HiDR: Hierarchical Decomposition Retrieval for Multi-Tool Discovery in Large-Scale MCP Ecosystems

Anonymous ACL submission

Abstract

The rapid growth of the Model Context Protocol(MCP) ecosystem brings heavy token and context-window pressure when every tool is injected into the LLM context. Anthropic's Tool Search Tool eases this by fetching tools on demand. Yet its built-in BM25 mode is weak on multi-intent queries, where the task is to assemble a tool set drawn from many servers that covers every sub-intent. Drawing on Retrieval-Augmented Generation(RAG) in text retrieval and on the execution characteristics of tool calls, we tackle this **multi-tool discovery** problem. We argue that **multi-tool discovery must achieve per-sub-intent independence and at the same time keep execution information coherent across sub-intents**. Following this principle, we propose **HiDR (Hierarchical Decomposition Retrieval)**. Small Language Model(SLM) decomposer splits a complex query into semantically independent sub-queries. A two-level Server→Tool index supports global server retrieval. A Scope Memory shares the confirmed servers across sub-intents, and the tool selector matches tools inside the union of the per-sub-query retrieved servers and the confirmed servers. On the MCP-Bench multi-tool test set, HiDR's T-F1 substantially exceeds both BM25 and existing RAG baselines. The result holds across four 7–8B SLM backbones. Code and Dataset: [1]

1 Introduction

LLM agents have become a mainstream paradigm of AI applications (Wölflein et al., 2025; Schick et al., 2023). The Model Context Protocol (MCP) (Anthropic, 2024a,b) standardises tool exposure and drives rapid ecosystem growth (Jing et al., 2025; Hou et al., 2025). For example, MCP.so already hosts more than 21,000 servers (idoubi, 2026). Injecting every available tool into the LLM context, however, brings significant token cost and context-window pressure. Anthropic reports that loading all tools at once can consume up to 134K tokens. In November 2025, Anthropic released the Tool Search function to fetch a tool subset on demand. It ships with BM25 and regex modes, but both are limited under multi-intent queries (Anthropic, 2025). In practice, a single agent often faces hundreds of servers and thousands of tools. Assembling a tool-call set that covers every sub-intent of the task, i.e., the **multi-tool discovery** problem, has become a key bottleneck of agent performance.

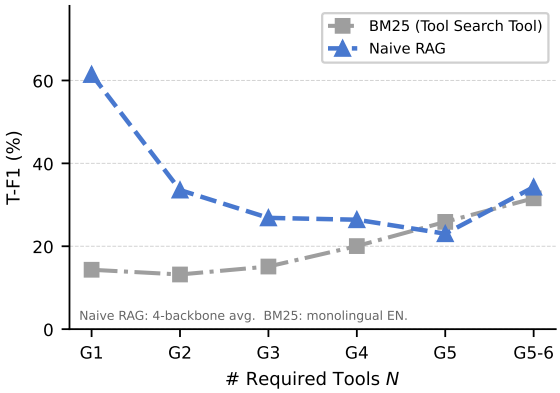

Figure 1: **Pre-experiment.** BM25 (Tool Search Tool's default) and Naive RAG both fail on multi-tool queries. BM25 stays below 30%, while Naive RAG fits a p^N curve ($p \approx 0.67$). Naive RAG is averaged over four SLM backbones. BM25 is LLM-independent.

To address the limitation of the BM25 mode in Tool Search, we draw on the RAG paradigm in text retrieval. A direct application of RAG stores tool descriptions in a vector database for semantic similarity retrieval (Shi et al., 2025; Qin et al., 2024; Jiang et al., 2023), and adopts multi-round retrieve-and-reason for complex queries (Yao et al., 2023; Trivedi et al., 2023; Singh et al., 2025a). Yet this naive approach compresses a multi-intent query

[1] https://anonymous.4open.science/r/HiDR-F308

into a single vector, so the semantic signals of sub-intents interfere with each other (Zhou et al., 2023; Liu et al., 2024b), and the multi-tool discovery task fails (Figure 1). A naive multi-round retrieve-and-reason loop further suffers from cross-step context contamination (Liu et al., 2024a; Trivedi et al., 2023). Both limitations come from ignoring the natural Server→Tool hierarchy of the MCP ecosystem and the decomposability of multi-MCP tasks (Wang et al., 2024; Fan et al., 2024).

We argue that multi-tool retrieval must preserve per-sub-intent independence, so that multi-intent queries are not compressed into a single vector and sub-intents do not interfere. It must also let servers needed by earlier sub-intents be reused directly by later ones, which achieves cross-sub-intent server coherence. Based on this principle, we propose **HiDR (Hierarchical Decomposition Retrieval)**, built from three mechanisms: a Small Language Model (SLM) driven query decomposer, a cross-subquery Scope Memory, and an L1/L2 Server→Tool cascaded index. On MCP-Bench, HiDR substantially outperforms all baselines, and the result holds across Qwen2-7B, Qwen2.5-7B, Llama-3.1-8B and GPT-4o-mini.

The main contributions of this paper are:

- We identify the core bottleneck of multi-tool discovery in MCP. Tool Search Tool's built-in BM25 and single-vector RAG both fail to preserve the independent signal of each sub-intent under multi-intent queries. Neither exploits the Server→Tool hierarchy to reuse server scope across sub-intents. Based on this, we propose the core principle that multi-tool retrieval must simultaneously preserve sub-intent independence and cross-sub-intent server coherence.

- We propose the **HiDR** framework, realised by three cooperating mechanisms that follow the principle above. *An SLM-driven query decomposer* preserves the independence of each sub-intent, so multi-intent queries are no longer compressed into a single vector. *A cross-subquery Scope Memory* preserves cross-sub-intent server coherence by accumulating the servers confirmed in earlier sub-queries, so that later sub-queries can reuse them. *An L1/L2 Server→Tool cascaded vector index* supports hierarchical retrieval that locates servers first and then searches tools inside them.

- We construct and release **MCP-Bench**, built on top of the MCP-tools (Fei et al., 2025) tool corpus, with 1,000 queries stratified by required tool count, covering 308 real MCP servers and 2,797 tools.

2 Related Work

Early work on LLM tool discovery is centred on model training. Toolformer (Schick et al., 2023) uses self-supervision to decide when to call which tool, and ToolLLM (Qin et al., 2024) scales this through large-scale tool-data training and evaluation. Both methods require retraining or fine-tuning whenever the tool inventory changes, which does not match the fast pace of real tool ecosystems. Now MCP is a standard protocol to connect LLMs with various tools. Anthropic released the Search Tool. Its built-in BM25 and regex modes fail to handle scenarios with multiple intents and multiple tools.s (Anthropic, 2025).

To further improve tool search, we draw on the RAG paradigm. Text retrieval first relied on keyword matching such as BM25, and the rise of dense vector retrieval led to RAG being widely adopted to extend the LLM context (Lewis et al., 2020; Guu et al., 2020; Dong et al., 2025; Patil et al., 2024). Some methods introduce hierarchy on top of RAG. RAPTOR (Sarthi et al., 2024), TreeRAG (Tao et al., 2025) and HippoRAG (Gutiérrez et al., 2024) build hierarchies via document semantic clustering, but those hierarchies differ from the Server→Tool structure of MCP and cannot be reused for tool-scope pruning. AnyTool (Du et al., 2024) introduces hierarchical API candidate filtering with self-reflection, but feeds the query as a whole and does not perform multi-intent decomposition. It is designed to find one matching API per call, so small models struggle once a query needs several APIs together.

