

Toolshed: Scale Tool-Equipped Agents with Advanced RAG-Tool Fusion and Tool Knowledge Bases

Elias Lumer^a, Vamse Kumar Subbiah, James A. Burke,
Pradeep Honaganahalli Basavaraju and Austin Huber
PricewaterhouseCoopers, U.S.A.

Keywords: Tool Learning, Tool Selection, Function Calling, Retrieval-Augmented-Generation, Tool Retrieval, Knowledge Retrieval, AI Agents, Large Language-Models.

Abstract: Recent advancements in tool-equipped agents (LLMs) have enabled complex tasks like secure database interactions and code development. However, scaling tool capacity beyond agent reasoning or model limits remains a challenge. In this paper, we address these challenges by introducing Toolshed Knowledge Bases, a tool knowledge base (vector database) designed to store enhanced tool representations and optimize tool selection for large-scale tool-equipped agents. Additionally, we propose Advanced RAG-Tool Fusion, a novel ensemble of tool-applied advanced retrieval-augmented generation (RAG) techniques across the pre-retrieval, intra-retrieval, and post-retrieval phases, without requiring fine-tuning. During pre-retrieval, tool documents are enhanced with key information and stored in the Toolshed Knowledge Base. Intra-retrieval focuses on query planning and transformation to increase retrieval accuracy. Post-retrieval refines the retrieved tool documents, enables self-reflection, and equips the tools to the agent. Furthermore, by varying both the total number of tools (*tool-M*) an agent has access to and the tool selection threshold (*top-k*), we address trade-offs between retrieval accuracy, agent performance, and token cost. Our approach achieves 46%, 56%, and 47% absolute improvements on the ToolE single-tool, ToolE multi-tool and Seal-Tools benchmarks, respectively (recall@5).

1 INTRODUCTION


The latest advancements in Large Language Models (LLMs) have enabled LLM agents to autonomously handle tasks through external tools or APIs. With tool calling, or function calling, these agents can execute complex actions such as interacting with data APIs, collaborating on code development, and performing domain-specific question answering. Current models handle up to 128 tool function definitions, though this limit presents challenges for scaling agent capabilities in production, where hundreds or thousands of tools may be required (Google Cloud, 2024).

Despite advancements in retriever-based tool selection systems, a significant gap remains compared to the advanced retrieval-augmented generation (RAG) community (Gao et al., 2024). Current tool retrievers rely on only 1–2 key tool components (tool name and description) to embed as vector representations, whereas advanced RAG methods append document summaries, questions, and key metadata. Additionally, inference-time solutions such as query planning, expansion, and reranking, remain unexplored.

In this paper, we introduce *Toolshed Knowledge Bases*, a knowledge base optimized for storing and retrieving tools for scalable tool-equipped agents, through enhancing tool documents with 5 tool components (Fig. 1). This approach also addresses and optimizes the trade-off of how the tool definition count (*tool-M*) and tool selection threshold (*top-k*) affect retrieval accuracy, agent performance, and cost.

We also introduce *Advanced RAG-Tool Fusion*, a modular ensemble of advanced RAG patterns applied to tool selection and planning without requiring model fine-tuning. They include 1) pre-retrieval techniques (optimizing tool document vector embeddings), 2) intra-retrieval strategies (query transformations to retrieve relevant tools), and 3) post-retrieval techniques (reranking or self-correction) (Fig. 1). Our Advanced RAG-Tool Fusion significantly advances tool retrieval, achieving 46%, 56%, and 47% absolute improvements over BM25 on ToolE single-tool, ToolE multi-tool, and Seal-Tools benchmarks, while outperforming current SOTA retrievers (recall@5).

The paper is organized as follows: Section 2 reviews advanced RAG and tool learning, Section 3 outlines methods, Section 4 covers evaluations, Section 5 concludes, and Section 6 discusses limitations.

^a  <https://orcid.org/0009-0000-9180-3690>

2 BACKGROUND

2.1 Advanced RAG

Advanced Retrieval-Augmented Generation (RAG) builds on naive RAG by improving relevance and efficiency, addressing the challenge of selecting correct documents from large knowledge bases for LLM reasoning (Gao et al., 2024). Query rewriting and hypothetical document embeddings (HyDE) transform queries and improve out-of-domain understanding (Ma et al., 2023; Gao et al., 2022). Query expansion adds relevant terms to improve accuracy (Jagerman et al., 2023; Wang et al., 2023; Peng et al., 2024). Document chunk enhancements such as summaries and potential questions (reverse HyDe) align documents to queries in semantic space (Setty et al., 2024; Gao et al., 2024). Query decomposition and planning break complex questions into steps, improving multi-step reasoning (Tang and Yang, 2024; Trivedi et al., 2023; Yao et al., 2023; Khattab et al., 2023; Joshi et al., 2024; Xu et al., 2023; Zheng et al., 2024a). Reranking algorithms reorder results for contextual relevance (Raudaschl, 2023; Sun et al., 2023; Sawarkar et al., 2024). Corrective methods discard poor documents or retrieve new ones (Yan et al., 2024; Asai et al., 2023). Agentic RAG equips agents with a RAG tool, while Adaptive-RAG adjusts strategies to query complexity (Roucher, 2023; Jeong et al., 2024).

Our approach, Advanced RAG-Tool Fusion, applies the aforementioned document RAG techniques to tool selection and planning for agents.

2.2 Task Planning for Tools

Similar to advanced RAG, task planning is essential for breaking down complex queries into manageable sub-tasks for tool retrieval. Chain-of-Thought (Wei et al., 2023) and ReAct (Yao et al., 2023) laid the foundation by enabling agents to systematically decompose tasks. EasyTool, PLUTO, and Re-Invoke extend this by retrieving tools for each sub-task (Yuan et al., 2024b; Huang et al., 2024a; Chen et al., 2024). Advanced RAG-Tool Fusion leverages task planning while also employing other pre-, intra-, and post-retrieval strategies to enhance tool retrieval (Fig. 1).

2.3 Tool Selection or Retrieval

2.3.1 Retriever-Based Tool Selection

Tool retrieval is tightly coupled to task planning for tools. Early retriever-based methods, such as TF-IDF (Papineni, 2001) and BM25 (Robertson and

Zaragoza, 2009), rely on exact term matching to align queries with documents or tools, forming the baseline for modern retrieval methods. ProTIP (Anantha et al., 2023) uses a BERT-base-uncased retriever to match decomposed queries with tool descriptions. CRAFT (Yuan et al., 2024a) retrieves tools using SimCSE embeddings and aligns generated names and descriptions to queries with function names. ToolRerank (Zheng et al., 2024b) combines Adaptive Truncation and Hierarchy-Aware Reranking with dual-encoder and cross-encoder models for queries. Re-Invoke (Chen et al., 2024) and Tool2Vec (Moon et al., 2024) employ synthetic queries to enhance embeddings.

Our approach showcases zero-shot usage with out-of-the-box embedders from providers like OpenAI that avoid reliance on labeled data for training. While Re-Invoke and ToolRerank enhance vector representations with synthetic queries, Advanced RAG-Tool Fusion extends this by generating synthetic queries, key topics, themes, and intents, as well as detailed descriptions and schema parameters for embedding. Furthermore, Advanced RAG-Tool Fusion’s ensemble modules include query rewriting, decomposition into sub-tasks (user intents), and multi-query expansion or variation for each sub-task, capturing diverse descriptions to better match tools.

