resource efficiency, and robustness, as summarized in Table A5.

Appendix F.3. Evaluation Gaps and Reporting Guidance

Current evaluations rarely report task utility together with all four resource dimensions. They also under-measure retries, stale observations, rate limits, tool failures, side effects, and budget depletion, despite progress in stateful benchmarks (Lu et al. 2025; Patil et al. 2025). Live APIs and mutable backends introduce further drift, while inconsistent hardware, network conditions, and tool availability limit reproducibility (Lei et al. 2025; Qin et al. 2023).

Future studies should report task performance alongside N_{tok}, N_{call}, L, and M, document operating conditions, and analyze failure sources. Such joint reporting helps distinguish genuine efficiency gains from improvements obtained by sacrificing reliability.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

References

Kurtis, L. K. Break chatbot speed limits with speculative tool calls, 2025.

GetStream.io. Speculative tool calling: How to build sub-second voice agents, 2025.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.R.; Cao, Y. React: Synergizing reasoning and acting in language models. In Proceedings of the The eleventh international conference on learning representations, 2022.

Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Hambro, E.; Zettlemoyer, L.; Cancedda, N.; Scialom, T. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems* **2023**, *36*, 68539–68551.

Patil, S.G.; Zhang, T.; Wang, X.; Gonzalez, J.E. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems* **2024**, *37*, 126544–126565.

Nakano, R.; Hilton, J.; Balaji, S.; Wu, J.; Ouyang, L.; Kim, C.; Hesse, C.; Jain, S.; Kosaraju, V.; Saunders, W.; et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332* **2021**.

Karpas, E.; Abend, O.; Belinkov, Y.; Lenz, B.; Lieber, O.; Ratner, N.; Shoham, Y.; Bata, H.; Levine, Y.; Leyton-Brown, K.; et al. MRKL Systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445* **2022**.

Gao, L.; Madaan, A.; Zhou, S.; Alon, U.; Liu, P.; Yang, Y.; Callan, J.; Neubig, G. Pal: Program-aided language models. In Proceedings of the International conference on machine learning. PMLR, 2023, pp. 10764–10799.

Lu, J.; Holleis, T.; Zhang, Y.; Aumayer, B.; Nan, F.; Bai, H.; Ma, S.; Ma, S.; Li, M.; Yin, G.; et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In Proceedings of the Findings of the Association for Computational Linguistics: NAACL 2025, 2025, pp. 1160–1183.

Wang, H.; Huang, W.; Wang, Y.; Xi, Y.; Lu, J.; Zhang, H.; Hu, N.; Liu, Z.; Pan, J.Z.; Wong, K.F. Rethinking Stateful Tool Use in Multi-Turn Dialogues: Benchmarks and Challenges. In Proceedings of the Findings of the Association for Computational Linguistics: ACL 2025, 2025, pp. 5433–5453.

Patil, S.G.; Mao, H.; Yan, F.; Ji, C.C.J.; Suresh, V.; Stoica, I.; Gonzalez, J.E. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In Proceedings of the Forty-second International Conference on Machine Learning, 2025.

Xu, H.; Wang, Z.; Zhu, Z.; Pan, L.; Chen, X.; Fan, S.; Chen, L.; Yu, K. Alignment for efficient tool calling of large language models. In Proceedings of the Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, 2025, pp. 17787–17803.

Vigel, J.; Cai, R.; Chen, E.; Neema, A.; Liao, A.; Zhu, K.; O'Brien, S. Self Knowledge-Tracing for Tool Use (SKT-Tool): Helping LLM Agents Understand Their Capabilities in Tool Use. In Proceedings of the The Sixth Workshop on Insights from Negative Results in NLP, 2025, pp. 150–156.

Ghoshal, S.; Al-Bustami, A. When Do Tools and Planning Help LLMs Think? A Cost-and Latency-Aware Benchmark. *arXiv preprint arXiv:2601.02663* **2026**.

Lei, F.; Yang, Y.; Sun, W.; Lin, D. Mcpverse: An expansive, real-world benchmark for agentic tool use. *arXiv preprint arXiv:2508.16260* **2025**.

Du, Y.; Wei, F.; Zhang, H. Anytool: Self-reflective, hierarchical agents for large-scale api calls. *arXiv preprint arXiv:2402.04253* **2024**.

Bian, S.; Yan, M.; Jayarajan, A.; Pekhimenko, G.; Venkataraman, S. What Limits Agentic Systems Efficiency? *arXiv preprint arXiv:2510.16276* **2025**.