For multi-hop tasks, IRCoT (Trivedi et al., 2023), ReAct (Yao et al., 2023), Adaptive-RAG (Jeong et al., 2024) and Self-RAG (Asai et al., 2024) handle complex queries through multi-round reason-and-retrieve loops (Singh et al., 2025b; Zhuang et al., 2024). IRCoT accumulates retrieved passages across rounds inside the prompt as a passage-content-grained state, but each round still operates independently over the entire open-

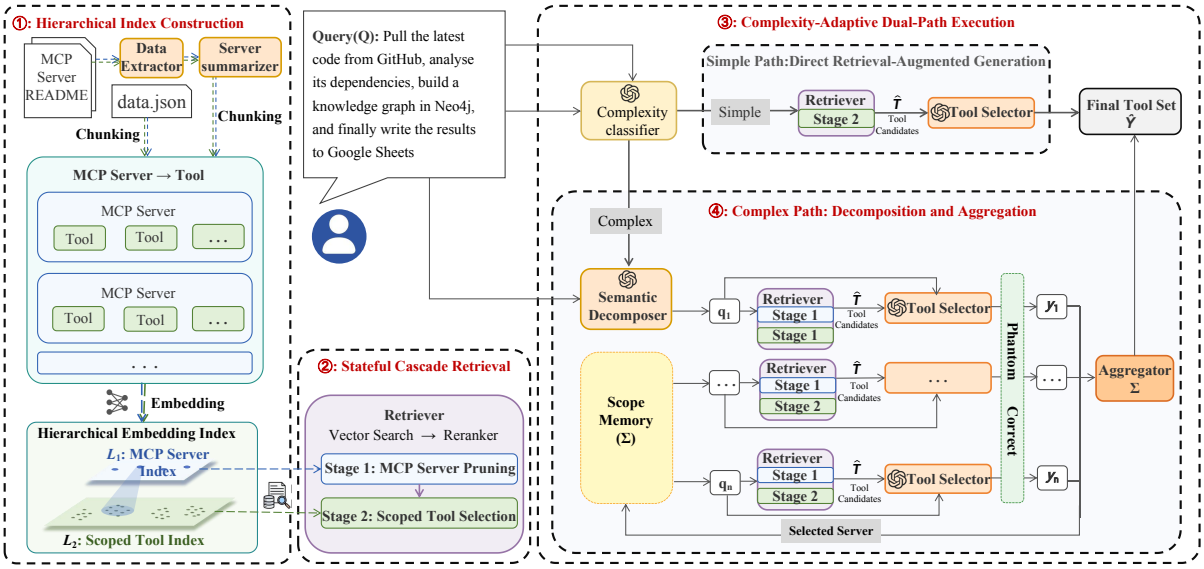

Figure 2: Overview of HiDR. Four modules: ① hierarchical index construction (offline; encodes MCP servers and tools into L1 Server / L2 Tool dual-layer vector indices); ② stateful cascade retrieval (Stage 1 L1 server pruning → Stage 2 L2 scoped selection); ③ complexity-adaptive dual-path execution (the Simple path runs cascade retrieval and tool selection directly; the Complex path triggers ④); ④ Complex path: decomposition, retrieval, generation and aggregation, incrementally building a tool set that covers all sub-intents.

document corpus. The MCP ecosystem is organised under a two-level Server→Tool hierarchy, so the more critical state is which servers have already been located.

3 Method: HiDR

HiDR's multi-tool discovery preserves both per-sub-intent independence and cross-sub-intent server coherence. The full architecture is shown in Figure 2, realised by three mechanisms. *An SLM-driven query decomposer* (§3.3) splits a complex query into semantically independent sub-queries, which preserves per-sub-intent independence. *A cross-subquery Scope Memory* (§3.4) accumulates the servers adopted by the tool selector after hallucination correction across sub-query iterations, which preserves cross-sub-intent server coherence. *An L1/L2 Server→Tool cascaded vector index* (§3.2) carries the other two. It provides the hierarchical retrieval space for query decomposition, and it gives Scope Memory the server as a clear unit of accumulation. Finally, an LLM tool selector generates the tool set inside the reranked candidate scope, followed by hallucination correction.

3.1 Task Definition

We define MCP ecosystem as a set of servers $\mathcal{S} = \{S_i\}_{i=1}^{n}$. Each server S_i is described by a name n_i^s, a functional summary d_i^s, and a hosted tool set $\mathcal{T}_i = \{t_{i,j}\}_{j=1}^{m_i}$, where each tool $t_{i,j} = (n_{ij}, d_{ij})$ consists of a tool name n_{ij} and a detailed functional description d_{ij}. The global tool pool is the union $\mathcal{T}_{\text{all}} = \bigcup_{i=1}^{n} \mathcal{T}_i$. Given a user query Q, the goal is to identify the optimal tool set from $\mathcal{T}_{\text{all}}$:

$$\hat{Y} = \arg\max_{Y \subseteq \mathcal{T}_{\text{all}}} P(Y \mid Q)$$

RAG-based retrieval typically scores tools at the global level, $\text{score}_{\text{flat}}(t, Q) = \text{Sim}(\phi(Q), \phi(d_{ij}))$, and ignores the two-level Server→Tool structure. HiDR instead implements retrieval as a two-stage cascade: first prune candidate servers, then rank tools only inside the pruned set,

$$\hat{\mathcal{S}}(Q) = \text{TopK}_{S \in \mathcal{S}} \, \text{Sim}(\phi(Q), \phi(d_s)),$$
$$\hat{Y}(Q) = \text{TopK}_{t \in \mathcal{T}_{\hat{\mathcal{S}}}} \, \text{Sim}(\phi(Q), \phi(d_t)),$$

where $\mathcal{T}_{\hat{\mathcal{S}}} = \bigcup_{S \in \hat{\mathcal{S}}} \mathcal{T}_S$ is the tool pool inside the pruned server set. The cascade reduces the L2 search space from $O(|\mathcal{T}_{\text{all}}|)$ to $O(|\mathcal{T}_{\hat{\mathcal{S}}}|)$, and provides the substrate on which the query decomposition (Sec. 3.3) and the Scope Memory (Sec. 3.4) operate.

3.2 L1/L2 Cascaded Index and Retrieval

The L1 index stores the functional summary of each Server, and the L2 index stores the tool de-

Algorithm 1 HiDR inference pipeline

Require: Query Q, L1 index $\mathcal{I}_s$, L2 index $\mathcal{I}_t$, LLM $\mathcal{M}$
Require: L1 top-k_s, L2 top-k_t, Reranker top-r_s, top-r_t
Ensure: Tool set $\hat{Y}$

```
is_simple ← M.classify(Q)                          ▷ complexity routing
if is_simple then
    Q ← {Q}
else
    Q ← M.decompose(Q)                             ▷ semantic decomposition
end if
Σ ← ∅                                              ▷ shared server set across sub-queries (Scope Memory)
Ŷ ← ∅
for each q_k ∈ Q do
    D_s ← I_s.search(q_k, k_s)                     ▷ L1 vector retrieval
    Ŝ_k ← Reranker(D_s, q_k, r_s)                  ▷ L1 rerank, top-r_s
    D_t ← I_t.search(q_k, k_t, filter = Ŝ_k ∪ Σ)   ▷ L2 scope = current L1 ∪ accumulated Σ
    T̂_k ← Reranker(D_t, q_k, r_t)                  ▷ L2 rerank, top-r_t
    Ŷ_k ← M.select_tools(q_k, T̂_k)                 ▷ LLM tool selection
    Ŷ_k ← CorrectHallucination(Ŷ_k, T̂_k)           ▷ hallucination correction
    Σ ← Σ ∪ servers(Ŷ_k)                           ▷ accumulate servers in Ŷ_k after correction
    Ŷ ← Ŷ ∪ Ŷ_k
end for
return dedup(Ŷ)
```

scriptions inside each Server. Built on this hierarchy, cascade retrieval first locates the Server and then searches the Tools inside it.

L1 Server index $\mathcal{I}_s$. We concatenate the server name and its summary into a text unit (n_{s_i}, d_{s_i}), encode it, and use it to build $\mathcal{I}_s$. At retrieval time, L1 recalls a candidate server subset $\hat{\mathcal{S}}$ for query Q. The L2 search is then restricted from the full tool pool to the tools inside the semantically relevant servers. L2 Tool index $\mathcal{I}_t$. We concatenate the tool name and description into a text unit $(n_{t_{i,j}}, d_{t_{i,j}})$, encode it, and record the parent server n_{s_i} in metadata. L2 retrieval is strictly restricted to $\hat{\mathcal{S}}$ via metadata filtering. The L1 server pruning result is forwarded to tool-level retrieval, and noise from the global pool is avoided.

Cascade retrieval based on the two-layer index. L1 coarse retrieval and re-ranking. For each sub-query q_k, HiDR first runs vector retrieval on $\mathcal{I}_s$ to take the top-k_s candidate servers. A Reranker then scores the server summaries against q_k and keeps the top-r_s, forming the candidate set $\hat{\mathcal{S}}_k$. L2 scoped retrieval and re-ranking. Once $\hat{\mathcal{S}}_k$ is fixed, HiDR runs metadata-filtered vector retrieval on $\mathcal{I}_t$ inside the tool subset of $\hat{\mathcal{S}}_k$, takes the top-k_t, and applies the Reranker again to keep the top-r_t tools, forming the reranked tool set $\hat{\mathcal{T}}_k$.

