algorithms

MDPI

Article

Too Many Tools, Too Much Confusion? Navigating Agentic Tool Selection at Scale

Jerzy Kamiński [1,*], Ilya Galyukshev [1,2], Sergey Chuprin [1], Artem Kuznetsov [1] and Anna Kalyuzhnaya [1]

[1] Artificial Intelligence Technologies Faculty, ITMO University, Saint-Petersburg 197101, Russia; galyukshev@itmo.ru (I.G.); 468028@niuitmo.ru (S.C.); kmetra1910@gmail.com (A.K.); anna.kalyuzhnaya@itmo.ru (A.K.)
[2] Faculty of Mathematics and Computer Science, Central University, Moscow 123056, Russia
* Correspondence: jkaminski@niuitmo.ru

Abstract

This paper addresses the critical scalability challenge that large language model agents face when operating over massive tool repositories. As tool catalogs expand to hundreds or thousands of functions, current architectures exhibit substantial performance degradation caused by semantic collisions between similar tools and ineffective handling of complex multi-tool scenarios. To address these bottlenecks, we propose a recall-first Retrieval–Plan–Select (RPS) framework that combines context-aware query decomposition with synthetic tool description augmentation. The proposed approach explicitly separates retrieval, planning, and final selection through step-local candidate generation, while augmented tool descriptions enriched with expanded summaries and synthetic user questions reduce representation collisions in dense embedding spaces. Evaluation across Ultratool, ToolLinkOS, and ToolRet demonstrates that contextual decomposition consistently improves end-to-end recall under large tool catalogs, increasing recall from 0.340 to 0.494 on Ultratool, from 0.208 to 0.323 on ToolLinkOS, and from 0.300 to 0.347 on ToolRet. Description augmentation further improves retrieval quality, increasing Recall@10 from 0.288 to 0.403 and reducing high-similarity semantic collisions by 41.9% at the 0.90 cosine-similarity threshold. The proposed framework highlights that scalable tool use should be approached primarily as a recall-oriented retrieval and planning problem rather than as a flat in-context selection task, providing practical guidance for building large-scale tool-augmented agents over modern API and MCP-based ecosystems.

Keywords: large language models; query decomposition; embedding spaces; multi-agent system; tool calling; tool selection

Academic Editor: Haifeng Zhang

Received: 9 April 2026
Revised: 17 May 2026
Accepted: 27 May 2026
Published: 1 June 2026

1. Introduction

Tool-augmented language models have transformed LLM applications by enabling interaction with external APIs and computational resources. However, as tool ecosystems expand beyond hundreds of functions, current agent architectures face critical scalability issues. Our preliminary analysis shows tool selection accuracy drops dramatically when moving from small to large tool environments, while inference and context-management costs increase substantially. Traditional approaches either overload context windows with an exhaustive number of tools or rely on simplistic retrieval methods that fail to capture the complex semantic space of tools. We propose a multi-agent approach paired with synthetic tool description augmentation.

Despite rapid progress in tool-using agents, the community lacks a clear problem decomposition and measurement protocol for the two dominant failure modes at scale: retrieval collisions among semantically similar tools and drift in in-context selection when candidate pools grow. Prior work also under-specifies how tool document design and stepwise planning interact and rarely reports behavior as catalogs cross the hundreds-to-thousands threshold.

To address these challenges, we propose a multi-agent approach paired with synthetic tool description augmentation. This work explores the following research questions: (1) Can current RAG-augmented multi-agent systems reliably operate over hundreds and thousands of tools? (2) How do RAG-enhancement techniques influence the quality of retrieval model outputs and LLM-agent tool selection capabilities? (3) What strategies are most effective for operating over massive tool repositories?

This paper is primarily a problem-driven study that addresses these gaps through a systematic investigation of tool selection at scale. Our contributions are (1) a recall-first Retrieval–Plan–Select decomposition that separates retrieval from selection and makes step-local candidate sets explicit; (2) a compact description augmentation recipe (clarified summaries and synthetic questions) that reduces collisions and improves retrieval recall under closest-negatives stress; and (3) a collision-aware stress-testing protocol (random vs. closest noise) that exposes when augmentation helps retrieval yet can burden in-context selection.

Our evaluation on Ultratool (Huang et al. [1]), ToolLinkOS (Lumer et al. [2]) and ToolRet (Shi et al. [3]) benchmarks compares architectural configurations and demonstrates that our approach yields modest but consistent gains that compound as tool catalogs scale from hundreds to thousands of tools.

2. Related Work

2.1. Tool-Using Agents and Scalability

A central challenge in the development of tool-augmented language models (LLMs) lies not in tool invocation itself but in the retrieval and selection of appropriate tools from increasingly large repositories. Shi et al. [3] highlight this problem in their benchmark study, demonstrating that existing information retrieval (IR) models—though strong on conventional IR tasks—perform poorly on large-scale tool retrieval. Their proposed ToolRet benchmark, consisting of 7.6k retrieval tasks over 43k tools, reveals that retrieval quality is a critical bottleneck: even high-performing IR models achieve less than 35% completeness at the top-10, leading to a sharp degradation in the downstream pass rate of tool-use LLMs. This work establishes that scaling tool-using agents is not merely a question of tool orchestration but fundamentally a problem of efficient and accurate retrieval from vast, heterogeneous toolsets.

Beyond retrieval completeness, Shi et al. [3] further report that even when gold tools appear in the candidate list, downstream pass rate remains low, with the best-performing LLM agents achieving only 20–30% task success on the ToolRet benchmark. This demonstrates that retrieval errors accumulate and propagate through execution, reinforcing that scalable tool selection is primarily limited by retrieval quality rather than tool calling ability. ToolRet therefore provides an important stress test for scalable tool-augmented LLM architectures: its large and heterogeneous tool space exposes semantic collisions, schema variation, and long-tail retrieval failures that are often hidden in smaller curated benchmarks. In contrast, our contextual decomposition approach improves recall from 0.247 to 0.465 on Ultratool and from 0.208 to 0.323 on ToolLinkOS, and it also achieves a competitive F1 on ToolRet, indicating more robust behavior under large catalog sizes.

While ToolRet focuses on the difficulty of retrieving tools from large repositories, another related line of work studies how LLMs handle tool chains and multi-step tool dependencies. Han et al. [4] present NESTools, a benchmark for evaluating nested tool learning in LLMs, where a tool's output serves as input to a subsequent tool. To simulate realistic conditions, the authors augment each ground-truth tool with five distractor tools, retrieved and filtered by Levenshtein distance to minimize name similarity. This method of ambiguity avoidance is robust but does not capture real-world API embedding-space collisions, where tools may be semantically close even when their names are distinct. The study concludes that even advanced LLMs perform poorly on complex nested tasks, highlighting the need for methods that can support both scalable retrieval and reliable multi-tool reasoning.

The retrieval bottleneck identified by ToolRet and the ambiguity issues highlighted by NESTools have motivated recent work that adapts advanced RAG principles to tool selection. Lumer et al. [5] introduce the Toolshed framework with Advanced RAG-Tool Fusion, a comprehensive three-phase approach (pre-retrieval, intra-retrieval, and post-retrieval) that systematically enhances tool document representation and retrieval accuracy. Unlike conventional approaches that store minimal tool metadata, Toolshed enhances tool documents with comprehensive descriptions, argument schemas, synthetically generated hypothetical questions, and key topics before indexing in specialized "Toolshed Knowledge Bases." During inference, the framework employs query decomposition, multi-query expansion, and LLM-based reranking to address the fundamental challenge that direct user queries often fail to capture the full essence of required tools. This approach demonstrates significant improvements in retrieval accuracy on datasets like Seal-Tools ($\sim$4000 tools) and ToolE ($\sim$200 tools), suggesting that the combination of enhanced indexing strategies and sophisticated query processing can substantially mitigate the retrieval bottleneck identified by ToolRet.

A related but distinct direction is representation learning for tool retrieval. ProTIP [6] proposes learning prototypical tool representations via contrastive alignment with tool descriptions and usage traces. Unlike Toolshed's reliance on document enrichment and reranking, ProTIP jointly embeds tools and queries into a shared space where semantically similar tools cluster naturally. ProTIP further reports improved retrieval accuracy over prior baselines, indicating the value of geometry-aware tool representations. This approach improves retrieval by structuring the embedding space around canonical usage semantics, enabling better generalization to unseen or sparse tools. Together, Toolshed and ProTIP show two complementary ways of improving tool retrieval: enriching the textual representation of tools before indexing, and learning a more suitable geometry for matching tools and queries.

To clarify the relation between these approaches, Table 1 summarizes how prior work addresses different aspects of scalable tool use. Existing methods typically focus on one main component: benchmarking retrieval failures, enriching tool representations, learning improved embedding geometries, or improving downstream orchestration. Our RPS framework combines these directions into a recall-first pipeline that uses contextual decomposition and synthetic tool descriptions to improve candidate coverage before final tool selection. A more detailed comparison is provided in Appendix A.

These scalable retrieval-oriented works extend earlier efforts in tool learning, which focused more narrowly on enabling LLMs to demonstrate tool usage under limited settings. Schick et al. [7] introduced Toolformer for self-supervised tool learning, establishing foundational paradigms but restricting evaluations to small toolsets. Liu et al. [8] designed ToolNet with directed graphs for tool selection. While effective at capturing structured dependencies, these approaches assume pre-annotated or pre-constructed tool relationships and do

not scale to repositories beyond hundreds of tools. In contrast, ToolRet, Toolshed, and ProTIP explicitly confront the issue of retrieval at scale, demonstrating how performance collapses when the simplifying assumption of small, curated toolsets is removed.

Table 1. Comparison of related approaches for scalable tool retrieval and tool-use agents.

Work	Main Focus	Key Mechanism	Relation to RPS
Toolformer [7]	Tool-use learning	Self-supervised API-call prediction	Foundational tool-use setting; not focused on large-scale retrieval.
ToolNet [8]	Structured tool selection	Directed graph over tool relations	Models dependencies, but assumes predefined tool structure.
ToolRet [3]	Retrieval benchmark	Large-scale tool-retrieval evaluation	Establishes retrieval bottleneck; does not propose a full RPS-like pipeline.
NESTools [4]	Nested tool use	Multi-step tool dependency benchmark	Tests tool chaining, but does not model dense semantic collisions directly.
Toolshed [5]	RAG-based tool retrieval	Enriched tool documents, query expansion, and reranking	Closely related; RPS emphasizes recall-first retrieval and downstream selection trade-offs.
ProTIP [6]	Tool representation learning	Contrastive alignment of tools and queries	Improves embedding geometry; complementary to RPS-style decomposition and augmentation.
Think-then-Act/Chain of Tools [9,10]	Tool orchestration	Planning and sequential tool execution	Improves downstream reasoning after candidates are available.
RPS (ours)	Recall-first tool retrieval	Contextual decomposition and synthetic description enhancement	Combines decomposition, enhanced descriptions, and recall-oriented candidate generation.