Huang, Z.; Zeng, W.; Fu, T.; Liu, T.; Sun, Y.; Hong, K.; Yang, X.; Liu, C.; Li, Y.; Zhang, Q.; et al. Reducing Latency of LLM Search Agent via Speculation-based Algorithm-System Co-Design. *arXiv preprint arXiv:2511.20048* **2025**.

Liu, D.; Li, Z.; Du, H.; Wu, X.; Gui, S.; Kuang, Y.; Sun, L. Graph-of-Skills: Dependency-Aware Structural Retrieval for Massive Agent Skills, 2026, [arXiv:cs.AI/2604.05333].

Xiao, J.; Li, B.; Li, X.; Li, J.; Jie, J.; Liu, X.; Jun, M.; Wang, C.; Tashi, N.; Yu, J. SkillSight: Calibrating Generic Content Bias for Skill Retrieval, 2026, [arXiv:cs.AI/2607.18785].

Krikorian, A.; Li, Y.; Corso, J.J. NTILC: Neural Tool Invocation via Learned Compression, 2026, [arXiv:cs.SE/2606.06566].

Jia, J.; Li, Q. AutoTool: Efficient tool selection for large language model agents. In Proceedings of the Proceedings of the AAAI Conference on Artificial Intelligence, 2026, Vol. 40, pp. 31265–31273.

Xiao, Q.; Shi, H.; Gao, Y.; Hu, W.; Jing, H.; Zheng, T.; Xu, B.; Zhang, Z.; Wang, W.; Li, H.; et al. SING: Synthetic Intention Graph for Scalable Active Tool Discovery in LLM Agents, 2026, [arXiv:cs.CL/2606.16591].

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

Miao, Y.; Yu, Z.; Zhao, L.; Zhu, B.; Haque, H. SkillLens: Adaptive Multi-Granularity Skill Reuse for Cost-Efficient LLM Agents, 2026, [arXiv:cs.AI/2605.08386].

Gaurav, N.; Akarsh, A.; Ranjan, A.; Bajaj, M. Dynamic react: Scalable tool selection for large-scale mcp environments. *arXiv preprint arXiv:2509.20386* **2025**.

Vadlapati, P. ToolSEE: Agent Tool Search Engine for Efficient and Scalable Tool Discovery Using Retrieval **2025**.

Patel, B.; Belli, D.; Jalalirad, A.; Arnold, M.; Ermolov, A.; Major, B. Dynamic Tool Dependency Retrieval for Efficient Function Calling. *arXiv preprint arXiv:2512.17052* **2025**.

Paramanayakam, V.; Karatzas, A.; Anagnostopoulos, I.; Stamoulis, D. Less is more: Optimizing function calling for llm execution on edge devices. In Proceedings of the 2025 Design, Automation & Test in Europe Conference (DATE). IEEE, 2025, pp. 1–7.

Robert, D.C.; Garg, A.; Punjabi, J.V.; Reddy M, P.K.; Balani, N.S.; Shah, R.; et al. Automatic Tool Selection to Reduce Large Language Model Latency **2024**.

Lu, Y.; Yu, H.; Khashabi, D. Gear: Augmenting language models with generalizable and efficient tool resolution. In Proceedings of the Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers), 2024, pp. 112–138.

Fore, M.; Singh, S.; Stamoulis, D. Geckopt: Llm system efficiency via intent-based tool selection. In Proceedings of the Proceedings of the Great Lakes Symposium on VLSI 2024, 2024, pp. 353–354.

Shemla, Y.; Yakobe, A.; Agarwal, T.; Patel, D.; Maghraoui, K.E. Internalizing Tool Knowledge in Small Language Models via QLoRA Fine-Tuning, 2026, [arXiv:cs.CL/2605.17774].

Yu, A.; Zhou, C.; Xu, T.; Guo, Z.; Shan, R.; Fu, Z.; Wang, J.; Liu, W.; Yu, Y.; Zhang, W.; et al. LatentSkill: From In-Context Textual Skills to In-Weight Latent Skills for LLM Agents, 2026, [arXiv:cs.CL/2606.06087].

Lu, Z.; Yao, Z.; Wu, J.; Han, C.; Gu, Q.; Cai, X.; Lu, W.; Xiao, J.; Zhuang, Y.; Shen, Y. SKILL0: In-Context Agentic Reinforcement Learning for Skill Internalization, 2026, [arXiv:cs.LG/2604.02268].