3.3 Query Decomposition and Complexity Classification

Query decomposition is the mechanism by which HiDR preserves per-sub-intent independence. HiDR uses a lightweight LLM classifier to judge the complexity of Q. If Q involves only a single primary operation, it is labelled **simple** and enters cascade retrieval directly. If Q requires cross-platform data transfer or a sequence of multiple independent operations, it is labelled **complex** and triggers the stateful decomposition-aggregation path. The classifier follows a conservative strategy. When uncertain, it defaults to *simple* to avoid unnecessary decomposition overhead on simple queries. For complex queries, the semantic decomposer breaks Q into a set of atomic sub-queries $\{q_1, q_2, \ldots, q_K\}$, each focused on a single functional intent, retaining critical context and resolving ambiguous references so that each sub-query is independently executable.

3.4 Scope Memory: Cross-Sub-intent Server Coherence

Scope Memory is the mechanism by which HiDR preserves cross-sub-intent server information. Algorithm 1 gives the pseudocode of HiDR pipeline. We analyse how it incrementally builds the tool set that covers all sub-intents of a complex query.

After query decomposition, a single round of L1 retrieval only covers Servers relevant to the current sub-query. Servers adopted by earlier sub-queries cannot be reused in later rounds, so cross-sub-intent Server information is broken. Scope Memory preserves this coherence by accumulating, across iterations of a complex query, the Servers adopted by the tool selector after hallucination correction. After the tool selection and correction of round k yield $\hat{Y}_k$, the Servers involved in $\hat{Y}_k$ are accumulated into the global state $\Sigma_k = \bigcup_{j \le k} \text{servers}(\hat{Y}_j)$. The L2 retrieval scope of the next round $(k+1)$ is then $\hat{\mathcal{S}}_{k+1} \cup \Sigma_k$, the union of the current L1 rerank candidates and the historically adopted Servers. Later sub-queries can thus both bring in new Servers from L1 and directly reuse Servers already confirmed in earlier rounds, reducing cross-step candidate misses. The final tool set is obtained by deduplicated aggregation:

$$\hat{Y} = \text{dedup}\left(\bigcup_{k=1}^{K} \hat{Y}_k\right)$$

The reranked tool set $\hat{\mathcal{T}}_k$ and sub-query q_k are

fed to an LLM tool selector, which picks the best-matching tools from the candidate list and outputs a structured list of `Server > Tool` identifiers. The selector is constrained to select only from the candidate list, which prevents fabrication of non-existent tools. If hallucination still occurs, a hallucination-correction module maps any tool name outside the candidate list back to the closest legal tool as a fallback.

4 Experiments

4.1 Setup

MCP-Bench. Due to the lack of a suitable open-source benchmark for this setting (**?**), we construct and release MCP-Bench. We reuse the tool corpus collected by MCP-Zero (Fei et al., 2025) (2,797 tools from 308 MCP servers) and produce 1,000 annotated queries on top of it, each paired with a ground-truth tool set. Queries are stratified into six groups by the required tool count: G1 corresponds to the simple subset; G2–G5-6 correspond to the medium and complex subsets. The full distribution is given in Appendix D.

Baselines. We compare against the following baselines (Lewis et al., 2020; Yao et al., 2023; Shao et al., 2023; Jeong et al., 2024; Anthropic, 2025). **B0 (BM25, Tool-Search-Tool-like)** reproduces the BM25 mode of Anthropic's Tool Search Tool. We build a BM25 index over `server_name + tool_name + description` and return the top-k tools by query-term scoring. This is the industrial baseline that Tool Search Tool exposes when no custom retriever is plugged in. **B1 (Dense Vector)** runs dense vector retrieval without an LLM tool selector and returns the top-k tools directly, with the same k as B0. Tool Search Tool also ships a regex mode. We do not reproduce it. Anthropic does not publish the regex construction, and any ad-hoc rule would inject substantial subjectivity. So BM25 is the only practically reproducible industrial baseline of Tool Search Tool, and we use B0 for that role. **B2 (Naive RAG)** runs one-shot vector retrieval and then an LLM generates the tool set. **B3 (ReAct RAG)** iterates reason-and-retrieve over the global tool pool. **B4 (Adaptive RAG)** routes between B2 and B3 by query complexity.

Metrics. The primary metric is **Tool-level F1 (T-F1)**, computed as set Precision/Recall/F1, ignoring tool order and duplicates, in line with the conventions of RAG and tool discovery (Thakur et al., 2021; Li et al., 2023). Auxiliary metrics are Server-level F1 (S-F1), average token consumption, and average latency.

Implementation. The embedding model is Qwen3-Embedding-0.6B, the reranker is bge-reranker-large, and the vector store is Milvus. The main experiments use Qwen2.5-7B-Instruct, Qwen2-7B-Instruct, Llama-3.1-8B-Instruct and GPT-4o-mini as backbones to verify cross-model generality, with inference temperature 0 throughout (Zhang et al., 2025; Yang et al., 2024; Qwen et al., 2025; OpenAI et al., 2024; Grattafiori et al., 2024). Hyperparameters, hardware configuration, and reproduction details are in Appendix G.

4.2 Main Results

Table 1 reports T-F1, T-Precision, T-Recall and latency on Simple/Medium/Complex/Overall for four backbones. **HiDR consistently outperforms all baselines on the multi-tool subsets.**

Tool Search Tool BM25 mode in practice. B0 (BM25, Tool-Search-Tool-like) reproduces the BM25 mode built into Anthropic's Tool Search Tool over a monolingual corpus. Its Overall T-F1 is only **18.5%**, and Complex is 29.1%. The score is slightly above B1 Dense Vector (14.4% Overall), which is also LLM-free. It remains **2.7× below HiDR** (49.7% Overall and 61.3% Complex). On the Complex subset the gap reaches **32.2 pt**. The result shows that single-vector keyword matching cannot preserve the independent signal of each sub-intent under multi-intent compression. It also cannot exploit the Server→Tool hierarchy to reuse server scope across sub-queries. This is the empirical reason why HiDR is positioned as a custom retriever that replaces the industrial BM25 mode in Tool Search Tool. The regex mode is not reproduced. Anthropic does not publish its construction, and any ad-hoc regex would inject substantial subjectivity into the comparison.

Cross-SLM-backbone robustness. Across all four SLM backbones, **HiDR consistently outperforms Tool Search Tool's industrial BM25 default (B0)**. Overall T-F1 improves by **+18.3 to +31.2 pt** (highest +31.2 on Qwen2.5-7B, lowest +18.3 on Llama-3.1-8B), and the Complex subset improves by **+14.1 to +32.2 pt**. HiDR also leads on every backbone over the RAG-based baselines (B2 Naive RAG, B3 ReAct RAG, B4 Adaptive RAG). Taking Naive RAG (B2) as a refer-

