Selection over Injection: Evaluating Tool Retrieval in High-Cardinality AgentOps

Vincent Caldeira and Ilya Kolchinsky
Red Hat, Germany

vcaldeir@redhat.com
ikolchin@redhat.com

Abstract: As Large Language Models (LLMs) transition into autonomous agentic systems via standardized interfaces like the Model Context Protocol (MCP), they are increasingly exposed to high-cardinality toolsets. While granting agents access to thousands of enterprise tools theoretically expands their utility, naively injecting these tool schemas into a model's context window leads to prompt bloat. This introduces operational and governance risks such as unsafe tool use, reduced accountability, and overcomplicated compliance. Additionally, polluting the context by an extensive number of tool definitions, degrades reasoning performance, increases latency, and triggers hallucinations. To address this, we present an evaluation of Semantic Tool Selection - a retrieval-augmented approach (Tool-RAG) that pre-filters tool registries to inject only the most relevant tools per query, enabling transparent and accountable tool use decisions.

In this paper, we implement a semantic routing architecture heavily anchored in a new open-source library for tool retrieval and subject it to a rigorous, AgentOps-driven evaluation framework. Rather than relying solely on end-to-end task completion, we trace and evaluate the agent's inner execution loop, introducing granular metrics for Information Retrieval alongside Agentic Execution metrics (Tool Selection Quality, Argument Correctness, and Context Efficiency). Evaluated against a high-cardinality adaptation of BFCL V4 Multiple dataset using a shared 443-tool catalog evaluated under a one-turn tool-binding protocol, our findings demonstrate that semantic tool retrieval reduces the number of tools exposed to the model by over 90%, thus drastically improving the accountability, reliability and explainability of tool selection and execution decisions compared to traditional static injection. This work provides a much-needed empirical foundation for AgentOps practitioners looking to scale multi-agent systems reliably without retraining foundational models.

Keywords: Large language models, AI agents, Tool use, Retrieval, Agent evaluation

1. Introduction

Large Language Models are increasingly being deployed in production settings requiring interaction with external tools through standardized interfaces such as the Model Context Protocol (MCP). LLM-powered agents may be exposed to large and continuously growing tool registries containing internal APIs, enterprise services and automation endpoints. As a result, the practical bottleneck in many agentic systems is no longer only model capability, but the ability to use tools reliably at scale.

Tool use in language models is commonly implemented by injecting all available tool definitions directly into the model context. This strategy is simple, but it does not scale naturally as the number of available tools grows. Full injection creates the so-called context-availability paradox: increasing tool availability expands the theoretical capabilities of an agent, yet exposing the complete tool catalogue to the model can reduce its practical effectiveness. The resulting prompt bloat gives the model a substantially larger action space from which to select. In practice, excessive tool exposure can lead the model to choose an inappropriate tool, fail to produce a valid call, or otherwise fail despite having access to the required capability.

High-cardinality tool use also creates a diagnostic challenge. A failed tool interaction may have several different causes: the relevant tool may not have been surfaced, the model may choose the wrong tool despite seeing the correct one, or it may select the correct tool but invoke it with invalid arguments. Evaluations collapsing these cases into a single end-to-end outcome provide little guidance about where the system failed or how it should be improved. This motivates the stage-wise evaluation used in this paper, which treats retrieval, selection and argument construction as distinct stages.

In this paper, we study a retrieval-augmented approach to tool use referred to as Tool RAG. Rather than exposing the entire tool registry, the system introduces a selection layer before model inference. For each query, this layer retrieves a small set of relevant tools and binds only that subset into the model context. We instantiate this general pattern using the *ToolScope* library for the dense retrieval mode, while also evaluating a sparse BM25 baseline. The approach decouples tool discovery from downstream reasoning, preserves ordinary function-calling interfaces, and provides an explicit intermediate decision that can be inspected independently of the eventual tool call.

The paper makes four contributions:

1. It formulates the context-availability paradox as a practical limitation of high-cardinality tool use in agentic systems.
2. It presents a Tool RAG architectural blueprint for dynamically identifying and binding relevant tools before model inference.
3. It applies an AgentOps-oriented evaluation decomposition that separately measures retrieval quality, downstream tool selection, argument correctness and context efficiency, allowing failures to be localized to different stages of the tool-use pipeline.
4. It reports an experimental evaluation over a shared 443-tool catalogue and five locally runnable models ranging from 3B to 70B parameters, showing that pre-inference selection improves tool-name accuracy across all evaluated configurations while reducing the serialized tool-schema size estimate by approximately 97.7%.

