

PA-RAG: Protocol-Aware Retrieval-Augmented Generation for Tool Discovery in MCP Ecosystems

Anonymous ACL submission

Abstract

MCP has emerged as foundational infrastructure for AI-native applications since its inception. The ecosystem’s rapid expansion makes it a significant challenge to discover tools across thousands of servers precisely. To address it, we propose Protocol-Aware Retrieval-Augmented Generation (PA-RAG) for Tool Discovery in MCP Ecosystems. PA-RAG utilizes topology-aware hierarchical indexing to model structural relationships between MCP servers and tools. This indexing facilitates multi-stage cascade retrieval across hierarchical layers. PA-RAG adopts a complexity-adaptive dual-path execution mechanism to handle varying query types. Simple queries employ direct RAG. Complex queries follow a decomposition-aggregation strategy to maximize functional tool coverage. Extensive experiments based on lightweight LLMs demonstrate that PA-RAG significantly outperforms the state-of-the-art, effectively balancing discovery latency, coverage, and precision.¹

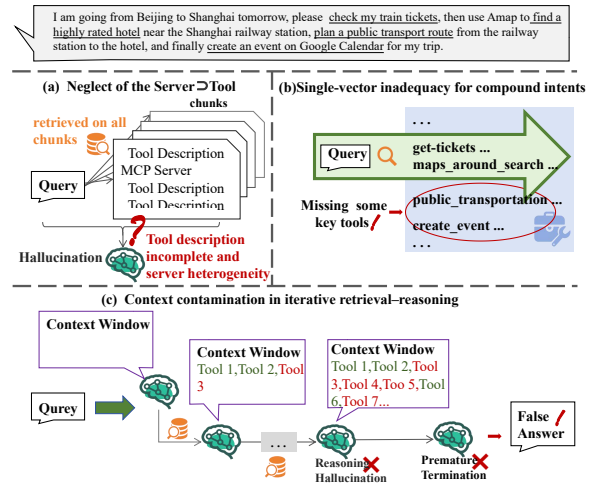


Figure 1: Three challenges in existing tool discovery methods. (a) Neglecting the MCP hierarchy leads to an incomplete context. (b) Compressing compound intents into a single vector causes tools to be missed. (c) Iterative retrieval results in context contamination, triggering reasoning drift and premature termination before the task is fully resolved.

1 Introduction

Model Context Protocol (MCP) is an open standard (Anthropic, 2024a,b) for connecting Large Language Models (LLMs) to external data sources and tools, such as local files, databases, and third-party services. As the ecosystem expands, the proliferation of available MCP servers (Jing et al., 2025; Hou et al., 2025) has introduced a critical bottleneck in tool discovery: precisely retrieving relevant tools from thousands of servers based on ambiguous natural language queries (Qin et al., 2024). The lack of standardized, high-quality corpora for MCP also hinders the development of effective retrieval systems. While Retrieval-Augmented Generation (RAG) is the dominant paradigm for acquiring external tool descriptions (Lewis et al., 2020; Guu et al., 2020; Dong et al.,

2025), its direct application to MCP yields suboptimal performance. The fundamental reason is that existing RAG frameworks are protocol-agnostic; they treat tool retrieval as a general text-matching task, while completely ignoring the structured priors and interaction constraints inherent in the MCP protocol. As shown in Fig.1, this protocol-agnostic leads to three key challenges:

Neglect of the Server-Tool hierarchy. MCP explicitly defines a hierarchical organization in which an MCP server encapsulates a set of tools (Anthropic, 2024b). But as illustrated in Fig. 1(a), due to their protocol-agnostic design, existing RAG frameworks disregard this hierarchy, treating MCP tools as a flat list (Wang et al., 2024) or applying fixed-length chunking strategies (Fan et al., 2024). This flattening disrupts the semantic and contextual dependencies between tools and their

¹<https://anonymous.4open.science/r/PA-RAG-301F/>

parent servers, leading to the retrieval of irrelevant tools and inefficient use of the context window. Incomplete tool descriptions increase the risk of reasoning hallucinations (Liu et al., 2024a). Moreover, retrieval across the entire tool set is $O(S \times T)$, where S represents the number of MCP servers and T denotes the average number of tools per server.

Single-vector inadequacy for compound intents. In MCP scenarios, user queries often express compound intents requiring coordinated invocation of tools across diverse servers. However, protocol-agnostic RAG frameworks typically perform one-shot retrieval by encoding such queries into a single vector (Jiang et al., 2023); the resulting representation is dominated by the primary sub-intent, thereby overshadowing secondary ones (Zhou et al., 2023; Liu et al., 2024b). This loss of semantic information prevents the framework from retrieving the complete set of required tool descriptions across disparate servers, as shown in Fig. 1(b).

Context contamination in iterative retrieval-reasoning. To address multi-step queries, Agentic RAG (Singh et al., 2025a; Fan et al., 2024) processes them by interleaving reasoning and retrieval (Trivedi et al., 2023), with the ReAct framework (Yao et al., 2023) being the most common implementation. However, its state maintenance typically relies on a linear accumulation strategy. In the dense MCP ecosystem, redundant tool descriptions and the continuous accumulation of intermediate outputs quickly exhaust the context window, leading to context contamination, which in turn induces reasoning hallucinations and ultimately results in premature task termination, as shown in Figure 1(c).

To address these challenges, we propose the Protocol-Aware Retrieval-Augmented Generation (**PA-RAG**) for Tool Discovery in MCP Ecosystems. PA-RAG leverages lightweight, open-source LLMs (e.g., 7B parameters) to implement its LLM-driven components, thereby ensuring both practicality and low latency. PA-RAG first aligns its indexing strategy with the inherent hierarchical topology of MCP, enabling a corresponding cascade retrieval. To optimize the performance-correctness trade-off, PA-RAG adaptively determines the inference flow via a complexity-aware dual-path execution, ensuring that simple queries remain rapid. In contrast, compound queries receive the necessary tool coverage.

The primary contributions of this paper are summarized as follows:

Topology-Aware Hierarchical Indexing and Corresponding Cascade Retrieval. We propose a hierarchical dual-layer indexing structure based on the hierarchical MCP Server \supset Tool topology. By employing a corresponding cascade retrieval strategy incorporating MCP server pruning and scoped tool selection, we narrow the search space, reduce the retrieval complexity from $O(S \times T)$ to $O(S + T)$, where S is the number of MCP servers and T denotes the average number of tools per server. By constraining retrieved tools to a minimal number of servers, this coarse-to-fine approach minimizes server heterogeneity. Consequently, it suppresses hallucinations in downstream LLM generation by providing a more focused and consistent context.

Complexity-Adaptive Dual-Path Execution. We introduce a complexity-adaptive router to balance inference latency with tool coverage. It routes queries into two paths: a simple path for direct RAG and a complex path that triggers our decomposition and aggregation strategy. This on-demand execution achieves an optimal equilibrium between response speed and high correctness.

Decomposition and Aggregation. We introduce a divide-and-conquer-based decomposition and aggregation strategy to address complex, multi-server long-chain queries. The strategy decomposes compound intents into independent sub-queries, performs targeted retrieval, and synthesizes sub-results through an aggregation verification module. This approach eliminates cross-step noise, significantly improving recall and accuracy in long-chain scenarios.

Experimental results on multiple lightweight 7B-class LLMs confirm that PA-RAG significantly enhances tool discovery compared to existing methods. PA-RAG achieves an average recall increase of 30% and an average precision gain of 18% (up to 38%). Particularly in complex multi-server scenarios, it delivers an average recall boost of 45% and an average precision increase of 25%.

2 Related Work

Large-Scale Tool Discovery and Standardization. The shift toward Agentic AI marks a milestone in enabling LLMs to interact with external tools (Schick et al., 2023; Wölflein et al., 2025).

As scaling API repositories makes parameter-based fine-tuning (Patil et al., 2024) infeasible, new approaches, such as ToolLLM (Qin et al., 2024; Li et al., 2023) and AnyTool (Du et al., 2024), have emerged to manage massive tool libraries. Concurrently, implementation standards for tool usage have evolved significantly. OpenAI first launched function calling (OpenAI, 2023), enabling LLM to use external APIs. Anthropic further introduced MCP in 2024, establishing a unified interface standard. But existing tool discovery approaches treat tools as isolated units, failing to account for the server-tool encapsulation inherent in the MCP ecosystem. Neglecting this hierarchical domain context leads to contextual inconsistency and suboptimal performance when applying traditional methods to the MCP ecosystem.

Structural Awareness and Dynamic Routing.

While standard RAG (Lewis et al., 2020) retrieves document chunks based on semantic similarity, recent advancements incorporate structural awareness and adaptive mechanisms (Gao et al., 2024). For instance, RAPTOR (Sarathi et al., 2024) for handling long-context reasoning, TreeRAG (Tao et al., 2025) for enhancing global context awareness, Adaptive-RAG (Jeong et al., 2024), Self-RAG (Asai et al., 2024) both introduced dynamic routing mechanisms. But these frameworks are primarily designed for unstructured text and rely on semantic clustering rather than the strict physical hierarchies of MCP. Consequently, they fail to capture the relationship logic of MCP servers and their tools. Their adaptive mechanisms often route queries solely on parametric knowledge within fine-tuned LLMs, neglecting the explicit metadata and functional descriptions of the MCP servers and tools.

Agentic Reasoning and Context Management. Agentic RAG integrates dynamic planning and feedback into the retrieval process (Singh et al., 2025b; Liu et al., 2024b; Zhuang et al., 2024). Agentic RAG leverages planning, reasoning, and memory capabilities to coordinate retrieval from multiple heterogeneous sources, typically building on the ReAct (Yao et al., 2023). Crucially, these paradigms often employ a linearly cumulative state management strategy, appending redundant tool descriptions and intermediate results to the history. When a large number of MCP servers encapsulating different tools are present (Jing et al., 2025), this accumulation quickly exhausts the context window and introduces signifi-

cant noise, rendering LLMs susceptible to hallucinations or premature termination.

3 Problem Formulation

Definitions and Objective.

