ToolSEE: Agent Tool Search Engine for Efficient and Scalable Tool Discovery Using Retrieval

Praneeth Vadlapati *

Posted Date: 19 December 2025

doi: 10.20944/preprints202512.1744.v1

Keywords: AI agents; Artificial Intelligence; Large Language Models; LLMs; LLM agents; context engineering; decision support; tool retrieval; tool selection; token efficiency

 doi:10.20944/preprints202512.1744.v1

Article

ToolSEE: Agent Tool Search Engine for Efficient and Scalable Tool Discovery Using Retrieval

Praneeth Vadlapati

Independent researcher; praneethv@arizona.edu

Abstract

Tool-augmented conversational agents increasingly operate over large and evolving tool catalogs, rendering the enumeration of all tool descriptions within the model context impractical due to excessive latency, token consumption, and an expanded action space that undermines selection reliability. This paper introduces ToolSEE, a lightweight retrieval layer that decouples tool discovery from model prompting by indexing structured tool metadata and dense semantic embeddings to retrieve a compact, query-conditioned subset of relevant tools. ToolSEE further supports controlled dynamic expansion during agent execution by allowing agents to issue targeted search queries when additional tools are required, thereby maintaining bounded context size while preserving flexibility. This paper describes the system architecture and retrieval mechanisms and empirically evaluates ToolSEE across multiple agent workloads. Results demonstrate that retrieval-based tool selection preserves tool-calling correctness relative to in-context enumeration baselines, while substantially reducing prompt token usage and input latency, highlighting ToolSEE's effectiveness as a scalable and production-ready solution for extensive tool catalogs. The source code is available at github.com/Pro-GenAI/Tool-SEE.

Keywords: AI agents; Artificial Intelligence; Large Language Models; LLMs; LLM agents; context engineering; decision support; tool retrieval; tool selection; token efficiency

I. Introduction

Agents powered by large language models (LLMs) increasingly rely on external tools to perform specialized tasks, such as file search, web queries, code execution, and domain-specific API calls [1,2]. For modest tool inventories, a model can select an appropriate tool from an enumerated prompt describing each capability. As tool catalogs scale to hundreds or thousands of entries, however, inclusion of all tool descriptions within the model context becomes impractical: prompt length increases, latency and monetary cost rise, and the agent must reason over a larger, noisier action space, which can reduce selection accuracy and increase the incidence of hallucinated or spurious tool invocations [3–5]. These considerations are particularly consequential in production settings, where latency and API expenditures directly impact user experience and operational costs.

A. Limitations of Existing Approaches

Existing approaches to tool-enabled agents generally fall into two categories: explicit prompt enumeration and implicit encoding of tool knowledge within model weights [1,3]. Prompt enumeration provides transparency but exhibits linear growth in context size and token costs as the number of tools increases [6,7]. Implicit encoding avoids prompt bloat but requires the model to internalize tool semantics and invocation conventions, thereby reducing adaptability and necessitating costly retraining to accommodate tool updates [1]. Both approaches may reduce reliability: extensive enumerations introduce distractor tools that the model may select erroneously, and implicit strategies can cause brittle behavior when the model encounters unfamiliar or ambiguously described tools [2,8]. These limitations motivate the development of a retrieval-based

 doi:10.20944/preprints202512.1744.v1

intermediary that preserves transparency and updatability while controlling the context size. A further practical disadvantage of prompt-enumeration approaches is their impact on model inference costs, as increasing the context window amplifies inference memory requirements because attention states and intermediate activations must be retained for more extended sequences. This growth complicates deployment, resulting in increased latency and resource consumption. Recent work has documented this scaling and proposed mitigations, including IO-aware attention kernels and long-context attention mechanisms [9,10].

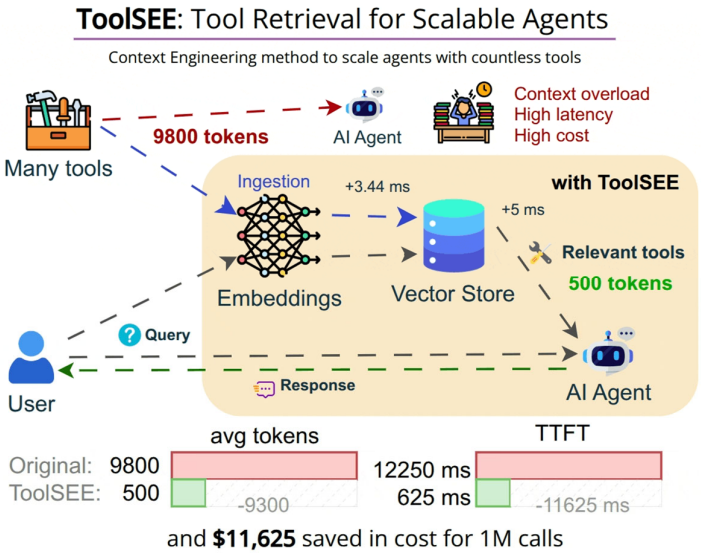

Figure 1. Graphical abstract of ToolSEE.