Gao, Y.; Li, Z.; Yuan, Y.; Ji, Z.; Ma, P.; Wang, S. SkillReducer: Optimizing LLM Agent Skills for Token Efficiency, 2026, [arXiv:cs.SE/2603.29919].

Yuan, S.; Song, K.; Chen, J.; Tan, X.; Shen, Y.; Ren, K.; Li, D.; Yang, D. Easytool: Enhancing llm-based agents with concise tool instruction. In Proceedings of the Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), 2025, pp. 951–972.

Vijayvargiya, S.; Lokesh, R. Efficient On-Device Agents via Adaptive Context Management. *arXiv preprint arXiv:2511.03728* **2025**.

Xu, Y.; Feng, Y.; Mu, H.; Hou, Y.; Li, Y.; Wang, X.; Zhong, W.; Li, Z.; Tu, D.; Zhu, Q.; et al. Concise and precise context compression for tool-using language models. In Proceedings of the Findings of the Association for Computational Linguistics: ACL 2024, 2024, pp. 16430–16441.

Anthropic. Code Execution with MCP: 98.7% Token Reduction Through Efficient Agent Architecture. Anthropic Engineering Blog, 2024.

Zhang, H.; Sun, Z. AGORA: Adapter-Grounded Observation-Action Retention for Inference-Free Prompt Compression in LLM Agents, 2026, [arXiv:cs.AI/2605.26596].

Xu, B.; Xue, Z.; Chen, D.; Fu, C.; Wu, C.; Huang, C.; Jiang, C.; Fang, J.; Deng, X.; Chen, Y.; et al. TokenPilot: Cache-Efficient Context Management for LLM Agents, 2026, [arXiv:cs.CL/2606.17016].

Hao, X.; Meng, H.; Yin, X.; Zhu, J.; Cao, C. Self-GC: Self-Governing Context for Long-Horizon LLM Agents, 2026, [arXiv:cs.AI/2607.00692].

Wu, Y.; Zheng, Y.; Xu, T.; Zhang, Z.; Yu, Y.; Zhu, J.; Ma, C.; Lin, B.; Dong, B.; Zhu, H.; et al. ContextBudget: Budget-Aware Context Management for Long-Horizon Search Agents, 2026, [arXiv:cs.AI/2604.01664].

Yi, L.; Lei, R.; Yao, L.; Xie, Y.; Li, Y.; Zhang, W.; Wei, Z.; Li, Y.; Nie, J.Y. Learning Agent-Compatible Context Management for Long-Horizon Tasks, 2026, [arXiv:cs.AI/2605.30785].

Chen, Z.; Pan, R.; Dai, Y.; Netravali, R. Slipstream: Trajectory-Grounded Compaction Validation for Long-Horizon Agents, 2026, [arXiv:cs.MA/2605.08580].

Wu, Y.; Zhang, Y.; Ghosh, S.; Basu, S.; Deoras, A.; Huan, J.; Gupta, G. ContextWeaver: Selective and Dependency-Structured Memory Construction for LLM Agents, 2026, [arXiv:cs.CL/2604.23069].

Xie, H.; Wang, X.; Wang, Y.; Zhao, P.; Ju, F. From History to State: Constant-Context Skill Learning for LLM Agents, 2026, [arXiv:cs.AI/2605.05413].

Wang, Z.; Chen, H.; Wang, J.; Wei, W. Memex(RL): Scaling Long-Horizon LLM Agents via Indexed Experience Memory, 2026, [arXiv:cs.CL/2603.04257].

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

Gu, Z.; Zhang, Q.; Khattab, O.; Madden, S. PEEK: Context Map as an Orientation Cache for Long-Context LLM Agents, 2026, [arXiv:cs.AI/2605.19932].

Lumer, E.; Gulati, A.; Subbiah, V.K.; Basavaraju, P.H.; Burke, J.A. Memtool: Optimizing short-term memory management for dynamic tool calling in llm agent multi-turn conversations. *arXiv preprint arXiv:2507.21428* **2025**.

Zhang, Y.; Shu, J.; Ma, Y.; Lin, X.; Wu, S.; Sang, J. Memory as action: Autonomous context curation for long-horizon agentic tasks. *arXiv preprint arXiv:2510.12635* **2025**.

Ye, R.; Zhang, Z.; Li, K.; Yin, H.; Tao, Z.; Zhao, Y.; Su, L.; Zhang, L.; Qiao, Z.; Wang, X.; et al. AgentFold: Long-Horizon Web Agents with Proactive Context Management. *arXiv preprint arXiv:2510.24699* **2025**.