We define the MCP ecosystem as a set of servers $S = \{S_i\}_{i=1}^n$, where each server $S_i = (n_{s_i}, d_{s_i})$ is characterized by its unique name n_{s_i} and a functional summary d_{s_i} . Each server S_i encapsulates a specific toolset $\mathcal{T}_i = \{t_{i,j}\}_{j=1}^m$, where each tool $t_{i,j} = (n_{t_{i,j}}, d_{t_{i,j}})$ comprises a tool name $n_{t_{i,j}}$ and a detailed functional description $d_{t_{i,j}}$. The universal tool set is defined as the union of all available tools across all servers, denoted as $\mathcal{T}_{\text{all}} = \bigcup_{i=1}^n \mathcal{T}_i$. Given a user query Q , the objective is to identify the optimal tool invocation set $\mathcal{Y} \subset \mathcal{T}_{\text{all}}$ by maximizing the conditional probability $P(\mathcal{Y} | Q)$. In cases involving compound intents, the total invocation set is derived as $\mathcal{Y} = \bigcup_k \mathcal{Y}_k$, where each \mathcal{Y}_k is the set of tools identified for a specific sub-query q_k within the set of decomposed intents $\{q_k\}$.

Hierarchical Probability Decomposition. Existing *Flat Retrieval* methods typically treat tool discovery as a direct matching problem over \mathcal{T}_{all} , modeled as $P_{\text{flat}}(t|Q) \propto \text{Sim}(\phi(Q), \phi(d_t))$. Here, $\phi(\cdot)$ is an encoder mapping inputs to a vector space where $\text{Sim}(\cdot, \cdot)$ measures their proximity. This ignores the intrinsic $S \supset t$ topology. To address this, we decompose the retrieval probability using the chain rule, transforming the global search into a two-stage cascade:

$$P_{\text{hier}}(t | Q) = \underbrace{P(\hat{S} | Q)}_{\text{Server Pruning}} \cdot \underbrace{P(t | \hat{S}, Q)}_{\text{Scoped Selection}}$$

MCP Server Pruning computes $P(\hat{S}|Q) \propto \text{Sim}(\phi(Q), \phi(d_S))$ to identify a candidate subset \hat{S} . Subsequently, **Scoped Tool Selection** computes $P(t|\hat{S}, Q)$ exclusively within \hat{S} , where the probability is proportional to $\text{Sim}(\phi(Q), \phi(d_t))$ if $S \in \hat{S}$, and 0 otherwise. This effectively reduces the retrieval complexity from \mathcal{T}_{all} to a localized subset $\bigcup_{S_{i_k} \in \hat{S}} \mathcal{T}_{i_k}$.

4 Method

PA-RAG operates through a coordinated flow involving four primary modules (see Fig. 2): a topology-aware hierarchical indexing, a corresponding cascade retrieval, a complexity-aware dual-path execution, and a complex path: decomposition-aggregation.

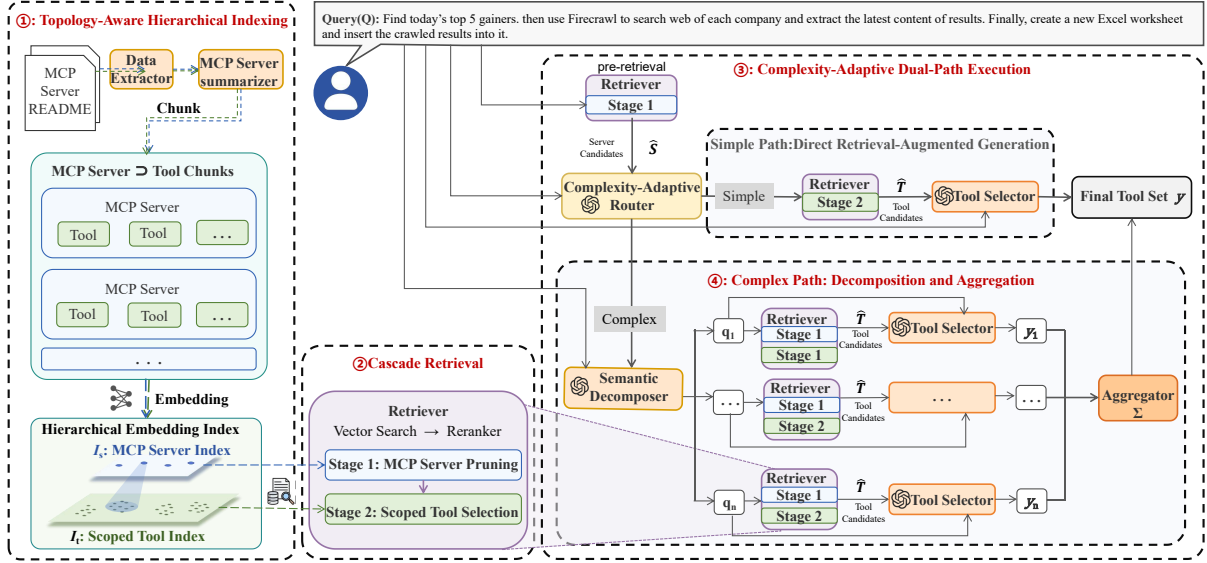


Figure 2: Overview of PA-RAG. It consists of three synergetic modules: ① Topology-Aware Hierarchical Indexing builds a dual-layer vector space mirroring the server-tool hierarchy; ② Cascade Retrieval performs multi-stage coarse-to-fine retrieval, transitioning from Server Pruning to Scoped Tool Selection; ③ Complexity-Adaptive Dual-Path Execution employs a router to evaluate query complexity with pre-retrieval, adaptively directing the flow toward either a Simple Path (Direct RAG) or a Complex Path (Decomposition-Aggregation); ④ Complex Path: Decomposition-Aggregation.

4.1 Overview

PA-RAG objective is to efficiently and accurately identify relevant tools from a large-scale, hierarchical MCP ecosystem with the given query. It begins with topology-aware hierarchical indexing, which transforms the MCP server README into a hierarchical vector space. The cascade retrieval acts as the connected engine, implementing retrieval from the MCP server and pruning to the scoped tool selection. Analyzing the query and its pre-retrieval results, the complexity-aware router determines the execution path. Simple queries execute a rapid direct RAG; complex queries require multi-server coordination via decomposition and aggregation. Complex queries are decomposed into atomic sub-queries by the semantic decomposer, each sub-query invoking multi-pass cascade retrieval and a tool selector to acquire a sub-result, and then aggregating to the final tool set.

The router, decomposer, and selector use lightweight LLM backbones to ensure the practical deployability and privacy of PA-RAG. This design choice also addresses the critical need for efficient, locally-deployable RAG systems in the MCP ecosystem, where low-latency tool discovery is paramount.

4.2 Topology-Aware Hierarchical Indexing

The MCP ecosystem inherently exhibits a clear hierarchical structure, where tools are subordinate to specific MCP servers. To leverage this structure to optimize both efficiency and accuracy in large-scale tool retrieval, we propose a decoupled dual-layer indexing structure (Module ① in Fig. 2). This structure implements the hierarchical probability model.

Data Extraction and Semantic-Aware Chunking. A structured data extractor parses the collected MCP server README files to extract key information, including server names, tool names, and functional descriptions. Building on this, we introduce an MCP server Summarizer that generates a comprehensive functional summary for each server. This step distills a server’s domain expertise into a compact representation, ensuring sufficient semantic density for subsequent hierarchical retrieval. Finally, the structured information is segmented into standardized metadata based on semantic boundaries.

Then we construct two distinct vector index layers to reflect the server \supset tool hierarchy:

MCP Server Index (\mathcal{I}_s). We encode the server name and its generated summary into vectors. By identifying candidate servers, this layer effectively

narrows the search space from the global tool set to a limited, server-specific tool subset. **Scoped Tool Index (\mathcal{I}_t)**. We encode individual tool names and tool descriptions into vectors. Crucially, each tool vector is tagged with its parent server’s metadata, enabling the following cascade retrieval to perform scoped tool selection within the specific servers.

4.3 Cascade Retrieval

To navigate the proposed hierarchical structure, we implement the corresponding two-stage coarse-to-fine cascade retrieval. As illustrated by ② Cascade Retrieval in Fig. 2, the retriever executes the progressive retrieval process from MCP servers to tools through two sequential stages, as defined in our formulation.

Stage 1: MCP Server Pruning. Given a query, it first retrieves and reranks the server candidates from \mathcal{I}_s . This stage approximates $P(\hat{S} | Q)$ by filtering out irrelevant servers, reducing the candidate pool from $|\mathcal{T}_{\text{all}}|$ to a manageable tool subset, which is critical for achieving efficient shrinkage of the search space. **Stage 2: Scoped Tool Selection.** Within the locked scope of \hat{S} , the stage performs localized vector distance calculation and reranking in \mathcal{I}_t to acquire tool candidates $\hat{\mathcal{T}}$. The Scoped selection is facilitated by the server metadata cataloging its hosted tools.

4.4 Complexity-Adaptive Dual-Path Execution

To strike a balance between processing efficiency and high tool coverage, we introduce a dual-path execution mechanism as shown in (Module ③ in Fig. 2). The core of this module is a complexity-aware router, implemented via a lightweight LLM. The router analyzes both the query and pre-retrieval results to determine the execution trajectory to a simple path or a complex path. The pre-retrieval results are functional descriptions of initial candidate servers obtained during Stage 1 by cascade retrieval.

For queries identified as simple, PA-RAG prioritizes execution speed. Since the complexity-aware router has determined that the initial candidate server set \hat{S} is likely sufficient to resolve the intent Q . Thereby, PA-RAG proceeds with scoped tool selection directly, as shown in the Simple Path part of Fig. 2.

In this path, the retriever performs a single-pass matching against functional descriptions within

the scope of \hat{S} to identify the tool candidates. These tool candidates $\hat{\mathcal{T}}$, along with the original query Q , are processed by a lightweight tool selector to generate the final tool invocation set \mathcal{Y} . By bypassing query decomposition and multi-round inference, this path minimizes processing overhead and latency while maintaining high precision for atomic tasks.

4.5 Complex Path: Decomposition and Aggregation

For queries classified as complex, we employ the decomposition and aggregation strategy, as shown in the Complex Path part of Fig. 2. The strategy is specifically designed to resolve the compound-intent task, which requires multiple MCP Servers. The strategy includes the following three phases.

Semantic Decomposition. A lightweight semantic decomposer partitions the original query Q into a sequence of semantically independent sub-queries $\{q_1, q_2, \dots, q_n\}$. This isolation of intents prevents the semantic dilution that typically occurs when a single vector representation is forced to encode multiple complex sub-queries. The decomposition ensures each sub-intent is explicitly represented.