B. Proposed Approach

We propose ToolSEE, a retrieval layer that mediates between an extensive tool catalog and an LLM-based agent. ToolSEE indexes tool metadata, such as tool names and descriptions, and stores vector embeddings, allowing for the retrieval and presentation to the agent of a compact set of relevant tools for a given user query. Conditioning the prompt on a small subset of tools reduces token consumption and may reduce erroneous tool selection by narrowing the action space. ToolSEE also supports controlled dynamic expansion in which the agent can invoke a search interface through a function called "search_tools" with a refined query during execution to obtain additional candidates. Under this design, the number of tool descriptions incorporated into the model context remains bounded through top-k retrieval, even as the tool catalog scales.

C. Applications

ToolSEE is designed for settings where assistants must operate over extensive and frequently changing tool inventories. Examples include internal developer platforms that expose extensive collections of utilities while maintaining interactive latency, evaluation pipelines that measure tool-selection behavior at scale without prohibitive prompt overhead, and agents operating over private catalogs whose metadata changes more rapidly than model weights. ToolSEE also supports interaction patterns in which an agent starts from a conservative top-k set and expands the candidate set only when needed, which can reduce time-to-first-token (TTFT) in typical cases.

D. Related Work

ToolSEE's design draws upon prior work in neural retrieval, vector databases, and routing techniques for modular systems. Research on retrieval-augmented generation has demonstrated how external indices and embeddings can extend the practical context available to language models

 doi:10.20944/preprints202512.1744.v1

[3,4,11]. Vector search libraries and approximate nearest neighbor methods provide the engineering foundation for efficient similarity search over embeddings [7,12]. Prior work on model tool use and modular agents has documented the benefits of exposing specialized capabilities to general-purpose models [1,2]. The literature on prompt engineering and instruction tuning has also articulated trade-offs between explicit context inclusion and implicit model knowledge. Recent work on scaling tool-equipped agents and RAG–tool fusion highlights practical considerations for selection, safety, and evaluation [13]. ToolSEE synthesizes these strands into a pragmatic retrieval layer tailored for tool selection in interactive agents.

II. Methods

A. System Architecture

ToolSEE is implemented as a lightweight middleware component positioned between an agent orchestration loop and a catalog of tool definitions. The system maintains a ToolMemory component that stores tool metadata alongside vector embeddings. The retrieval-oriented pattern and API design are informed by prior retrieval-augmented approaches and pretraining strategies that integrate retrieval into model workflows [2–4]. For each user query, the agent invokes the ToolSEE retrieval API. ToolSEE then computes a query embedding and executes a nearest-neighbor search in the embedding space to rank candidates by semantic relevance. A compact top-k set of tools, together with metadata and usage templates, is returned and incorporated into the agent's prompt. Subsequently, the agent may invoke a selected tool or, if additional capabilities are required, call the search_tools endpoint with a refined query to obtain further candidates.

B. Tool Ingestion and Indexing

Tool ingestion begins with a structured representation for each tool comprising a human-readable name, a concise description, optional tags and metadata fields, such as input/output schemas and example invocations, and an associated function or API handle. Tool metadata is converted into dense vector embeddings via an embeddings provider compatible with common LLM ecosystems, such as sentence encoders like Sentence-BERT [6]. These embeddings are stored with tool records in ToolMemory, which supports efficient cosine-similarity search. For index construction and nearest-neighbor retrieval, standard vector search backends, such as FAISS, and graph-based ANN algorithms, such as HNSW, are appropriate choices depending on scale and latency requirements [7,12]. ToolSEE supports both in-memory indexes for small to medium catalogs and external vector stores for larger deployments. The implementation abstracts storage, allowing practitioners to select an embedding model and index infrastructure that are appropriate to their latency and capacity requirements.

C. Retrieval and Ranking

At query time, ToolSEE generates an embedding of the user query and retrieves the top-k nearest tool embeddings by cosine similarity. Retrieved candidates undergo post-processing using heuristics such as metadata filters that exclude tools flagged as private or contextually inappropriate, score thresholds to exclude low-confidence matches, and optional re-ranking that combines embedding similarity with lexical or tag-based signals. Standard coarse-to-fine retrieval and re-ranking patterns are applicable in this stage, including dense bi-encoders and DPR [5], as well as late-interaction techniques such as ColBERT [14] and cross-encoder rescoring for high-precision results [15]. The returned candidate set is designed to optimize both precision and diversity, ensuring that the agent receives highly relevant tools. In contrast, diversity exposes the agent to complementary capabilities when multiple tools could satisfy the query. The retrieval pipeline is intentionally lightweight to maintain low selection latency; more computationally intensive re-ranking strategies may be applied when application constraints permit.