Kang, M.; Chen, W.N.; Han, D.; Inan, H.A.; Wutschitz, L.; Chen, Y.; Sim, R.; Rajmohan, S. Acon: Optimizing context compression for long-horizon llm agents. *arXiv preprint arXiv:2510.00615* **2025**.

Chen, J.; Liang, J.; Wang, B. Smurfs: Leveraging multiple proficiency agents with context-efficiency for tool planning. *arXiv e-prints* **2024**, pp. arXiv–2405.

Sun, C.E.; Liu, L.; Yan, G.; Wang, Z.; Weng, T.W. LLM Agents Already Know When to Call Tools – Even Without Reasoning, 2026, [arXiv:cs.CL/2605.09252].

Qian, C.; Acikgoz, E.C.; Wang, H.; Chen, X.; Sil, A.; Hakkani-Tur, D.; Tur, G.; Ji, H. SMART: Self-aware agent for tool overuse mitigation. In Proceedings of the Findings of the Association for Computational Linguistics: ACL 2025, 2025, pp. 4604–4621.

Li, W.; Li, D.; Dong, K.; Zhang, C.; Zhang, H.; Liu, W.; Wang, Y.; Tang, R.; Liu, Y. Adaptive tool use in large language models with meta-cognition trigger. In Proceedings of the Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2025, pp. 13346–13370.

Ross, H.; Mahabaleshwarkar, A.S.; Suhara, Y. When2Call: When (not) to call tools. In Proceedings of the Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), 2025, pp. 3391–3409.

Trombino, D.; Pecorella, V.; De Giulii, A.; Tresoldi, D. Knowledge Base-Aware Orchestration: A Dynamic, Privacy-Preserving Method for Multi-Agent Systems. *arXiv preprint arXiv:2509.19599* **2025**.

Shen, Y.; Zhu, X.; Chen, L. Smartcal: An approach to self-aware tool-use evaluation and calibration. In Proceedings of the Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track, 2024, pp. 774–789.

Gui, A.; Li, J.; Dai, Y.; Du, N.; Xiao, H. Look before you leap: towards decision-aware and generalizable tool-usage for large language models. *arXiv preprint arXiv:2402.16696* **2024**.

Fang, Z.; Sun, R. AdaTIR: Adaptive Tool-Integrated Reasoning via Difficulty-Aware Policy Optimization. *arXiv preprint arXiv:2601.14696* **2026**.

Chu, K.; Xiang, D.; Zhang, W. Latency-Quality Routing for Functionally Equivalent Tools in LLM Agents, 2026, [arXiv:cs.LG/2605.14241].

Liu, H.; Tian, C.; An, N.; Wang, Z.; Lu, P.; Yu, C.; Qi, Q. Budget-Constrained Agentic Large Language Models: Intention-Based Planning for Costly Tool Use. *arXiv preprint arXiv:2602.11541* **2026**.

Wang, B. A Stackelberg Framework for Resource-Aware LLM Agents: Learning, Repair, and Conditional Guarantees, 2026, [arXiv:cs.AI/2606.23026].

Wei, Y.; Yu, X.; Weng, Y.; Pan, T.; Li, A.; Du, L. Autotir: Autonomous tools integrated reasoning via reinforcement learning. *arXiv preprint arXiv:2507.21836* **2025**.

WANG, H.; Qian, C.; Zhong, W.; Chen, X.; Qiu, J.; Huang, S.; Jin, B.; Wang, M.; Wong, K.F.; Ji, H. Acting Less is Reasoning More! Teaching Language Model to Act Efficiently. In Proceedings of the NeurIPS 2025 Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning, 2025.

Liu, T.; Wang, Z.; Miao, J.; Hsu, I.; Yan, J.; Chen, J.; Han, R.; Xu, F.; Chen, Y.; Jiang, K.; et al. Budget-aware tool-use enables effective agent scaling. *arXiv preprint arXiv:2511.17006* **2025**.

Gul, M.O.; Cardie, C.; Goyal, T. MASH: Modeling Abstention via Selective Help-Seeking, 2026, [arXiv:cs.CL/2510.01152].

Zheng, Y.; Li, P.; Yan, M.; Zhang, J.; Huang, F.; Liu, Y. Budget-constrained tool learning with planning. In Proceedings of the Findings of the Association for Computational Linguistics: ACL 2024, 2024, pp. 9039–9052.