Method	Simple			Medium			Complex			Overall				
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	Lat	Tok
B0 (BM25)	7.9	77.0	14.3	8.9	31.9	13.7	21.5	45.9	29.1	12.5	45.2	18.5	0.05	0
B1 (Dense Vector)	8.5	82.0	15.4	8.0	29.5	12.4	12.5	26.9	17.0	9.5	39.2	14.4	0.10	0
SLM Backbone: Qwen2.5-7B-Instruct														
B2 (Naive RAG)	**64.0**	**75.0**	**67.1**	28.2	26.2	26.5	43.5	28.3	32.8	40.0	36.6	36.5	**1.27**	**833**
B3 (ReAct RAG)	58.2	59.0	58.5	27.7	23.7	24.7	39.2	21.5	26.7	37.3	30.1	32.0	7.03	3105
B4 (Adaptive RAG)	63.4	70.0	65.2	27.8	23.7	24.7	39.2	21.5	26.7	38.4	32.3	33.4	6.77	3106
HiDR (Ours)	58.7	65.5	60.7	**38.4**	**41.1**	**38.1**	**61.4**	**64.4**	**61.3**	**49.5**	**53.1**	**49.7**	5.12	3437
SLM Backbone: Qwen2-7B-Instruct														
B2 (Naive RAG)	**62.0**	**72.5**	**65.2**	28.4	27.6	27.3	37.2	27.9	30.8	37.8	36.7	35.9	**1.25**	**832**
B3 (ReAct RAG)	54.8	55.0	54.8	23.3	16.7	18.6	30.1	15.6	19.7	31.7	24.0	26.2	8.17	3285
B4 (Adaptive RAG)	59.3	61.0	59.8	23.3	16.7	18.6	30.1	15.6	19.7	32.6	25.2	27.1	8.33	3407
HiDR (Ours)	55.0	66.5	57.3	**35.6**	**42.1**	**36.9**	**56.2**	**64.6**	**58.9**	**45.8**	**53.8**	**47.7**	5.20	3402
SLM Backbone: Llama-3.1-8B-Instruct														
B2 (Naive RAG)	39.5	**82.0**	**51.3**	**23.1**	30.3	25.2	27.1	30.4	27.3	27.6	40.6	31.1	**2.41**	**943**
B3 (ReAct RAG)	38.2	44.0	40.1	14.7	11.0	12.0	23.1	10.3	13.7	22.0	17.4	18.1	8.63	3729
B4 (Adaptive RAG)	40.4	69.0	48.4	14.8	11.2	12.1	23.1	10.3	13.7	22.5	22.5	19.8	8.29	3573
HiDR (Ours)	**40.8**	72.0	50.6	21.1	**44.0**	**27.3**	**33.4**	**66.1**	**43.2**	**28.8**	**56.3**	**36.8**	11.16	4025
SLM Backbone: GPT-4o-mini														
B2 (Naive RAG)	56.6	**76.5**	**62.2**	30.7	28.9	29.2	49.2	29.9	36.1	41.5	38.7	37.9	**1.68**	**854**
B3 (ReAct RAG)	55.8	60.5	57.3	31.5	24.6	26.4	54.8	27.2	35.4	43.5	32.6	35.3	8.58	3775
B4 (Adaptive RAG)	**56.8**	69.5	60.4	31.8	24.0	26.0	52.1	26.5	34.3	43.0	33.9	35.4	9.73	3786
HiDR (Ours)	55.9	67.5	59.2	**35.1**	**43.5**	**37.0**	**55.7**	**66.7**	**59.2**	**45.5**	**55.4**	**48.2**	8.29	3635

Table 1: Main results (T-Precision / T-Recall / T-F1 %). Subsets are bucketed jointly by tool count and task-chain depth (**Simple**: 1 tool, 200; **Medium**: 2–4 tools, 495; **Complex**: 5+ tools or deep chain, 305; full mapping in Appendix D). Best value within each SLM backbone block is bold. B0/B1 do not invoke an LLM and are listed once. Lat = end-to-end Overall latency (s); Tok = Overall mean LLM tokens. B0 reproduces the BM25 mode of Anthropic Tool Search Tool.

ence, HiDR leads by +5.7 to +13.2 pt on Overall and by +15.9 to +28.5 pt on Complex. Paired bootstrap (B=1000, n=305) gives the 95% CIs for HiDR vs B2's Complex ΔF1 as $[+25.6, +31.4]$ / $[+25.3, +31.0]$ / $[+13.5, +18.3]$ / $[+20.2, +26.1]$ pt respectively, all significant at $p < 0.05$. Per-backbone full numbers are in Appendix A.

Latency–accuracy trade-off. On Qwen2.5-7B, HiDR achieves a favourable latency–accuracy–cost balance among 7B-class SLM backbones (Figure 3, Table 1). HiDR reaches **49.7%** Overall T-F1 at only 5.12 s latency, well above B3/B4 (32–33% at ∼7 s). The latency profile is backbone-dependent. On Llama-3.1-8B HiDR runs slightly slower than B3/B4 because Llama emits longer decomposition outputs, so the accuracy gain there should be read primarily as an accuracy result rather than a Pareto-latency result. On token cost, **HiDR (3402 tokens on Qwen2-7B) is on par with the ReAct-class B3/B4 (3285/3407)** but delivers 21–22 pt higher T-F1. It uses 4× the tokens of B2 (832) yet also yields large F1 gains (+11.8 pt Overall, +28.1 pt Complex), showing that HiDR's improvement is not bought by extra LLM calls.

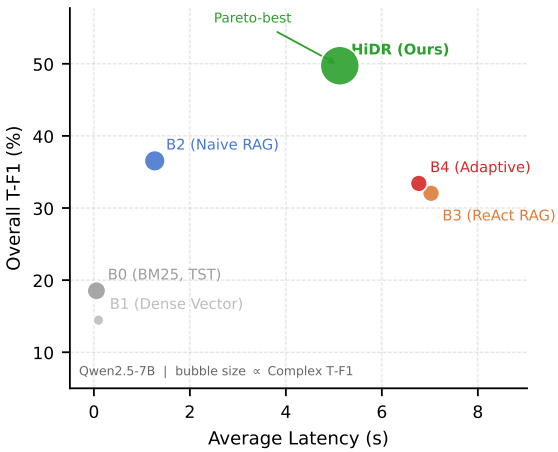

Figure 3: Latency–accuracy trade-off scatter (Qwen2.5-7B-Instruct). Dot size is proportional to Complex T-F1.

Analysis on the Simple subset. HiDR's T-F1 on the Simple subset is slightly below B2/B4. This is because a Simple query needs only one tool, so global retrieval over the tool pool is already sufficient. HiDR's L1 pruning introduces additional

Variant	Simple		Medium		Complex		Overall			
	F1	Lat	F1	Lat	F1	Lat	F1	Lat	Tok	ΔOverall
Full (HiDR)	60.7	1.84	38.1	4.77	61.3	7.85	49.7	5.12	3437	—
A1 w/o Hierarchy (flat L2)	59.9	1.81	36.0	4.56	60.2	7.02	48.1	4.76	3519	−1.6
A2.1 Simple-only (no classifier)	61.6	0.91	29.8	1.34	34.8	1.98	37.7	1.45	820	−12.0
A2.2 Complex-only (no classifier)	61.0	2.41	38.0	4.49	61.2	7.55	49.7	5.01	3081	—
A3 w/o Reranker	62.7	2.00	38.1	5.47	61.2	8.97	50.0	5.85	8566	+0.3
A4 w/o Decomposer	62.3	1.32	29.4	1.74	34.6	2.47	37.6	1.88	1273	−12.1

Table 2: Ablation (Qwen2.5-7B-Instruct, 1000-query full-set mean). Each column reports T-F1 % / Latency (s). Tok is the weighted average LLM token usage (from the Qwen2-7B backbone, trend matches Qwen2.5-7B). ΔOverall is the absolute change in T-F1 (pt) relative to Full (HiDR). The targeted ablation of Scope Memory (S1) is evaluated separately on the main-table Complex subset (n=305) under a paired design, see Sec.4.3.

filtering risk when shrinking the candidate server range, while Scope Memory and query decomposition contribute near-zero gain on single-tool tasks. The two-stage structure is therefore not competitive in this regime. By contrast, HiDR's gain on Medium and Complex is substantial (**+11.6 pt** and **+28.5 pt** over Naive RAG (B2), on Qwen2.5-7B), and these two multi-tool subsets are the core design target of our method. The small loss on Simple is an acceptable trade-off between hierarchical filtering and multi-tool gain. The lead on Overall T-F1 (**+13.2 pt** on Qwen2.5-7B) confirms that the trade-off is justified.

4.3 Ablation Study

Table 2 reports five ablation variants (Qwen2.5-7B-Instruct). **Query decomposition (Decomposer) and the complexity classifier are the two most critical components of HiDR**—removing either drops Overall T-F1 by more than 11 pt.

Removing query decomposition (A4) reduces Overall T-F1 by 12.1 pt and drops Complex sharply from 61.3 to 34.6, showing that decomposition is the core capability for multi-tool tasks. The classifier is equally critical. A2.1 forces every query into the simple path and reduces Overall to 37.7 with the same failure mode as A4. A2.2 routes every query through decomposition; its Overall matches Full (49.7 vs 49.7) but about one-third of simple queries are forced into the decomposition path at the cost of 3.6 s extra latency, which is precisely why HiDR is slightly below B2 on the Simple subset in the main experiment (per-backbone classifier accuracy and analysis are in Appendix H). The L1/L2 hierarchical index (A1) contributes only 1.6 pt on Overall T-F1. Its real value is not direct F1 gain but providing the server-set substrate on which Scope Memory accumulates state—once the hierarchy is removed (A1), Scope Memory has no scope to bind to. For A3 (without Reranker), Overall is slightly higher than Full (+0.3 pt), indicating that the MCP-untuned Reranker contributes little to F1 directly and may even be slightly negative; its practical value is to suppress low-quality candidates at the cost of 0.73 s extra latency (5.12→5.85 s) and 2.5× tokens (3437→8566), trading for Precision stability.