2. Background and Related Work

The LLM tool calling scalability problem has motivated a substantial body of work on tool retrieval, tool representation and more efficient tool use. A recent survey by Yang et al. (2026) places tool learning alongside memory and planning as one of the major efficiency axes of agentic systems. Within this landscape, one prominent family of approaches formulates tool scaling explicitly as a retrieval problem. ProTIP progressively retrieves tools to improve planning over multi-step tasks (Anantha et al., 2023). AnyTool uses a hierarchical architecture to operate over large API collections (Du, Wei and Zhang, 2024). DRAFT improves tool use indirectly by refining tool documentation through self-driven interactions (Qu et al., 2025), while Kachuee et al. (2025) target the query side by generating retrieval queries with an LLM rather than relying only on the original user request. At the system level, RAG-MCP uses retrieval to reduce prompt bloat in MCP-based environments (Gan and Sun, 2025), while Toolshed extends the retrieval stack with enrichment, transformation, reranking and reflection (Lumer et al., 2025b). These approaches motivate treating tool selection explicitly rather than leaving the complete catalogue inside the prompt.

A second line of work enriches retrieval with information about tool usage and relationships. Tool2Vec learns tool representations from usage-driven synthetic data rather than relying only on static descriptions (Moon et al., 2024). Completeness-oriented retrieval considers relationships among tools and the need to retrieve complementary capabilities rather than merely the individually most similar tools (Qu et al., 2024). Graph RAG-Tool Fusion further incorporates inter-tool dependency structure so that retrieval can surface tools together with related capabilities on which they depend (Lumer et al., 2025a). These approaches are particularly relevant to high-cardinality environments because they treat the tool ecosystem itself as a structured retrieval space.

A related but distinct family of methods integrates tools more tightly into the model. ToolkenGPT represents tools through learned embeddings that function similarly to vocabulary items, enabling invocation through token prediction (Hao et al., 2023), while Toolken+ augments this approach with reranking and a reject option (Yakovlev, Nikolenko and Bout, 2024). ToolGen unifies tool retrieval and calling as generation over tool tokens (Wang et al., 2025), and Chain-of-Tools targets large pools of previously unseen tools while keeping the underlying language model frozen (Wu et al., 2025). Context-compression approaches attack the same scalability problem from another direction by reducing the amount of tool documentation placed into the prompt while attempting to preserve its relevant information (Xu et al., 2024). Together, these methods show that scalable tool use can be addressed through external retrieval, learned representations, generative selection or context compression.

Broader research on LLM tool use focuses on tool calling and tool-integrated reasoning. Toolformer is an influential early example that trains a language model to determine whether, when and how to invoke external tools (Schick et al., 2023). Chain-of-Abstraction, LLMCompiler and LLM-Tool Compiler address reasoning abstraction, execution efficiency or parallel function calling once relevant capabilities are available (Gao et al., 2025; Kim et al., 2024; Singh et al., 2024). Agent-FLAN studies tuning strategies for stronger agent behaviour and reduced hallucinated tool use (Chen et al., 2024). Gorilla combines model specialization with large-scale API-oriented evaluation (Patil et al., 2024), whereas MCP-Zero explores active capability discovery and task-specific toolchain construction (Fei, Zheng and Feng, 2025). These approaches primarily address how an agent should use an available action space; our focus is on how that action space should be exposed to the model in the first place.

3. Problem Statement: The Context-availability Paradox

In modern agentic workflows, the number of tools accessible to a model can become very large. Enterprise deployments may involve dozens, hundreds or thousands of internal APIs, service endpoints and automation interfaces. Although this breadth increases theoretical capability, it also creates a difficult selection problem at inference time.