In parallel to retrieval-focused methods, several recent works attempt to improve tool selection and orchestration once a smaller candidate tool pool is already available. Shen et al. [9] proposed Think-then-Act for retrieval-augmented generation (RAG) optimization, Min et al. [11] introduced UniHGKR for unified heterogeneous knowledge retrieval, and Qian et al. [12] developed Toolink for chain-of-solving paradigms. Shi et al. [10] advanced Chain of Tools for multi-tool orchestration. These methods improve sequential reasoning, planning, and tool-use coordination, but they do not directly address the first-stage retrieval bottleneck emphasized by ToolRet and Toolshed, where failures in selecting appropriate candidate tools severely limit downstream performance. This distinction is important: improving orchestration is valuable, but in large repositories the agent must first retrieve a sufficiently complete and relevant candidate set.

Our work is positioned at the intersection of these lines of research. Like ToolRet, we treat retrieval quality as a central bottleneck for scalable tool-augmented LLMs. Like Toolshed, we recognize that tool descriptions can be enriched to improve retrieval. Like orchestration-oriented methods, we also consider the multi-step nature of user tasks. However, our contribution is the combination of these ideas in a recall-first RPS pipeline: user requests are decomposed into contextual subqueries, tool descriptions are synthetically enhanced to reduce semantic mismatch, and candidate generation is separated from downstream tool selection. This combination is designed specifically for large and semantically dense tool repositories, where the key challenge is not only to select the best tool from a small set but first to avoid losing relevant tools during retrieval.

Direct quantitative comparison with Toolshed and ProTIP is currently difficult because neither framework provides a fully reproducible open-source evaluation pipeline compatible with our benchmark setup. In particular, Toolshed relies on a proprietary retrieval knowledge base construction pipeline, while ProTIP does not provide a public implementation of the full retrieval and orchestration stack.

2.2. Multi-Agent Systems for Tool Coordination

Multi-agent approaches have emerged as promising solutions for tool utilization scalability. Yang et al. [13] introduced XAgents for rule-based cooperation but focused on predefined domains without automatic sector identification. Recent coordination improvements include the following: Sprigler et al. [14] demonstrated emergent behaviors through synergistic simulations, Huang et al. [15] developed Romas for role-based specialization in database monitoring, and Liu et al. [16] improved efficiency through group discussion mechanisms.

These approaches shift the focus from retrieval to coordination: rather than asking how to retrieve the right tools from a large repository, they explore how multiple agents can divide responsibilities, specialize roles, and improve decision-making once the relevant environment is already constrained. More recently, Xu et al. [17] propose an iterative feedback mechanism, where the LLM repeatedly refines its retrieval queries based on execution feedback. This approach leads to improved retrieval accuracy compared to single-pass methods and could potentially be integrated into broader agentic tool-use pipelines.

Complementing these coordination strategies, Zhang et al. [18] introduce a similarity- and dependency-aware tool selection framework that explicitly models inter-tool relationships. By optimizing multi-tool selection via experience networks and training on simulated agent–tool interactions, ToolExpNet improves tool selection quality in complex environments. While not directly addressing the retrieval bottleneck, it highlights the value of structured dependency modeling for downstream execution.

However, these approaches typically remain grounded in small or curated benchmarks and do not fully address the scale and heterogeneity of real-world tool ecosystems. In contrast, our RPS framework focuses on the earlier retrieval stage under large-catalog conditions, where semantic collisions, long-tail tools, and incomplete candidate generation can prevent even strong downstream agents from succeeding. Thus, multi-agent coordination and dependency modeling are complementary to our approach, while RPS specifically targets recall-oriented tool retrieval as the prerequisite for reliable large-scale tool use.

3. Proposed Approach

We study tool selection at scale for LLM agents operating over hundreds to thousands of tools. Given a natural-language request q, the agent must retrieve a small candidate set of relevant tools and decide which tool(s) to call for each subtask. We design a recall-first pipeline whose components can be ablated independently in Section 4.

Our design follows three principles:

1. Recall-first selection. Missing a required tool prevents task completion; therefore, we optimize for recall at the candidate-generation stage, tolerating additional false positives that can later be filtered by schema checks and execution-time validation.
2. Context-efficient reasoning. Decomposition reduces the cognitive load of the LLM by exposing smaller, step-specific candidate sets.
3. Collision mitigation. We explicitly reduce vector-space collisions between semantically similar tools via representation shaping of tool documents (expanded descriptions and synthetic questions).

We adopt recall-first selection because false negatives at candidate generation are unrecoverable: a required tool omitted early cannot be resurrected later by reasoning. Conversely, false positives are inexpensive to prune by schema checks or during execution. This motivates our emphasis on context-assisted decomposition and document expansion, both of which empirically improve recall under semantically close noise stress.

Algorithm 1 and Figure 1 summarize the end-to-end pipeline used in our experiments and ablations. Constants follow our implementation: $M = 15$ planner tools, $K \in \{5, 10\}$

worker tools per subtask, Chroma vector store with cosine similarity, and instruction-tuned embeddings.

Algorithm 1 Unified Retrieval–Plan–Select (RPS) pipeline (recall-first).

Require: User request q; tool catalog $\mathcal{T}$; vector DB over $\{D_t\}$; budgets M, K.
Index. For all $t \in \mathcal{T}$, build D_t and add to vector DB.
Planner context. $C \leftarrow \mathsf{ANN}(q, M)$.
Decomposition. $S \leftarrow \textsc{Decompose}(q, C)$ ▷ dummy or context-based
$\mathcal{S}_{\text{sel}} \leftarrow \varnothing$
for each subtask $s \in S$ **do**
    $C_s \leftarrow \mathsf{ANN}(s, K)$
    $\Pi_s \leftarrow \textsc{WorkerLLM}(s, q, C_s)$
    $\mathcal{S}_{\text{sel}} \leftarrow \mathcal{S}_{\text{sel}} \cup \{\pi.\texttt{tool} : \pi \in \Pi_s\}$
end for
return $\mathcal{S}_{\text{sel}}$

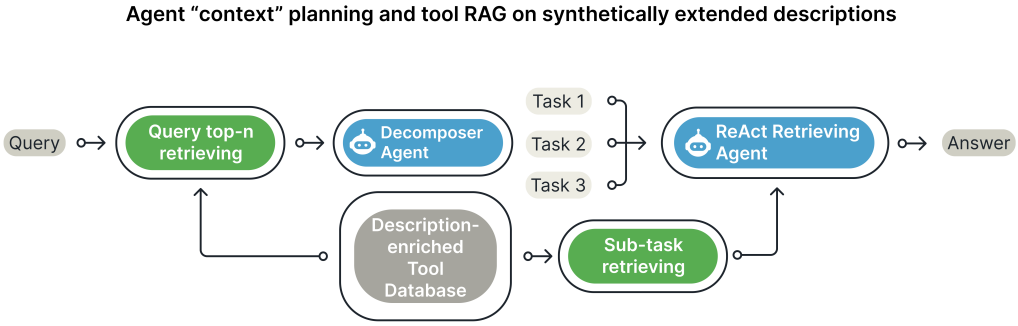


Figure 1. Proposed approach: augmented tool documents with "context" decomposition (Context+Aug).

3.1. Tool Corpus Preparation

Let $\mathcal{T}$ be the tool catalog. For each tool $t \in \mathcal{T}$ with original description d_t and argument schema A_t, we support two document variants.

We construct an augmented tool document:

$$D_t^{\text{aug}} = \big[\underbrace{\texttt{description\_expanded}(d_t)}_{\text{LLM-expanded}} \;\|\; \texttt{Arguments}(A_t) \;\|\; \texttt{SyntheticQuestions}(t)\big],$$

where `description_expanded` is a concise LLM rewrite of the vendor description that adds clarifications (capabilities, preconditions, side-effects, and typical failure modes) without changing semantics; `Arguments` is a flat, human-readable listing of required/optional parameters with types and short semantics (extracted from A_t); and `SyntheticQuestions` is a compact set of potential user queries solvable with t, generated once by an LLM and stored with the tool. (Expanded descriptions and synthetic questions reduce representation collisions between near-duplicate tools by injecting discriminative lexical/semantic cues.) Here, $\|$ denotes context concatenation.

For controlled comparisons we also use an unmodified variant

$$D_t^{\text{van}} = \big[d_t \;\|\; \texttt{Arguments}(A_t)\big],$$

i.e., the original description and argument schema without synthetic questions or LLM expansion.

All D_t (augmented by default unless otherwise stated) are embedded and indexed in a vector database (Chroma) with cosine similarity.

3.2. Vector Index and Notation

We denote by $\mathsf{ANN}(\cdot, k)$ the approximate nearest-neighbor search that returns the top-k documents under cosine similarity. Two index access patterns are used:

1. Planner retrieval with budget M (default $M = 15$): $\mathsf{ANN}(q, M)$ returns a planning context of tool documents.
2. Worker retrieval with budget K (default $K \in \{5, 10\}$): for each subtask s, $\mathsf{ANN}(s, K)$ returns a step-local candidate set.

3.3. Step-Local Selection

Our context decompositor receives q and the M nearest tools $\mathsf{ANN}(q, M)$ as planning context. The context grounds the splitter in concrete capabilities and argument schemas, producing finer-grained steps whose slots align with available tools. This is our default mode. Decomposition returns an array of natural-language sub-requests (no tool names), enabling downstream retrieval per subtask.

For each $s \in S$, we instantiate a stateless worker LLM whose input contains the current subtask s, the full original request q to preserve global constraints that might be lost during splitting, and a step-local candidate set of K tools from $\mathsf{ANN}(s, K)$. The worker outputs a JSON array of tool calls:

$$\big[\{\texttt{tool} : \tau,\ \texttt{param} : \cdot,\ \texttt{input\_source} : \text{question or tool}\}, \ldots\big],$$

where $\tau \in \mathcal{T}$. We collect the union of selected tool names across all steps. In recall-first mode we keep all candidates selected by any worker and postpone pruning to execution-time validation.

3.4. LLM Prompt Design and Template Specification

For reproducibility, we provide the core prompts used for each stage of RPS in Appendix B.

4. Experiments

The experimental setup utilized a computing platform equipped with an AMD Ryzen 7 5800X CPU, an NVIDIA GeForce RTX 4080 Super GPU, and 32 GB of system memory. LLM inference was executed remotely via the OpenRouter API.