Multi-pass Cascade Retrieval and Tool Selector. For each sub-query q_i , we execute the complete cascade retrieval process ($q_i \rightarrow \hat{S}_i \rightarrow \hat{\mathcal{T}}_i$). Then, the tool selector filters the tool candidates to produce the optimal tool subset ($q_i, \hat{\mathcal{T}}_i \rightarrow \mathcal{Y}_i$). This multi-pass ensures each sub-query is resolved in a contextually pure environment, effectively mitigating error propagation caused by irrelevant tool noise or semantic cross-contamination.

Aggregation and Verification. The aggregator synthesizes the results of sub-queries into the final invocation set $\mathcal{Y} = \bigcup_{i=1}^n \mathcal{Y}_i$. This module deduplicates redundant tool calls and performs a global consistency check to ensure that the aggregated set \mathcal{Y} provides comprehensive functional tool coverage for the compound-intent task without exceeding the LLM’s context limits.

5 Experiments

5.1 Experimental Setup

Datasets. Due to the lack of widely recognized open-source datasets for MCP Servers and tools, we develop **MCP-ToolBench**. We systematically collect and clean README documents from 50 real-world MCP servers, covering 400+ tools

Methods	Simple Set				Complex Set				Comprehensive Set			
	Prec.	Rec.	F1	Lat.(s)	Prec.	Rec.	F1	Lat.(s)	Prec.	Rec.	F1	Lat.(s)
<i>LLM Backbone: Qwen2-7B-Instruct</i>												
Naive RAG (B1)	77.1	81.3	78.3	0.76	47.6	33.6	38.3	1.40	61.2	55.4	56.6	1.03
Agentic RAG based on ReAct (B2)	65.0	63.8	64.2	4.67	43.3	28.7	33.4	8.98	54.2	46.2	48.8	6.82
Adaptive RAG (B3)	68.3	71.3	69.2	1.92	43.3	28.7	33.4	9.42	57.1	51.2	52.5	5.69
PA-RAG (Ours)	93.8	95.0	94.2	2.04	87.5	83.0	84.1	6.36	89.9	88.4	88.5	4.12
<i>LLM Backbone: Qwen2.5-7B-Instruct</i>												
Naive RAG (B1)	82.5	82.5	82.5	0.51	49.0	37.8	41.7	1.14	66.2	60.4	62.4	0.85
Agentic RAG based on ReAct (B2)	47.5	48.8	47.9	4.61	32.3	21.6	25.2	9.39	39.9	35.2	36.6	6.99
Adaptive RAG (B3)	83.8	83.8	83.8	1.07	32.3	21.6	25.2	9.76	57.4	52.0	53.9	5.41
PA-RAG (Ours)	92.1	95.0	92.9	1.76	79.7	75.3	75.4	6.59	85.6	84.4	83.8	4.15
<i>LLM Backbone: GPT-4o Mini</i>												
Naive RAG (B1)	82.1	87.5	83.8	1.30	68.4	41.8	50.2	1.78	75.5	64.3	67.1	1.59
Agentic RAG based on ReAct (B2)	82.5	85.0	83.3	5.39	59.4	34.3	42.1	10.68	66.4	55.2	58.0	9.00
Adaptive RAG (B3)	79.2	85.0	81.2	3.09	56.7	30.8	38.4	13.06	69.3	58.4	60.4	6.92
PA-RAG (Ours)	88.3	92.5	89.6	3.08	80.4	85.7	81.3	10.55	84.3	88.5	85.2	6.20
<i>LLM Backbone: Llama-3.1-8B-Instruct</i>												
Naive RAG (B1)	62.5	81.3	68.3	0.89	40.3	35.0	36.0	1.91	51.5	58.9	52.7	1.35
Agentic RAG based on ReAct (B2)	60.4	68.8	62.4	4.22	32.5	20.8	24.9	6.39	45.8	44.4	43.2	5.23
Adaptive RAG (B3)	69.2	77.5	71.6	1.84	32.5	20.8	24.9	6.77	51.3	50.1	48.9	4.25
PA-RAG (Ours)	78.1	90.0	81.3	4.02	57.4	82.5	66.9	11.62	64.7	86.1	70.7	7.34

Table 1: Main performance comparison on MCP-ToolBench across various lightweight LLM backbones. We report Precision (Prec.), Recall (Rec.), F1-Score (F1) in percentage (%), and Average End-to-End Latency (Lat.) in seconds. Best results for each metric within the same backbone are bold.

across domains such as databases, web search, and file operations. To assess various capabilities (Liu et al., 2024b), we construct the following three query evaluation subsets. The simple set consists of 40 queries that can be resolved by invoking tools from only 1-2 MCP Servers. The complex set consists of 40 queries that require coordinating tools from at least three different MCP Servers, designed to simulate multi-hop reasoning challenges often found in complex tool-use scenarios (Yang et al., 2018; Geva et al., 2021). The comprehensive set is the combination of the simple and complex sets, used to evaluate overall performance.

Baselines. We compare PA-RAG against three representative retrieval paradigms, Naive RAG (B1) Utilizes a one-shot flat retrieval strategy combined with a tool selector to generate the final answer (Lewis et al., 2020); Agentic RAG based on ReAct (B2) Represents the Agentic RAG paradigm based on the ReAct framework (Yao et al., 2023; Shao et al., 2023), employing an iterative process of flat retrieval and reasoning; Adaptive RAG (B3) incorporates a general-purpose complexity classifier (Jeong et al., 2024) to route tasks to either Naive RAG for simple queries dynamically or Agentic RAG based on ReAct for

complex queries.

Evaluation Metrics. Following the evaluation paradigms in RAG and tool learning (Thakur et al., 2021; Li et al., 2023; Qin et al., 2024), we evaluate the effectiveness of various approaches for tool discovery by comparing the final tool set of our PA-RAG and baselines against the correct tool set. Specifically, we report recall, precision, and F1-score as evaluation metrics. Recall, precision, and F1-score are calculated per query and subsequently averaged to obtain the final results. As our primary objective is to find the required tool and its hosting MCP server based on a user’s natural language query, we must prioritize coverage of all relevant tools and hosting servers, which makes **recall the primary evaluation metric**. We can tolerate some loss of precision when necessary. Additionally, we use average end-to-end latency to assess inference efficiency.

Implementation Details. PA-RAG and all baselines are implemented using the LangChain framework (Chase, 2022). For cascade retrieval, we employ Qwen3-Embedding-0.6B (Zhang et al., 2025) for text embedding and BGE-Reranker-Large (Xiao et al., 2023) for retrieval result reranking. To verify the generalization of PA-RAG, we

implement PA-RAG and baselines using widely used open source and closed source LLMs, including Qwen2-7B-Instruct (Yang et al., 2024), Qwen2.5-7B-Instruct (Qwen et al., 2025), GPT-4o Mini (OpenAI et al., 2024), and Llama-3.1-8B-Instruct (Grattafiori et al., 2024). All experiments using open source LLMs are conducted on NVIDIA Tesla V100 GPUs (32GB). To validate the contribution of each component, we define three ablation variants: A1 (w/o Hierarchy), A2 (w/o Classifier), including A2.1 (Simple Path Only) and A2.2 (Complex Path Only), and A3 (w/o Reranker).

5.2 Main Results

Table 1 presents the performance of PA-RAG and baselines across four representative lightweight LLM backbones, including open source and closed source. We summarize the following four key findings.

Mastery of Complex Multi-Server Scenarios. PA-RAG consistently outperforms competitive baselines to achieve state-of-the-art performance on various lightweight LLMs. The performance improvement is particularly significant on the complex set. For instance, when using Qwen-2.5-7B as the backbone, our framework achieves an F1 improvement of 33.7% over Naive RAG (B1) and up to 50.2% over Agentic RAG based on ReAct (B2). This advantage shows that single vector retrieval is challenging to capture cross-server dependencies due to semantic information loss. Our "decompose and aggregate" strategy effectively keep the semantic information of the query by decomposing the complex query into manageable and fine-grained sub-tasks, thereby achieving high accurate coverage.

The Failure of Iterative Reasoning in MCP. A counterintuitive observation is that Agentic RAG based on ReAct frequently underperforms even Naive RAG in terms of recall and F1-score. This degradation stems from protocol-agnostic context contamination. Such contamination is driven by the linear state maintenance of the ReAct mechanism. The mechanism will accumulate tool descriptions and reasoning content in the model’s context. This dense context will obscure core task semantics, thereby triggering reasoning hallucinations. These hallucinations will lead to premature task termination before all necessary tools are identified. In contrast, PA-RAG preserves contextual purity by decomposing tasks into independent

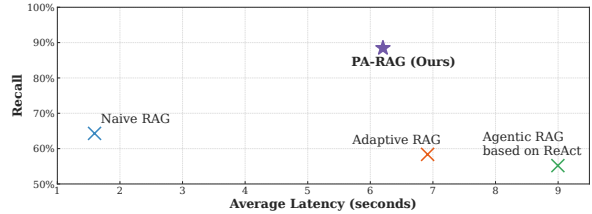


Figure 3: Performance-Efficiency Pareto frontier on GPT 4o Mini. PA-RAG (purple star) achieves the optimal trade-off, outperforming Agentic RAG based on ReAct in speed while surpassing Naive RAG in coverage.

sub-queries. Such isolation effectively mitigates error propagation inherent in iterative baselines.

Universality Across Diverse Lightweight LLM Backbones. PA-RAG demonstrates consistent superiority across a diverse range of lightweight LLM backbones. These include closed-source GPT-4o Mini and various open-source models. When using Llama-3.1-7B-Instruct as the backbone, the approach elevates Recall on the complex set from 35.0% to 82.5%. This finding validates the universality of PA-RAG, which integrates topology-aware indexing, cascade retrieval, and complexity-adaptive dual-path execution. This systematic design ensures structural robustness. It reduces the retrieval space noise and the reasoning complexity of compound tasks, thereby alleviating the processing overhead for lightweight LLMs.

Balancing Inference Efficiency and Tool Coverage. As visualized in Figure 3, we analyze the trade-off between recall and latency. While Naive RAG exhibits the lowest latency due to its one-shot nature, its insufficient recall renders it impractical for reliable tool discovery. In contrast, PA-RAG reduces average latency by approximately 30%- 40% compared to the Agentic RAG based on ReAct, and achieves a massive gain in tool coverage. Notably, the plot reveals that PA-RAG dominates both Adaptive and Agentic RAG based on ReAct (i.e., higher recall and lower latency), establishing a superior performance-latency Pareto front enabled by complexity-adaptive dual-path execution.