When a model is presented with a large collection of tool schemas, its prompt becomes cluttered with structured definitions that are mostly irrelevant to the current request. The model must identify the appropriate capability while simultaneously reasoning over the input task. As the action space grows, similar or overlapping tools also become increasingly difficult to distinguish. More available tools can therefore make the agent less effective unless tool exposure is controlled. Static injection also carries substantial efficiency costs since every inference step includes the complete tool catalogue. In the experiment reported in this paper, the serialized representation of all 443 tool definitions corresponds to approximately 60,051 tool-token equivalents under the harness's approximation, compared with approximately 1,362 for a dense top-10 shortlist. This repeated context burden affects both resource consumption and latency regardless of how many tools are actually relevant.

The problem is not limited to efficiency. Excessive tool exposure can also complicate operational control and governance. When hundreds of capabilities are simultaneously available to a model, it becomes harder to inspect what the model was allowed to use, why a specific tool was chosen and where policy constraints should be applied. Restricting the candidate set does not by itself guarantee safety or compliance, but it creates a more explicit control point at which capability exposure can be logged, filtered and audited.

A further challenge is evaluation. Final task success alone cannot distinguish whether a failure came from poor retrieval, incorrect choice among available tools or invalid argument construction. These failures call for different remedies: improving the retriever will not fix bad arguments, while upgrading the calling model will not help if the required tool was never made available. This lack of diagnostic resolution creates an AgentOps blind spot. A useful evaluation of high-cardinality tool use should therefore separate the retrieval decision from the downstream model decision.

4. Proposed Methodology: Tool RAG

To address these limitations, we use a retrieval-mediated architecture in which tool selection takes place before LLM inference. The central idea of Tool RAG is straightforward: instead of binding an entire registry to every model call, retrieve a query-relevant subset and expose only that subset to the model for the current reasoning step. The downstream model continues to use its ordinary native tool-calling interface.

The first component is canonical indexing. Heterogeneous tool definitions, including OpenAPI specifications and MCP manifests, are normalized into a common internal representation. The representation captures the information needed for retrieval, including the tool name, a normalized description and the parameter schema. This creates a common retrieval surface across tools originating from different interfaces and ecosystems. The indexed tools can then be ranked for each incoming request. Tool RAG is not inherently tied to a particular retrieval algorithm: lexical, dense or more sophisticated reranking approaches can occupy the same architectural position. In our reference implementation, dense semantic retrieval embeds normalized tool representations and performs similarity search against the current query. The system then dynamically binds only the selected subset into the LLM context.

We instantiate this architecture using *ToolScope*, an open-source tool-retrieval library. ToolScope provides schema normalization, dense retrieval and dynamic binding while remaining independent of the underlying calling model. The library also supports optional capabilities relevant to larger agent systems.

5. Evaluation Strategy and Experimental Setup

5.1 Benchmark and Experimental Environment

Our experimental evaluation uses a high-cardinality adaptation of the BFCL V4 Non-Live Multiple split (Patil et al., 2025). Every query is evaluated against the same catalogue of 443 unique function definitions extracted from the Multiple split. As a controlled baseline constraint, duplicate function names are deduplicated by retaining only the first-seen schema. Each model is evaluated on all 200 queries. This “shared catalogue” design differs from an evaluation in which each query is paired with a smaller instance-specific distractor set. Its purpose is to

create a common, deliberately large action space in which full injection and retrieval-mediated selection can be compared directly. The resulting scores are BFCL-derived, but they are not official Gorilla leaderboard scores because the catalogue construction, agent wrapper, grader and serving environment are specific to this experiment.

Three modes were evaluated: full injection (all 443 tool definitions are bound to the model), sparse retrieval (BM25-based keyword search + top-k tool definition extraction), and dense retrieval (semantic search implementation provided by ToolScope). The sparse and dense retrieval modes represent two implementations of the same broader architectural idea: reduce the candidate action space to a predefined size of *k* before model inference. The main setting used in this study is *k=10*, with nested ablations at *k=5* and *k=20*. To isolate schema retrieval and binding capabilities from multi-turn state drift or execution side-effects, we formulate our evaluation as a stateless single-turn tool discovery protocol. Under this protocol, each query consists of a single, isolated agentic turn: the candidate tool set is dynamically bound, the model is invoked through its native tool-calling interface, and the returned function call is parsed and validated via abstract syntax tree (AST) grading without executing the underlying tool.