SLM Backbone	ΔF1	ΔTok	ΔLat
Qwen2.5-7B-Instruct	**+2.54**	+0.14%	+0.31
Qwen2-7B-Instruct	**+4.62**	+0.41%	+0.38
Llama-3.1-8B-Instruct	**+3.19**	−0.06%	+0.68
GPT-4o-mini	**+2.68**	+0.28%	+2.29

Table 3: Scope Memory ablation (A4.5 vs. Full HiDR, paired on n=305, identical to the main-table Complex subset). ΔF1 is in pt, ΔTok is the relative token increment, and ΔLat is the absolute latency increment (s).

Targeted ablation of Scope Memory. Scope Memory (S1) only activates in multi-tool iterations and contributes zero on the single-tool subset. We evaluate directly on the main-table Complex subset (n=305, drawn from complex1+complex2+complex3, with the same scope as the Complex column of Table 1). We use a paired design. The same queries are run under Full HiDR and under A4.5 with Scope Memory removed. ΔF1 is then computed per query, which removes the variance from query difficulty.

Table 3 shows that S1 brings a positive gain on all four SLM backbones. ΔF1 ranges from **+2.54 to +4.62 pt**, the token overhead is uniformly under 0.5%, and the local inference latency cost is only +0.31 ∼ +0.68 s. **Open-source (Qwen / Llama) and closed-source (GPT) models show the same direction and magnitude**, validating

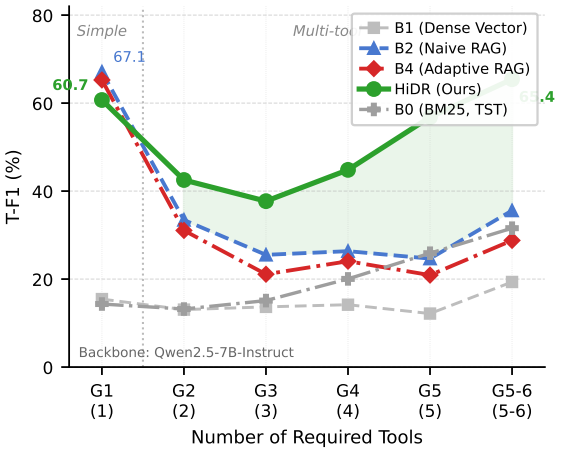

Figure 4: T-F1 (%) as a function of the required tool count (G1–G5-6) for Qwen2-7B-Instruct. The green shaded region marks HiDR's gain over B2. B2's decay curve is fitted by p^N ($p = 0.67$, $R^2 = 0.94$). The p^N decay comes from a single-round retriever having to hit all target tools at once; HiDR's query decomposition reduces this to one target per step, keeping accuracy stable on multi-tool tasks and avoiding the p^N collapse.

that S1 delivers a near-zero-cost quality gain.

Broken down by the required tool count, S1's contribution peaks in the 4-tool and 6+-tool ranges and is weaker in the 5-tool range. This indicates that the benefit of Scope Memory is tied to the cross-subquery server-reuse pattern rather than monotonic in tool count.

4.4 Result Analysis

Scalability in tool count. B2's T-F1 decays exponentially in the required tool count ($F1 \approx p^N$, $p = 0.67$, $R^2 = 0.94$), while HiDR's curve trends upward (Figure 4). B2 stays at 23.3 on G5 and 35.4 on G5-6, while HiDR climbs from 42.0 on G2 up to 57.8 on G5 and 63.7 on G5-6. The reason: flat retrieval in a multi-tool query has to hit N tools at once, so its hit probability decreases exponentially in N; HiDR's query decomposition breaks the multi-intent query into N single-intent sub-queries, each retrieved independently and aggregated via Scope Memory, so the gain from decomposition grows with the tool count. This also explains why the gain on the multi-tool subsets is much larger than on the single-tool subset in the main experiment.

Error analysis. We split HiDR's retrieval errors into *Server-miss* (L1 fails to recall the target server) and *Tool-miss* (the server is recalled but L2 fails to hit the target tool). **Server-miss accounts for 89%–91% on G1–G5** (Figure 6) and remains 72% on G5-6. It is HiDR's main current bottleneck, because tools under an un-recalled server are necessarily missing. Tool-miss is only 9%–11% on G1–G5 and rises to 28% on G5-6, suggesting that L2 still has reranking headroom when many targets coexist. **Improving L1 recall (e.g., better server-summary generation) thus has higher marginal value than further optimising L2 reranking**.

Phantom Tool Confusion. In MCP-Bench, 4.9% of distinct tool names appear in multiple servers (Phantom Tools), and 43.7% of queries involve at least one such tool. By locating the target server first with L1 pruning, HiDR shrinks the T-F1 gap under Phantom cases from 12.8 pt (B2) to 6.3 pt. See Appendix B (Figure 7) for details.

5 Conclusion

This paper studies the multi-tool discovery problem in MCP. Tool Search Tool's industrial BM25 default fails to preserve the independent signal of each sub-intent under multi-intent queries. It also fails to reuse confirmed servers across sub-intents. We therefore propose the idea that multi-tool retrieval must simultaneously preserve sub-intent independence and cross-sub-intent server coherence. Based on this idea, we propose HiDR, with three cooperating mechanisms. *An SLM-driven query decomposer* preserves the independence of sub-queries. *A cross-subquery Scope Memory* accumulates the confirmed servers and delivers coherence. *An L1/L2 Server→Tool cascaded vector index* supports low-cost accumulation for Scope Memory and global tool search. On MCP-Bench, HiDR reaches **61.3%** T-F1 on the Complex multi-tool subset (Qwen2.5-7B). This is **+32.2 pt** above Tool Search Tool's industrial BM25 default. It is **+28.5 pt** above Naive RAG (B2) under the RAG approach. The advantage holds across all four SLM backbones. Future work will combine retrieval accuracy with end-to-end task execution success rate, and will study finer routing mechanisms to mitigate classifier mis-routing.

Limitations

This paper has the following limitations. The MCP-Bench tool corpus is derived from MCP-Zero's snapshot of 308 real MCP servers. Its coverage is bounded by that snapshot, does not reflect the ongoing evolution of the MCP ecosystem, and

will need periodic refresh as new servers appear. HiDR's query decomposition and complexity classification rely on 7B-class SLMs. On extremely complex long-chain queries (e.g., more than six tools), decomposition quality is bounded by the model's reasoning capability, and sub-queries can be too coarse or have unclear intent boundaries. L1 server summaries are auto-generated by an LLM, and their quality depends strongly on README completeness. For servers with sparse or vague documentation, L1 recall can degrade. Fourth, all LLM-invoking methods (HiDR and B2–B4) are evaluated under task-agnostic prompt templates that share the same tool-list formatting, ensuring no MCP-specific prior is leaked selectively to any single method. We observe Naive RAG outperforming Adaptive RAG (B4) on this distribution because B4's complexity router under-routes on queries dominated by tool-level rather than reasoning-level complexity, an effect that warrants further study. Finally, our evaluation centres on set P/R/F1 and does not consider tool call order or argument correctness. The relationship to end-to-end task execution success rate requires further study.

References

Anthropic. 2024a. Architecture Overview of Model Context Protocol. Anthropic Technical Documentation. Accessed: 2026-05-18.

Anthropic. 2024b. Introduction to Model Context Protocol (MCP). Anthropic Technical Documentation. Accessed: 2026-05-18.

Anthropic. 2025. Introducing advanced tool use on the Claude Developer Platform. Anthropic Technical Documentation. Accessed: 2026-05-18.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.

Guanting Dong, Jiajie Jin, Xiaoxi Li, Yutao Zhu, Zhicheng Dou, and Ji-Rong Wen. 2025. RAG-Critic: Leveraging automated critic-guided agentic workflow for retrieval augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 3551–3578.

Yu Du, Fangyun Wei, and Hongyang Zhang. 2024. AnyTool: Self-Reflective, Hierarchical Agents for Large-Scale API Calls,. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11812–11829. PMLR.

Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pages 6491–6501.

Xiang Fei, Xiawu Zheng, and Hao Feng. 2025. Mcp-zero: Active tool discovery for autonomous llm agents. *2506.01056*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *arXiv:2407.21783*.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 59532–59569. Curran Associates, Inc.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: retrieval-augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org.

Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv:2503.23278*.

idoubi. 2026. Find Awesome MCP Servers and Clients. Accessed: 2026-05-18.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong Park. 2024. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7036–7050, Mexico City, Mexico. Association for Computational Linguistics.

Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992.

Huihao Jing, Haoran Li, Wenbin Hu, Qi Hu, Xu Heli, Tianshu Chu, Peizhao Hu, and Yangqiu Song. 2025.

MCIP: Protecting MCP safety via model contextual integrity protocol. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 1177–1194, Suzhou, China. Association for Computational Linguistics.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2024b. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report. *arXiv:2303.08774*.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*, volume 37, pages 126544–126565. Curran Associates, Inc.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *arXiv:2412.15115*.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. 2024. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551. Curran Associates, Inc.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9248–9274, Singapore. Association for Computational Linguistics.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. 2025. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24497–24524, Vienna, Austria. Association for Computational Linguistics.

Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei. 2025a. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv:2501.09136*.

Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. 2025b. Agentic reasoning and tool integration for llms via reinforcement learning. *arXiv:2505.01441*.

Wenyu Tao, Xiaofen Xing, Yirong Chen, Linyi Huang, and Xiangmin Xu. 2025. TreeRAG: Unleashing the power of hierarchical storage for enhanced knowledge retrieval in long documents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 356–371, Vienna, Austria. Association for Computational Linguistics.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long*

Papers), pages 10014–10037, Toronto, Canada. Association for Computational Linguistics.

Yu Wang, Nedim Lipka, Ruiyi Zhang, Alexa Siu, Yuying Zhao, Bo Ni, Xin Wang, Ryan Rossi, and Tyler Derr. 2024. Topology-aware retrieval augmentation for text generation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, page 24422452, New York, NY, USA. Association for Computing Machinery.

Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelovic, and Jakob Nikolas Kather. 2025. LLM agents making agent tools. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 26092–26130, Vienna, Austria. Association for Computational Linguistics.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 43 others. 2024. Qwen2 technical report. *arXiv:2407.10671*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv:2506.05176*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

Ziyuan Zhuang, Zhiyang Zhang, Sitao Cheng, Fangkai Yang, Jia Liu, Shujian Huang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2024. EfficientRAG: Efficient retriever for multi-hop question answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3392–3411, Miami, Florida, USA. Association for Computational Linguistics.

A Supplementary Experimental Results

This section gives a unified cross-backbone ablation comparison (Table 4). Table 2 in the main text already shows the per-subset ablation for Qwen2.5-7B. This section adds the Overall F1, Token usage and ΔF1 (relative to Full) for the other three backbones, enabling a horizontal comparison of each component's contribution and cost across models.

B Supplementary Analysis Figures

This appendix collects the auxiliary analysis figures and detailed discussion omitted from the main text for brevity, covering the ablation visualisation (Figure 5), HiDR's error-type composition (Figure 6), and the Phantom Tool Confusion comparison (Figure 7).

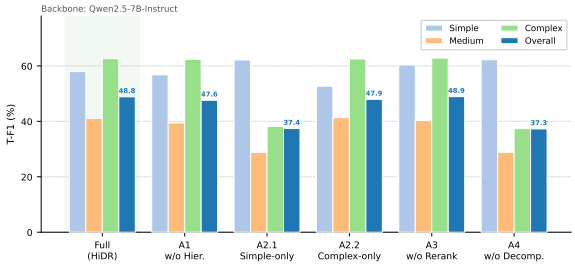

Figure 5: Per-component ablation (Qwen2.5-7B-Instruct). Each bar group corresponds to Simple / Medium / Complex / Overall T-F1; Overall is annotated on top of each bar.

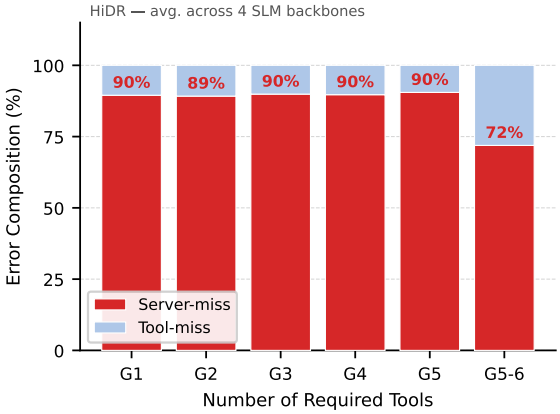

Figure 6: HiDR's error composition by tool-count group (G1–G5-6), averaged over four SLM backbones. Server-miss accounts for 89%–91% on G1–G5 (72% on G5-6), and is the current main bottleneck.

Phantom Tool Confusion detailed analysis. In MCP-Bench, 4.9% of distinct tool names (126/2593) appear in multiple servers, that is, tools sharing the same name across servers but with different functionality. Many such collisions are high-frequency generic tools (e.g., `search`, `create`), so 43.7% of queries (1748/4000) involve

Variant	Qwen2.5-7B			Qwen2-7B			Llama-3.1-8B			GPT-4o-mini		
	F1	Tok	ΔF1	F1	Tok	ΔF1	F1	Tok	ΔF1	F1	Tok	ΔF1
Full (HiDR)	**49.7**	3437	—	**47.7**	3402	—	**36.8**	4025	—	**48.2**	3635	—
A1 w/o Hier.	48.1	3519	−1.6	47.1	3614	−0.6	33.6	4125	−3.3	46.3	3718	−1.9
A2.1 Simple-only	37.7	820	−12.0	37.8	820	−9.9	31.9	942	−4.9	39.7	843	−8.5
A2.2 Complex-only	49.7	3081	—	45.8	3364	−1.9	33.6	4173	−3.2	45.7	3487	−2.5
A3 w/o Reranker	50.0	8566	+0.3	47.6	9020	−0.1	34.6	10336	−2.2	48.6	8551	+0.4
A4 w/o Decomp.	37.6	1273	−12.1	37.0	1274	−10.7	31.8	1436	−5.1	39.4	1284	−8.8
A4.5 w/o Scope Mem.	48.3	3435	−1.4	46.2	3508	−1.5	35.2	3972	−1.6	43.4	3256	−4.8

Table 4: Cross-backbone ablation (1000-query full-set mean). Each backbone reports Overall T-F1 (%) / mean Token / ΔF1 (pt, relative to Full). **A4.5 (w/o Scope Memory)** keeps Token nearly identical to Full ($|\Delta\text{Tok}| \leq 379$, $\leq 10\%$ relative), consistent with the paired analysis in Sec.4.3, confirming Scope Memory's near-zero token cost.

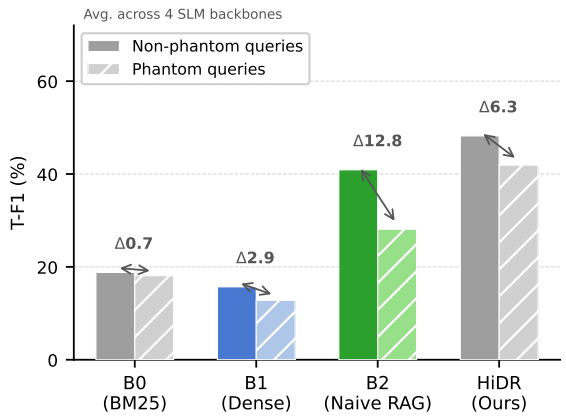

Figure 7: Phantom Tool Confusion: T-F1 of non-phantom queries (solid) vs. phantom queries (hatched). Dense Vector / Naive RAG / HiDR are 4-backbone averages (n=2252 vs. 1748); BM25 is LLM-independent (n=563 vs. 437). Naive RAG suffers the largest Phantom drop (12.8 pt). HiDR cuts the gap to 6.3 pt via L1 server pruning. BM25's gap is the smallest (0.7 pt) only because its absolute F1 is already low.

at least one phantom tool, which heavily interferes with flat retrieval. As shown in Figure 7, HiDR shrinks the T-F1 gap under Phantom cases from 12.8 pt (B2) to 6.3 pt, nearly halving it. The improvement comes from the constraining effect of L1 pruning. By locating the target server first, HiDR confines the candidate range to semantically relevant servers, which reduces same-name confusion. The residual 6.3 pt gap is mainly due to the limited semantic granularity of L1 summaries when distinguishing functionally similar same-name tools. When two server summaries are semantically close, L1 may still recall both, leaving L2 with ambiguity.