Figure 2 displays all experimental configurations used for the ablation study. We evaluate the proposed RPS pipeline under controlled stress tests and on three public benchmarks: Ultratool, ToolLinkOS, and ToolRet. The numbers of tools and queries are presented in Table 2. The experiments are organized to answer three questions: (i) how the choice of the embedding model impacts selection quality; (ii) how retrieval and LLM selection behave under random versus semantically close noise; and (iii) how these effects translate to end-to-end performance on the benchmarks.

Table 2. Ultratool, ToolLinkOS and ToolRet-Web benchmark parameters.

Dataset	Number of Tools	Number of Queries
Ultratool	2032	1000
ToolLinkOS	573	1569
ToolRet-Web	37,292	5230

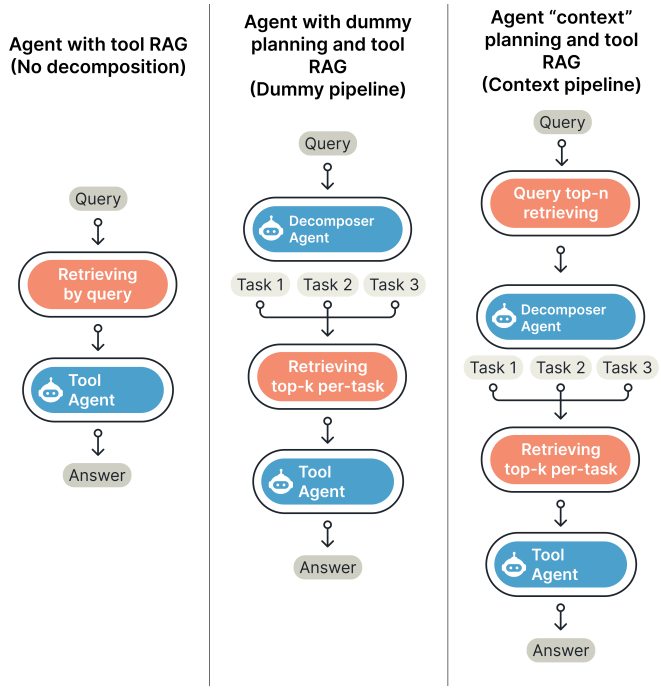


Figure 2. Compared configurations: (1) simple agent with VectorDB (no decomposition); (2) agent system with dummy planner and VectorDB (dummy pipeline); (3) agent system with "context" planner and VectorDB (context pipeline).

The same RPS backbone described in Section 3 yields all configurations we evaluate later:

1. Embedding ablation. Context decomposition with $M = 15$, augmented tool docs, and worker $K \in \{5, 10\}$; we swap the embedding model and measure selection metrics on the Ultratool subset of 100 difficult cases.
2. VectorDB retrieval test. Given a reference tool pool plus injected noise, we only run retrieval: planner $\Rightarrow$ subtasks $\Rightarrow$ for each subtask return $\mathrm{ANN}(s, K)$; union over subtasks forms the predicted set. We vary decomposition (none/dummy/context) and document expansion (vanilla vs. augmented).
3. LLM context selection test. We provide the worker LLM with a mixed candidate pool without calling the vector DB and ask it to select tools. We vary decomposition (none/dummy/context) and document expansion.
4. End-to-end. Full pipeline with retrieval and worker selection; we vary $K \in \{5, 10\}$, decomposition (context).

4.1. Setup

Indexes and models. Unless noted, we store tools in Chroma as augmented documents D_t^{aug} (Section 3). The planner retrieves $M = 15$ nearest tools as context; each worker retrieves $K \in \{5, 10\}$ nearest tools per subtask. The default chat model is `gpt-4o-mini`. For ablations that isolate the effect of document expansion, we switch to the vanilla variant D_t^{van} (original descriptions; no synthetic questions) while keeping all other settings identical.

Parameter choice. The values of M and K control two different stages of the pipeline. The planner budget M determines how many nearest tools are provided as contextual evidence for task decomposition, whereas the worker budget K determines how many tools are retrieved for each decomposed subtask. Thus, M is not part of the final candidate selection pool directly; it is used only to ground the planner in the local structure of the tool space.

We set $M = 15$ as a bounded planning-context budget rather than as a tuned hyperparameter. This value is intentionally larger than the typical number of reference tools per query in Ultratool: the mean toolset size is 2.039, the median is 2, the 90th percentile is 3, and the maximum is 7. Therefore, $M = 15$ provides the planner with several plausible tools per potential reasoning step while keeping the prompt size manageable and avoiding excessive distraction from semantically related but irrelevant tools.

The worker retrieval budget is set to $K \in \{5, 10\}$. In the decomposed setting, K is applied independently to each subtask, and the resulting tool candidates are merged before final evaluation. Therefore, $K = 5$ in the RPS pipeline should be interpreted as a step-local retrieval budget rather than as a global candidate limit. In contrast, the RAG-only baseline performs a single retrieval pass over the original query; we therefore report it with $K = 10$, a common top-k retrieval setting also used in prior tool-retrieval evaluations. All comparisons keep these budgets fixed within each experimental regime.

Decomposition modes. No decomposition: the full request is used directly. Dummy: prompt-only splitter. Context (default): splitter receives the $M = 15$ nearest tools to the original query.

Noise model. Given a reference toolset for a query, we inject $n \in \{0, 10, 20, 30, 40, 50\}$ random tools (uniformly from non-reference) or closest tools (hard negatives chosen by cosine similarity to each reference tool).

Metrics. We report macro-averaged Precision, recall, and F1 over queries. Given a reference set R and a predicted set S, $P = \frac{|R \cap S|}{|S|}$, $R = \frac{|R \cap S|}{|R|}$, and $F1 = 2PR/(P+R)$. In this domain, recall is primary: missing a required tool prevents completion, whereas superfluous tools can be pruned at execution time.

4.2. Embedding Model Selection

Because retrieval quality governs the entire pipeline, we empirically select the embedding model by running the full decomposition pipeline on the 100 most tool-intensive Ultratool queries (largest reference toolsets) where the number of reference tools is four or higher (Figure 3). We use this difficult subset because multi-tool queries are the most sensitive to retrieval failures: missing any required tool can prevent successful task completion. Selecting the most tool-intensive cases therefore provides a stronger stress test for the retriever than randomly sampling mostly single-tool or two-tool queries.

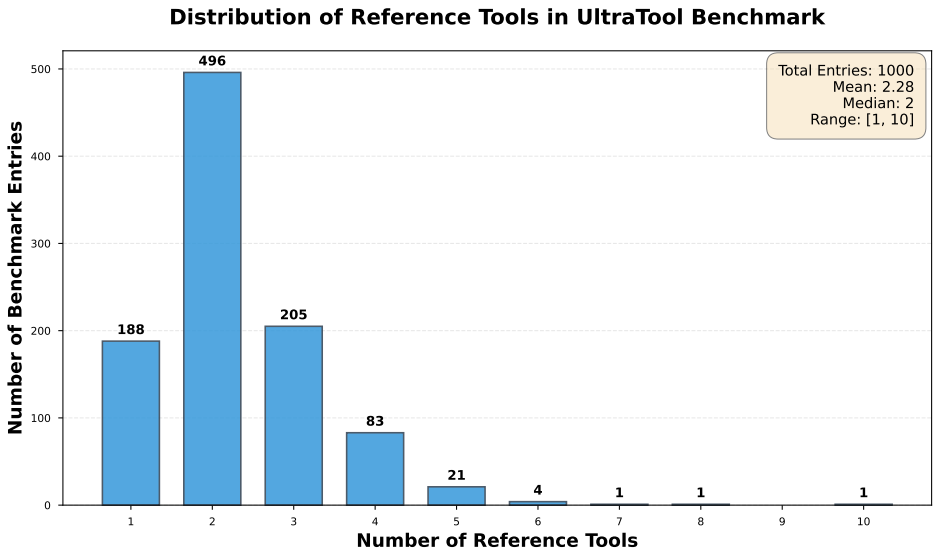


Figure 3. Distribution of reference tools in Ultratool benchmark.

To avoid selecting the retriever backbone in an ad hoc manner, we define the candidate embedding models before the main experiments. The candidate set covers several retrieval families: a lexical baseline (TF–IDF), compact dense encoders, larger dense encoders, and an instruction-tuned embedding model. The selection was also constrained by practical reproducibility requirements, including local availability, feasible inference cost, and compatibility with the same Chroma-based indexing pipeline. In addition, the candidate list was informed by prior comparisons in tool-retrieval settings such as ToolRet.

The embedding model is selected only once on the difficult Ultratool subset and is then fixed for all subsequent experiments. We do not tune the embedding backbone separately for each benchmark or experimental configuration. This ablation isolates the effect of the embedding backbone while holding prompts, indexing, and downstream selection constant.

We first fix the full pipeline (context decomposition with $M = 15$, worker $K = 5$, and augmented tool documents) and swap the embedding model. The results in Table 3 show that gte-Qwen2-1.5B-instruct achieves the best Precision and F1 while remaining competitive on recall; we adopt it henceforth.

Table 3. Embedding ablation on Ultratool 100 difficult cases (full pipeline). Best in **bold**.

Embedding Model	Precision	Recall	F1
TF–IDF	0.258	0.299	0.268
TinyBERT	0.292	0.328	0.297
BGE-Small	0.300	0.343	0.306
E5-Large	0.313	0.337	0.311
gte-Qwen2-1.5B-instruct	**0.343**	**0.363**	**0.337**

The candidate embedding models were selected to cover different retrieval regimes: a sparse lexical baseline, compact dense encoders, a larger dense encoder, and an instruction-tuned embedding model. Table 4 summarizes their main characteristics. The embedding backbone was selected once on the difficult Ultratool subset and then fixed for all subsequent experiments, avoiding benchmark-specific tuning.

Table 4. Embedding model characteristics used in the ablation study.

Embedding Model	Type	Params	Dim.	Max Tokens
TF–IDF	Sparse lexical	–	$\|V\|$	–
TinyBERT	Compact encoder	∼14 M	312	512
BGE-Small	Dense encoder	∼33 M	384	512
E5-Large	Dense encoder	∼335 M	1024	512
gte-Qwen2-1.5B-instruct	Instruction-tuned embedder	1.5B	1536	32k

4.3. Description Augmentation for Tool Selection

A critical challenge in tool-use benchmarks is the presence of semantically similar tools that are difficult to distinguish based on their descriptions alone. These semantic collisions—tool pairs with high embedding similarity—can artificially inflate or complicate performance metrics when models must select from large toolsets. We quantify this phenomenon by computing pairwise cosine similarities between tool embeddings and counting collision pairs above various similarity thresholds.