Five open-weight models were evaluated: Llama 3.2 3B Instruct, Qwen2.5 7B Instruct, Llama 3.1 8B Instruct, Qwen3 32B, and Llama 3.3 70B Instruct. All models were served sequentially through llama.cpp on an NVIDIA DGX Spark with 128 GB of unified memory using an OpenAI-compatible endpoint.

Context size is reported using two distinct measures. The approximately 60,051-token figure used for context compression calculations is not a tokenizer-specific prompt length, but a model-independent estimate computed from the JSON representation of the 443 tool definitions at approximately four characters per token. The evaluation also recorded server-reported effective prompt lengths after model-specific templating and context handling. Under full injection, these averaged 16,424 tokens for Llama 3.2 3B, 28,363 for Qwen2.5 7B, 16,424 for Llama 3.1 8B, 61,115 for Qwen3 32B and 81,960 for Llama 3.3 70B. The configured context windows were 32,768, 32,768, 32,768, 65,536 and 131,072 tokens respectively.

Our experimental harness performed no truncation and always submitted the full 443-tool catalogue. However, for the 3B, 7B and 8B runs llama.cpp internally applied context shifting, removing blocks from the prompt when an initial tokenized request exceeded the available context. Consequently, the effective prompt seen by these models could contain only part of the submitted catalogue. The 32B and 70B baselines used enlarged contexts within which the observed full-injection prompts fit.

5.2 Metrics and Analysis

To make failures transparent and easily identifiable, we evaluate the tool-use pipeline at three distinct stages: retrieval, selection and tool calling. At the retrieval stage, *Recall@k* measures whether a ground-truth tool appears in the selected candidate set, while *NDCG@k* additionally captures ranking quality. Context efficiency is measured from the serialized tool definitions using a standardized reference tokenizer to compute approximate token counts across configurations. At the selection stage, *tool name accuracy* determines whether the model emits a ground-truth function name, separating choosing the appropriate capability from constructing its arguments. At the tool calling stage, *AST accuracy* compares the generated call against BFCL's possible-answer representation and measures argument correctness. This metric captures both the tool selection and the tool argument estimation quality.

The harness also records diagnostic failure labels, including retrieval misses, wrong tool selections, bad arguments, parse failures, missing calls and condition-level inference failures. We use these labels selectively in the analysis to explain the dominant failure modes observed in particular model and retrieval regimes. For the dense retrieval comparison, changes in name accuracy relative to full injection are evaluated with exact two-sided McNemar tests over the same 200 queries. Conditions that encounter inference failures are scored fail-closed while the other conditions for the query remain evaluable.

6. Results and Discussion

6.1 Selection over injection Across Model Scales

Table 1: Tool name accuracy at k=10

Model	Full injection	Sparse retrieval	Dense retrieval	Dense Δ
Llama 3.2 3B	2.5%	85.5%	84.5%	+82.0 pp
Llama 3.1 8B	6.0%	91.5%	92.0%	+86.0 pp
Qwen2.5 7B	40.0%	86.0%	87.0%	+47.0 pp
Qwen3 32B	72.5%	89.0%	88.0%	+15.5 pp
Llama 3.3 70B	79.0%	90.5%	91.0%	+12.0 pp

Selection improves downstream accuracy for every evaluated model, although the magnitude varies considerably with full-catalogue performance. The effect broadly decreases as a model's full-catalogue handling becomes stronger, but it does not disappear when full injection is already usable. Qwen3 32B still gains 15.5 percentage points and Llama 3.3 70B gains 12.0 points.

The form of the baseline failure differs across models. Llama 3.2 3B and Llama 3.1 8B exhibit 183 and 161 parse failures respectively under full injection, while Qwen2.5 7B makes 110 wrong-tool selections. These three models were configured with 32,768-token contexts, and oversized full-catalogue requests could be truncated by llama.cpp before inference. Their large retrieval gains therefore reflect the evaluated deployment configuration, combining candidate-space reduction with relief from context-capacity pressure, rather than a controlled comparison in which all 443 tools are fully visible. By contrast, the 32B and 70B full-injection prompts fit within their enlarged contexts, yet dense retrieval still improves tool-name accuracy by 15.5 and 12.0 percentage points respectively.