Additionally, we comprehensively conducted a case study of PA-RAG and all baselines dealing with the specific process of a complex query, which requires tools across multiple MCP servers. The detailed analysis are illustrated in (Sec B in Appendix).

Methods	Simple Set				Complex Set				Comprehensive Set			
	Prec.	Rec.	F1	Lat.(s)	Prec.	Rec.	F1	Lat.(s)	Prec.	Rec.	F1	Lat.(s)
PA-RAG (Full)	88.3	92.5	89.6	3.08	80.4	85.7	81.3	10.55	84.3	88.5	85.2	6.20
A1: w/o Hierarchy	87.9	91.3	88.3	2.14	69.8	82.5	74.0	9.74	79.0	86.9	81.4	6.16
A2.1: Simple Only	85.0	86.3	85.0	1.54	77.7	56.0	64.1	2.59	80.8	71.1	74.4	1.74
A2.2: Complex Only	61.2	88.8	67.8	8.37	80.9	84.4	80.8	10.18	69.8	84.6	72.8	8.20
A3: w/o Reranker	82.1	87.5	83.8	2.78	78.9	82.8	79.6	8.77	81.4	84.6	82.1	5.86

Table 2: Partial ablation results on MCP-ToolBench on **GPT-4o Mini** backbone, and complete results of all ablation variants are detailed in Table 3 in the Appendix

5.3 Ablation Study

To evaluate the contribution of each component within PA-RAG, we conduct comprehensive ablation experiments on the MCP-ToolBench. For brevity, we present the ablation analysis using GPT-4o-mini as a representative backbone in Table 2. The complete ablation results across all evaluated lightweight LLM backbones are detailed in Table 3 in the Appendix.

Effectiveness of Topology-Aware Hierarchy (A1). Removing the hierarchical indexing and server pruning (A1) results in a noticeable degradation across all metrics, particularly on the complex set, where the F1-score drops from 81.3% to 74.0%. This performance dip highlights the critical role of the server-tool topology, demonstrating its effectiveness in noise reduction. By narrowing the retrieval space, the hierarchical structure acts as an architectural prior, effectively filtering out domain-level noise. Without it, the retriever is easily distracted by semantically similar but functionally irrelevant tools from incorrect servers, leading to a significant drop in precision from 80.4% to 69.8%.

The Criticality of Complexity-Adaptive Dual-Path Execution (A2). We analyze the tool coverage and efficiency trade-off by forcing PA-RAG into a single execution path. The simple path only (A2.1) achieves the lowest latency (1.54s) but suffers from a catastrophic recall collapse on the complex set from 85.7% to 56.0%, confirming that direct RAG lacks the semantic granularity for multi-server queries. Conversely, Complex Path Only (A2.2) maintains high recall but incurs prohibitive computational overhead, increasing latency by approximately 3 \times on simple queries. Furthermore, it causes a severe drop in precision on the simple set from 88.3% to 61.2%. This is attributed to over-decomposition, where straightforward intents are fragmented, introducing un-

necessary semantic noise. PA-RAG effectively navigates this trade-off. Although A2.2 occasionally yields marginally higher recall (see Table 3 in Appendix), negligible routing errors are tolerable to achieve substantial latency reduction and enhanced precision on simple tasks.

The Necessity of Fine-Grained Reranking (A3). Removing the intermediate reranker (A3) leads to a consistent decline in both precision and recall. This exposes the limitations of relying solely on dense vector retrieval of the Bi-encoder, which fails to distinguish subtle semantic variations. By integrating the Cross-Encoder-based BGE-Reranker, PA-RAG leverages deep token-level interactions to capture subtle relevance signals that are missed by vector dot products. This ensures high-fidelity candidates are passed to the next stage, preventing noise propagation in the cascade pipeline.

6 Conclusion

In this paper, we present PA-RAG, a novel approach designed to address the tool discovery bottleneck in the MCP ecosystem. PA-RAG effectively overcomes the protocol-agnostic limitation of existing approaches. It achieves this through integrating topology-aware indexing, cascade retrieval, and complexity-adaptive dual-path execution. Extensive experiments with multiple real-world MCP servers across diverse lightweight LLMs confirm that PA-RAG consistently achieves high tool discovery coverage and accuracy. Ultimately, PA-RAG provides a scalable and efficient paradigm for tool discovery, serving as a robust foundation for future LLM applications in complex MCP environments.

Limitations

Dependency on Intrinsic Backbone Capabilities. The core components of PA-RAG, such

as the adaptive router and semantic decomposer, function as soft-coded logic modules driven by prompts rather than deterministic algorithms. Although PA-RAG effectively structures the hierarchical indexing and cascade retrieval to minimize ambiguity, the precision of the final output remains partially influenced by the underlying LLM’s reasoning stability. This implies that while our PA-RAG ensures high robustness, the system’s ability to reach its theoretical optimum, particularly in strictly adhering to complex constraints, is correlated with the backbone’s alignment quality. Consequently, limitations in the model’s instruction-following capabilities may constrain the granularity or optimality of the generated solutions, rather than compromising the system’s overall functionality.

Lack of Fine-Grained Parameter Schema Alignment. The current retrieval paradigm primarily focuses on semantic matching between the query and the tool’s functional description, without explicitly verifying the availability and compatibility of specific tool parameters within the context. For complex queries involving multiple servers, this limitation can create a gap between finding the right tool and executing it correctly. For instance, the retriever might correctly identify a `calculate_loan` tool. Still, the subsequent generation phase might hallucinate missing arguments or cause type mismatches due to the lack of an explicit schema validation module during the aggregation stage. This could lead to execution failures in rigorous industrial environments.

Discovery Challenges in a Decentralized Ecosystem. The MCP landscape is highly decentralized and community-driven, with diverse servers developed by independent organizations and hosted on disparate platforms, such as GitHub repositories and private registries. While PA-RAG supports automated indexing given a specific server README, it currently lacks a proactive discovery mechanism to aggregate scattered resources in real time efficiently. Consequently, the system’s coverage is bounded by the completeness of the input source list. Developing a scalable crawler or a federated protocol to detect and ingest updates from this fragmented ecosystem autonomously remains an open challenge.

References

Anthropic. 2024a. [Architecture Overview of Model](#)

[Context Protocol](#). Anthropic Technical Documentation. Accessed: 2025-12-18.

Anthropic. 2024b. [Introduction to Model Context Protocol \(MCP\)](#). Anthropic Technical Documentation. Accessed: 2025-12-18.

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*.

Harrison Chase. 2022. [Langchain](#). GitHub repository. Accessed: 2025-12-19.

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.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. [Retrieval-augmented generation for large language models: A survey](#). *arXiv:2312.10997*.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. [Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies](#). *Transactions of the Association for Computational Linguistics*, 9:346–361.

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*.

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.

735	Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model context protocol (mcp): Landscape, security threats, and future research directions . <i>arXiv:2503.23278</i> .	793
736		794
737		795
738		796
739	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 <i>Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)</i> , pages 7036–7050, Mexico City, Mexico. Association for Computational Linguistics.	797
740		798
741		799
742		800
743		
744		801
745		802
746		803
747		804
748		805
749	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 <i>Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing</i> , pages 7969–7992.	806
750		807
751		808
752		809
753		810
754		811
755	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 <i>Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing</i> , pages 1177–1194, Suzhou, China. Association for Computational Linguistics.	812
756		813
757		
758		814
759		815
760		816
761		817
762		818
763		819
764		
765		820
766		821
767		822
768		823
769		824
770	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 <i>Advances in Neural Information Processing Systems</i> , volume 33, pages 9459–9474. Curran Associates, Inc.	825
771		826
772		827
773		828
774		829
775		830
776		831
777		
778		832
779		833
780		834
781		835
782		836
783		837
784		838
785		
786		839
787		840
788		841
789		
790		842
791		843
792		844
		845
		846
		847
		848
		849