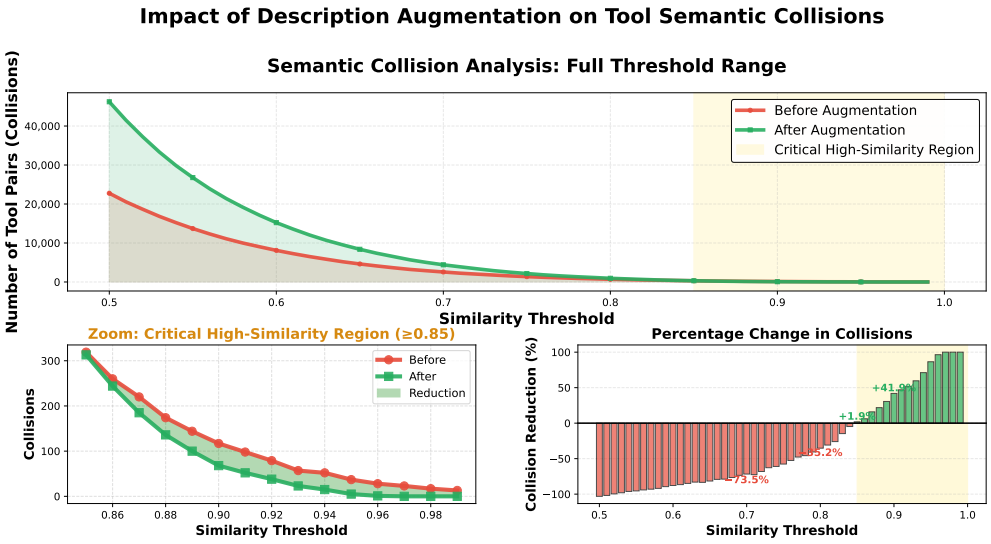
Figure 4 presents our analysis of how description augmentation affects semantic collisions across the Ultratool benchmark's 2032 tools. At low similarity thresholds (0.5–0.80), augmentation increases collision counts by 35–74%. This is expected: richer, more detailed descriptions naturally create more surface-level semantic overlap. For instance, at threshold 0.70, collisions increase from 2886 to 5008.

However, the critical insight emerges at high similarity thresholds (>0.85), where collisions represent genuinely ambiguous tool pairs that are problematic for tool selection. At these thresholds, augmentation demonstrates its value: at threshold 0.90, collisions decrease by 41.9% (from 117 to 68 pairs), and at 0.85, we observe a modest 1.9% reduction. This indicates that augmentation with expanded descriptions and synthetic questions successfully adds disambiguating context that helps differentiate highly similar tools.

The gold-highlighted region in the top panel of Figure 4 emphasizes this critical zone where reduction occurs. The bottom-left panel's detailed view shows the consistent separation between the before and after curves, with the green shaded area representing the achieved collision reduction. The percentage change visualization (bottom-center) clearly delineates the threshold at which augmentation's effect transitions from increasing general semantic richness to reducing problematic ambiguity.

This analysis validates our augmentation strategy: while expanded descriptions increase low-level semantic matches, they effectively reduce the high-similarity collisions that genuinely challenge tool selection systems. The 42% reduction in 0.90-threshold collisions represents a meaningful improvement in tool distinguishability, which we expect to positively impact downstream tool selection accuracy.

We provide examples of augmented tool descriptions illustrating different augmentation strategies applied in our experiments in Appendix C.



Threshold	Before	After	Reduction	% Change
0.70	2886	5008	−2122	−73.5%
0.75	1373	2164	−791	−57.6%
0.80	698	944	−246	−35.2%
0.85	319	313	+6	+1.9%
0.90	117	68	+49	+41.9%

Figure 4. Analysis of semantic collision reduction through tool description augmentation across similarity thresholds.

To connect collision statistics with retrieval on real user requests, we run a pure VectorDB evaluation on the full Ultratool benchmark (1000 queries), comparing an index built over vanilla documents D_t^{van} to the augmented variant D_t^{aug}. Table 5 shows consistent gains in Recall@k and nDCG$_{\text{bin}}$@k for $k \in \{5, 10\}$: augmentation increases coverage of reference tools and ranks them earlier within the top-k list. A representative heatmap case is provided in Appendix D.

Table 5. Effect of description augmentation on pure vector retrieval quality on Ultratool. Absolute and relative gains are computed with respect to vanilla tool documents. Bootstrap p-values are computed by paired resampling over queries. The best results are shown in **bold**, and p-values are shown in *italics*.

Metric	Vanilla	Augmented	Δ	Relative Δ	p_{boot}
Recall@5	0.184	**0.281**	+0.097	+52.7%	*0.005*
nDCG@5	0.160	**0.241**	+0.081	+5.7%	*0.007*
Recall@10	0.288	**0.403**	+0.115	+4.1%	*0.005*
nDCG@10	0.199	**0.289**	+0.090	+45.3%	*0.008*

In addition to recall, we report nDCG_{bin} to capture rank quality under binary relevance. Given a reference set R and the 1-based ranks $\text{rank}(t)$ in the retriever output, we compute

$$\text{DCG}_{\text{bin}}@k = \sum_{t \in R:\, \text{rank}(t) \le k} \frac{1}{\log_2(\text{rank}(t)+1)},$$

$$\text{IDCG}_{\text{bin}}@k = \sum_{i=1}^{\min(|R|,k)} \frac{1}{\log_2(i+1)},$$

$$\text{nDCG}_{\text{bin}}@k = \frac{\text{DCG}_{\text{bin}}@k}{\text{IDCG}_{\text{bin}}@k}.$$

While Recall@k measures coverage of the reference set, $\text{nDCG}_{\text{bin}}@k$ distinguishes early from late hits within the top-k window.

4.4. Tool Selection Stress Tests

We evaluate retrieval and selection under controlled candidate pools formed by combining reference tools with injected noise. To probe robustness under scale and collisions, we use two controlled noise models when forming candidate pools:

- Random noise. Inject n uniformly random tools from $\mathcal{T}^q$.
- Semantically close noise. Inject n nearest distractors to each reference tool by cosine similarity over D_t. This explicitly simulates representation collisions between look-alike tools.

We vary $n \in \{0, 10, 20, 30, 40, 50\}$ and evaluate both pure retrieval and LLM selection.

4.4.1. Random Noise, No Decomposition

Retrieving $K = 10$ tools directly from the full query (no decomposition) is robust to random noise: recall remains ≥ 0.890 even at 50 injected tools. Without decomposition, LLM selection Precision stays high while recall decays mildly with noise. See Table 6. To assess whether the observed differences are attributable to the proposed configurations rather than random variation over queries, we additionally report bootstrap-based statistical estimates. We use paired non-parametric bootstrap resampling over benchmark queries. For each bootstrap sample, we recompute macro-averaged recall and F1. The reported 95% confidence intervals correspond to the 2.5th and 97.5th percentiles of the bootstrap distribution. One-sided paired bootstrap p-values are computed for recall degradation relative to the zero-noise condition within the same stage.

For compactness, Tables 6–8 report representative noise levels $n \in \{0, 10, 30, 50\}$.

Table 6. Random-noise stress test without decomposition on Ultratool 100 difficult cases.

Noise	Stage	Recall	F1	Recall 95% CI	Recall p_{boot} vs. Noise 0
0	VectorDB retrieval	1.000	1.000	[1.000, 1.000]	–
10	VectorDB retrieval	0.992	0.493	[0.985, 1.000]	0.030
30	VectorDB retrieval	0.934	0.460	[0.897, 0.971]	<0.01
50	VectorDB retrieval	0.890	0.436	[0.843, 0.937]	<0.01
0	LLM selection	0.674	0.761	[0.642, 0.705]	–
10	LLM selection	0.645	0.741	[0.608, 0.682]	0.035
30	LLM selection	0.598	0.694	[0.554, 0.642]	<0.01
50	LLM selection	0.586	0.668	[0.543, 0.639]	<0.01

4.4.2. Semantically Close Noise

Hard negatives sharply reduce recall. Table 7 compares four retrieval settings: (A) no decomposition ($K = 10$ per full query), (B) dummy decomposition ($K = 5$ per subtask), (C) context decomposition ($K = 5$), and (D) the same as (C) but with augmented tool documents. Contextual decomposition consistently improves recall over dummy; document expansion yields the highest recall across all noise levels, indicating reduced representation collisions.

Table 7. VectorDB retrieval under semantically close hard negatives on UltraTool 100 difficult cases. The table reports recall/F1 for representative noise levels. Recall 95% confidence intervals are estimated with bootstrap resampling over queries. One-sided paired bootstrap *p*-values are computed for recall improvement over RAG-only at the same noise level. The best results are shown in **bold**.

Noise	Configuration	Recall	F1	Recall 95% CI	p_{boot} vs. RAG-Only
0	RAG-only	1.000	1.000	[1.000, 0..000]	–
10	RAG-only	0.857	0.422	[0.825, 0.889]	–
30	RAG-only	0.632	0.302	[0.588, 0.676]	–
50	RAG-only	0.597	0.283	[0.551, 0.643]	–
0	RPS-Dummy	1.000	1.000	[1.000, 0..000]	–
10	RPS-Dummy	0.848	**0.471**	[0.821, 0.875]	0.39
30	RPS-Dummy	0.732	0.336	[0.698, 0.766]	<0.01
50	RPS-Dummy	0.699	**0.305**	[0.662, 0.736]	<0.01
0	RPS-Context	1.000	1.000	[1.000, 0..000]	–
10	RPS-Context	0.859	0.449	[0.822, 0.896]	0.44
30	RPS-Context	0.763	0.324	[0.717, 0.809]	<0.001
50	RPS-Context	0.743	0.298	[0.692, 0.794]	<0.001
0	RPS-Context+Aug	1.000	1.000	[1.000, 0..000]	–
10	RPS-Context+Aug	**0.863**	0.458	[0.832, 0.894]	0.65
30	RPS-Context+Aug	**0.784**	**0.340**	[0.732, 0.836]	<0.001
50	RPS-Context+Aug	**0.752**	0.303	[0.695, 0.809]	<0.001

For LLM selection, Table 8 mirrors the retrieval variants: no decomposition, dummy, context decomposition, and context decomposition with augmented tool docs. Under semantically close noise, context decomposition improves recall the most, but longer tool descriptions can hurt selection in-context, so the best F1 sometimes comes from context without augmentation.

Table 8. LLM selection under semantically close hard negatives on UltraTool 100 difficult cases. The table reports recall/F1 for representative noise levels. Recall 95% confidence intervals are estimated with bootstrap resampling over queries. One-sided paired bootstrap p-values are computed for recall improvement over RAG-only at the same noise level. The best results are shown in **bold**.