The result also empirically demonstrates the context-availability paradox. The full catalogue has a model-independent serialized-schema estimate of approximately 60,051 token equivalents, compared with approximately 1,362 for dense retrieval at k=10. Selection therefore reduces tool schema model context by approximately 97.7% while improving name accuracy rather than trading accuracy for compression. While absolute token counts vary slightly across Llama and Qwen tokenizers, the relative compression ratio remains consistently within 97.5%–98.0%.

6.2 Retrieval Method and k-ablation

Table 2: Retrieval quality and context size

Retriever	k	Recall@k	NDCG@k	Mean tool tokens
BM25	5	95.0%	0.874	699
BM25	10	97.0%	0.881	1,401
BM25	20	99.0%	0.886	2,789
Dense retrieval	5	96.0%	0.877	683
Dense retrieval	10	98.5%	0.885	1,362
Dense retrieval	20	99.5%	0.888	2,700

Dense retrieval maintains a modest advantage in raw retrieval quality: at k=10, recall reaches 98.5% compared with 97.0% for BM25, and NDCG is 0.885 compared with 0.881. Downstream model performance remains largely equivalent across both retrieval methods within this benchmark.

However, this apparent retriever equivalence must be qualified in light of dataset characteristics. BM25 achieves highly competitive recall on BFCL primarily because the benchmark queries exhibit strong lexical alignment with ground-truth function names and parameter descriptions. While sparse keyword matching suffices for lexically transparent benchmarks, real-world enterprise AgentOps environments present distinct challenges like vocabulary mismatches, ambiguous natural language, and multi-turn query drift. As a result, relying solely on lexical retrieval can easily lead to failures in these environments, highlighting the potential need for hybrid retrieval architectures that combine sparse keyword indexing with dense semantic routing to ensure reliable tool discovery across varied query formats. This should be the subject of a separate analysis and the results shared in this paper should be reviewed in the context of our benchmark's specific lexical structure.

The k-ablation also shows that increasing the shortlist does not monotonically improve model selection. Retrieval recall rises from 95–96% at k=5 to 99–99.5% at k=20, but name accuracy sometimes remains unchanged or declines. A larger shortlist can recover a missed ground-truth tool while simultaneously reintroducing more confusable alternatives. The practical tradeoff is therefore a balance between coverage and shortlist purity.

6.3 Tool Calling Errors After Retrieval

High retrieval recall does not imply that the correct tool is selected by the model and correctly invoked with the right parameters. At dense k=10, the AST accuracy values as reported in Table 3 below remain far below the corresponding tool name accuracies.

Table 3: AST accuracy at k=10

Model	Full injection	Sparse retrieval	Dense retrieval
Llama 3.2 3B	2.0%	47.0%	46.5%
Llama 3.1 8B	3.5%	50.5%	52.0%
Qwen2.5 7B	23.5%	53.5%	53.5%
Qwen3 32B	45.0%	55.5%	55.0%
Llama 3.3 70B	46.5%	60.0%	61.0%

It can be observed that retrieval-mediated selection improves AST accuracy across all five models, particularly where full-injection tool selection is weak. However, the gap between tool-name accuracy and AST accuracy remains substantial. Under dense retrieval at k=10, name accuracy ranges from 84.5% to 92.0%, whereas AST accuracy ranges from only 46.5% to 61.0%. Decomposing this post-retrieval error gap reveals two distinct failure modes: candidate confusion (where the model selects an incorrect tool despite the ground-truth tool being present in the top-10 shortlist) and parameter generation errors (where the correct tool name is selected, but the invocation fails due to invalid argument types, missing required keys, or malformed syntax). While candidate confusion accounts for a smaller fraction of post-retrieval failures, parameter syntax and type errors constitute the majority of the remaining degradation—failing approximately 33–45% of total calls even after successful tool name selection. Thus, while retrieval successfully resolves the tool discovery bottleneck, precise parameter generation remains the primary downstream reasoning challenge for smaller-scale and open-weight models.