850	<i>the Association for Computational Linguistics: ACL</i>	Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin	906
851	2025, pages 356–371, Vienna, Austria. Association	Zhang, Huan Lin, Baosong Yang, Pengjun Xie,	907
852	for Computational Linguistics.	An Yang, Dayiheng Liu, Junyang Lin, Fei Huang,	908
853	Nandan Thakur, Nils Reimers, Andreas Rücklé, Ab-	and Jingren Zhou. 2025. <i>Qwen3 embedding: Ad-</i>	909
854	hishek Srivastava, and Iryna Gurevych. 2021. <i>BEIR:</i>	<i>Advancing text embedding and reranking through founda-</i>	910
855	<i>A heterogeneous benchmark for zero-shot evalua-</i>	<i>tion models.</i> <i>arXiv:2506.05176.</i>	911
856	<i>tion of information retrieval models.</i> In <i>Thirty-fifth</i>	Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei,	912
857	<i>Conference on Neural Information Processing Sys-</i>	Nathan Scales, Xuezhi Wang, Dale Schuurmans,	913
858	<i>tems Datasets and Benchmarks Track (Round 2).</i>	Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H.	914
859	Harsh Trivedi, Niranjan Balasubramanian, Tushar	Chi. 2023. <i>Least-to-most prompting enables com-</i>	915
860	Khot, and Ashish Sabharwal. 2023. <i>Interleav-</i>	<i>plex reasoning in large language models.</i> In <i>The</i>	916
861	<i>ing retrieval with chain-of-thought reasoning for</i>	<i>Eleventh International Conference on Learning Rep-</i>	917
862	<i>knowledge-intensive multi-step questions.</i> In <i>Pro-</i>	<i>resentations.</i>	918
863	<i>ceedings of the 61st Annual Meeting of the Associa-</i>	Ziyuan Zhuang, Zhiyang Zhang, Sitao Cheng, Fangkai	919
864	<i>tion for Computational Linguistics (Volume 1: Long</i>	Yang, Jia Liu, Shujian Huang, Qingwei Lin, Saravan	920
865	<i>Papers)</i> , pages 10014–10037, Toronto, Canada. As-	Rajmohan, Dongmei Zhang, and Qi Zhang. 2024.	921
866	sociation for Computational Linguistics.	<i>EfficientRAG: Efficient retriever for multi-hop ques-</i>	922
867	Yu Wang, Nedim Lipka, Ruiyi Zhang, Alexa Siu,	<i>tion answering.</i> In <i>Proceedings of the 2024 Con-</i>	923
868	Yuying Zhao, Bo Ni, Xin Wang, Ryan Rossi, and	<i>ference on Empirical Methods in Natural Language</i>	924
869	Tyler Derr. 2024. <i>Topology-aware retrieval aug-</i>	<i>Processing</i> , pages 3392–3411, Miami, Florida, USA.	925
870	<i>mentation for text generation.</i> In <i>Proceedings of</i>	Association for Computational Linguistics.	926
871	<i>the 33rd ACM International Conference on Informa-</i>		
872	<i>tion and Knowledge Management, CIKM '24</i> , page		
873	24422452, New York, NY, USA. Association for		
874	Computing Machinery.		
875	Georg Wölflein, Dyke Ferber, Daniel Truhn, Ogn-		
876	jen Arandjelovic, and Jakob Nikolas Kather. 2025.		
877	<i>LLM agents making agent tools.</i> In <i>Proceedings</i>		
878	<i>of the 63rd Annual Meeting of the Association for</i>		
879	<i>Computational Linguistics (Volume 1: Long Papers)</i> ,		
880	pages 26092–26130, Vienna, Austria. Association		
881	for Computational Linguistics.		
882	Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas		
883	Muennighoff. 2023. <i>Bge: Beijing academy of artifi-</i>		
884	<i>cial intelligence general embedding.</i> GitHub reposi-		
885	tory. Accessed: 2025-12-19.		
886	An Yang, Baosong Yang, Binyuan Hui, Bo Zheng,		
887	Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan		
888	Li, Dayiheng Liu, Fei Huang, Guanting Dong, Hao-		
889	ran Wei, Huan Lin, Jialong Tang, Jialin Wang,		
890	Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin		
891	Ma, and 43 others. 2024. <i>Qwen2 technical report.</i>		
892	<i>arXiv:2407.10671.</i>		
893	Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio,		
894	William Cohen, Ruslan Salakhutdinov, and Christo-		
895	pher D. Manning. 2018. <i>HotpotQA: A dataset</i>		
896	<i>for diverse, explainable multi-hop question answer-</i>		
897	<i>ing.</i> In <i>Proceedings of the 2018 Conference on Em-</i>		
898	<i>pirical Methods in Natural Language Processing</i> ,		
899	pages 2369–2380, Brussels, Belgium. Association		
900	for Computational Linguistics.		
901	Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak		
902	Shafran, Karthik R. Narasimhan, and Yuan Cao.		
903	2023. <i>ReAct: Synergizing reasoning and acting</i>		
904	<i>in language models.</i> In <i>The Eleventh International</i>		
905	<i>Conference on Learning Representations.</i>		

A Additional Experiments

In this section, we provide supplementary empirical evidence to further validate the versatility, robustness, and efficiency of PA-RAG.

To demonstrate the comprehensive superiority of our method, we present a multi-dimensional evaluation in Figure 4. This radar chart highlights that PA-RAG achieves the most balanced performance across complex set recall, complex set precision, comprehensive F1-Score, and inference speed, effectively mitigating the trade-offs observed in all baselines.

We delve into the component-level contributions through detailed ablation studies. Figure 5 illustrates the capability loss Δ . Recall when key components are removed, confirming the necessity of the topology-aware hierarchy and cascade retrieval, complexity-adaptive router, and reranker. To complement this, Figure 6 illustrates the trade-off between capability loss and efficiency gain, justifying our adaptive routing strategy while the simple only path offers significant speedups (up to 78%), it incurs unacceptable recall degradation in complex scenarios, necessitating the full PA-RAG pipeline.

Finally, to verify the generalization ability of our PA-RAG, Figure 7 reports the performance across four distinct lightweight LLM backbones: Qwen2, Qwen2.5, GPT-4o Mini, and Llama-3.1. PA-RAG consistently outperforms all baselines on the Comprehensive Set, proving that its effectiveness is architectural rather than model-specific. Notably, we observe that for Llama-3.1, removing retrieval constraints, e.g., hierarchy or reranker, increases latency, suggesting that the structured context of PA-RAG also helps reduce redundant reasoning tokens in verbose models.

B Case Study

To intuitively understand the superiority of PA-RAG, we present a real-world case study involving a complex, cross-server query: "Find today's top 5 gainers, use Firecrawl to ... and finally create a new Excel worksheet to insert the results." This task requires coordinating tools across three distinct MCP Servers: the investor-agent, Firecrawl, and Excel MCP Servers. The process of PA-RAG and all baselines to deal with this problem is indicated in Figs. 8–12.

Baselines Failure Analysis. As shown in Figs. 8–10, Naive RAG performs a single flat retrieval,

fetching a mixture of finance and search tools but completely failing to retrieve the necessary Excel tools, rendering the downstream task impossible. Agentic RAG initiates a ReAct loop and successfully executes the first two steps, fetching stock data and searching the web. However, it fails at the final "productivity" step. Due to the limited context window and retrieval noise from previous steps, the agent suffers from reasoning hallucinations; it attempts to invoke the Excel API > create_worksheet action without successfully retrieving the corresponding tool into its context, causing the process to terminate prematurely. Adaptive RAG, recognizing the query's complexity, routes it to the Agentic path and suffers the same failure.

PA-RAG Success Analysis. In contrast, PA-RAG completes the full workflow, as shown in Figs. 10–12. First, the Semantic Decomposer breaks the complex query into four granular sub-queries (e.g., Sub-query 3: "Create a new Excel worksheet"). Then the Cascade Retrieval Strategy plays a crucial role. For sub-queries 3 and 4, the Server Pruning stage explicitly identifies the Excel MCP Server, filtering out irrelevant noise from the Finance and Search domains. This allows the subsequent Scoped Tool Selection to retrieve create_worksheet and write_data_to_excel accurately. While the total inference time (8.70s) is marginally higher than the failed Agentic attempt (8.05s), PA-RAG is the only one that correctly orchestrates the multi-hop process without hallucination, demonstrating the efficacy of our hierarchical pruning mechanism in handling multi-server tasks.

Methods	Simple Set				Complex Set				Comprehensive Set			
	Prec.	Rec.	F1	Lat.(s)	Prec.	Rec.	F1	Lat.(s)	Prec.	Rec.	F1	Lat.(s)
<i>LLM Backbone: Qwen2-7B-Instruct</i>												
PA-RAG (Full)	93.8	95.0	94.2	2.04	87.5	83.0	84.1	6.36	89.9	88.4	88.5	4.12
A1: w/o Hierarchy	90.4	88.8	88.1	1.19	80.8	78.1	78.4	5.58	84.9	83.1	82.9	3.39
A2.1: Simple Only	87.5	85.0	85.8	0.96	66.0	51.8	56.7	1.76	77.0	68.4	71.4	1.36
A2.2: Complex Only	73.3	81.3	75.4	4.65	86.5	83.7	84.5	5.68	79.8	81.0	79.3	5.02
A3: w/o Reranker	81.3	82.5	81.7	1.92	79.5	72.9	74.5	6.08	82.2	77.1	78.4	3.88
<i>LLM Backbone: Qwen2.5-7B-Instruct</i>												
PA-RAG (Full)	92.1	95.0	92.9	1.76	79.7	75.3	75.4	6.59	85.6	84.4	83.8	4.15
A1: w/o Hierarchy	92.5	91.3	91.7	1.03	71.9	72.4	70.1	5.94	81.6	82.3	80.7	3.51
A2.1: Simple Only	89.6	91.3	89.6	1.07	74.3	53.5	60.3	1.85	81.9	72.4	75.0	1.45
A2.2: Complex Only	74.4	81.3	76.0	2.62	80.3	78.6	78.7	6.11	75.8	77.5	75.1	4.36
A3: w/o Reranker	83.8	85.0	84.2	1.78	77.6	73.2	73.5	6.28	80.4	78.5	78.6	4.06
<i>LLM Backbone: GPT-4o Mini</i>												
PA-RAG (Full)	88.3	92.5	89.6	3.08	80.4	85.7	81.3	10.55	84.3	88.5	85.2	6.20
A1: w/o Hierarchy	87.9	91.3	88.3	2.14	69.8	82.5	74.0	9.74	79.0	86.9	81.4	6.16
A2.1: Simple Only	85.0	86.3	85.0	1.54	77.7	56.0	64.1	2.59	80.8	71.1	74.4	1.74
A2.2: Complex Only	61.2	88.8	67.8	8.37	80.9	84.4	80.8	10.18	69.8	84.6	72.8	8.20
A3: w/o Reranker	82.1	87.5	83.8	2.78	78.9	82.8	79.6	8.77	81.4	84.6	82.1	5.86
<i>LLM Backbone: Llama-3.1-8B-Instruct</i>												
PA-RAG (Full)	78.1	90.0	81.3	4.02	57.4	82.5	66.9	11.62	64.7	86.1	70.7	7.34
A1: w/o Hierarchy	75.0	87.5	78.1	4.31	58.0	78.3	62.3	10.15	60.5	83.8	66.9	8.68
A2.1: Simple Only	68.8	86.3	74.2	1.13	60.2	52.9	54.7	2.05	64.9	70.2	64.8	1.60
A2.2: Complex Only	32.5	77.5	42.1	6.53	55.8	81.3	65.4	7.10	44.7	78.8	54.4	6.66
A3: w/o Reranker	60.5	82.5	66.9	5.59	55.7	79.6	64.3	11.82	56.4	81.2	63.9	8.89

Table 3: Comprehensive ablation studies validating the contribution of each component across various lightweight LLM backbones. We report Precision (Prec.), Recall (Rec.), F1-Score (F1) in percentage (%), and End-to-End Average Latency (Lat.) in seconds.