Scope Memory does not collapse to flat retrieval. A natural concern is that Σ accumulates monotonically with K and may eventually cover the entire server pool, reducing L2 retrieval to flat search. On the Qwen2.5-7B main batch we measure the number of unique servers finally selected per query as an empirical lower bound on $|\Sigma_K|$ (Σ_K is a superset of the final selected servers). Mean $|\Sigma_K|$ across tool-count groups is 2.0 (G2), 2.7 (G3), 3.1 (G4), 4.5 (G5), 3.5 (G5-6), with corresponding p95 values 4, 4, 5, 6, 5. The analytical upper bound is $K \cdot r_s = 3K$ (with r_s=3 the L1 reranker top-r), peaking at 18 for G5-6. Both the empirical lower bound and the analytical upper bound stay far below $|\mathcal{S}| = 308$ (coverage $\leq 5.8\%$), confirming that Scope Memory does not degenerate into flat retrieval under our experimental setup.

C Case Study

We pick two cases from the Qwen2.5-7B main-experiment logs (complex2 subset, 5–6 tools per query) where the HiDR–B2 F1 gap is largest, to qualitatively show how HiDR's query decomposition and Scope Memory address the multi-intent compression failure of Naive RAG. Queries are translated from the original Chinese annotations in MCP-Bench; tool names are kept verbatim.

Case 1: cross-domain 5-tool query. *Query.* Post a message in the Slack channel `#movie-club` announcing tonight's movie recommendation. Then, based on the movie *Inception* (ID: 27205), retrieve recommended films and query this week's trending movie list. Next, list all running Pod resources in our Kubernetes cluster to verify the recommendation service is healthy. Finally, generate an optimised set of SEO meta information for the curated movie-recommendation page.

Ground truth (5 tools). Slack > slack_post_message, TMDB > get_recommendations, TMDB > get_trending, mcp-k8s-go > List and get any Kubernetes resources, Open Strategy Partners Marketing Tools > OSP Meta Information Generator.

HiDR output (F1 = 1.00). Exactly matches GT. The decomposer splits the query into five atomic sub-queries (post a Slack message / fetch movie recommendations / fetch trending list / list K8s pods / generate SEO meta), retrieves them independently, and Scope Memory aggregates the four confirmed servers across iterations.

B2 output (F1 = 0.22). Graphlit > Posts, Kubernetes and OpenShift > resources_list, Video Editor > search-videos, Open Strategy Partners Marketing Tools > OSP Meta Information Generator. The Slack "post message" intent is compressed into a generic "post" signal and lands on Graphlit Posts; the K8s intent lands on a same-named tool on a wrong server (Kubernetes and OpenShift instead of mcp-k8s-go, a textbook Phantom Tool Confusion); both TMDB movie APIs are entirely missed; only the trailing marketing tool happens to be hit.

Analysis. Five tools come from five heterogeneous services. B2 compresses the multi-intent query into a single embedding, so the dominant semantic signals interfere with one another, and only the marketing-SEO intent survives the dilution. HiDR's decomposition ensures each sub-query carries a single intent, so recall does not decay with tool count.

Case 2: 6-tool query with multiple tools per server. *Query.* Delete the failed Pod etl-worker-crashed in the data-pipeline namespace of the Kubernetes cluster. Then, in the etl-worker-main Pod of the same namespace, execute python check_status.py to check the data-processing status. Next, query the list of all running jobs in the Keboola project, and retrieve the detailed configuration of the main data bucket. Then delete an obsolete tag named deprecated-feature from the Ghost content platform. Finally, use Riza to execute a Python snippet that summarises the results of the above operations.

Ground truth (6 tools). Kubernetes and OpenShift > pods_delete, pods_exec; Keboola > retrieve_jobs, get_bucket_detail; Ghost > Delete Tag; Riza > execute_tool.

HiDR output (F1 = 0.73, 5/6 hit). Kubernetes and OpenShift > pods_delete, pods_exec; Keboola > retrieve_jobs, get_bucket_detail; Riza > execute_code. Five of six are recalled correctly; only Ghost > Delete Tag is missed. The multi-tool-per-server hits (K8s' pods_delete+pods_exec, Keboola's retrieve_jobs+get_bucket_detail) benefit directly from Scope Memory: once a server is confirmed, L2 retrieval inside that scope can recall several tools across iterations.

B2 output (F1 = 0.00). Returns an empty set. With one flat embedding over the 2,797-tool pool, no candidate in top-k passes the LLM tool selector's relevance threshold.

Analysis. This is the extreme form of B2's failure mode: 6 tools spread across 4 servers, two of which need two tools each. Flat retrieval can neither preserve any sub-intent signal under multi-intent compression nor exploit server-scope information to recall multiple tools inside a confirmed server. HiDR's 5/6 hit rate illustrates the synergy of "L1 Scope Memory + L2 same-scope multi-tool recall".

D MCP-Bench Distribution

MCP-Bench contains 1,000 annotated queries, stratified into six groups (G1–G5-6) by the ground-truth tool count. G1 is the Simple subset (200 queries, single-tool), and G2–G5-6 are the Medium and Complex subsets (800 queries). Table 5 gives the full distribution.

Group	#Tools	#Queries	Subset
G1	1	200	Simple
G2	2	200	Medium
G3	3	150	Medium
G4	4	250	Medium / Complex
G5	5	100	Medium
G5-6	5–6	100	Complex
Total	1–6	**1,000**	—

Table 5: MCP-Bench query distribution by required tool count. G4 merges medium3 (145) and complex1 (105). G5-6 is complex2 as a whole (100 queries, with 80 5-tool and 20 6-tool). 6-tool annotations are sparse (n=20) and pooled with 5-tool for statistical reliability, in line with prior practice such as ToolLLM.

Bucketing rule for Simple/Medium/Complex in the main table.

Tool-name leakage audit. We scan all 1,000 queries for verbatim substring occurrences of `server_name` and `tool_name`. Server names (e.g. "Slack", "PostgreSQL", "GitHub") appear in 52.9% of queries as platform-level mentions. This is a natural part of how users describe tasks. It does *not* constitute a retrieval shortcut. HiDR and the B0/B1/B2–B4 baselines still need to distinguish among multiple tools hosted by the same platform across 308 servers. After removing platform-level mentions and generic English verbs, the share of queries containing a concrete tool function name (with an underscore, a dash, or a camel-case form, and not coinciding with any server name) drops to **28/1000 (2.8%)**. Among these, 24 hit "Kubernetes" or "SharePoint". These are generic platform terms that happen to also be registered as a tool name on some server. The number of queries containing a truly function-style tool identifier (e.g. `get_user`) is only 1. We therefore conclude that MCP-Bench does not exhibit tool-name leakage at a level that would reduce retrieval to keyword matching. The audit script and a 30-query random sample are at `scripts/audit/query_leakage.py` and `query/logs/leakage_audit_sample30.json`.

E MCP-Bench Annotation Protocol

Tool corpus construction. The tool corpus is derived from MCP-Zero (Fei et al., 2025), which aggregates README files from public MCP server repositories. We use an LLM-based structured extraction pipeline (`src/data_extractor.py`) to parse each README into structured records, including `server_name`, several (`tool_name`, `tool_description`) entries, and an auto-generated server-level summary. After deduplication by the composite key (`server_name`, `tool_name`), the final corpus contains 308 servers and 2,797 tools.

Query generation. All queries are written in natural language, expressing user intent without naming any server or tool, simulating realistic users who do not know the underlying API surface. Generation proceeds in two stages. In stage 1, the authors manually write seed queries that cover a diverse set of server categories and tool combinations, forming the initial entries of each file (`simple1.json`, `simple2.json`, `medium1-4.json`, `complex1.json`, `complex2.json`). In stage 2, an LLM-assisted generation pipeline (`supplement_queries.py`) expands each file to the target size, using Claude (claude-haiku-4-5) as the generator. For single-tool queries (G1), the model is given the server summary and tool description and asked to produce a corresponding natural-language user request; for multi-tool queries (G2–G5-6), the model is given an ordered (`server`, `tool`) chain and asked to write a coherent multi-step task scenario. Single-tool queries are constrained to 20–60 characters and multi-tool queries to 50–120. Results that fail JSON parsing or are empty are discarded and resampled.