In many post-retrieval errors observed during the evaluation, the correct tool was already present in the shortlist but the model chose a closely related alternative. For Qwen3 32B, nine of the ten cases in which full injection selects the correct tool but dense retrieval misses still contained the ground-truth tool in the retrieved set. For Llama 3.3 70B, this was true for all four such cases. The dominant post-retrieval challenge is therefore increasingly one of disambiguation among plausible neighboring tools rather than retrieval coverage. From an AgentOps perspective, this decomposition is useful because it localizes failure: a retrieval miss points to the indexing or ranking stage, a wrong-tool choice with the correct tool already present points to selection, and a

correct name with invalid arguments points to calling. This is the practical value of evaluating retrieval, selection and argument construction separately rather than collapsing them into a single metric.

7. Limitations

The study has several limitations. First, for the three models configured with 32k contexts, oversized full-catalogue requests could be truncated before inference as explained in Section 5. Their full-injection results therefore conflate tool-selection difficulty with context-capacity effects. Second, the evaluation relies on a stateless, single-turn protocol using static AST grading without executing external tool code. While this single-turn AST grading serves as a controlled upper-bound for schema-binding accuracy, it does not assess multi-turn trajectory planning, state recovery, or real-world API failures. Third, the dense retrieval mode is evaluated using standard default settings and relies on a single embedding model architecture. Finally, schema versioning and dynamic registry mutations were held constant during evaluation to isolate the impact of raw cardinality. In production environments, tool schemas and parameters may evolve dynamically; positioning and evaluating tool retrieval under conditions of active schema drift remain important targets for future work.

8. Conclusion

High-cardinality tool use is not only a question of giving an agent access to more capabilities, but also a question of deciding which capabilities should be exposed at each inference step. The evaluation in this paper shows that retrieval-mediated tool selection can substantially reduce tool-schema context while improving downstream tool-name accuracy across models ranging from 3B to 70B parameters. The effect is the largest where full-catalogue handling is the weakest, but remains significant even for models that already handle the complete catalogue comparatively well. Once retrieval recall is high, the primary operational challenge transitions from coverage to disambiguating similar tools and constructing correct execution arguments. Pre-inference tool selection should therefore be treated as a distinct, observable stage in an agent architecture: it eliminates a critical high-cardinality prompt-bloat bottleneck, while delegating downstream tool-calling fidelity to focused model reasoning. Our planned future work will extend this evaluation framework beyond the stateless discovery benchmark to an execution-backed, multi-turn protocol.

Ethics Declaration: Ethical clearance was not required for this study because the reported experiments evaluate software models using benchmark function-calling data and do not involve human participants or personal data.

AI Declaration: Generative AI tools were used to assist with language editing, restructuring and drafting of portions of the manuscript. The authors reviewed and revised the generated material and remain responsible for the paper's content, analysis and conclusions.

References

Anantha, R., Bandyopadhyay, B., Kashi, A., Mahinder, S., Hill, A.W. and Chappidi, S. (2023). *ProTIP: Progressive Tool Retrieval Improves Planning*. arXiv:2312.10332.

Chen, Z., Liu, K., Wang, Q., Zhang, W., Liu, J., Lin, D., Chen, K. and Zhao, F. (2024). "Agent-FLAN: Designing Data and Methods of Effective Agent Tuning for Large Language Models", *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 9354–9366. doi:10.18653/v1/2024.findings-acl.557.

Du, Y., Wei, F. and Zhang, H. (2024). "AnyTool: Self-Reflective, Hierarchical Agents for Large-Scale API Calls", *Proceedings of the 41st International Conference on Machine Learning*.

Fei, X., Zheng, X. and Feng, H. (2025). *MCP-Zero: Proactive Toolchain Construction for LLM Agents from Scratch*. arXiv:2506.01056.

Gan, T. and Sun, Q. (2025). *RAG-MCP: Mitigating Prompt Bloat in LLM Tool Selection via Retrieval-Augmented Generation*. arXiv:2505.03275.

Gao, S., Dwivedi-Yu, J., Yu, P., Tan, X.E., Pasunuru, R., Golovneva, O., Sinha, K., Celikyilmaz, A., Bosselut, A. and Wang, T. (2025). "Efficient Tool Use with Chain-of-Abstraction Reasoning", *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 2727–2743.

Hao, S., Liu, T., Wang, Z. and Hu, Z. (2023). "ToolkenGPT: Augmenting Frozen Language Models with Massive Tools via Tool Embeddings", *Advances in Neural Information Processing Systems*.