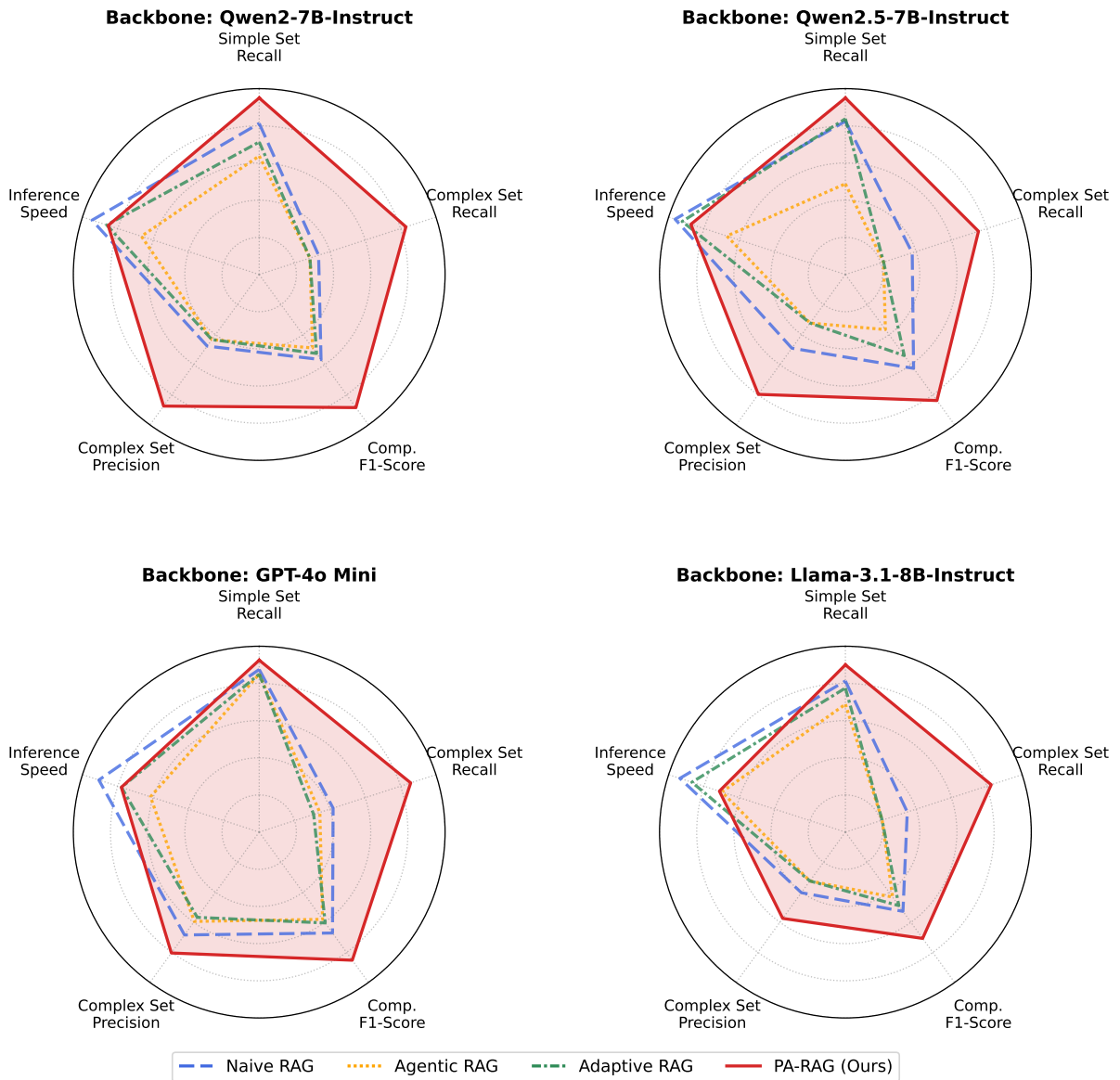


Figure 4: Multi-dimensional evaluation of PA-RAG and representative baselines across four lightweight LLM backbones. The radar charts visualize the balance between tool discovery effectiveness (Recall, Precision, and F1-score) and inference efficiency (Inference Speed). A larger and more balanced polygon area indicates superior overall performance. PA-RAG consistently demonstrates the highest versatility, effectively mitigating the trade-off between accuracy and speed that often hampers Naive RAG and Agentic paradigms.

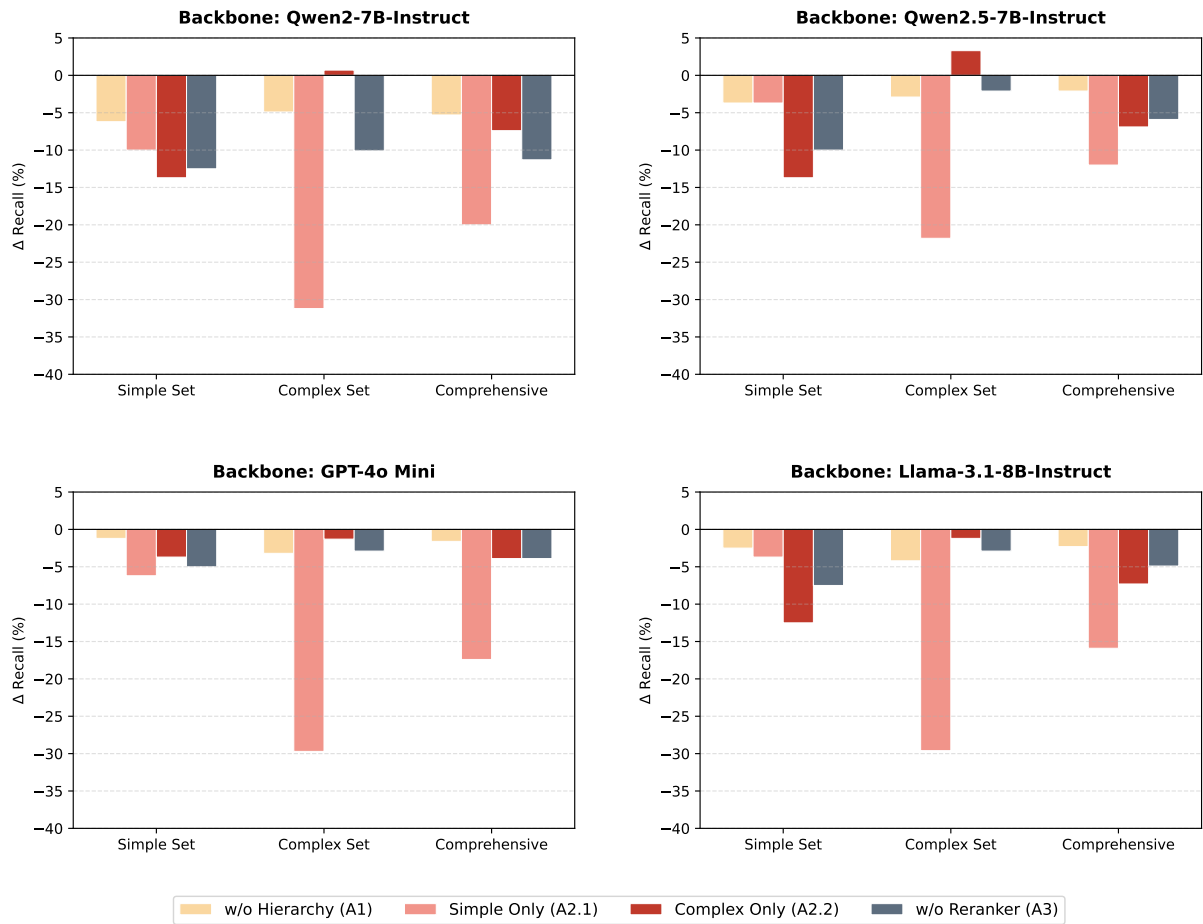


Figure 5: Ablation analysis showing the performance degradation (Δ Recall) when specific components are removed from PA-RAG. The results cover four backbones and three evaluation subsets. Variations include: A1 (w/o Hierarchy), A2.1 (Simple Path Only), A2.2 (Complex Path Only), and A3 (w/o Reranker). The significant drop in Complex Set Recall for the "Simple Only" variant (A2.1) underscores the necessity of the complexity-aware classifier and the agentic reasoning path for multi-hop tool coordination.

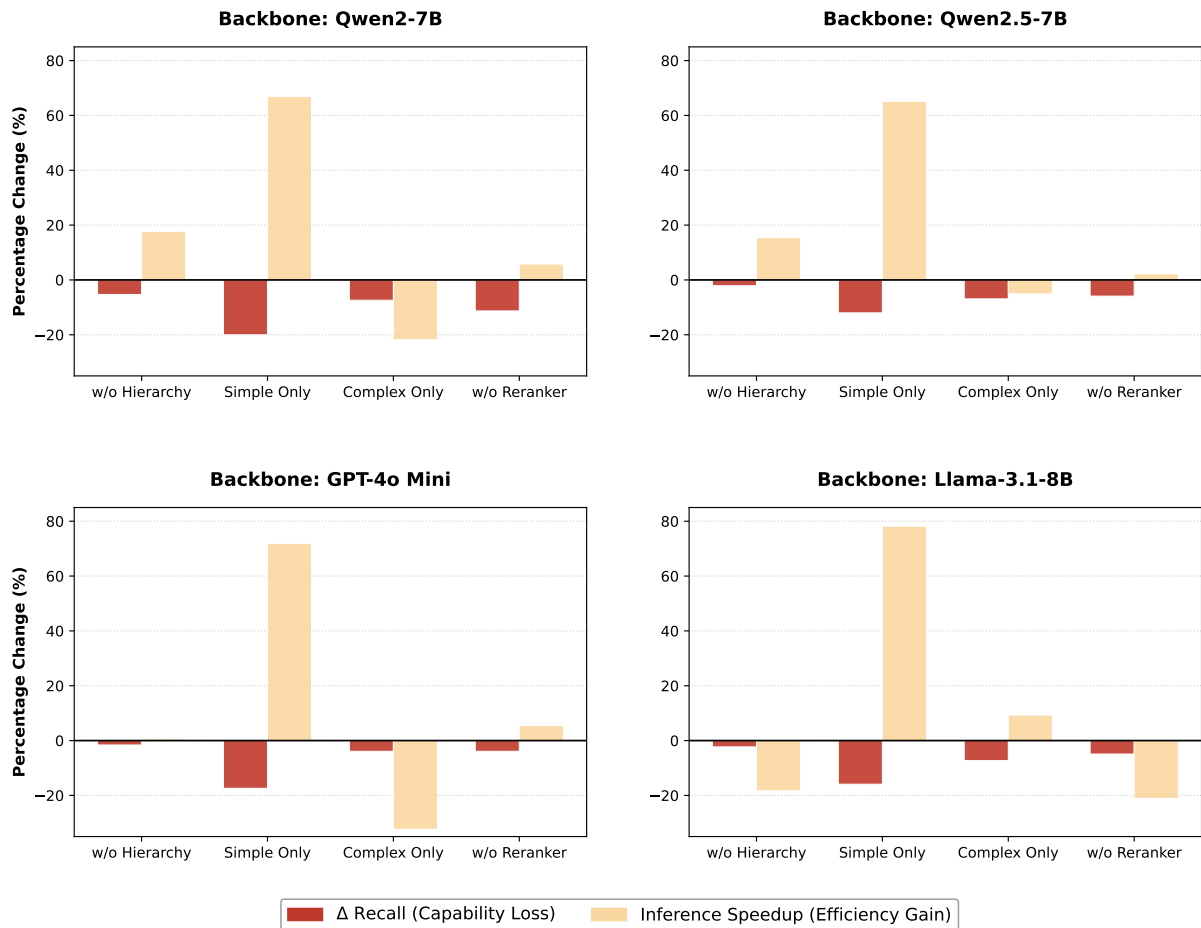


Figure 6: Detailed trade-off analysis on the Comprehensive Set, comparing the capability loss (Δ Recall) against the inference efficiency gain (Speedup) for different ablation variants. The visualization justifies the adaptive cascading architecture of PA-RAG: while simpler paths (e.g., Simple Only) provide substantial speedups, they incur unacceptable recall losses. PA-RAG achieves an optimal balance by dynamically routing queries to ensure high coverage while maintaining competitive inference speeds.