Cui, X.; Zou, Y.; Li, Z.; Li, P.; Xu, X.; Liu, X.; Huang, H. T2Agent: A Tool-augmented Multimodal Misinformation Detection Agent with Monte Carlo Tree Search. In Proceedings of the Proceedings of the AAAI Conference on Artificial Intelligence, 2026, Vol. 40, pp. 175–183.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

Yang, S.; Han, S.C.; Ding, Y.; Wang, S.; Hoy, E. ToolTree: Efficient LLM Agent Tool Planning via Dual-Feedback Monte Carlo Tree Search and Bidirectional Pruning. *arXiv preprint arXiv:2603.12740* **2026**.

Li, Y.; Deng, W.; Li, J.; Li, X. Spend Less, Reason Better: Budget-Aware Value Tree Search for LLM Agents, 2026, [arXiv:cs.LG/2603.12634].

Wei, R.; Shi, G.; Cheng, M.; Zhang, N.; Li, P.; Ghosh, S.; Gorde, V.; Akoglu, L. Long-Horizon Plan Execution in Large Tool Spaces through Entropy-Guided Branching, 2026, [arXiv:cs.AI/2604.12126].

Winston, C.; Wang, R.Y.; Mirhoseini, A.; Kozyrakis, C. Agent JIT Compilation for Latency-Optimizing Web Agent Planning and Scheduling, 2026, [arXiv:cs.LG/2605.21470].

Pham, D.; Katevas, K.; Shamsabadi, A.S.; Haddadi, H. AgentStop: Terminating Local AI Agents Early to Save Energy in Consumer Devices, 2026, [arXiv:cs.LG/2605.15206].

Ruan, K.; Huang, Z.; Zhou, Z.; Wei, Q.; Lin, J.; Wang, X.; Sun, H. Doomed from the Start: Early Abort of LLM Agent Episodes via a Recall-Controlled Probe Cascade, 2026, [arXiv:cs.AI/2607.06503].

Chen, Y.; Dong, G.; Dou, Z. ET-Agent: Incentivizing Effective Tool-Integrated Reasoning Agent via Behavior Calibration. *arXiv preprint arXiv:2601.06860* **2026**.

Yang, J.; Hou, B.; Wei, W.; Bao, Y.; Chang, S. Ares: Adaptive Reasoning Effort Selection for Efficient LLM Agents, 2026, [arXiv:cs.AI/2603.07915].

Ma, Z.; Liu, J.; Luo, X.; Huang, Z.; Zhu, Q.; Che, W. Advancing tool-augmented large language models via meta-verification and reflection learning. In Proceedings of the Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2, 2025, pp. 2078–2089.

Lu, Y.; Ye, F.; Li, J.; Gao, Q.; Liu, C.; Luo, H.; Du, N.; Li, X.; Ren, F. Codetool: Enhancing programmatic tool invocation of llms via process supervision. *arXiv preprint arXiv:2503.20840* **2025**.

Wu, B.; Meij, E.; Yilmaz, E. A joint optimization framework for enhancing efficiency of tool utilization in llm agents. In Proceedings of the Findings of the Association for Computational Linguistics: ACL 2025, 2025, pp. 22361–22373.

Wu, Q.; Liu, W.; Luan, J.; Wang, B. Toolplanner: A tool augmented llm for multi granularity instructions with path planning and feedback. In Proceedings of the Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, 2024, pp. 18315–18339.

Chen, S.; Wang, Y.; Wu, Y.F.; Chen, Q.G.; Xu, Z.; Luo, W.; Zhang, K.; Zhang, L. Advancing tool-augmented large language models: Integrating insights from errors in inference trees. *Advances in Neural Information Processing Systems* **2024**, *37*, 106555–106581.

Zhuang, Y.; Chen, X.; Yu, T.; Mitra, S.; Bursztyn, V.; Rossi, R.A.; Sarkhel, S.; Zhang, C. Toolchain*: Efficient action space navigation in large language models with a* search. *arXiv preprint arXiv:2310.13227* **2023**.

Hu, M.; Mu, Y.; Yu, X.; Ding, M.; Wu, S.; Shao, W.; Chen, Q.; Wang, B.; Qiao, Y.; Luo, P. Tree-planner: Efficient close-loop task planning with large language models. *arXiv preprint arXiv:2310.08582* **2023**.

Kim, H.; Wu, Y.; Tambe, T. AgenticCache: Cache-Driven Asynchronous Planning for Embodied AI Agents, 2026, [arXiv:cs.LG/2604.24039].