2.3.2 LLM-Based Tool Selection

Researchers have also used LLMs for tool retrieval alongside retriever-based methods. API-Bank (Li et al., 2023) uses a Plan+Retrieve+Call paradigm, similar to Agentic RAG (Roucher, 2023), but struggles with GPT-4’s limited use of the search API tool. AnyTool (Du et al., 2024) retrieves tools via a hierarchical API structure and incorporates self-reflection when retrieved tools are insufficient.

Our approach uses retriever-based tool selection with additional post-retrieval strategies. Unlike Agentic RAG (Roucher, 2023), we prompt LLMs to first decompose queries for tool retrieval and can self-correct itself if the retrieval does not yield all necessary tools. Furthermore, Advanced RAG-Tool Fusion can utilize metadata filtering or hierarchy groupings in the Toolshed Knowledge Base (Appendix C).

2.4 Tool Calling

Prior work focuses on tool invocation through parameter extraction and fine-tuning. GorillaLLM fine-tunes LLaMA-7B with retriever-aware training to access tool documentation (Patil et al., 2023), while ToolLLM is trained on 16,000 APIs from the ToolBench dataset using a retriever with DFSDT (Qin et al., 2023). ToolACE uses multi-agents to train

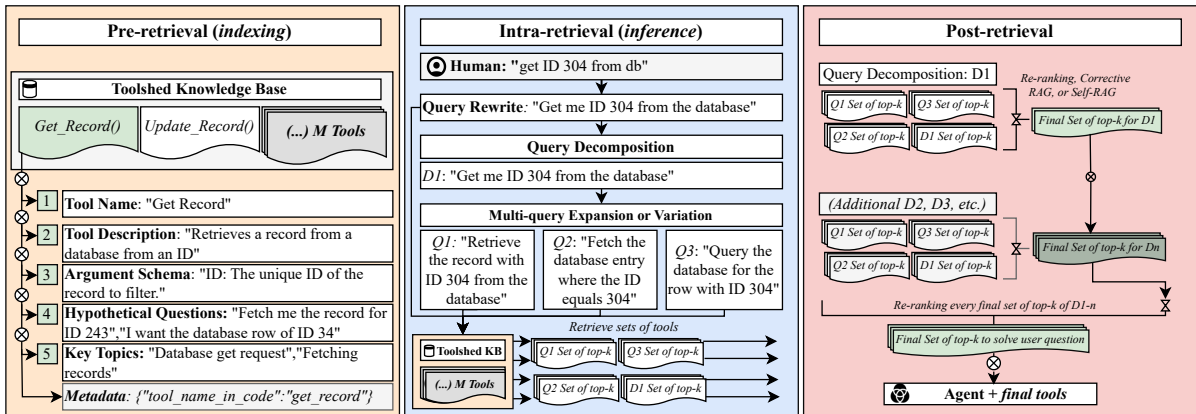


Figure 1: Advanced RAG-Tool Fusion within three phases. The pre-retrieval phase optimizes the tool document by appending a high-quality name, description, argument schema, hypothetical questions the tool can answer, related key topics, and metadata. The intra-retrieval and post-retrieval phases transform the user question into decomposed queries and expanded/varied queries to retrieve the top- k relevant tools from the Toolshed Knowledge Base and rerank the final tools to the agent.

LLaMA-8B (Liu et al., 2024), CITI uses MOLoRA (Hao et al., 2024b), and ToolkenGPT trains output tokens as tools (Hao et al., 2024a). Unlike prior work, we do not fine-tune LLMs for tool calling. We use function-calling LLMs (e.g., OpenAI, Anthropic) as a plug-and-play tool selection and planning solution, analyzing how tool count and top- k thresholds affect retrieval accuracy, agent performance, and cost.

3 METHOD

3.1 Tool Datasets

Notable datasets in the tool-calling community include ToolBench (Qin et al., 2023), ToolAlpaca (Tang et al., 2023), ToolE (Huang et al., 2024b), τ -bench (Yao et al., 2024), and Seal-Tools (Wu et al., 2024).

Upon reviewing these datasets and golden query-tool-parameter pairings, we identified several issues: unclear tool descriptions, missing parameter details, overlapping tools, and queries solvable by multiple similar tools. For this study, we selected Seal-Tools and ToolE as primary datasets. Seal-Tools ($\sim 3,500$) and ToolE (~ 200 tools) both contain high tool counts and minimal tool overlap to reduce retrieval errors.

3.2 LLM and Embedder Models

We use Azure OpenAI gpt-4o, 2024-05-13 (*final result*), gpt-4o (0613), gpt-35-turbo-16k (0613) for the LLM. For the embedders, we use Azure OpenAI text-embedding-3-large (*final result*), text-embedding-3-small, text-embedding-ada-002.

3.3 Toolshed Knowledge Bases

The *Toolshed Knowledge Base* serves as the vector database for storing tools that will be retrieved and equipped to a Single agent during inference. The strategy we use to represent tool documents stems from the pre-retrieval phase of Advanced RAG-Tool Fusion. Each tool’s vector representation combines up to 5 components: tool name, description, argument schema, synthetic queries, and key topics (Fig. 1). Since tool names cannot contain spaces when using OpenAI function definitions, we modify tool names by adding spaces (e.g., “GetRecord” becomes “Get Record”) to better represent them in the vector space, along with other features for enhanced retrieval. Each tool document also includes a metadata dictionary (“tool_name”) that links its unique name to its corresponding Python function. During inference, with the user query or decomposed query well-represented across the vector space, the top- k tools are retrieved and mapped to Python functions via the dictionary.

3.4 Advanced RAG-Tool Fusion

Having an agent retrieve and choose the correct tool(s) from a large collection of tools is fundamentally the same problem as document RAG (Gao et al., 2024; Kamradt, 2023). Thus, we can apply advanced RAG principles to the tool selection and planning field. While previous tool scaling work touched on few, if any, individual components of advanced RAG, our approach, Advanced RAG-Tool Fusion (Fig. 1), introduces an ensemble of state-of-the-art advanced RAG patterns applied to tool selection and planning in three phases (pre-retrieval, intra-retrieval, post-retrieval). See Appendix C for a detailed case study.

3.4.1 Pre-Retrieval (Indexing)

In the pre-retrieval or indexing phase of Advanced RAG-Tool Fusion, our goal is to enhance the quality of the tool document to be retrieved at a higher accuracy rate in the retrieval stage. Prior work demonstrates storing a tool’s name and description in a vector database does not yield optimal results (Chen et al., 2024; Moon et al., 2024). Our approach in Advanced RAG-Tool Fusion enhances the tool documents with 5 tool components and stores them in the Toolshed Knowledge Base: 1) tool name, 2) tool description, 3) argument schema (parameters & description), 4) hypothetical questions, and 5) key topics/intents (both questions/topics are synthetically generated). The corresponding advanced RAG methods involve appending document chunks with metadata, summaries, key topics, and hypothetical questions (Gao et al., 2024). See Appendix C, Fig. 16.

Recommendations for Pre-Retrieval. If opting to enhance the tool document with any 5 components:

- Tool functionality should not overlap, and tool names should be unique with an “embedded version” (spaces instead of underscores/dashes).
- Tool descriptions should be long, unique, and descriptive (e.g., explain when to use or not use it).
- Appending the tool’s argument schema can help retrieval. Ensure parameter names and descriptions are descriptive with no abbreviations.
- Appending synthetic questions can increase retrieval. Ensure questions are diverse, mirror future user questions, and utilize required and optional parameters in the question.
- Appending key topics/intents can increase retrieval. Ensure key topics are based on tool names, descriptions, and any synthetic questions.