Noise	Configuration	Recall	F1	Recall 95% CI	p_{boot} vs. RAG-Only
0	RAG-only	0.635	0.725	[0.590, 0.680]	–
10	RAG-only	0.407	0.453	[0.341, 0.473]	–
30	RAG-only	0.347	0.393	[0.272, 0.422]	–
50	RAG-only	0.323	0.364	[0.252, 0.394]	–
0	RPS-Dummy	0.665	0.750	[0.633, 0.697]	0.07
10	RPS-Dummy	0.449	0.494	[0.407, 0.491]	0.07
30	RPS-Dummy	0.388	0.433	[0.343, 0.433]	0.12
50	RPS-Dummy	0.347	0.384	[0.300, 0.394]	0.27
0	RPS-Context	**0.799**	**0.868**	[0.762, 0.836]	<0.001
10	RPS-Context	**0.557**	0.533	[0.510, 0.604]	<0.001
30	RPS-Context	**0.474**	0.433	[0.417, 0.531]	<0.01
50	RPS-Context	**0.448**	0.405	[0.396, 0.500]	<0.01
0	RPS-Context+Aug	0.773	0.854	[0.718, 0.828]	<0.001
10	RPS-Context+Aug	0.555	**0.559**	[0.507, 0.603]	<0.001
30	RPS-Context+Aug	0.463	**0.453**	[0.407, 0.519]	<0.01
50	RPS-Context+Aug	0.432	**0.422**	[0.368, 0.496]	<0.05

Hard negatives strongly depress both retrieval and selection when semantically similar tools collide in embedding space. Document expansion measurably increases retrieval recall under closest noise (Table 7), supporting our collision-mitigation hypothesis. For LLM selection, however, longer per-tool text can reduce effective recall/F1 at high noise (Table 8), consistent with context compression effects; here, context decomposition without augmentation often yields the best F1.

Semantically close noise exposes collisions: under hard negatives, context decomposition reliably raises recall in both retrieval and LLM selection. Augmented tool documents produce the highest retrieval recall under the closest noise (Table 7), supporting our disentanglement hypothesis. Augmented descriptions can lower recall at high noise; the best recall often comes from context decomposition without augmentation (Table 8).

4.5. Visualization of Embedding Spaces

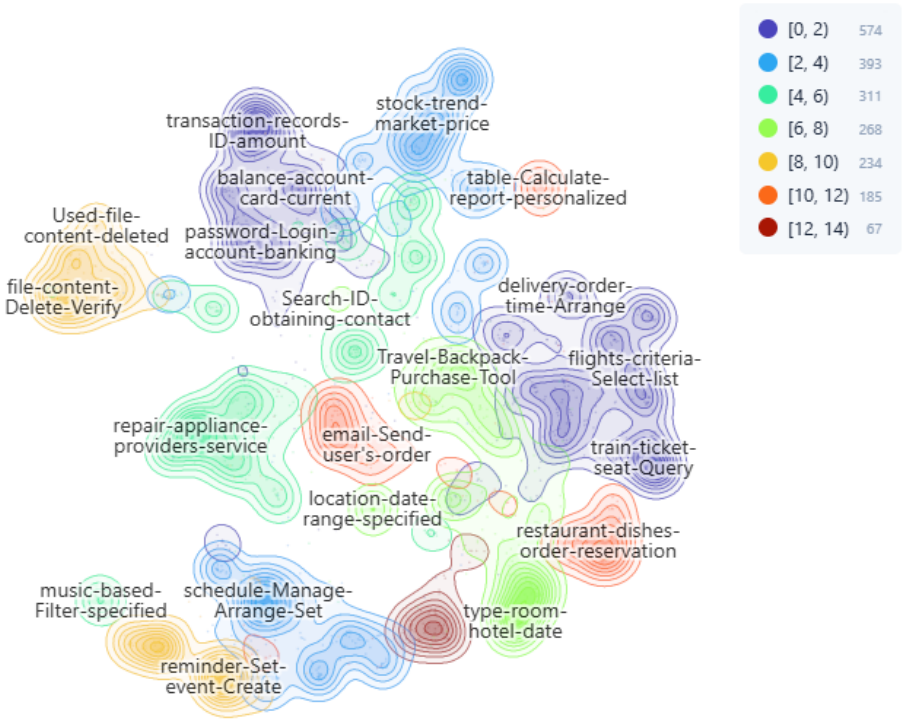
To qualitatively examine how different tool description strategies shape the vector space organization, we compute sentence embeddings for all tools using the all-MiniLM-L6-v2 encoder and apply UMAP for 2-dimensional projection (Figure 5). To illustrate semantic clustering, we visualize the UMAP projection as a density plot with colors assigned by kNN-derived local clusters. The clusters are computed from embeddings of expanded descriptions and are used only for visualization.

Colors denote kNN-derived local density clusters in the UMAP projection; they are used only to visualize neighborhood structure and do not correspond to manually assigned API categories.

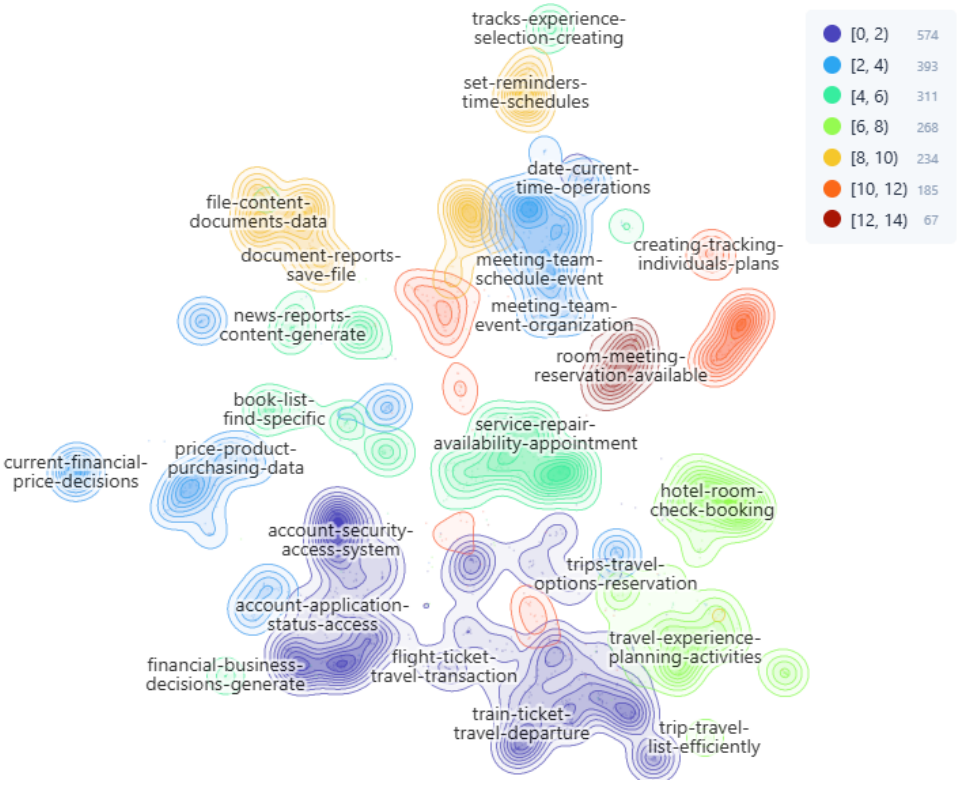
We compare three index variants:

- Vanilla descriptions (vendor-provided).
- Expanded descriptions (augmented with clarifications + synthetic questions).
- Out-of-distribution noise matched version (original descriptions padded to match expanded length using random language noise).

This setup allows us to isolate whether improvements in retrieval performance are due to semantic disentanglement, or simply due to increased token length.

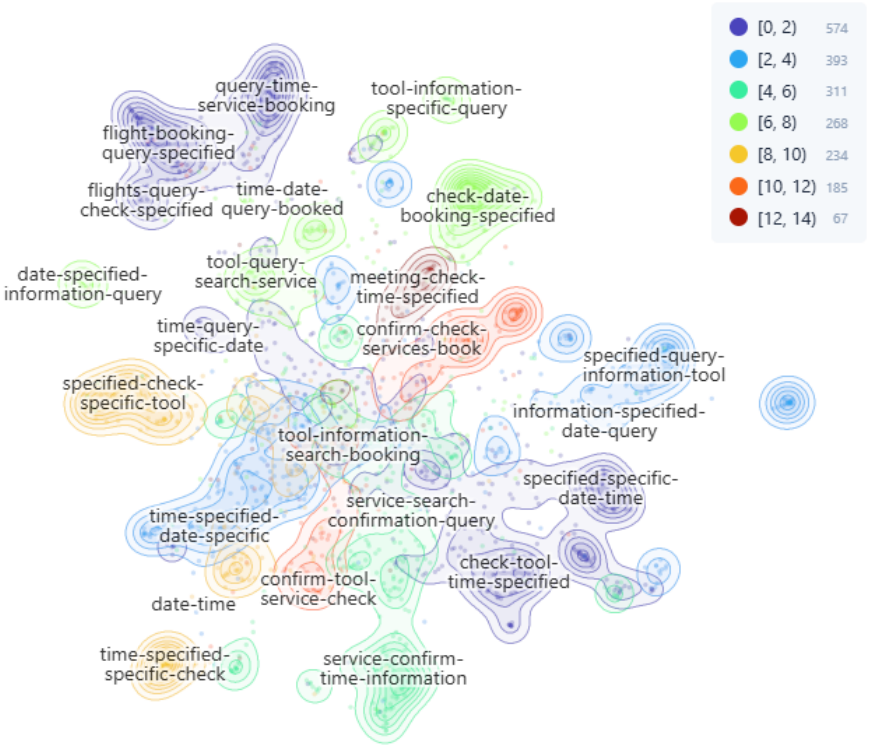


(**a**)



(**b**)

Figure 5. *Cont.*



(**c**)

Figure 5. UMAP projections of tool-description embeddings under three document variants: (**a**) vanilla tool descriptions; (**b**) expanded tool descriptions with synthetic questions, which form tighter local clusters of semantically related APIs; and (**c**) random-noise padding, where the embedding structure becomes more diffuse and less semantically organized.

The visualizations support our quantitative findings:

- Expanded descriptions produce sharper and more stable local clusters, revealing meaningful semantic structure.
- Random-noise padding leads to diffusion and cluster breakup, indicating the importance of semantic rather than syntactic expansion.