 doi:10.20944/preprints202512.1744.v1

D. Dynamic Expansion Pattern

ToolSEE implements a two-phase interaction pattern to balance compactness and flexibility. Initially, the agent is provided with a small top-k selection intended to minimize token and latency overhead. If the agent's internal reasoning determines that the provided tools are insufficient, following an unsuccessful attempt or upon receipt of additional contextual information, the agent may trigger the "search_tools" facility with a focused query that articulates the refined requirement. ToolSEE returns additional candidates, which the agent may incorporate into a subsequent prompt. This dynamic expansion reduces unnecessary context in typical interactions while preserving the capacity to discover specialized tools as required. Such multi-stage and iterative retrieval patterns align with RAG-style systems and recent tool-fusion proposals that enable controlled expansion of context on demand [3,4,13].

E. Implementation Details

The reference implementation utilizes off-the-shelf embedding models and an in-memory index, which is exposed via the ToolMemory API, as documented in the repository. Tools are represented as tuples comprising an identifier and a metadata dictionary; ingestion routines embed textual descriptions and store embeddings alongside metadata. The select_tools_for_query function accepts a top_k parameter and returns a list of tool records that include a _score, representing cosine similarity, and the original description. The agent orchestration demonstrates how to incorporate selected tools into model prompts and how to expose a search_tools facility for dynamic expansion. Benchmarks were computed by measuring the correctness of tool selection against labeled queries, end-to-end latency for selection and TTFT, and token consumption associated with attaching tool descriptions to model inputs.

F. Evaluation

The evaluation assesses ToolSEE along three principal dimensions: tool selection correctness, latency, and token efficiency. Experiments were designed to quantify the extent to which retrieval-based selection preserves or improves selection accuracy relative to baseline strategies that either include the entire tool catalog in the prompt or present a single representative tool per type. Reported benchmarks follow the repository's evaluation scripts and include selection correctness, median selection latency, TTFT improvements, and token savings computed over representative datasets and simulated tool catalogs. The correctness of tool selection is measured using the DeepEval tool-correctness metric [16]. The experiments utilize the MetaTool toolset and dataset as reference tool inventories [17].

III. Results

A. Selection Accuracy

In benchmark evaluations designed to represent typical multi-tool agent workloads, the selective retrieval strategy implemented by ToolSEE produced a measured tool correctness of 88.33% when a limited set of relevant tools was presented per query. By contrast, a single-tool baseline evaluated on a matched sample exhibited 81.0% correctness. These findings indicate that retrieval-based restriction of the agent's action space increases the probability that an appropriate tool is available to the agent at decision time, thereby enhancing task-level tool-calling accuracy. Statistical hypothesis testing and confidence interval estimation were not performed for these preliminary measurements; subsequent work should quantify uncertainty and assess robustness across larger and more diverse datasets.

 doi:10.20944/preprints202512.1744.v1

B. Latency and Token Savings

Selection latency for the in-memory retrieval configuration was observed to be low, with median query times in the sub-10 millisecond range. The end-to-end time-to-first-tool (TTFT) was substantially reduced in the experiments. The reported median TTFT decreased from 6196.5 ms when the complete tool catalog was provided to the model to 436.0 ms when five selected tools were presented, corresponding to a median reduction of 5760.5 ms. Input token footprints were reduced when only top-k tool descriptions were attached to prompts, yielding substantial reductions in tokens consumed per interaction and concomitant decreases in input-side API costs. The magnitude of latency and cost savings is deployment-dependent, varying with embedding provider latency, index configuration, and the downstream LLM pricing model.

IV. Discussion

The experimental results indicate that a compact, retrieval-driven selection layer can reconcile the competing requirements of scalability, reliability, and responsiveness in multi-tool agent architectures. ToolSEE leverages dense vector representations to capture semantic correspondences between user queries and tool descriptions, employing lightweight post-retrieval filtering and optional re-ranking to produce compact, high-precision candidate sets. Nonetheless, several practical considerations and limitations warrant attention. First, the quality of embeddings materially affects ranking fidelity; domain-specific tools may require specialized embedding models or enriched metadata, such as usage examples or input/output schemas, to ensure robust matching. Second, index design and location influence latency: in-memory indexes provide the lowest query latencies for moderate catalog sizes, whereas externally hosted vector stores may be required for scalability at the expense of higher network latencies. Third, the current evaluation suite focuses on selection correctness and latency; downstream task-level effects, long-term user satisfaction, and failure-mode analyses remain to be systematically studied. Finally, the system inherits broader concerns associated with retrieval-augmented pipelines, such as potential biases in textual descriptions and the need for access controls and provenance tracking when tools can perform privileged actions.