3.4.2 Intra-Retrieval (Inference-Time)

In the retrieval stage of Advanced RAG-Tool Fusion, our goal is to retrieve the correct tool(s) needed for a user question. Previous work demonstrates that, because users often use shorthand, rely on pronouns instead of using the subject, or ask unclear queries, the user query may not capture the full intent of which tool should be retrieved (Ma et al., 2023). Therefore, our approach in Advanced RAG-Tool Fusion (Fig. 1) initially rewrites the query to fix any typos, errors, unclear pronouns (with available chat history), and overall conciseness. Additionally, a single user question may consist of multiple distinct steps requiring several tools. Directly embedding and querying

the entire question leads to poor retrieval results for tools (Tang and Yang, 2024). Advanced RAG-Tool Fusion then breaks the query into logical, independent steps, then rewrites/expands each step, capturing ways to solve the decomposed query. Finally, for each individual expanded query, we retrieve the initial *top-k* tools. The corresponding applied advanced RAG methods are query decomposition, query rewriting, multi-query expansion or variation, and step-back prompting (Joshi et al., 2024; Gao et al., 2024; Zheng et al., 2024a). See Appendix C, Fig. 18.

Recommendations for Intra-Retrieval. If opting to add query decomposition, transformation, or more:

- Query planning or decomposition helps retrieve different tools for a multi-hop query.
- If a user’s question uses shorthand, contains grammatical or spelling errors, or relies on pronouns, rewriting it initially (and utilizing previous chat history, if applicable) can improve retrieval.
- If there are multiple tools that can solve the same question, multi-query expansion or variation can help identify diverse pathways to solve the question by broadening the search scope of tools.
- Step-back query rewriting can help answer abstract questions in the planning module, but is an optional module in the framework.
- Test the retrieval accuracy for various *top-k* values (e.g., 1, 5, 10, 20... ≤ 128), and adjust threshold as needed based on the tool dataset complexity.

3.4.3 Post-Retrieval

In the post-retrieval phase of Advanced RAG-Tool Fusion, the goal is to finalize the list of tools for the agent. Reranking can occur in the query decomposition level, the multi-query expansion/variation level, and/or the individual query variation level. Some irrelevant tools may pass through the intra-retrieval phase because they are similar enough to be retrieved but not useful for answering the user’s question. To address this, we rerank and discard irrelevant tools, selecting only the *top-k* most relevant tools. While we use an LLM-based reranker (due to increased reasoning), an embedder cross-encoder reranker (Theja, 2023) can be used as well. Finally, using self-reflection, the agent can autonomously re-search the Toolshed Knowledge Base if it identifies missing tools. The associated advanced RAG patterns include reranking (Sun et al., 2023; Theja, 2023), corrective RAG (Yan et al., 2024), and self-RAG (Asai et al., 2023). See Appendix C, Fig. 19.

Table 1: Retriever results comparison on the Seal-Tools and ToolE datasets. We compare our Advanced RAG-Tool Fusion approach against a BM25 baseline (Robertson and Zaragoza, 2009), and the SOTA retrievers, Seal-Tool’s DPR (Wu et al., 2024) and Re-Invoke (Chen et al., 2024). The metrics are reported as recall@k, some k values were not calculated or clearly defined in the original papers, thus not reproduced for our approach. The best-performing method is highlighted in boldface.

Dataset	Retriever	Recall @ 1	Recall @ 5	Recall @ 10
Seal-Tools	BM25		0.410	0.550
	Seal-Tools DPR		0.480	0.680
	Advanced RAG-Tool Fusion with Toolshed Knowledge Base		0.876	0.965
ToolE - Single Tool	BM25	0.272	0.462	
	Re-Invoke’s	0.672	0.871	
	Advanced RAG-Tool Fusion with Toolshed Knowledge Base	0.726	0.928	
ToolE - Multi Tool	BM25	0.093	0.335	
	Re-Invoke’s	0.333	0.801	
	Advanced RAG-Tool Fusion with Toolshed Knowledge Base	0.400	0.894	

Recommendations for Post-Retrieval. If opting to add reranking, corrective, or self-RAG:

- A post-requisite to intra-retrieval Recommendations 1 and 3 is reranking the N sets of retrieved tools from the decomposed/expanded queries to the final condensed *top-k* (if limiting k).
- Explore the retrieval accuracy vs. cost/latency trade-off of an embedder vs. LLM-based reranker.
- Self-RAG can help if not all tools were retrieved.
- Remove duplicates in each sub-query tool set.

3.4.4 Advanced RAG-Tool Fusion Equation

The Advanced RAG-Tool Fusion equation is modeled for any given *tool-M* and *top-k* value. The abbreviated ARTF Agent Accuracy can be measured by its retrieval accuracy (*tool-M*, *top-k*), multiplied by Base Agent’s accuracy where $tool-M_b = top-k$. A Base Agent is a simple LLM with M_b tools equipped. This equation is critical when optimizing the *top-k* value (from 1-128) due to the trade-off of Base Agent Accuracy, ARTF retrieval accuracy, and cost.

$$\begin{aligned} & \mathbb{E}[\text{ARTF Agent Acc.}(tool-M, top-k)] = \\ & \mathbb{E}[\text{Base Agent Acc.}(tool-M_b = top-k)] \times \\ & \mathbb{E}[\text{ARTF Retrieval Acc.}(tool-M, top-k)] \end{aligned} \quad (1)$$

4 EVALUATIONS

In this section, we describe two experiments, evaluate results, and discuss impacts to gauge the 1) tool selection effectiveness of Advanced RAG-Tool Fusion and the 2) impact of varying the number of tools (*tool-M*) an agent has on Base Agent accuracy and the tool selection threshold (*top-k*) has on retrieval accuracy, and cost. The former is a study on our approach and the latter dictates the optimization and trade-off of *top-k* in Advanced RAG-Tool Fusion.

4.1 Scaling Tool Selection and Planning with Advanced RAG-Tool Fusion

4.1.1 Experiment Settings

We assess Advanced RAG-Tool Fusion’s retrieval accuracy compared to baselines and SOTA retrievers at recall@k ($k = 1, 5, 10$). Section 3.1 specifies the dataset. Section 3.2 states the models used.

4.1.2 Results Analysis

Table I compares retrieval results on the Seal-Tools and ToolE datasets. Each dataset section begins with a baseline BM25 result, followed by comparisons of our approach against Seal-Tools DPR and Re-Invoke.

Advanced RAG-Tool Fusion outperforms the baseline by approximately 46% across all datasets. On the Seal-Tools dataset (both single and multi-tool evaluations), our approach shows an improvement of 41% over Seal-Tools DPR. On the ToolE single-tool dataset, it outperforms Re-Invoke by 5% and 9% on the single- and multi-tool datasets, respectively (recall@5). All metrics are absolute improvements.

4.1.3 Discussion

Comparing results in Table 1, the key differentiator (among all other modules) on the Seal-Tools DPR benchmark is the query decomposition module. Similarly, the distinction between our approach and the Re-Invoke ToolE (single and multi tool) benchmark lies in the pre-retrieval, multi-query expansion, query rewriting, and reranking modules.