Annotation scheme. Each query is paired with an `expected_docs` list containing one or more `{mcp_server, tool}` objects as the ground-truth tool set. For LLM-generated queries, the ground truth is directly determined by the pre-sampled (`server`, `tool`) combination, ensuring exact alignment between query intent and label without post-hoc human labelling. For manually authored queries, the ground truth is annotated by the author who wrote the query, referencing the tool descriptions in the corpus.

Difficulty stratification. Queries are stratified by required tool count into eight files: `simple1-2` (G1, 1 tool, 100 each), `medium1-4` (G2–G5, 2–5 tools, 200/150/145/100 respectively), and `complex1-2` (G4 and G5-6, 4–6 tools, 105/100). G4 covers both medium3 and complex1, providing denser samples in the medium-to-high complexity transition. See Table 5 for per-group sizes and the pooling scheme. The benchmark contains 1,000 queries in total, covering 308 servers and 2,797 tools.

F Hyperparameters and Implementation Details

Vector index. The vector store is Milvus, using HNSW with $M = 16$ and efConstruction $= 200$, and inner product (IP) as the similarity metric. L1 server-summary retrieval uses ef $= 64$, while L2 tool-description retrieval (targeted search) uses ef $= 200$ to keep high recall accuracy over a larger candidate space.

Retrieval hyperparameters. The key hyperparameters of HiDR's main pipeline are listed in Table 6. The Simple and Complex paths share the L1 and Reranker configurations. The difference is that the Complex path uses an L2 retrieval limit of 5 (instead of 10) per sub-query iteration, to control the per-iteration compute, relying on Scope Memory's cumulative effect to gradually expand the candidate coverage.

Hyperparameter	Simple	Complex
L1 retrieval limit	10	10
L1 Reranker top-k	3	3
L2 retrieval limit	10	5 (per iter.)
L2 Reranker top-k	10	10
Final #tools returned	10	10

Table 6: HiDR's main-pipeline retrieval hyperparameters (shared across all SLM backbones).

Language models. All SLM backbones use inference temperature 0. The main experiments use Qwen2.5-7B-Instruct, while cross-model robustness experiments additionally use Qwen2-7B-Instruct, Llama-3.1-8B-Instruct and GPT-4o-mini. The embedding model is Qwen3-Embedding-0.6B, and the reranker is bge-reranker-large. See Sec.G for deployment details.

G Hardware and Runtime Environment

All experiments run on a server equipped with an NVIDIA Tesla V100 (32 GB) GPU. The service stack is deployed with Docker Compose and consists of six containers: Milvus (etcd + MinIO + standalone), retrieval-service, vllm-service, and app-service. The embedding and reranker models are loaded inside retrieval-service. Open-source SLMs (Qwen2.5-7B-Instruct, Qwen2-7B-Instruct, Llama-3.1-8B-Instruct) are deployed in FP16 via vLLM, while GPT-4o-mini is called via the OpenAI API.

H Complexity Classifier Evaluation

The complexity classifier is the entry point of HiDR's routing: its judgement decides whether a query goes through the Simple direct-retrieval path or the Complex decomposition–aggregation path. The classifier follows a conservative strategy (defaulting to *simple* when uncertain) to avoid unnecessary decomposition overhead on truly simple queries.

Evaluation protocol. Labels come from MCP-Bench. `simple1-2` (200 queries) carry the ground-truth label *simple*, and `medium1-4` + `complex1-2` (800 queries) carry the ground-truth label *complex*. We invoke the classifier on every query (with the same temperature-0 setting as in the main experiments) and compute accuracy and the confusion matrix.

Analysis. (1) Structural bias of the conservative strategy. On all four SLM backbones, Complex accuracy is $\geq 97.5\%$ (multi-tool queries are almost never mis-labelled as simple), while Simple accuracy spans a wide range (64–99.5%). The two Qwen models mis-label about one-third of single-tool queries as complex, whereas Llama-3.1 and GPT-4o-mini have much lower mis-label rates (0.5% and 15%, respectively). **(2) Explanation of the Simple subset behaviour in the main experiment.** On Qwen2.5-7B, 36% of simple queries are mis-labelled as complex, so HiDR's measured Simple T-F1 is a weighted mixture of the simple and complex paths. Substituting the ablation numbers (A2.1 Simple-only T-F1 $= 62.1$ and A2.2 Complex-only T-F1 $= 52.7$), the weighted average $0.64 \times 62.1 + 0.36 \times 52.7 = 58.7$ matches the main-table HiDR Simple $= 58.0$ almost exactly, confirming that mis-routing is the direct cause of HiDR being slightly below B2 on the Simple subset.

Typical mis-classifications. Simple queries mis-labelled as complex are mostly state-modifying requests ("create a branch", "grant permission", "create a hash table"), where the classifier tends to read "create + several parameter descriptions" as a multi-step operation.

I Prompt Templates

HiDR uses three online LLM modules: complexity classifier, query decomposer, and tool selector. All prompts are zero-shot with temperature set to 0.

Complexity Classifier. The classifier receives only the raw query and outputs a single JSON field `single_step`. A conservative bias minimises unnecessary decomposition overhead on simple queries.

Query Decomposer. The decomposer splits a complex query into a list of semantically independent sub-queries, each targeting a single functional

SLM Backbone	Simple Acc	Complex Acc	Overall Acc
Qwen2.5-7B-Instruct	64.0	99.9	92.7
Qwen2-7B-Instruct	78.0	99.6	95.3
Llama-3.1-8B-Instruct	99.5	97.5	97.9
GPT-4o-mini	85.0	100.0	97.0

Table 7: Complexity classifier accuracy (%) across SLM backbones.

intent. Vague pronouns are resolved back to their referents, and unnecessary specifics (URLs, file paths) are stripped.

Tool Selector. The selector receives the sub-query together with the reranked candidate tool list and outputs a JSON array of `Server > Tool` identifiers. It is explicitly forbidden from hallucinating tools outside the candidate list.

Prompt 1 Complexity Classifier

You are a task-complexity expert. Judge whether the [User Query] can be completed with a single atomic operation.
When single_step = true (simple):
- Only one core action (read, write, search, query, send, etc.), even if described in detail.
- Default to **true** when uncertain.

When single_step = false (complex):
- Requires cross-platform data transfer (e.g., fetch from A then write to B).
- Two or more independent operations where the output of one feeds the next.

The following do NOT count as multi-step:
- Path separators (/, .) inside a file path.
- Conjunctions modifying the same action ("search and return", "read and display").
- "Output/display the result" as a descriptive suffix of a single operation.

Examples:

[User Query]: Read /home/user/config.yaml ⇒ {"single_step": true}

[User Query]: Search GitHub for rag repos, then write results to a local database ⇒ {"single_step": false}

[User Query]: Search Python repos on GitHub and return results ⇒ {"single_step": true}
Output only JSON. No preamble or explanation.

[User Query]: {original_query}

JSON output:

Prompt 2 Query Decomposer

You are a meticulous task planner that strictly follows the steps below.
Step 1. Sub-query list.

Original query: {original_query}

Decompose the query following the rules below:
1. Full coverage: every key action must appear in at least one sub-query.
2. Retain necessary context: keep tool names, platform names, targets, and key constraints.
3. Drop unnecessary specifics: remove concrete URLs, file names, and directory paths.
4. Resolve ambiguous references: replace "it" / "they" with the explicit referent from the original query.
5. Semantic independence: each sub-query must be self-contained and non-redundant.

Step 2. Strict JSON output.

Return a JSON object with key **"sub_querys"** whose value is a list of sub-query description strings. No explanation or extra text.

Example:

```
{"sub_querys": ["sub-query 1", "sub-query 2", ...]}
```

Prompt 3 Tool Selector

You are an efficient tool-selection expert. Given the [User Query] and the [Available Tool List], select ALL tools required to resolve the query, following the rules below.
Core rules:
1. Full coverage: select every tool whose function best matches a need in the query; return multiple tools when necessary.
2. Copy identifiers verbatim: output each tool's exact [Tool Name] in the format `[aaa] > [bbb]`.
3. List format: [Tool Name] and [Description] constitute each entry.
4. Empty result: return `[]` if no tool matches.
5. No parameters or explanations: output tool name strings only.
6. Specificity preference: prefer a specialised tool over a generic one when both apply.

[Available Tool List]: {context}

[User Query]: {query}

Output format (strict JSON array of name strings):

```
["xxx > aaaa", "yyy > bbbb", ...]
```