V. Conclusions

This work presents ToolSEE, a retrieval-oriented layer for tool selection that enables multi-tool agents to operate effectively over extensive tool catalogs, while preserving responsiveness and reducing token costs. Empirical evidence from repository evaluations demonstrates that retrieval-based selection can maintain or improve tool-calling correctness and substantially lower time-to-first-tool and input token consumption. ToolSEE's architecture supports dynamic expansion to accommodate exploratory tool discovery without incurring constant prompt bloat. Future research directions include integrating stronger re-ranking components, such as cross-encoder rescoring, to enhance the model's performance further. The deployment of ToolSEE in production environments should be accompanied by careful tuning of embedding models, index topology, and retrieval thresholds to balance recall and precision with latency.

References

1. T. Schick et al., "Toolformer: language models can teach themselves to use tools," in Proceedings of the 37th International Conference on Neural Information Processing Systems, in NIPS '23. Red Hook, NY, USA: Curran Associates Inc., 2023.
2. S. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models," Mar. 10, 2023, arXiv: arXiv:2210.03629. doi: 10.48550/arXiv.2210.03629.
3. P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in Proceedings of the 34th International Conference on Neural Information Processing Systems, in NIPS '20. Red Hook, NY, USA: Curran Associates Inc., 2020.

 doi:10.20944/preprints202512.1744.v1

4. K. Guu et al., "REALM: retrieval-augmented language model pre-training," in Proceedings of the 37th International Conference on Machine Learning, in ICML'20. JMLR.org, 2020.
5. V. Karpukhin et al., "Dense Passage Retrieval for Open-Domain Question Answering," in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), B. Webber, T. Cohn, Y. He, and Y. Liu, Eds., Online: Association for Computational Linguistics, Nov. 2020, pp. 6769–6781. doi: 10.18653/v1/2020.emnlp-main.550.
6. N. Reimers et al., "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), K. Inui, J. Jiang, V. Ng, and X. Wan, Eds., Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3982–3992. doi: 10.18653/v1/D19-1410.
7. J. Johnson, M. Douze, H. Jégou, J. Johnson, M. Douze, and H. Jégou, "Billion-Scale Similarity Search with GPUs," IEEE Transactions on Big Data, vol. 7, no. 3, pp. 535–547, 2021, doi: 10.1109/TBDATA.2019.2921572.
8. J. Shi et al., "Prompt Injection Attack to Tool Selection in LLM Agents," Aug. 24, 2025, arXiv: arXiv:2504.19793. doi: 10.48550/arXiv.2504.19793.
9. T. Dao et al., "FLASHATTENTION: fast and memory-efficient exact attention with IO-awareness," in Proceedings of the 36th International Conference on Neural Information Processing Systems, in NIPS '22. Red Hook, NY, USA: Curran Associates Inc., 2022.
10. T. Munkhdalai, M. Faruqui, S. Gopal, T. Munkhdalai, M. Faruqui, and S. Gopal, "Leave No Context Behind: Efficient Infinite Context Transformers with Infini-attention," Aug. 09, 2024, arXiv: arXiv:2404.07143. doi: 10.48550/arXiv.2404.07143.
11. [11]A. J. Oche et al., "A Systematic Review of Key Retrieval-Augmented Generation (RAG) Systems: Progress, Gaps, and Future Directions," July 25, 2025, arXiv: arXiv:2507.18910. doi: 10.48550/arXiv.2507.18910.
12. Y. A. Malkov, D. A. Yashunin, Y. A. Malkov, and D. A. Yashunin, "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs," IEEE Trans. Pattern Anal. Mach. Intell., vol. 42, no. 4, pp. 824–836, Apr. 2020, doi: 10.1109/TPAMI.2018.2889473.
13. E. Lumer et al., "Toolshed: Scale Tool-Equipped Agents with Advanced RAG-Tool Fusion and Tool Knowledge Bases," Oct. 22, 2024, arXiv: arXiv:2410.14594. doi: 10.48550/arXiv.2410.14594.
14. O. Khattab, M. Zaharia, O. Khattab, and M. Zaharia, "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT," in Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, in SIGIR '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 39–48. doi: 10.1145/3397271.3401075.
15. R. Nogueira, K. Cho, R. Nogueira, and K. Cho, "Passage Re-ranking with BERT," Apr. 14, 2020, arXiv: arXiv:1901.04085. doi: 10.48550/arXiv.1901.04085.
16. DeepEval, "Tool Correctness," DeepEval. [Online]. Available: https://deepeval.com/docs/metrics-tool-correctness
17. Y. Huang et al., "MetaTool Benchmark for Large Language Models: Deciding Whether to Use Tools and Which to Use," in The Twelfth International Conference on Learning Representations, 2024. [Online]. Available: https://openreview.net/forum?id=R0c2qtalgG