Our findings show our ensemble-based Advanced RAG-Tool Fusion consistently outperforms individual (1-2) applications of advanced RAG within their isolated tool selection frameworks. Notably, for the Seal-Tools dataset, our approach enables tool-equipped agents to scale to thousands of tools without a significant drop in retrieval accuracy.

Advanced RAG-Tool Fusion and Toolshed Knowledge Bases require no fine-tuning, allowing easy implementation for researchers and practitioners. We encourage benchmarking your tool datasets to identify which tool components are most impactful in pre-retrieval, and which modules are beneficial in intra-retrieval and post-retrieval phases.

4.2 Varying Number of Tools (*tool-M*) and Selection Threshold (*top-k*)

4.2.1 Experiment Settings

We assess the tool-calling ability of a Base LLM Agent by incrementally equipping it with M tools, ranging from 1 to 128. The evaluation uses our Toolshed Evaluation Framework (Appendix A).

Next, we evaluate the retrieval accuracy of Advanced RAG-Tool Fusion by varying both the selection threshold $top-k$ (from 1 to 128) and the total number of tools in the Toolshed Knowledge Base (from 1 to $\sim 3,500$). For each $tool-M$ level, we vary $top-k$ such that $top-k \leq M$. We also explore different configurations within Advanced RAG-Tool Fusion, including the components in the tool embedding, the embedder for indexing and retrieving, and the LLM.

4.2.2 Results Analysis

Base Agent Accuracy. Across all M values (1–128), Base Agent accuracy remains around 97–100% (See Appendix B). Thus, for the Seal-Tools dataset, the number of tools equipped to an agent ($tool-M$) does not significantly affect Base Agent’s accuracy (likely due to the dataset’s distinct non-overlapping query-tool pairs and lack of sequential tool calls).

Retrieval Accuracy. In Fig. 2 and Fig. 3, we compare the retrieval accuracy of multi-hop queries for Seal-Tools DPR and Advanced RAG-Tool Fusion, respectively. As $tool-M$ and $top-k$ increase for Seal-Tools DPR, accuracy decreases significantly. However, Advanced RAG-Tool Fusion maintains high accuracy (~ 95 –100%) across all $tool-M$ and $top-k$.

Other Variations. Adding the argument schema hypothetical questions and key topics improved retrieval. However, this depends on the query-tool dataset. Varying embedders and LLMs (Appendix B, Fig. 13 and Fig. 14) showed minor gains (1–3%); `text-embedding-large-3` and `gpt-4o` outperformed others (See Section 3.2 for models).

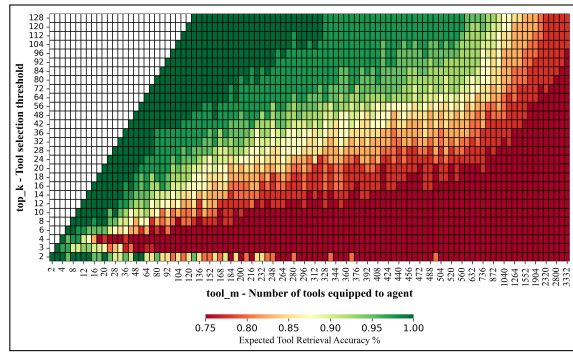


Figure 2: Impact of varying the selection threshold ($top-k$) (y-axis) and number of total tools ($tool-M$) (x-axis) from 1–3,500 on retrieval accuracy (recall@ $top-k$) of Seal-Tools DPR benchmark, without query decomposition.

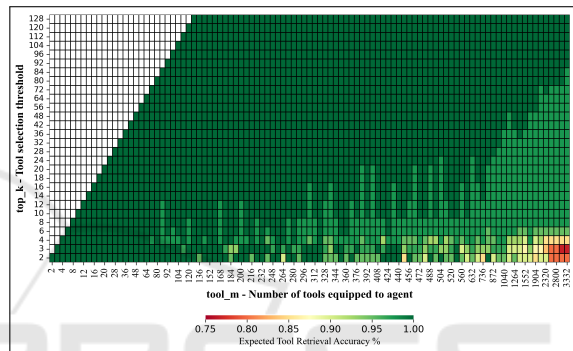


Figure 3: Impact of varying the selection threshold ($top-k$) (y-axis) and number of total tools ($tool-M$) (x-axis) from 1–3,500 on retrieval accuracy of Advanced RAG-Tool Fusion. This approach uses query decomposition among other patterns.

4.2.3 Discussion

Our work highlights how $tool-M$ and $top-k$ impact retrieval accuracy, Base Agent accuracy, and cost (Appendix B). As $top-k$ increases, retrieval improves but raises token costs and may lower Base LLM Agent accuracy on complex datasets. Thus, optimizing $top-k$ involves a trade-off between retrieval accuracy, Base Agent accuracy, and cost (Anthropic, 2024). In scenarios with complex tool datasets (Appendix B), a Base Agent may struggle to select the correct tools for high $tool-M$. Setting a lower $top-k$ can help the agent reason with fewer tools and reduce cost, though it may hinder retrieval accuracy if $top-k$ is too low.

We recommend first analyzing how a Base Agent performs on your tool dataset ($tool-M$). While the number of tools did not affect accuracy in the Seal-Tools dataset, other datasets with overlapping tools, intra-tool dependencies, or sequential reasoning (Lu et al., 2024; Yao et al., 2024) may show varied results. Finally, after customizing pre-retrieval, intra-retrieval, and post-retrieval modules, optimize $top-k$ by considering Base Agent accuracy $tool-M_b = top-k$.

5 CONCLUSION

As agent applications become more complex and scale to hundreds or thousands of tools, there is a need to consistently retrieve the correct tools to answer a user question. In this work, we present Advanced RAG-Tool Fusion, an ensemble of advanced RAG patterns novelly applied to tool selection and planning. Our framework consists of strategies within three phases: pre-retrieval, intra-retrieval, and post-retrieval. We have demonstrated that this ensemble of methods enables scalable tool-equipped agents and significantly outperforms both the baseline and an approach using single applications of advanced RAG, without fine-tuning LLMs or retrievers. Furthermore, we present Toolshed Knowledge Bases, the vector database to efficiently store the collection of tools during the pre-retrieval stage. Lastly, we study the impact of varying both 1) the total tools ($tool-M$) in the Toolshed Knowledge Base and 2) the tool selection threshold ($top-k$) on retrieval accuracy, Base Agent tool calling ability, and cost. Advanced RAG-Tool Fusion moves the needle for scaling tool-equipped agents and sheds light on the trade-off between retrieval accuracy, agent accuracy, and cost.

6 LIMITATIONS

Challenges remain for production-grade scalable tool-equipped agents. The first limitation is the need for human-in-the-loop planner modules to ask clarifying questions, such as “to confirm, you want to do X and Y?” Although deviating from zero-shot tool calling, this could confirm users’ true intent, refine sub-intent breakdowns, improving retrieval accuracy.

The second limitation concerns optimizing the tool selection threshold ($top-k$) for sub-queries. Currently, a fixed threshold is split even across sub-intents. However, if one sub-intent is more complex, the fixed $top-k$ may hinder accuracy. Future research could explore dynamic thresholds based on sub-intent complexity, capped at the overall tool threshold.