Thus, augmentation improves retrieval primarily by reducing representation collisions among semantically similar tools rather than by simply inflating description length. This tighter clustering boosts recall for tool retrieval under hard-negative stress, since gold tools and their closest paraphrases are more likely to appear in the same neighborhood of the index. At the same time, denser local neighborhoods make the candidate lists more homogeneous, which increases the difficulty of the final in-context selection step for the LLM and helps explain the recall dynamics between the retrieval and selection stages.

4.6. End-to-End Pipeline Estimation

We now run the complete pipeline (VectorDB retrieval $\rightarrow$ LLM selection). We compare three regimes: context decomposition with augmented tool documents, context decomposition with vanilla tool documents (RAG), and a no-decomposition baseline that retrieves a single candidate set per query (RAG only). We first evaluate on the 100 most tool-intensive Ultratool queries (Table 9) and then on the full Ultratool, ToolLinkOS, and ToolRet benchmarks (Table 10). Since missing a required tool is unrecoverable, we treat recall as the primary metric and use F1 as an overall summary of selection quality.

Table 9 shows that, on the hardest Ultratool cases, the iterative end-to-end configuration with augmented descriptions ($K = 5 \rightarrow 15$, multi-query, fallback) achieves the highest

recall (0.557), while adding post-selection filtering yields the best F1 (0.396) by pruning false positives at the cost of some coverage. Context-guided retrieval with vanilla descriptions and $K = 10$ reaches a recall of 0.465 and an F1 of 0.353, outperforming the no-decomposition baseline but trailing the iterative augmented pipeline in recall. The "Ultratool Reference Plan" rows, which use ground-truth decomposition plans, set an upper bound, a recall of 0.640 with a post-selection F1 of 0.465, indicating that further gains remain possible through improved planning.

Table 9. Ultratool 100 difficult cases, end-to-end. "Context" uses $M = 15$ for planning. The best results are shown in **bold**.

Configuration	Recall	F1	Recall 95% CI
End-to-End (context decomposition, augmented description)			
Context decomp., $K = 5$, max 3 loops	0.231	0.246	[0.167, 0.295]
$K = 5 \rightarrow 15$ (+5/loop), multi-query, fallback	**0.557**	0.325	[0.496, 0.618]
+ post-selection	0.424	**0.396**	[0.359, 0.489]
RAG, context decomposition			
$K = 5$ per subtask	0.435	0.336	[0.366, 0.504]
$K = 10$ per subtask	0.465	0.353	[0.392, 0.538]
RAG only (no decomposition)			
$K = 5$ from VectorDB	0.251	0.298	[0.207, 0.295]
$K = 10$ from VectorDB	0.247	0.277	[0.212, 0.282]
Ultratool Reference Plan			
$K = 5 \rightarrow 15$ (+5/loop), multi-query, fallback	0.640	0.359	[0.587, 0.693]
+ post-selection	0.492	0.465	[0.433, 0.551]

Table 10. Ultratool, ToolLinkOS, and ToolRet benchmarks, end-to-end. The best results are shown in **bold**.

Configuration	Recall	F1	Recall 95% CI
Ultratool			
End-to-End (context, augmented tools), $K = 5$	0.392	0.353	[0.333, 0.451]
RAG (context, vanilla tools), $K = 5$	**0.494**	0.329	[0.442, 0.546]
RAG only (no decomposition), $K = 10$	0.340	**0.362**	[0.305, 0.375]
ToolLinkOS			
End-to-End (context, augmented tools), $K = 5$	**0.323**	**0.438**	[0.297, 0.349]
RAG (context, vanilla tools), $K = 5$	0.313	0.429	[0.285, 0.341]
RAG only (no decomposition), $K = 10$	0.208	0.314	[0.187, 0.229]
ToolRet			
End-to-End (context, augmented tools), $K = 5$	0.343	0.214	[0.312, 0.374]
RAG (context, vanilla tools), $K = 5$	**0.347**	0.227	[0.323, 0.371]
RAG only (no decomposition), $K = 10$	0.300	**0.242**	[0.272, 0.328]

On the full benchmarks (Table 10), a similar picture emerges. For Ultratool, context decomposition with vanilla tool descriptions attains the highest recall (0.494), while the no-decomposition baseline yields the best F1 (0.362) at the cost of a lower recall (0.340), illustrating a trade-off between aggressively recovering all relevant tools and producing a tighter predicted set. On ToolLinkOS, context decomposition with augmented descriptions performs best on both recall and F1 (0.323 and 0.438), indicating that augmentation and step-local retrieval reinforce each other when the catalog size is moderate. On ToolRet, the largest catalog, context decomposition with vanilla descriptions, achieves the highest recall (0.347), whereas the no-decomposition baseline slightly surpasses it in F1 (0.242 vs. 0.227),

consistent with the observation that decomposition primarily improves coverage of relevant tools, while flatter retrieval can sometimes lead to marginally cleaner final selections.

Computational Overhead

Since RPS improves recall by adding decomposition and step-local retrieval, it introduces additional retrieval and LLM calls compared with a standard RAG-only pipeline. Table 11 summarizes this overhead analytically. Let m denote the number of decomposed subtasks, M the planner retrieval budget, and K the worker retrieval budget. Standard RAG performs one retrieval call and one LLM selection call over a single candidate set. In contrast, RPS performs one planning call and then repeats retrieval and selection for each subtask. Thus, the overhead scales approximately linearly with the number of subtasks while keeping each worker context bounded by K tools.

Table 11. Analytical comparison of retrieval and LLM-call overhead for standard RAG and RPS-style pipelines. Here m denotes the number of decomposed subtasks, M is the planner retrieval budget, and K is the worker retrieval budget.

Configuration	Retrieval Calls	LLM Calls	Tool Docs Shown	Scaling
RAG-only	1	1	K	constant
RPS-Dummy	m	$1+m$	mK	linear in m
RPS-Context	$1+m$	$1+m$	$M+mK$	linear in m
RPS-Context+Aug	$1+m$	$1+m$	$M+mK$ (aug.)	linear in m

A detailed latency and cost optimization study is beyond the scope of this paper and is left for future work, with the understanding that it depends on provider-side server availability.

5. Conclusions

Our investigation reveals that scaling tool-using agents to massive repositories faces critical challenges rooted in semantic similarity and task complexity rather than pure catalog size. The experimental results demonstrate several key findings with concrete performance metrics. Semantic collisions prove significantly more problematic than random noise in large tool catalogs. This finding has critical implications for real-world deployments, where API catalogs such as Microsoft Graph, Google Workspace, or large MCP-based tool repositories contain numerous functionally similar tools that confuse both retrieval systems and final selection mechanisms.

Query decomposition emerges as a crucial technique for complex scenarios, providing substantial recall improvements from 0.208 to 0.323 on ToolLinkOS, from 0.340 to 0.494 on Ultratool, and from 0.300 to 0.347 on ToolRet. These consistent gains across different benchmarks indicate that breaking down complex requests into step-local tool selection problems effectively addresses the cognitive load of large candidate sets.

Across all benchmarks, our results highlight a complementary interaction between representation shaping and stepwise planning. Augmenting tool descriptions with expanded summaries and synthetic questions primarily improves retrieval by reducing representation collisions among semantically similar tools rather than by simply increasing description length. This reshaping of the embedding space yields tighter local clusters and higher recall under hard-negative stress, because gold tools and their closest paraphrases are more likely to occupy the same neighborhood of the index. At the same time, denser local neighborhoods make candidate lists more homogeneous, which increases the difficulty of the final in-context selection step for the LLM and helps explain the recall dynamics between the retrieval and selection stages. Taken together, these findings support a practical, modular

framework in which representation shaping and decomposition are tuned jointly to balance retrieval coverage and downstream discriminability in large tool ecosystems.

Future work should focus on adaptive reranking mechanisms, cross-model validation, and closed-loop evaluation of task completion rates in dynamic tool environments. The measurement protocols and stress-testing framework introduced here offer a foundation for systematic evaluation of tool selection systems at scale.

6. Limitations

This work emphasizes problem framing and measurement, and our improvements, while consistent, are modest. The effectiveness of disentanglement depends on the quality and stability of tool metadata and synthetic expansions, which may vary across providers and over time. Multi-stage retrieval and decomposition introduce additional latency and cost, and our fixed budgets (M, K) are not learned or adaptively tuned. Evaluations focus on selection under static catalogs; they do not measure closed-loop task success, safety, or performance under rapidly changing tool inventories and strict real-time constraints. Finally, results are contingent on specific embedding and chat models and prompt designs; alternative models or finetuning could shift trade-offs, suggesting the need for broader cross-model validation and adaptive reranking in future work.

We note that direct end-to-end comparison with recent retrieval-oriented systems such as Toolshed and ProTIP remains challenging due to limited reproducibility of the published pipelines. Toolshed relies on a proprietary tool knowledge-base construction process and does not provide an open-source implementation of the complete retrieval stack, while ProTIP currently lacks a publicly available end-to-end evaluation framework.

To mitigate this limitation, we position our evaluation primarily around public large-scale benchmarks (Ultratool, ToolLinkOS, and ToolRet) and additionally reuse ToolRet-style embedding comparisons when selecting retrieval backbones.

Author Contributions: Conceptualization, J.K., A.K. (Anna Kalyuzhnaya), I.G.; methodology, J.K., I.G., A.K. (Anna Kalyuzhnaya), S.C.; software, J.K., I.G., S.C.; validation, J.K., I.G., S.C., A.K. (Artem Kuznetsov); formal analysis, I.G., J.K., A.K. (Anna Kalyuzhnaya); investigation, J.K., I.G., S.C., A.K. (Anna Kalyuzhnaya); resources, A.K. (Anna Kalyuzhnaya); data curation, J.K., I.G., S.C.; writing—original draft preparation, J.K., I.G., A.K. (Artem Kuznetsov), A.K. (Anna Kalyuzhnaya), S.C.; writing—review and editing, A.K. (Anna Kalyuzhnaya), A.K. (Artem Kuznetsov), I.G.; visualization, J.K., A.K. (Artem Kuznetsov); supervision, A.K. (Anna Kalyuzhnaya); project administration, Anna.K.; funding acquisition, A.K. (Anna Kalyuzhnaya). All authors have read and agreed to the published version of the manuscript.

Funding: This research is financially supported by the Russian Science Foundation, Agreement No. 24-71-10093, https://rscf.ru/en/project/24-71-10093/ (accessed on 9 April 2026).

Data Availability Statement: Source code is available on GitHub platform: https://github.com/jrzkaminski/too-many-tools-supplementary (commit c4254ab, accessed on 25 November 2025).