Chillara, V.; Kline, D.; Alvares, C.; Wooten, E.; Yang, H.; Khetan, S.; Bauer, C.; Guillory, T.; Shah, T.; Dhariwal, Y.; et al. SemanticALLI: Caching Reasoning, Not Just Responses, in Agentic Systems. *arXiv preprint arXiv:2601.16286* **2026**.

He, S.; Cui, Y.; Wu, F.; Ma, X.; Lu, J.; Li, Y.; Ding, B.; Chowdhury, M. Beyond Domains: Reusing Web Skills via Transferable Interaction Patterns, 2026, [arXiv:cs.AI/2606.17645].

Ding, H.; Xie, Y.; Wei, Z.; Li, Y.; Ding, B. From Atomic Actions to Standard Operating Procedures: Iterative Tool Optimization for Self-Evolving LLM Agents, 2026, [arXiv:cs.AI/2607.07321].

Kujanpää, K.; Liu, N.; Alam, S.; Sura, Y.R.; Yang, T.; Klinkner, K.; Malmasi, S. Tool-Making and Self-Evolving LLM Agents in Low-Latency Systems. *arXiv preprint arXiv:2607.08010* **2026**.

Guopeng, L.; Ruiqi, W.; Haisheng, T.; Guoliang, C. A Plan Reuse Mechanism for LLM-Driven Agent. *Journal of Computer Research and Development* **2024**, *61*, 2706–2720. https://doi.org/10.7544/issn1000-1239.202440380.

Zhang, Q.; Wornow, M.; Wan, G.; Olukotun, K. Agentic Plan Caching: Test-Time Memory for Fast and Cost-Efficient LLM Agents. *arXiv preprint arXiv:2506.14852* **2025**.

Kang, M.; Jeong, J.; Lee, S.; Cho, J.; Hwang, S.J. Distilling llm agent into small models with retrieval and code tools. *arXiv preprint arXiv:2505.17612* **2025**.

Zhang, H.; Yang, J.; Li, H.; He, Y.; Gong, F. ODIA: Oriented Distillation for Inline Acceleration of LLM-based Function Calling. *arXiv preprint arXiv:2507.08877* **2025**.

Qiu, J.; Juan, X.; Wang, Y.; Yang, L.; Qi, X.; Zhang, T.; Guo, J.; Lu, Y.; Yao, Z.; Wang, H.; et al. Agentdistill: Training-free agent distillation with generalizable mcp boxes. *arXiv preprint arXiv:2506.14728* **2025**.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

Erdogan, L.E.; Lee, N.; Jha, S.; Kim, S.; Tabrizi, R.; Moon, S.; Hooper, C.R.C.; Anumanchipalli, G.; Keutzer, K.; Gholami, A. Tinyagent: Function calling at the edge. In Proceedings of the Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 2024, pp. 80–88.

Zhai, Y.; Shen, D.; Luo, J.; Yang, B. ToolCaching: Towards Efficient Caching for LLM Tool-calling. *arXiv preprint arXiv:2601.15335* **2026**.

Ruan, C.; Bi, C.; Zheng, K.; Shi, Z.; Wan, X.; Li, J. Cortex: Achieving {Low-Latency},{Cost-Efficient} Remote Data Access For {LLM} via {Semantic-Aware} Knowledge Caching. In Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26), 2026, pp. 2407–2421.

Boateng, E.A.; MacDonald, K.; Kumar, A.; Kodwani, S.; Das, S. Decoupling Search from Reasoning: A Vendor-Agnostic Grounding Architecture for LLM Agents, 2026, [arXiv:cs.AI/2606.18947].

Kumar, A.V.; Kataria, B.; Oh, B.; Manzoor, E.; Singh, R. TVCACHE: A Stateful Tool-Value Cache for Post-Training LLM Agents. *arXiv preprint arXiv:2602.10986* **2026**.

Kulkarni, V.; Jha, V.; Reddy, N.; Eswaran, A.; Jayachandran, P.; Simmhan, Y. Optimizing FaaS Platforms for MCP-enabled Agentic Workflows. *arXiv preprint arXiv:2601.14735* **2026**.

Merchant, A.M.; Veera, K.; Goyla, S.K.; Bhure, S.; Patel, D.; Maghraoui, K.E. Evaluating Temporal Semantic Caching and Workflow Optimization in Agentic Plan-Execute Pipelines, 2026, [arXiv:cs.AI/2605.20630].