The third limitation involves multi-turn chat history. For instance, if a chatbot calculates the net present value of the user’s cash flows and follows up with “what if the initial cost was \$500 more?”, research is needed to determine whether to reuse the initial tool set or rerun the retrieval process. We hope that future contributions build on Advanced RAG-Tool Fusion and Toolshed Knowledge Bases to maximize the tool-calling ability of LLM agents.

REFERENCES

- Anantha, R., Bandyopadhyay, B., Kashi, A., Mahinder, S., Hill, A. W., and Chappidi, S. (2023). ProTIP: Progressive Tool Retrieval Improves Planning. *Preprint*, arXiv:2312.10332.
- Anthropic (2024). Tool use system prompt.
- Asai, A., Wu, Z., Wang, Y., Sil, A., and Hajishirzi, H. (2023). Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. *Preprint*, arXiv:2310.11511.
- Chen, Y., Yoon, J., Sachan, D. S., Wang, Q., Cohen-Addad, V., Bateni, M., Lee, C.-Y., and Pfister, T. (2024). Re-Invoke: Tool Invocation Rewriting for Zero-Shot Tool Retrieval. *Preprint*, arXiv:2408.01875.
- Du, Y., Wei, F., and Zhang, H. (2024). AnyTool: Self-Reflective, Hierarchical Agents for Large-Scale API Calls. *Preprint*, arXiv:2402.04253.
- Gao, L., Ma, X., Lin, J., and Callan, J. (2022). Precise Zero-Shot Dense Retrieval without Relevance Labels. *Preprint*, arXiv:2212.10496.
- Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., and Wang, H. (2024). Retrieval-Augmented Generation for Large Language Models: A Survey. *Preprint*, arXiv:2312.10997.
- Google Cloud (2024). Function declarations.
- Hao, S., Liu, T., Wang, Z., and Hu, Z. (2024a). ToolkenGPT: Augmenting Frozen Language Models with Massive Tools via Tool Embeddings. *Preprint*, arXiv:2305.11554.
- Hao, Y., Cao, P., Jin, Z., Liao, H., Chen, Y., Liu, K., and Zhao, J. (2024b). CITI: Enhancing Tool Utilizing Ability in Large Language Models without Sacrificing General Performance. *Preprint*, arXiv:2409.13202.
- Huang, T., Jung, D., and Chen, M. (2024a). Planning and Editing What You Retrieve for Enhanced Tool Learning. *Preprint*, arXiv:2404.00450.
- Huang, Y., Shi, J., Li, Y., Fan, C., Wu, S., Zhang, Q., Liu, Y., Zhou, P., Wan, Y., Gong, N. Z., and Sun, L. (2024b). MetaTool Benchmark for Large Language Models: Deciding Whether to Use Tools and Which to Use. *Preprint*, arXiv:2310.03128.
- Jagerman, R., Zhuang, H., Qin, Z., Wang, X., and Bender-sky, M. (2023). Query Expansion by Prompting Large Language Models. *Preprint*, arXiv:2305.03653.
- Jeong, S., Baek, J., Cho, S., Hwang, S. J., and Park, J. C. (2024). Adaptive-RAG: Learning to Adapt Retrieval-Augmented Large Language Models through Question Complexity. *Preprint*, arXiv:2403.14403.
- Joshi, A., Sarwar, S. M., Varshney, S., Nag, S., Agrawal, S., and Naik, J. (2024). REAPER: Reasoning based Retrieval Planning for Complex RAG Systems. *Preprint*, arXiv:2407.18553.
- Kamradt, G. (2023). Needle in a haystack - pressure testing LLMs: A simple ‘needle in a haystack’ analysis to test in-context retrieval ability of long context LLMs.
- Khattab, O., Santhanam, K., Li, X. L., Hall, D., Liang, P., Potts, C., and Zaharia, M. (2023). Demonstrate-Search-Predict: Composing retrieval and language

- models for knowledge-intensive NLP. *Preprint*, arXiv:2212.14024.
- Li, M., Zhao, Y., Yu, B., Song, F., Li, H., Yu, H., Li, Z., Huang, F., and Li, Y. (2023). API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. *Preprint*, arXiv:2304.08244.
- Liu, W., Huang, X., Zeng, X., Hao, X., Yu, S., Li, D., Wang, S., Gan, W., Liu, Z., Yu, Y., Wang, Z., Wang, Y., Ning, W., Hou, Y., Wang, B., Wu, C., Wang, X., Liu, Y., Wang, Y., Tang, D., Tu, D., Shang, L., Jiang, X., Tang, R., Lian, D., Liu, Q., and Chen, E. (2024). ToolACE: Winning the Points of LLM Function Calling. *Preprint*, arXiv:2409.00920.
- Lu, J., Holleis, T., Zhang, Y., Aumayer, B., Nan, F., Bai, F., Ma, S., Ma, S., Li, M., Yin, G., Wang, Z., and Pang, R. (2024). ToolSandbox: A Stateful, Conversational, Interactive Evaluation Benchmark for LLM Tool Use Capabilities. arXiv:2408.04682 [cs].
- Ma, X., Gong, Y., He, P., Zhao, H., and Duan, N. (2023). Query Rewriting for Retrieval-Augmented Large Language Models. *Preprint*, arXiv:2305.14283.
- 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. *Preprint*, arXiv:2409.02141.
- Papineni, K. (2001). Why Inverse Document Frequency? In *Second Meeting of the North American Chapter of the Association for Computational Linguistics*.
- Patil, S. G., Zhang, T., Wang, X., and Gonzalez, J. E. (2023). Gorilla: Large Language Model Connected with Massive APIs. *Preprint*, arXiv:2305.15334.
- Peng, W., Li, G., Jiang, Y., Wang, Z., Ou, D., Zeng, X., Xu, D., Xu, T., and Chen, E. (2024). Large Language Model based Long-tail Query Rewriting in Taobao Search. *Preprint*, arXiv:2311.03758.
- Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Hong, L., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., and Sun, M. (2023). ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. *Preprint*, arXiv:2307.16789.
- Raudaschl, A. H. (2023). Forget RAG, the future is RAG-Fusion: The next frontier of search: Retrieval Augmented Generation meets Reciprocal Rank Fusion and generated queries.
- Robertson, S. and Zaragoza, H. (2009). The Probabilistic Relevance Framework: BM25 and Beyond. ISSN: 1554-0669 Issue: 4 Pages: 333-389 Publication Title: Foundations and Trends® in Information Retrieval Volume: 3.
- Roucher, A. (2023). Agentic RAG: Turbocharge your RAG with query reformulation and self-query!
- Sawarkar, K., Mangal, A., and Solanki, S. R. (2024). Blended RAG: Improving RAG (Retriever-Augmented Generation) Accuracy with Semantic Search and Hybrid Query-Based Retrievers. *Preprint*, arXiv:2404.07220.
- Setty, S., Thakkar, H., Lee, A., Chung, E., and Vidra, N. (2024). Improving Retrieval for RAG based Question Answering Models on Financial Documents. *Preprint*, arXiv:2404.07221.
- Sun, W., Yan, L., Ma, X., Wang, S., Ren, P., Chen, Z., Yin, D., and Ren, Z. (2023). Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents. *Preprint*, arXiv:2304.09542.
- Tang, Q., Deng, Z., Lin, H., Han, X., Liang, Q., Cao, B., and Sun, L. (2023). ToolAlpaca: Generalized Tool Learning for Language Models with 3000 Simulated Cases. *Preprint*, arXiv:2306.05301.
- Tang, Y. and Yang, Y. (2024). MultiHop-RAG: Benchmarking Retrieval-Augmented Generation for Multi-Hop Queries. *Preprint*, arXiv:2401.15391.
- Theja, R. (2023). Boosting RAG: Picking the best embedding & reranker models.
- Trivedi, H., Balasubramanian, N., Khot, T., and Sabharwal, A. (2023). Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions. *Preprint*, arXiv:2212.10509.
- Wang, L., Yang, N., and Wei, F. (2023). Query2doc: Query Expansion with Large Language Models. *Preprint*, arXiv:2303.07678.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. (2023). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Preprint*, arXiv:2201.11903.
- Wu, M., Zhu, T., Han, H., Tan, C., Zhang, X., and Chen, W. (2024). Seal-Tools: Self-Instruct Tool Learning Dataset for Agent Tuning and Detailed Benchmark. *Preprint*, arXiv:2405.08355.
- Xu, B., Peng, Z., Lei, B., Mukherjee, S., Liu, Y., and Xu, D. (2023). ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models. *Preprint*, arXiv:2305.18323.
- Yan, S.-Q., Gu, J.-C., Zhu, Y., and Ling, Z.-H. (2024). Corrective Retrieval Augmented Generation. *Preprint*, arXiv:2401.15884.
- Yao, S., Shinn, N., Razavi, P., and Narasimhan, K. (2024). τ -bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *Preprint*, arXiv:2406.12045.
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. *Preprint*, arXiv:2210.03629.
- Yuan, L., Chen, Y., Wang, X., Fung, Y. R., Peng, H., and Ji, H. (2024a). CRAFT: Customizing LLMs by Creating and Retrieving from Specialized Toolsets. *Preprint*, arXiv:2309.17428.
- Yuan, S., Song, K., Chen, J., Tan, X., Shen, Y., Kan, R., Li, D., and Yang, D. (2024b). EASYTOOL: Enhancing LLM-based Agents with Concise Tool Instruction. *Preprint*, arXiv:2401.06201.
- Zheng, H. S., Mishra, S., Chen, X., Cheng, H.-T., Chi, E. H., Le, Q. V., and Zhou, D. (2024a). Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models. *Preprint*, arXiv:2310.06117.
- Zheng, Y., Li, P., Liu, W., Liu, Y., Luan, J., and Wang, B. (2024b). ToolRerank: Adaptive and Hierarchy-Aware Reranking for Tool Retrieval. *Preprint*, arXiv:2403.06551.