Acknowledgments: The authors acknowledge the use of Claude Sonnet 4.5 and GPT-4o (Anthropic and OpenAI, respectively) for grammar correction, stylistic refinement, and text preparation during the writing process. All AI-assisted content underwent thorough human review and revision. The authors assume complete responsibility for the content, accuracy, and conclusions presented in this article.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

LLM	Large Language Model
RAG	Retrieval-Augmented Generation
IR	Information Retrieval
CPU	Central Processing Unit
GPU	Graphics Processing Unit
API	Application Programming Interface
DB	Database
UMAP	Uniform Manifold Approximation and Projection
RPS	Retrieval–Plan–Select
JSON	JavaScript Object Notation
MCP	Model Context Protocol
ANN	Approximate Nearest-Neighbor search

Appendix A. Extended Comparison of Related Methods

Table A1. Extended comparison of tool retrieval, representation, and benchmarking methods.

Method	Core Idea	Retrieval Strategy	Decomposition Usage	Synthetic Descriptions	Strengths	Limitations
Toolformer [7]	Self-supervised learning of when and how to call external tools.	No explicit large-scale retrieval stage; tools are used in a constrained setting.	Not central.	No.	Foundational paradigm for tool-augmented language modeling.	Not designed for retrieval from large heterogeneous tool repositories.
ToolNet [8]	Tool selection through directed dependency graphs.	Selection is guided by predefined or constructed graph relations.	Partially, through graph-structured dependencies.	No.	Captures structured relations between tools and supports multi-tool reasoning.	Requires available or constructed tool relationships; scaling to very large repositories is limited.
ToolRet [3]	Benchmarking large-scale tool retrieval as a bottleneck for tool-using LLMs.	IR-style retrieval over a large tool collection.	No explicit decomposition mechanism.	No.	Demonstrates that retrieval quality strongly limits downstream tool-use success.	Primarily a benchmark and analysis framework rather than a complete retrieval-selection pipeline.
NESTools [4]	Evaluation of nested tool use where one tool output becomes another tool input.	Uses controlled distractor tools for evaluation.	Captures nested tool dependencies but not query decomposition for retrieval.	No.	Useful for studying multi-step tool dependency and compositional tool use.	Does not directly model dense embedding-space semantic collisions.
Toolshed [5]	Advanced RAG-style tool retrieval through enriched tool documentation.	Enhanced tool documents, tool knowledge bases, query expansion, and reranking.	Yes; uses query decomposition and multi-query expansion.	Yes.	Strongly improves retrieval by combining document enrichment with advanced retrieval and reranking.	Closely related to RPS, but mainly framed as RAG-tool fusion rather than recall-first retrieval with explicit retrieval-selection trade-off analysis.

Table A1. *Cont.*

Method	Core Idea	Retrieval Strategy	Decomposition Usage	Synthetic Descriptions	Strengths	Limitations
ProTIP [6]	Learning prototypical tool representations aligned with descriptions and usage traces.	Contrastive alignment between tools and queries in a shared embedding space.	Not central.	No.	Learns stronger semantic representations for tool matching.	Focuses on representation learning rather than combining decomposition, synthetic descriptions, and recall-first candidate generation.
RPS (ours)	Recall-first pipeline for scalable tool retrieval and selection.	Contextual step-local candidate generation before downstream selection.	Yes; decomposes user requests into contextual subqueries.	Yes.	Combines decomposition, description enhancement, and recall-oriented candidate generation for large semantically dense repositories.	Adds preprocessing and retrieval-time complexity; downstream selection may still be affected by denser candidate neighborhoods.

Table A2. Extended comparison of tool orchestration, coordination, and downstream tool-use methods.

Method	Core Idea	Retrieval Strategy	Decomposition Usage	Synthetic Descriptions	Strengths	Limitations
Think-then-Act [9]	Improving tool use through explicit reasoning before action.	Planning-oriented selection in a constrained tool setting.	Yes, at the reasoning and planning level.	No.	Strengthens sequential decision-making and tool-use reasoning.	Does not primarily target first-stage recall in very large tool repositories.
UniHGKR [11]	Unified heterogeneous knowledge retrieval.	Retrieval over heterogeneous knowledge sources and structures.	Not primarily designed as tool-query decomposition.	No.	Useful for integrating different knowledge structures into retrieval.	Focuses on heterogeneous knowledge retrieval rather than large-scale tool retrieval and selection.
Toolink [12]	Tool use through chain-of-solving paradigms.	Selects or organizes tools within a multi-step solving process.	Yes, at the chain-of-solving level.	No.	Supports structured multi-step tool use and reasoning.	Does not directly address recall degradation in large tool catalogs.
Chain of Tools [10]	Multi-tool orchestration through chained tool execution.	Focuses on selecting and executing tool chains.	Yes, through sequential tool planning.	No.	Effective for modeling dependencies among tool calls during execution.	Assumes relevant tools can be found or provided; retrieval recall is not the central focus.

Table A2. *Cont.*

Method	Core Idea	Retrieval Strategy	Decomposition Usage	Synthetic Descriptions	Strengths	Limitations
XAgents [13]	Multi-agent cooperation for tool utilization.	Tool access and coordination through agent roles and rules.	Partially, via role-based task division.	No.	Provides a coordination mechanism for complex agentic workflows.	Usually assumes predefined domains or constrained tool environments.
Iterative feedback retrieval [17]	Refining retrieval queries based on execution feedback.	Iterative query refinement after retrieval or execution outcomes.	Partially; refinement is feedback-driven rather than decomposition-centered.	No.	Improves retrieval over single-pass methods and can complement agentic pipelines.	Adds iterative overhead and does not focus on synthetic tool representation enhancement.
ToolExpNet [18]	Tool similarity and dependency modeling through experience networks.	Selection based on simulated or learned agent–tool interaction experience.	Partially, through dependency-aware multi-tool selection.	No.	Captures inter-tool relations and improves downstream selection quality.	Focuses more on dependency-aware selection than on first-stage recall from large repositories.

Appendix B. LLM Prompts

Appendix B.1. Tool Description Expansion Prompt

Tool description expansion prompt

Expand the following tool description in 5 sentences, focusing on what the tool does, typical use-cases, and important details. Then propose {*n_questions*} diverse user requests that could be solved *using only this tool*.

Return the response strictly as valid JSON without additional text or code fences. Use this format: {"*name*" : "*str*","*description_expanded*" : "*str*","*synthetic_questions*" : ["*str*",..."]}

Tool name: "{*tool_name*}"
Original description: "{*original_description*}"
Arguments schema: {*arguments_schema*}
Results schema: {*results_schema*}.
Return the response in JSON format only, with no additional text or explanations.

Appendix B.2. Planner Agent System Prompt

Planner agent system prompt

Rewrite the USER REQUEST as the smallest sequence of independent, solvable sub-requests.

Context: you have access to 15 tools provided in a separate context section. Use them to think how the request can be decomposed so that every sub-request can be solved by SOME tool from the list. Split as aggressively as possible – the more fine-grained the better (but preserve logical order).

Rules:
1. Each sub-request must be a self-contained natural-language instruction; no code or tool names.
2. Preserve order and any quoted literals (file names, texts, numbers).
3. Return ONLY a JSON array of strings. No keys, no commentary.

User request: "{*user_request*}"

Appendix B.3. Execution Agent System Prompt

Agent system prompt

You are an execution agent working on a SINGLE sub-task: "{*current_subtask*}".
Full-task for context: "{*user_request*}"

Available tools:
{*tool_descriptions*}

Select the best tool(s) (one or many or all) to accomplish the sub-task. For tools that perform the same high-level action (e.g., file_write vs create_document) include ALL candidates.

Return ONLY a JSON array of tool calls following the format:

```
[
  {
    "tool" : "tool_name",
    "param" : {...}
  },
  ...
]
```

Do NOT output any commentary.

Appendix C. Augmented Tool Descriptions

Pair A: `set_schedule_priority` vs. `set_schedule_tag`

Vanilla (vendor) descriptions.

```
set_schedule_priority
  description: "Set priority for schedule"

set_schedule_tag
  description: "Set a tag for the schedule"
```

Augmented description + synthetic questions (indexed).

```
set_schedule_priority
  description_expanded:
    "The set_schedule_priority tool allows users to assign a specific priority
      level to a particular schedule, ensuring that critical tasks are addressed
      promptly. This tool is essential for project managers and team leaders who
      need to allocate resources effectively based on task urgency. Typical
      use-cases include setting deadlines for project milestones, prioritizing
      client meetings, or managing personal tasks in a busy schedule. By defining
      a clear priority, users can streamline their workflows and focus on the most
      important activities first. Additionally, this tool provides feedback on
      whether the priority assignment was successful, allowing users to confirm
      changes instantly."
  synthetic_questions:
    - "Can you set the priority of the team meeting to high?"
    - "I need to prioritize my project deadline; can you help me with that?"
    - "How do I change the priority of my schedule for next week's tasks?"
    - "Can you confirm if the priority for the client presentation was set
      successfully?"
    - "I want to ensure that my weekly planning session is marked as urgent; can
      you do that?"

set_schedule_tag
  description_expanded:
    "The 'set_schedule_tag' tool is designed to assign specific tags to schedules,
      enhancing organization and retrieval of information. By utilizing a tagging
      system, users can categorize their schedules based on various criteria such
      as project type, priority, or status. Typical use-cases include labeling
      schedules for team projects, differentiating between personal and
      work-related tasks, or highlighting urgent deadlines. This tool allows users
      to define tags with an ID, name, color, and description, providing
      comprehensive details for each tag. The outcome of using this tool indicates
      whether the tagging process was successful, ensuring users can effectively
      manage their scheduling needs."
  synthetic_questions:
    - "Can you help me set a tag named 'Urgent' with a red color for my project
      schedule titled 'Q4 Planning'?"
    - "I need to add a green tag called 'Completed' to my meeting schedule titled
      'Weekly Team Sync', can you do that?"
    - "Please create a tag with ID 'proj123', name 'Research', and description
      'Time allocated for research activities' for my schedule
       'Project Development'."
    - "How can I set a blue tag named 'Review' for the schedule titled 'Client
      Feedback Meeting'?"
    - "I want to categorize my 'Marketing Campaign' schedule with a yellow tag
      called 'Pending Approval', could you assist me with that?"
```

Pair B: `set_schedule_date_range` vs. `set_schedule_repetition`

Vanilla (vendor) descriptions.

```
set_schedule_date_range
  description: "Set the start and end dates of the schedule"

set_schedule_repetition
  description: "Set the repetition type and end date of the schedule"
```