Kwon, J. Tool Cache Agent: Accelerating LLM Agent Through Intelligent Tool Call Caching, 2025.

Singh, S.; Fore, M.; Karatzas, A.; Lee, C.; Jian, Y.; Shangguan, L.; Yu, F.; Anagnostopoulos, I.; Stamoulis, D. Llm-dcache: improving tool-augmented llms with gpt-driven localized data caching. *arXiv preprint arXiv:2406.06799* **2024**.

Bang, F. Gptcache: An open-source semantic cache for llm applications enabling faster answers and cost savings. In Proceedings of the Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023), 2023, pp. 212–218.

Xia, H.; Li, Y.; Du, C.; Song, M.; Li, W. ToolSpec: Accelerating Tool Calling via Schema-Aware and Retrieval-Augmented Speculative Decoding, 2026, [arXiv:cs.CL/2604.13519].

Feng, G.; Mao, H.; Dutta, P.; Gonzalez, J.E. Concurrency without Model Changes: Future-based Asynchronous Function Calling for LLMs, 2026, [arXiv:cs.CL/2605.15077].

Choi, D.; Park, K.; Song, W.; Dingliwal, S.; Jayanthi, S.M.; Shin, J.; Galstyan, A. IdleSpec: Exploiting Idle Time via Speculative Planning for LLM Agents, 2026, [arXiv:cs.AI/2605.22154].

Biswas, A.; Goel, K.; Mohan, J.; Khare, A.; Parayil, A.; Ramjee, R.; Bansal, C. Sutradhara: An Intelligent Orchestrator-Engine Co-design for Tool-based Agentic Inference. *arXiv preprint arXiv:2601.12967* **2026**.

Zhe, T.; Wang, H.; Luo, B.; Wu, M.; Fan, W.; Luo, X.; Yao, Z.; Chen, H.; Wang, D. Robust and Efficient Tool Orchestration via Layered Execution Structures with Reflective Correction. *arXiv preprint arXiv:2602.18968* **2026**.

CodeAnt AI. Why Parallel Tool Calling Matters for LLM Agents. Blog post, 2026.

Gao, S.; Dwivedi-Yu, J.; Yu, P.; Tan, X.E.; Pasunuru, R.; Golovneva, O.; Sinha, K.; Celikyilmaz, A.; Bosselut, A.; Wang, T. Efficient tool use with chain-of-abstraction reasoning. In Proceedings of the Proceedings of the 31st International Conference on Computational Linguistics, 2025, pp. 2727–2743.

Gong, J.; Huang, Y.; Yuan, B.; Zhu, M.; Liao, Z.; Liang, J.; Zhan, J.; Wang, J.; Shu, H.; Xiong, M.; et al. GhostShell: Streaming LLM Function Calls for Concurrent Embodied Programming. *arXiv preprint arXiv:2508.05298* **2025**.

Liu, X.; Di, P.; Li, C.; Sun, J.; Wang, J. Efficient Function Orchestration for Large Language Models. *IEEE Transactions on Software Engineering* **2025**.

Wu, D.; Wang, J.; Meng, Y.; Zhang, Y.; Sun, L.; Wang, Z. Catp-llm: Empowering large language models for cost-aware tool planning. In Proceedings of the Proceedings of the IEEE/CVF International Conference on Computer Vision, 2025, pp. 8699–8709.

Wu, J.; Zhao, Q.; Chen, Z.; Qin, K.; Zhao, Y.; Wang, X.; Yao, Y. GAP: Graph-Based Agent Planning with Parallel Tool Use and Reinforcement Learning. *arXiv preprint arXiv:2510.25320* **2025**.

Zhu, D.; Shi, W.; Shi, Z.; Ren, Z.; Wang, S.; Yan, L.; Yin, D. Divide-then-aggregate: An efficient tool learning method via parallel tool invocation. In Proceedings of the Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2025, pp. 28859–28875.

Qin, T.; Chen, Q.; Wang, S.; Xing, H.; Zhu, K.; Zhu, H.; Shi, D.; Liu, X.; Zhang, G.; Liu, J.; et al. Flash-searcher: Fast and effective web agents via dag-based parallel execution. *arXiv preprint arXiv:2509.25301* **2025**.

Xu, Y.; Kong, X.; Chen, T.; Zhuo, D. Conveyor: Efficient tool-aware llm serving with tool partial execution. *arXiv preprint arXiv:2406.00059* **2024**.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 August 2026 doi:10.20944/preprints202608.0365.v1

Gim, I.; Lee, S.s.; Zhong, L. Asynchronous llm function calling. *arXiv preprint arXiv:2412.07017* **2024**.