APPENDIX

A: Toolshed Evaluation Framework

The Toolshed Evaluation Framework enhances Tool-Eval (Qin et al., 2023) by introducing granular metrics: 1) tool name, 2) parameter keys, and 3) parameter values. These metrics pinpoint if poor agent performance stems from errors in selecting tool names, understanding parameter keys, or inputting values. Recall is computed at the sub-metric level using the golden dataset and agent responses. A weighted score combines these metrics (50% tool name, 25% each for keys and values), averaged for multiple tool calls in a golden QA set.

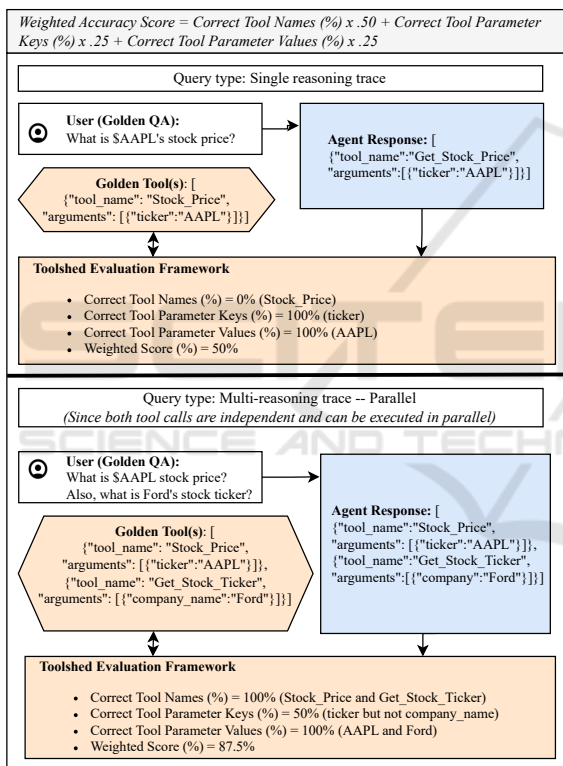


Figure 4: Toolshed Evaluation Framework.

B: Varying tool-M and top-k

Measuring Base Agent Accuracy. The graphs compare Seal-Tools weighted accuracy (Fig. 4) of the Base Agent and Advanced RAG-Tool Fusion at fixed top-k across tool-M levels (1–3,500). The Base Agent drops to 0% at tool-M=129 due to API limits (128 tools) from providers (Anthropic, 2024; Google Cloud, 2024). Advanced RAG-Tool Fusion, with top-k ≤ 128, consistently outperforms the Base Agent.

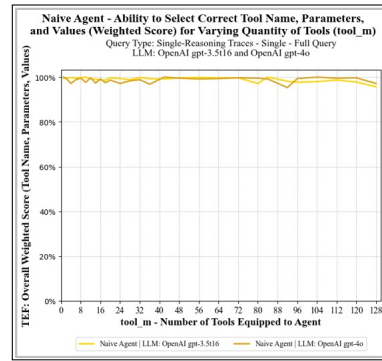


Figure 5: Base Agent accuracy for Single Reasoning Traces, varying tools 1–128.

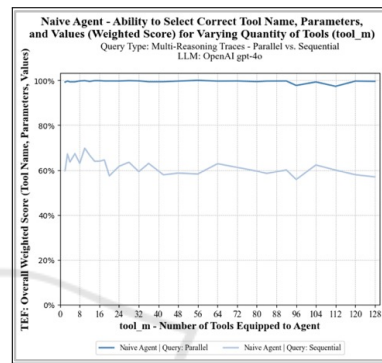


Figure 6: Base Agent accuracy for Multi-Reasoning Traces, varying tools 1–128. Sequential reasoning performs lower.

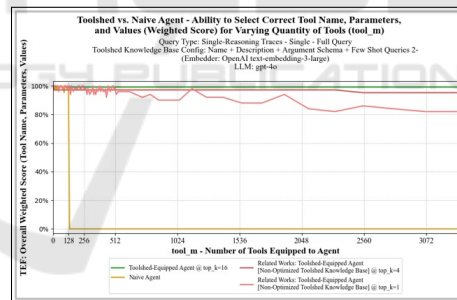


Figure 7: Comparison of Single-Reasoning Traces, varying tools 1–3,500 and top-k. Higher top-k improves retrieval.

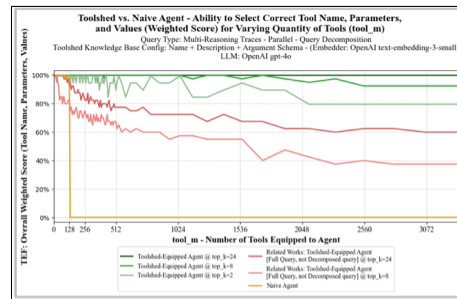


Figure 8: Comparison of Multi-Reasoning Traces, varying tools 1–3,500 and top-k. Red retriever has no query decomposition.