Kachuee, M., Ahuja, S., Kumar, V., Xu, P. and Liu, X. (2025). "Improving Tool Retrieval by Leveraging Large Language Models for Query Generation", *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*.

Kim, S., Moon, S., Tabrizi, R., Lee, N., Mahoney, M.W., Keutzer, K. and Gholami, A. (2024). "An LLM Compiler for Parallel Function Calling", *Proceedings of the 41st International Conference on Machine Learning*, Vol. 235, pp. 24370–24391.

Lumer, E., Basavaraju, P.H., Mason, M., Burke, J.A. and Subbiah, V.K. (2025a). *Graph RAG-Tool Fusion*. arXiv:2502.07223.

Lumer, E., Subbiah, V.K., Burke, J.A., Basavaraju, P.H. and Huber, A. (2025b). "Toolshed: Scale Tool-Equipped Agents with Advanced RAG-Tool Fusion and Tool Knowledge Bases", *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025)*, Vol. 3, pp. 1180–1191. doi:10.5220/0013303000003890.

Moon, S., Jha, S., Erdogan, L.E., Kim, S., Lim, W., Keutzer, K. and Gholami, A. (2024). *Efficient and Scalable Estimation of Tool Representations in Vector Space*. arXiv:2409.02141.

Patil, S.G., Zhang, T., Wang, X. and Gonzalez, J.E. (2024). "Gorilla: Large Language Model Connected with Massive APIs", *Advances in Neural Information Processing Systems*, Vol. 37 (NeurIPS 2024). doi:10.52202/079017-4020.

Patil, S.G., Mao, H., Yan, F., Ji, C.C.-J., Suresh, V., Stoica, I. and Gonzalez, J.E. (2025). "The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models", *Proceedings of the 42nd International Conference on Machine Learning*, Vol. 267, pp. 48371–48392.

Qu, C., Dai, S., Wei, X., Cai, H., Wang, S., Yin, D., Xu, J. and Wen, J.-R. (2025). "From Exploration to Mastery: Enabling LLMs to Master Tools via Self-Driven Interactions", *International Conference on Learning Representations (ICLR 2025)*.

Qu, C., Dai, S., Wei, X., Cai, H., Wang, S., Yin, D., Xu, J. and Wen, J.-R. (2024). "Towards Completeness-Oriented Tool Retrieval for Large Language Models", *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM 2024)*, pp. 1930–1940. doi:10.1145/3627673.3679847.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E**.**, Zettlemoyer, L., Cancedda, N. and Scialom, T. (2023). "Toolformer: Language Models Can Teach Themselves to Use Tools", *Advances in Neural Information Processing Systems*, Vol. 36.

Singh, S., Karatzas, A., Fore, M., Anagnostopoulos, I. and Stamoulis, D. (2024). *An LLM-Tool Compiler for Fused Parallel Function Calling*. arXiv:2405.17438.

Wang, R., Han, X., Ji, L., Wang, S., Baldwin, T. and Li, H. (2025). "ToolGen: Unified Tool Retrieval and Calling via Generation", *International Conference on Learning Representations (ICLR 2025)*.

Wu, M., Zhu, T., Han, H., Zhang, X., Shao, W. and Chen, W. (2025). *Chain-of-Tools: Utilizing Massive Unseen Tools in the CoT Reasoning of Frozen Language Models*. arXiv:2503.16779.

Xu, Y., Feng, Y., Mu, H., Hou, Y., Li, Y., Wang, X., Zhong, W., Li, Z., Tu, D., Zhu, Q., Zhang, M. and Che, W. (2024). "Concise and Precise Context Compression for Tool-Using Language Models", *Findings of the Association for Computational Linguistics: ACL 2024*.

Yakovlev, K., Nikolenko, S. and Bout, A. (2024). "Toolken+: Improving LLM Tool Usage with Reranking and a Reject Option", *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 5967–5974. doi:10.18653/v1/2024.findings-emnlp.345.

Yang, X., Li, L., Zhou, H., Zhu, T., Qu, X., Fan, Y., Wei, Q., Ye, R., Kang, L., Qin, Y., Kou, Z., Liu, D., Li, Q., Ding, N., Chen, S. and Shao, J. (2026). *Toward Efficient Agents: Memory, Tool Learning, and Planning*. arXiv:2601.14192.