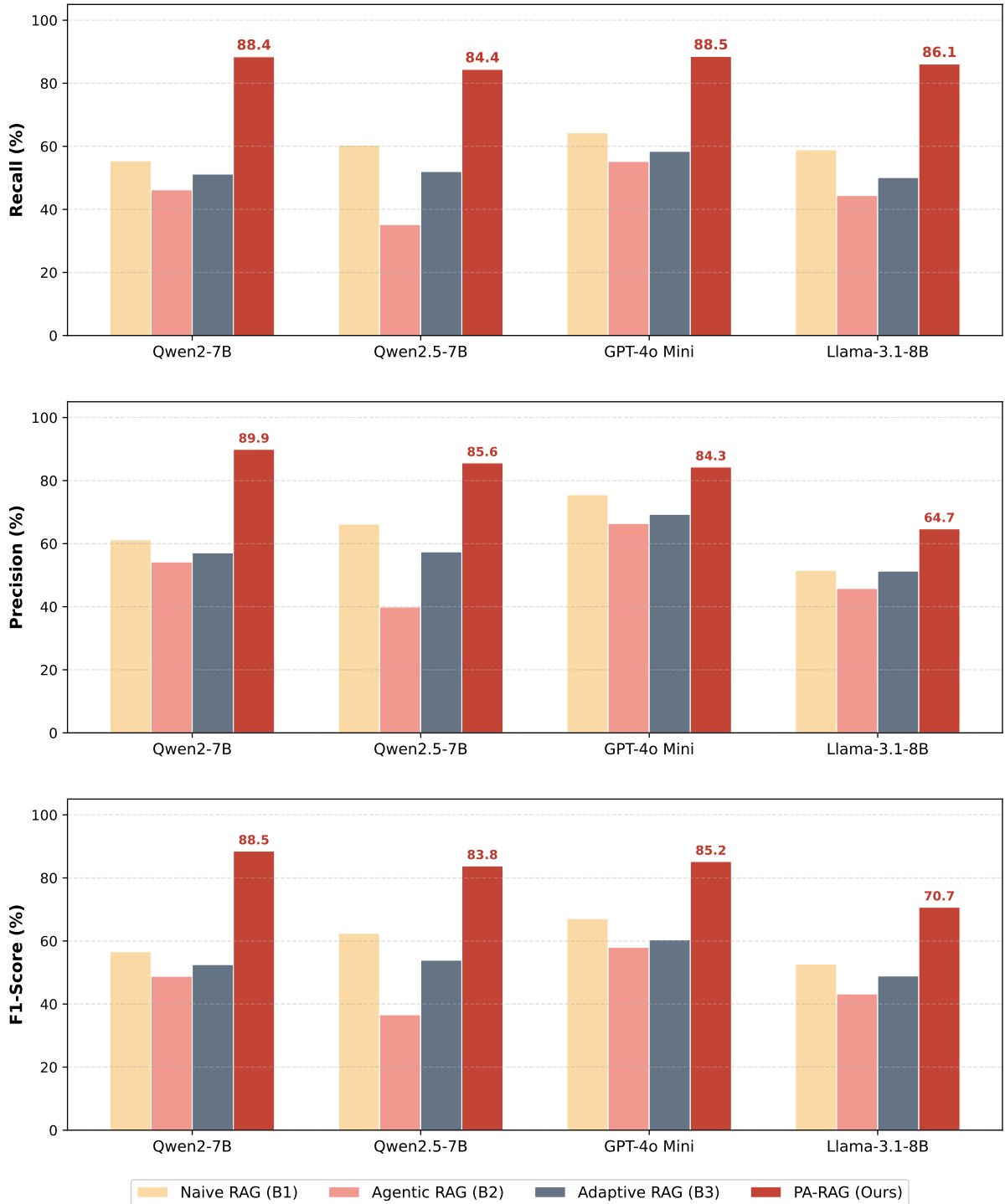


Figure 7: Comparison of PA-RAG and baselines on the Comprehensive Set across four distinct LLM architectures. Results are reported for Recall (primary metric), Precision, and F1-score. PA-RAG consistently outperforms all baseline retrieval paradigms by a significant margin across all models, demonstrating robust generalization. The consistent performance gap indicates that PA-RAG’s superiority stems from its architectural design rather than from specific backbone optimizations.

Query

Find today's top 5 gainers. then use Firecrawl to search web of each company and extract the latest content of results. Finally, create a new Excel worksheet and insert the crawled results into it.

Naive RAG

-----B1: Naive RAG process begins-----

Received User Query: Find today's top 5 gainers. then use Firecrawl to search web of each company and extract the latest content of results. Finally, create a new Excel worksheet and insert the crawled results into it.

Flat Retrieval completed, result:

1: MCP Server: investor-agent	Tool: get_ticker_data
2: MCP Server: Exa MCP Server	Tool: company_research
3: MCP Server: investor-agent	Tool: get_market_movers
4: MCP Server: Exa MCP Server	Tool: web_search_exa
5: MCP Server: investor-agent	Tool: get_google_trends
6: MCP Server: investor-agent	Tool: get_nasdaq_earnings_calendar
7: MCP Server: investor-agent	Tool: get_earnings_history
8: MCP Server: Firecrawl MCP Server	Tool: Search Tool
9: MCP Server: Brave Search MCP Server	Tool: Web Search (brave_web_search)
10: MCP Server: Firecrawl MCP Server	Tool: Crawl Tool

Total time: 1.50s, final answer:

```
[{'mcp_server': 'investor-agent', 'tool': 'get_market_movers'}, {'mcp_server': 'Firecrawl MCP Server', 'tool': 'Crawl Tool'}, {'mcp_server': 'Brave Search MCP Server', 'tool': 'Web Search (brave_web_search)'}, {'mcp_server': 'Firecrawl MCP Server', 'tool': 'Search Tool'}]
```

Agentic RAG

-----B2: Agentic RAG based on ReAct process begins-----

Received User Query: Find today's top 5 gainers. then use Firecrawl to search web of each company and extract the latest content of results. Finally, create a new Excel worksheet and insert the crawled results into it.

Launch ReAct Agent on an empty toolset ...

--- ReAct Iteration 1 ---

ReAct Agent newly generated output:

I need to find today's top 5 gainers, search the web for each company, and then create a new Excel worksheet to insert the crawled results. I should start by finding today's top 5 gainers.

Action: Tool Search > tool_search

Action Input: top 5 gainers today

Agent Action: Tool Search > tool_search. New Query: 'top 5 gainers today'

Use Flat Retrieval search the candidate toolset of New Query ...

Flat Retrieval completed, result:

1: MCP Server: investor-agent	Tool: get_market_movers
2: MCP Server: investor-agent	Tool: get_nasdaq_earnings_calendar
3: MCP Server: investor-agent	Tool: get_institutional_holders
4: MCP Server: investor-agent	Tool: get_ticker_data
5: MCP Server: investor-agent	Tool: get_earnings_history
6: MCP Server: investor-agent	Tool: get_insider_trades
7: MCP Server: investor-agent	Tool: get_crypto_fear_greed_index
8: MCP Server: Redis MCP Server	Tool: sorted set
9: MCP Server: Exa MCP Server	Tool: company_research
10: MCP Server: Brave Search MCP Server	Tool: News Search (brave_news_search)

--- ReAct Iteration 2 ---

ReAct Agent newly generated output:

I now have a list of tools to choose from. I should select the 'get_market_movers' tool from the investor-agent API to find today's top 5 gainers.

Figure 8: Detailed visualization of the Case Study (Page 1). This page illustrates the comparison between Naive, Agentic, Adaptive RAG, and the proposed PA-RAG across a multi-server query.

Action: investor-agent > get_market_movers
Action Input: {"category": "top gainers", "session": "regular"}
Observation: Tool 'investor-agent > get_market_movers' successfully executes and get the right result.

--- ReAct Iteration 3 ---
ReAct Agent newly generated output:
I have the top 5 gainers for today. Next, I need to search the web for each company and extract the latest content. I should use Firecrawl for this task.

Action: Exa MCP Server > company_research
Action Input: {"companies": ["Company A", "Company B", "Company C", "Company D", "Company E"], "actions": ["web_search"]}
Observation: Tool 'Exa MCP Server > company_research' successfully executes and get the right result.

--- ReAct Iteration 4 ---
ReAct Agent newly generated output:
I have the web search results for each of the top 5 gainers. Now, I need to create a new Excel worksheet and insert the crawled results into it. I will use the Excel API to accomplish this task.

Action: Excel API > create_worksheet
Action Input: {"worksheet_name": "Today's Top Gainers", "columns": ["Company", "Latest News"]}

Agent occur **Reasoning Hallucination**, generated an Action that was not in the available Toolset: Excel API > create_worksheet, the process **terminate prematurely**

Total time: 8.05s, final answer:
[{'mcp_server': 'investor-agent', 'tool': 'get_market_movers'}, {'mcp_server': 'Exa MCP Server', 'tool': 'company_research'}]

Adaptive RAG

-----B3: Adaptive RAG process begins-----

Received User Query: Find today's top 5 gainers. then use Firecrawl to search web of each company and extract the latest content of results. Finally, create a new Excel worksheet and insert the crawled results into it.