Measuring Token Cost. The graphs compare Seal-Tools token counts for Base Agent and Advanced RAG-Tool Fusion across *tool-M* levels (1–3,500) at fixed *top-k*. Token costs rise with more tools.

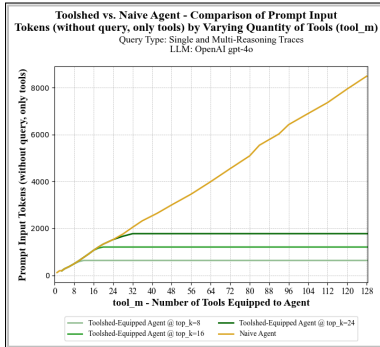


Figure 9: Prompt tokens (tools only) for Base Agent, varying tools 1–128 and top-k at 8, 16, 24.

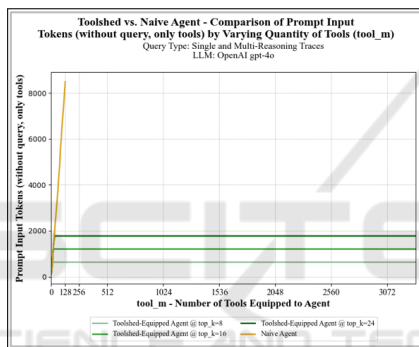


Figure 10: Prompt tokens (tools only) for Base Agent, varying tools 1–3,500 and top-k at 8, 16, 24.

Optimizing Trade-Offs in Accuracy, Performance, and Cost. A Base Agent struggles with more than 20 tools, while Advanced RAG-Tool Fusion scales to thousands or limits tools to a manageable *top-k*. Fixing *top-k* balances retrieval accuracy with token cost.

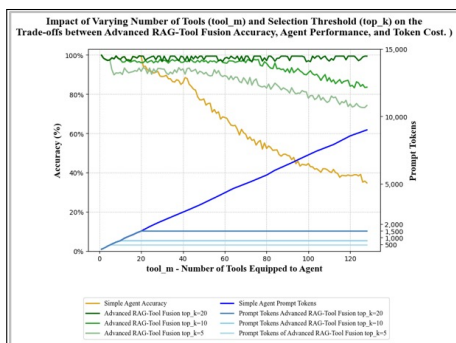


Figure 11: Trade-off between retrieval accuracy, agent performance, and token cost across tool-M and top-k values. Uses the Advanced RAG-Tool Fusion equation.

Impact of Embedders, Tool Configurations, *tool-M*, and *top-k*. Retrieval accuracy improves slightly with advanced embedders and richer tool components, while relying only on tool name and description leads to lower accuracy.

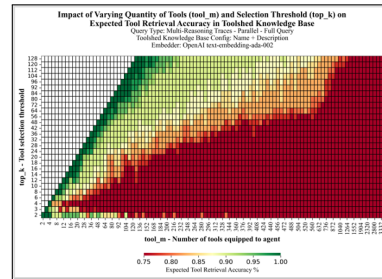


Figure 12: No query decomposition: Embedder is text-embedding-ada-002, using tool name and description.

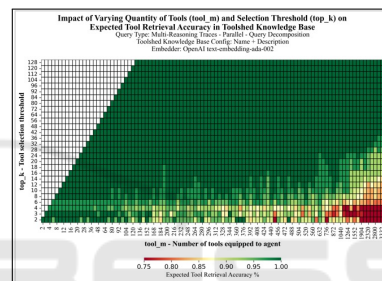


Figure 13: With query decomposition: Embedder is text-embedding-ada-002, using tool name and description.

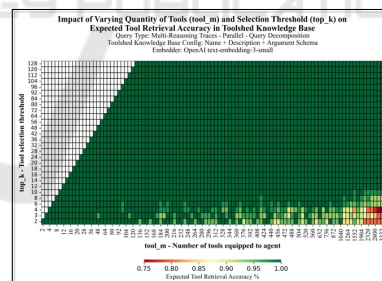


Figure 14: Query decomposition, embedder is text-embedding-3-small, with tool name, description, and arg schema.

C: Case Study

Configuration (per each Toolshed Knowledge Base).

- **LLM:** AOI gpt-4o-2024-08-06
- **Embedder:** AOI text-embedding-3-small
- **Tool-M:** 1,000 (per each TSKB)
- **Top-k:** 10 (final agent)

Hypothetical Situation

You have 3,000 tools or functions and aim to create a multi-agent system. Each agent uses a dedicated *Toolshed Knowledge Base* and implements *Advanced RAG-Tool Fusion* for optimized retrieval and execution. The tools are divided among the following sub-agents:

- **1,000 Finance Tools:** Focused on financial operations and analytics.
- **1,000 Database Operation Tools:** Designed for database management and query execution.
- **1,000 Healthcare Tools:** Tailored for healthcare-related tasks and insights.

Figure 15: Hypothetical Situation for Case Study. See above for configuration of each Toolshed Knowledge Base.

Pre-retrieval (indexing)

The following steps outline the pre-processing pipeline needed for each set of tools. After step 6, store the tool documents in a vector database.

1. Create Clear, Descriptive Names for Each Tool
 - Each tool should have a name that clearly describes its function.

e.g. "get_net_present_value"
2. Develop Clear Descriptions for Each Tool
 - Provide a detailed description of each tool, including specific scenarios where it would be used and where it wouldn't.

"Calculates the net present value (NPV) of a series of cash inflows and outflows over a specified period, discounted to present value based on a given rate. Useful for determining the value of future cash flows, particularly in investment scenarios, when provided with initial investment, discount rate, and time period."
3. Define Clear Argument Schema with Parameter Names and Descriptions for Each Tool
 - List and define the input arguments for each tool.

initial_value: "The initial cash flow at the start of the period, which could be an investment, cost, or inflow."
start_date: "The beginning date of the cash flow period."
end_date: "The end date of the cash flow period."
discount_rate: "The rate used to discount future cash flows to their present value."
scrap_value: "The final residual value of the asset at the end of the period."
cash_flows: "A series of inflows and outflows over the specified period."
4. Optional: Generate Hypothetical Questions for Each Tool (may help with retrieval)
 - Add 1-10 diverse user questions related to the tool. Use existing questions if available. Try to include parameters.

1. "What is the NPV for a project starting on January 1, 2025, with an initial outflow of \$100,000, annual cash flows of \$15,000, and a discount rate of 8%?"
2. "Calculate the net present value for cash flows of \$20,000 per year over 10 years, with a 7% discount rate."
3. "What is the NPV if my project ends in December 2030, with an initial cost of \$50,000 and a scrap value of \$5,000?"
5. Optional: Generate Key Topics, Themes, or Intents for Each Tool (may help with retrieval)
 - Add 1-10 key topics or intents associated with the tool. Be concise and differentiable from other tools. Can generate from the hypothetical questions and tool name, description, and argument schema.

1. "Investment Valuation"
2. "Cash Flow Analysis"
6. Metadata for Tool Name in Code Repository

{"tool_name": "get_net_present_value"}
7. Optional: Metadata filtering for hierarchical or group-based tool groups

{"sub_group": "financial_calculations"}

Figure 16: Case study pre-retrieval phase.

Logistics and Maintenance of Toolshed Knowledge Base with Tool Calling

The following steps outline best practice considerations when using *Advanced RAG-Tool Fusion* and *Toolshed Knowledge Bases* in production.