Kim, S.; Moon, S.; Tabrizi, R.; Lee, N.; Mahoney, M.W.; Keutzer, K.; Gholami, A. An llm compiler for parallel function calling. In Proceedings of the Forty-first International Conference on Machine Learning, 2024.

Li, Y.; Ye, Q.; Choubey, P.K.; Zhang, J.; Wu, C.S. Speculate with Memory: Lossless Acceleration for LLM Agents, 2026, [arXiv:cs.LG/2607.12236].

Bai, H.; Lv, W.; Zheng, H.; Lu, Y.; Shu, J. SPORK: Self-Speculative Forking to Accelerate Agentic LLM Inference, 2026, [arXiv:cs.DC/2607.03333].

Hooper, C.; Kang, M.; Moon, S.; Lee, N.; Wen, E.; Wawrzynek, J.; Mahoney, M.W.; Shao, Y.S.; Gholami, A.; Keutzer, K. Speculative Interaction Agents: Building Real-Time Agents with Asynchronous I/O and Speculative Tool Calling, 2026, [arXiv:cs.LG/2605.13360].

Wong, M.; Hsieh, K.; Nath, S.; Netravali, R. Skim: Speculative Execution for Fast and Efficient Web Agents, 2026, [arXiv:cs.AI/2605.16565].

Ye, N.; Ahuja, A.; Liargkovas, G.; Lu, Y.; Kaffes, K.; Peng, T. Speculative Actions: A Lossless Framework for Faster Agentic Systems, 2025, [arXiv:cs.AI/2510.04371].

Nichols, D.; Singhania, P.; Jekel, C.; Bhatele, A.; Menon, H. Optimizing Agentic Language Model Inference via Speculative Tool Calls. *arXiv preprint arXiv:2512.15834* **2025**.

Arora, S.; Khan, H.; Sun, K.; Dong, X.L.; Choudhary, S.; Moon, S.; Zhang, X.; Sagar, A.; Appini, S.T.; Patnaik, K.; et al. Stream rag: Instant and accurate spoken dialogue systems with streaming tool usage. *arXiv preprint arXiv:2510.02044* **2025**.

Singh, S.; Karatzas, A.; Fore, M.; Anagnostopoulos, I.; Stamoulis, D. An llm-tool compiler for fused parallel function calling. *arXiv preprint arXiv:2405.17438* **2024**.

Zhang, J.; Zhao, B.; Yang, W.; Foerster, J.; Clune, J.; Jiang, M.; Devlin, S.; Shavrina, T. Hyperagents, 2026, [arXiv:cs.AI/2603.19461].

Wang, L.; Ma, C.; Feng, X.; Zhang, Z.; Yang, H.; Zhang, J.; Chen, Z.; Tang, J.; Chen, X.; Lin, Y.; et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science* **2024**, *18*, 186345.

Guo, T.; Chen, X.; Wang, Y.; Chang, R.; Pei, S.; Chawla, N.V.; Wiest, O.; Zhang, X. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680* **2024**.

Luo, J.; Zhang, W.; Yuan, Y.; Zhao, Y.; Yang, J.; Gu, Y.; Wu, B.; Chen, B.; Qiao, Z.; Long, Q.; et al. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460* **2025**.

Qu, C.; Dai, S.; Wei, X.; Cai, H.; Wang, S.; Yin, D.; Xu, J.; Wen, J.R. Tool learning with large language models: A survey. *Frontiers of Computer Science* **2025**, *19*, 198343.

Yehudai, A.; Eden, L.; Li, A.; Uziel, G.; Zhao, Y.; Bar-Haim, R.; Cohan, A.; Shmueli-Scheuer, M. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416* **2025**.

Du, S.; Zhao, J.; Shi, J.; Xie, Z.; Jiang, X.; Bai, Y.; He, L. A survey on the optimization of large language model-based agents. *ACM Computing Surveys* **2026**, *58*, 1–37.

Yang, X.; Li, L.; Zhou, H.; Zhu, T.; Qu, X.; Fan, Y.; Wei, Q.; Ye, R.; Kang, L.; Qin, Y.; et al. Toward Efficient Agents: Memory, Tool learning, and Planning. *arXiv preprint arXiv:2601.14192* **2026**.

Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B.; et al. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. *arXiv preprint arXiv:2307.16789* **2023**.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.