Augmented description + synthetic questions (indexed).

```
set_schedule_date_range
  description_expanded:
    "The 'set_schedule_date_range' tool is designed to establish a defined period
      for scheduling events by setting both a start and end date. This tool is
      particularly useful in scenarios where project timelines, appointments, or
      any scheduled activities need to be clearly defined to avoid conflicts and
      ensure proper planning. Users can set the title for the schedule, which
      helps in identifying the purpose of the scheduled events easily. The tool
      ensures that all scheduling adheres to the specified dates, improving
      organization and time management. A successful operation returns a status
      indicating whether the date range was accurately established, providing
      immediate feedback on the action taken."
  synthetic_questions:
    - "Can you set the schedule for my project planning from January 10 to
      January 20?"
    - "I need to create a schedule titled 'Team Meetings' starting on February 1
      and ending on February 28; can you do that?"
    - "Please set the date range for my training sessions from March 5 to
      March 12."
    - "Can you establish a schedule for the annual review process with a start
      date of April 1 and an end date of April 15?"
    - "I want to set up a schedule for the conference preparation from May 10 to
      May 20; can you assist with that?"

set_schedule_repetition
  description_expanded:
    "The set_schedule_repetition tool is designed to establish a recurring
      schedule by allowing users to set the type of repetition and specify an end
      date for that schedule. This tool is particularly useful in various
      scenarios such as planning regular meetings, setting up reminders for
      recurring tasks, or organizing events that need to happen at fixed
      intervals, such as weekly training sessions or monthly reports. Users can
      select different repetition types, such as daily, weekly, or monthly, to fit
      their scheduling needs. Additionally, specifying an end date helps in
      managing the duration of the recurrence, ensuring that users have control
      over how long the repetition will last. Overall, this tool simplifies the
      process of creating and managing repetitive schedules, making it easier for
      users to stay organized and on track."
  synthetic_questions:
    - "Can you set a weekly meeting to repeat every Monday until the end of
      the year?"
    - "How do I schedule a monthly report submission that ends in six months?"
    - "I need to set a daily reminder for a task that should repeat until the end
      of next month, can you help?"
    - "Can you create a bi-weekly training schedule that runs for three months?"
    - "How can I set up a quarterly review meeting that repeats until December?"
```

Appendix D. Augmentation Heatmaps

Query:

First, I need to check the progress of my credit card application under my ID card, then check the balance of my ICBC (Industrial and Commercial Bank of China) card, and inquire about the debt amount of my China Merchants Bank credit card. Finally, I want to use the balance of my ICBC card to pay off the 500 yuan debt on my China Merchants Bank credit card.

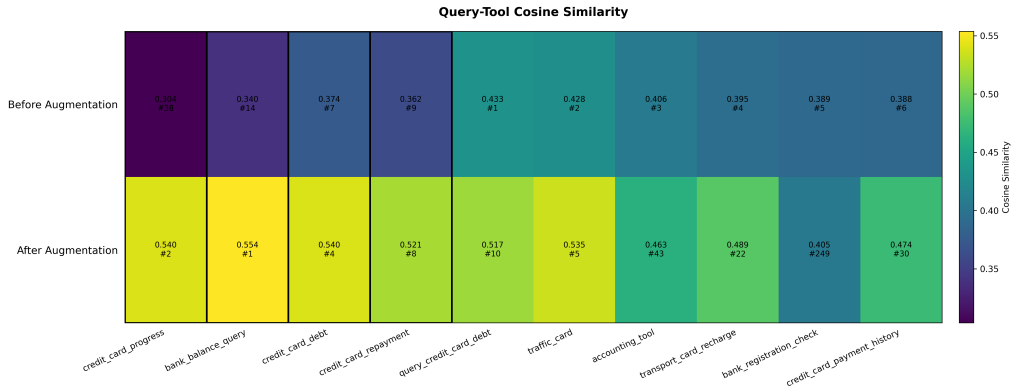


Figure A1. Query→Tool cosine similarity before vs. after augmentation.

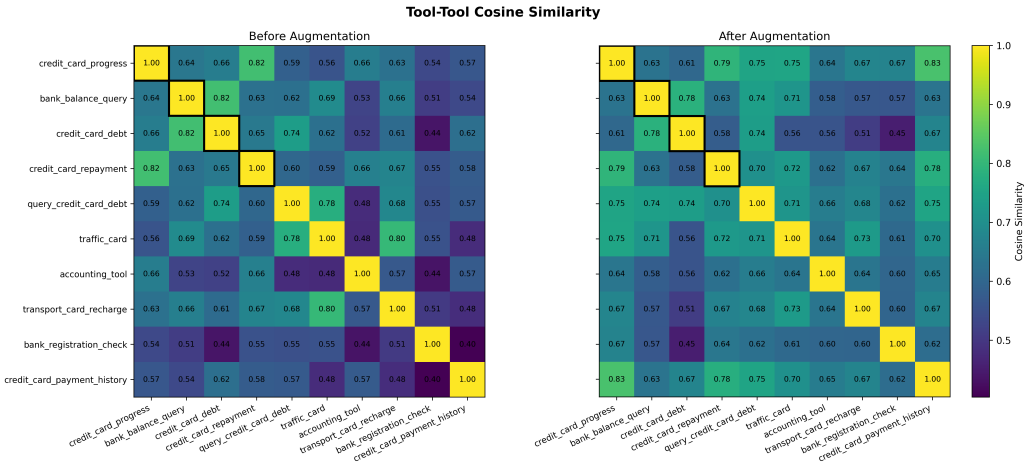


Figure A2. Tool→Tool cosine similarity before vs. after augmentation.

Appendix E. Benchmark Datasets and Licenses

This section details the licensing information for the benchmark datasets utilized in our experimental evaluation.

- The Ultratool benchmark is distributed under the Apache 2.0 License.
- The ToolLinkOS benchmark is released under the MIT License.
- The ToolRet benchmark is available under the Apache 2.0 License.

References

1. Huang, S.; Zhong, W.; Lu, J.; Zhu, Q.; Gao, J.; Liu, W.; Hou, Y.; Zeng, X.; Wang, Y.; Shang, L.; et al. Planning, Creation, Usage: Benchmarking LLMs for Comprehensive Tool Utilization in Real-World Complex Scenarios. *arXiv* **2024**, arXiv:2401.17167. [CrossRef]
2. Lumer, E.; Basavaraju, P.H.; Mason, M.; Burke, J.A.; Subbiah, V.K. Graph RAG-Tool Fusion. *arXiv* **2025**, arXiv:2502.07223. [CrossRef]

3. Shi, Z.; Wang, Y.; Yan, L.; Ren, P.; Wang, S.; Yin, D.; Ren, Z. Retrieval Models Aren't Tool-Savvy: Benchmarking Tool Retrieval for Large Language Models. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics, Vienna, Austria, 27 July–1 August 2025.
4. Han, H.; Zhu, T.; Zhang, X.; Wu, M.; Xiong, H.; Chen, W. NesTools: A Dataset for Evaluating Nested Tool Learning Abilities of Large Language Models. *arXiv* **2024**, arXiv:2410.11805.
5. Lumer, E.; Subbiah, V.K.; Burke, J.A.; Basavaraju, P.H.; Huber, A. Toolshed: Scale Tool-Equipped Agents with Advanced RAG-Tool Fusion and Tool Knowledge Bases. *arXiv* **2024**, arXiv:2410.14594.
6. Anantha, R.; Bandyopadhyay, B.; Kashi, A.; Mahinder, S.; Hill, A.W.; Chappidi, S. ProTIP: Progressive tool retrieval improves planning. *arXiv* **2023**, arXiv:2312.10332. [CrossRef]
7. Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Hambro, E.; Zettlemoyer, L.; Cancedda, N.; Scialom, T. Toolformer: Language models can teach themselves to use tools. *Adv. Neural Inf. Process. Syst.* **2023**, *36*, 68539–68551.
8. Liu, X.; Peng, Z.; Yi, X.; Xie, X.; Xiang, L.; Liu, Y.; Xu, D. ToolNet: Connecting large language models with massive tools via tool graph. *arXiv* **2024**, arXiv:2403.00839. [CrossRef]
9. Shen, Y.; Jiang, H.; Qu, H.; Zhao, J. Think-then-Act: A Dual-Angle Evaluated Retrieval-Augmented Generation. *arXiv* **2024**, arXiv:2406.13050.
10. Shi, Z.; Gao, S.; Chen, X.; Feng, Y.; Yan, L.; Shi, H.; Yin, D.; Chen, Z.; Verberne, S.; Ren, Z. Chain of tools: Large language model is an automatic multi-tool learner. *arXiv* **2024**, arXiv:2405.16533.
11. Min, D.; Xu, Z.; Qi, G.; Huang, L.; You, C. UniHGKR: Unified Instruction-aware Heterogeneous Knowledge Retrievers. *arXiv* **2024**, arXiv:2410.20163.
12. Qian, C.; Xiong, C.; Liu, Z.; Liu, Z. Toolink: Linking toolkit creation and using through chain-of-solving on open-source model. *arXiv* **2023**, arXiv:2310.05155.
13. Yang, H.; Gu, M.; Zhao, R.; Hu, F.; Deng, Z.; Chen, Y. XAgents: A Framework for Interpretable Rule-Based Multi-Agents Cooperation. *arXiv* **2024**, arXiv:2411.13932.
14. Sprigler, A.; Drobek, A.; Weinstock, K.; Tapsoba, W.; Childress, G.; Dao, A.; Gral, L. Synergistic Simulations: Multi-Agent Problem Solving with Large Language Models. *arXiv* **2024**, arXiv:2409.13753. [CrossRef]
15. Huang, Y.; Cheng, F.; Zhou, F.; Li, J.; Gong, J.; Yang, H.; Fan, Z.; Jiang, C.; Xue, S.; Chen, F. Romas: A role-based multi-agent system for database monitoring and planning. *arXiv* **2024**, arXiv:2412.13520.
16. Liu, T.; Wang, X.; Huang, W.; Xu, W.; Zeng, Y.; Jiang, L.; Yang, H.; Li, J. Groupdebate: Enhancing the efficiency of multi-agent debate using group discussion. *arXiv* **2024**, arXiv:2409.14051.
17. Xu, Q.; Li, Y.; Xia, H.; Li, W. Enhancing tool retrieval with iterative feedback from large language models. *arXiv* **2024**, arXiv:2406.17465. [CrossRef]
18. Zhang, Z.; Chen, Z.; Zhu, H.; Chen, Z.; Du, N.; Li, X. ToolExpNet: Optimizing multi-tool selection in llms with similarity and dependency-aware experience networks. In Proceedings of the Findings of the Association for Computational Linguistics: ACL 2025, Vienna, Austria, 27 July–1 August 2025; pp. 15706–15722.