1. Unified tools.py for each *Toolshed Knowledge Base*
 - To aid in maintaining the tools or functions, each set of tools used for a *Toolshed Knowledge Base* should be separated.

finance_tools.py -- the collection of 1,000 finance tools in whatever tool creation framework of your choice.
2. Ability to add a new tool to the *Toolshed Knowledge Base*
 - Automated systems in place to add a new tool/function to the *Toolshed Knowledge Base*.
 - *Recommended solution:* Use hashes to track changes in tool name, description, argument schema, and any appended questions or key topics/intents. Steps: generate a unique hash for each tool and compare it to the previously stored hash to identify when re-indexing or updates are necessary.
3. Ability to delete a tool to the *Toolshed Knowledge Base*
 - Automated systems in place to add a new tool/function to the *Toolshed Knowledge Base*.
 - *Recommended solution:* Use hashes to track changes in tool name, description, argument schema, and any appended questions or key topics/intents. Steps: generate a unique hash for each tool and compare it to the previously stored hash to identify when re-indexing or updates are necessary.
4. Ability to update a tool to the *Toolshed Knowledge Base*
 - Automated systems in place to update an existing tool/function to the *Toolshed Knowledge Base*.
 - *Recommended solution:* Use hashes to track changes in tool name, description, argument schema, and any appended questions or key topics/intents. Steps: generate a unique hash for each tool and compare it to the previously stored hash to identify when re-indexing or updates are necessary.
5. Generate a Toolshed Dictionary from the tools.py file for actual agent-tool execution
 - The Toolshed dictionary will serve as an in-app, inference-time look-up, where each key is the tool name in tools.py, and the value is the actual python tool or function.
 - After *Advanced RAG-Tool Fusion* retrieves the *top-k* relevant tools to equip to an agent, these tools are actually the documents represented by the vector database. For each retrieved tool document, use the tool document metadata key "tool_name" that we set up in step 6 of phase 1 pre-retrieval/indexing to access the key-value pair of the Toolshed Dictionary. You can then attach these functions to the agent in the framework of your choice.

```

graph LR
    subgraph Tools [top-k tools]
        T1[tool.metadata["tool_name"]]
        T2[tool.metadata["tool_name"]]
        T3[...]
    end
    subgraph Dictionary [Toolshed Dictionary]
        D["{"get_net_present_value":  
GetNetPresentValue(),  
"...": Tool()}"]
    end
    subgraph Agent [Agent]
        A["[ GetNetPresentValue(), Tool() ]"]
    end
    Tools --> Dictionary
    Dictionary -- Attach tools to agent --> Agent
            
```

Figure 17: Case study logistics in production.

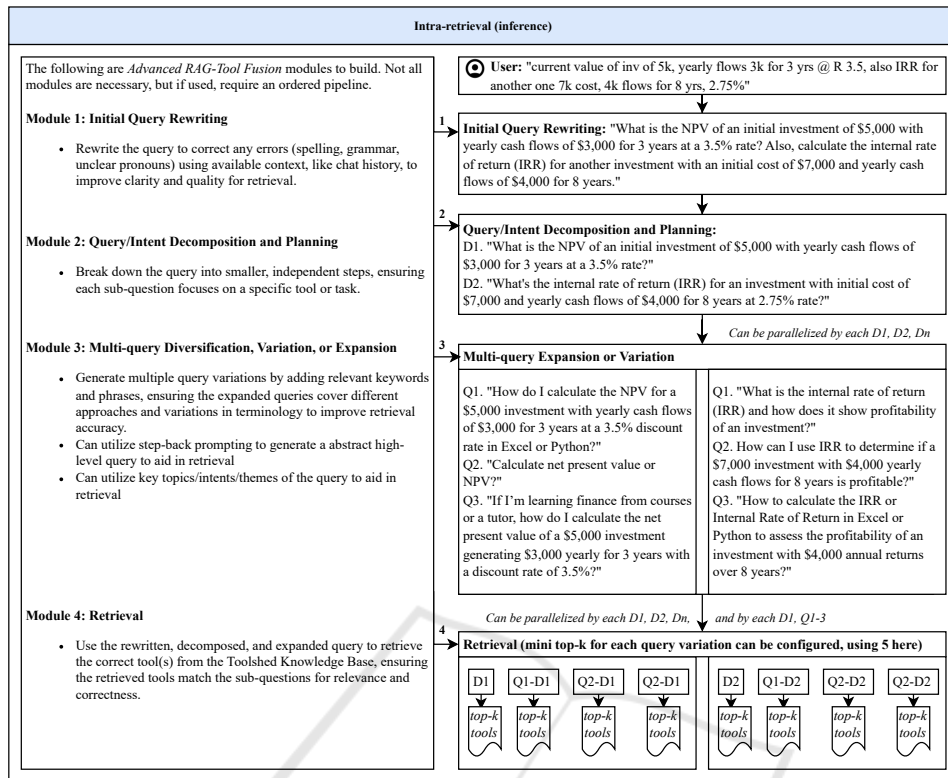


Figure 18: Case study intra-retrieval phase.

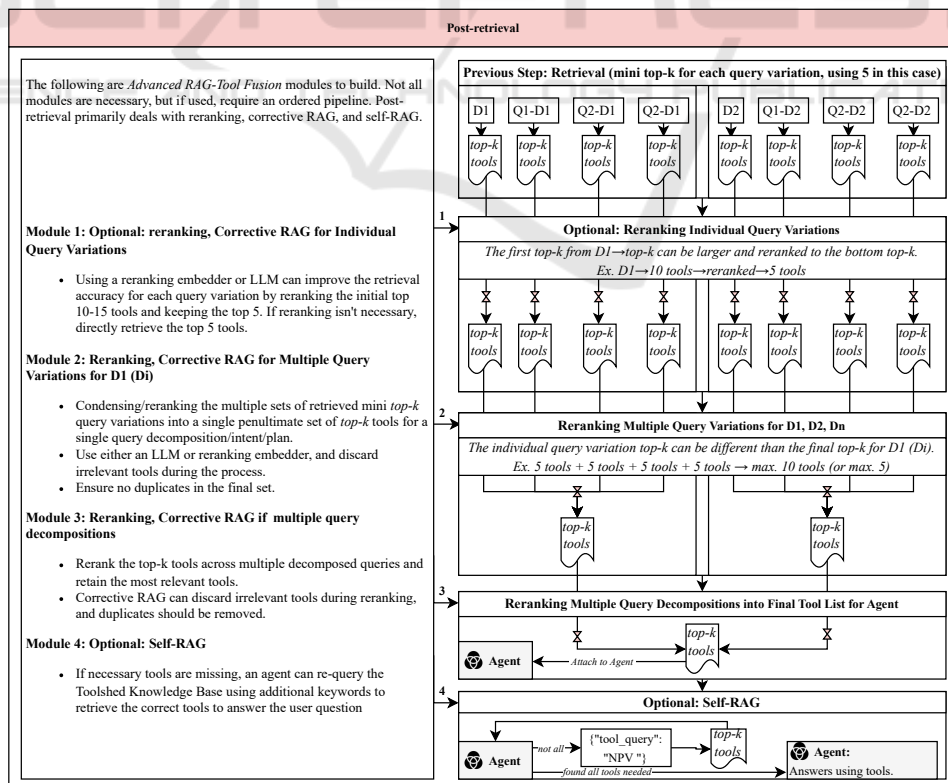


Figure 19: Case study post-retrieval phase.