Use the Classifier to determine the Query's complexity...
Route to a [Complex Path] process (Agentic RAG based on ReAct)

Launch ReAct Agent on an empty toolset...
--- ReAct Iteration 1 ---
ReAct Agent newly generated output:
I need to find today's top 5 gainers, search the web for each company, and then create a new Excel worksheet to insert the crawled results. I should start by finding today's top 5 gainers.

Action: Tool Search > tool_search
Action Input: top 5 gainers today
Agent Action: Tool Search > tool_search. New Query: 'top 5 gainers today'

Use Flat Retrieval search the candidate toolset of New Query ...
Flat Retrieval completed, result:

1: MCP Server: investor-agent	Tool: get_market_movers
2: MCP Server: investor-agent	Tool: get_nasdaq_earnings_calendar
3: MCP Server: investor-agent	Tool: get_institutional_holders
4: MCP Server: investor-agent	Tool: get_ticker_data
5: MCP Server: investor-agent	Tool: get_earnings_history
6: MCP Server: investor-agent	Tool: get_insider_trades
7: MCP Server: investor-agent	Tool: get_crypto_fear_greed_index
8: MCP Server: Redis MCP Server	Tool: sorted set
9: MCP Server: Exa MCP Server	Tool: company_research
10: MCP Server: Brave Search MCP Server	Tool: News Search (brave_news_search)

Figure 9: Detailed visualization of the Case Study (Page 2). This page illustrates the comparison between Naive, Agentic, Adaptive RAG, and the proposed PA-RAG across a multi-server query.

```

--- ReAct Iteration 2 ---
ReAct Agent newly generated output:
I now have a list of tools to choose from. I should select the 'get_market_movers' tool from the investor-agent API to find today's top 5 gainers.
Action: investor-agent > get_market_movers
Action Input: {"category": "top gainers", "session": "regular"}
Observation: Tool 'investor-agent > get_market_movers' successfully executes and get the right result.

--- ReAct Iteration 3 ---
ReAct Agent newly generated output:
I have the top 5 gainers for today. Next, I need to search the web for each company and extract the latest content. I should use Firecrawl for this task.
Action: Exa MCP Server > company_research
Action Input: {"companies": ["Company A", "Company B", "Company C", "Company D", "Company E"], "actions": ["web_search"]}
Observation: Tool 'Exa MCP Server > company_research' successfully executes and get the right result.

--- ReAct Iteration 4 ---
ReAct Agent newly generated output:
I have the web search results for each of the top 5 gainers. Now, I need to create a new Excel worksheet and insert the crawled results into it. I will use the Excel API to accomplish this task.
Action: Excel API > create_worksheet
Action Input: {"worksheet_name": "Today's Top Gainers", "columns": ["Company", "Latest News"]}

Agent occur Reasoning Hallucination, generated an Action that was not in the available Toolset: Excel API > create_worksheet, the process terminate prematurely.

Total time: 8.35s, final answer:
[{'mcp_server': 'investor-agent', 'tool': 'get_market_movers'}, {'mcp_server': 'Exa MCP Server', 'tool': 'company_research'}]

```

PA-RAG

===== PA-RAG process begins =====

Received User Query: Find today's top 5 gainers. then use Firecrawl to search web of each company and extract the latest content of results. Finally, create a new Excel worksheet and insert the crawled results into it.

Pre-retrieval : obtain the candidate MCP Servers for the **Complexity-Aware Router**
Perform **Stage 1: MCP Server Pruning of Cascade Retrieval Strategy**

- 1: MCP Server: investor-agent Tool: None
- 2: MCP Server: Context7 MCP - Up-to-date Code Docs For Any Prompt Tool: None
- 3: MCP Server: Fetch MCP Server Tool: None

Complexity-Based Dual-Path Execution
Intent classification complete, results: intent='complex', target_servers=[]

Execute the [Complex Path] process
Semantic Decomposer decompose the original query into **sub-queries**: ["Find today's top 5 gainers.", "Use Firecrawl to search web of each company and Extract the latest content of results.", "Create a new Excel worksheet.", "Insert the crawled results into the Excel worksheet."]

==== Execute sub-query 1: 'Find today's top 5 gainers.' ====

Execute the complete **Cascade Retrieval Strategy** for candidate toolset of sub-query 1

Stage 1: MCP Server Pruning: ['Fetch MCP Server', 'Context7 MCP - Up-to-date Code Docs For Any Prompt', 'investor-agent']

Stage 2: Scoped Tool Selection: obtain all the tools of these three MCP sServers were and 10 tools were obtained after reranking:

Figure 10: Detailed visualization of the Case Study (Page 3). This page illustrates the comparison between Naive, Agentic, Adaptive RAG, and the proposed PA-RAG across a multi-server query.

```

1: MCP Server: investor-agent      Tool: get_market_movers
2: MCP Server: investor-agent      Tool: calculate_technical_indicator
3: MCP Server: investor-agent      Tool: get_cnn_fear_greed_index
4: MCP Server: investor-agent      Tool: get_nasdaq_earnings_calendar
5: MCP Server: investor-agent      Tool: get_options
6: MCP Server: investor-agent      Tool: get_google_trends
7: MCP Server: investor-agent      Tool: get_earnings_history
8: MCP Server: investor-agent      Tool: get_crypto_fear_greed_index
9: MCP Server: Context7 MCP - Up-to-date Code Docs For Any PromptTool: get-library-docs
10: MCP Server: investor-agent     Tool: get_ticker_data

Tool Selector select: ["investor-agent > get_market_movers"]

Subquery 1 is completed. Result: [{'mcp_server': 'investor-agent', 'tool': 'get_market_movers'}]

=== Execute sub-query 2: 'Use Firecrawl to search web of each company.' ===
Execute the complete Cascade Retrieval Strategy for candidate toolset of sub-query 2

Stage 1: MCP Server Pruning: ['Airbnb Search & Listings - Desktop Extension (DXT)', 'Exa MCP Server', 'Firecrawl MCP Server']

Stage 2: Scoped Tool Selection: obtain all the tools of these three MCP sServers were and 10 tools were obtained after reranking:
1: MCP Server: Exa MCP Server      Tool: company_research
2: MCP Server: Firecrawl MCP Server Tool: Search Tool
3: MCP Server: Firecrawl MCP Server Tool: Crawl Tool
4: MCP Server: Firecrawl MCP Server Tool: Scrape Tool
5: MCP Server: Firecrawl MCP Server Tool: Map Tool
6: MCP Server: Firecrawl MCP Server Tool: Batch Scrape Tool
7: MCP Server: Firecrawl MCP Server Tool: Extract Tool
8: MCP Server: Firecrawl MCP Server Tool: Check Batch Status
9: MCP Server: Exa MCP Server      Tool: crawling
10: MCP Server: Exa MCP Server     Tool: linkedin_search

Tool Selector select: ["Firecrawl MCP Server > Search Tool"]

Subquery 2 is completed. Result: [{'mcp_server': 'Firecrawl MCP Server', 'tool': 'Search Tool'}]

=== Executing sub-query 3: 'Create a new Excel worksheet.' ===
Execute the complete Cascade Retrieval Strategy for candidate toolset of sub-query 3

Stage 1: MCP Server Pruning: ['Excel MCP Server', 'Airbnb Search & Listings - Desktop Extension (DXT)', 'Context7 MCP - Up-to-date Code Docs For Any Prompt']

Stage 2: MCP Server Pruning: obtain all the tools of these three MCP sServers were and 10 tools were obtained after reranking:
1: MCP Server: Excel MCP Server    Tool: create_worksheet
2: MCP Server: Excel MCP Server    Tool: create_workbook
3: MCP Server: Excel MCP Server    Tool: copy_worksheet
4: MCP Server: Excel MCP Server    Tool: create_table
5: MCP Server: Excel MCP Server    Tool: rename_worksheet
6: MCP Server: Excel MCP Server    Tool: write_data_to_excel
7: MCP Server: Excel MCP Server    Tool: create_chart
8: MCP Server: Excel MCP Server    Tool: apply_formula
9: MCP Server: Excel MCP Server    Tool: insert_rows
10: MCP Server: Excel MCP Server   Tool: insert_columns

Tool Selector select: ["Excel MCP Server > create_worksheet"]

Subquery 3 is completed. Result: [{'mcp_server': 'Excel MCP Server', 'tool': 'create_worksheet'}]

=== Executing sub-query 4: 'Insert the crawled results into the Excel worksheet.' ===
Execute the complete Cascade Retrieval Strategy for candidate toolset of sub-query 4

```

Figure 11: Detailed visualization of the Case Study (Page 4). This page illustrates the comparison between Naive, Agentic, Adaptive RAG, and the proposed PA-RAG across a multi-server query.

```
Stage 1: MCP Server Pruning: ['Excel MCP Server', 'Firecrawl MCP Server', 'Fetch MCP Server']

Stage 2: MCP Server Pruning: obtain all the tools of these three MCP sServers were and 10 tools were
obtained after reranking:
1: MCP Server: Excel MCP Server      Tool: write_data_to_excel
2: MCP Server: Excel MCP Server      Tool: insert_rows
3: MCP Server: Excel MCP Server      Tool: insert_columns
4: MCP Server: Excel MCP Server      Tool: get_data_validation_info
5: MCP Server: Excel MCP Server      Tool: read_data_from_excel
6: MCP Server: Excel MCP Server      Tool: apply_formula
7: MCP Server: Excel MCP Server      Tool: copy_range
8: MCP Server: Excel MCP Server      Tool: create_chart
9: MCP Server: Excel MCP Server      Tool: validate_excel_range
10: MCP Server: Excel MCP Server     Tool: format_range

Tool Selector LLM select: ["Excel MCP Server > write_data_to_excel"]

Subquery 4 is completed. Result: [{'mcp_server': 'Excel MCP Server', 'tool': 'write_data_to_excel'}]

The summary subtask results were completed, and a total of 4 results were found.
----- Complex Path complete -----

Total time: 8.70s, final answer:
[{'mcp_server': 'investor-agent', 'tool': 'get_market_movers'}, {'mcp_server': 'Firecrawl MCP Server',
'tool': 'Search Tool'}, {'mcp_server': 'Excel MCP Server', 'tool': 'create_worksheet'}, {'mcp_server': 'Excel
MCP Server', 'tool': 'write_data_to_excel'}]
```

Figure 12: Detailed visualization of the Case Study (Page 5). This page illustrates the comparison between Naive, Agentic, Adaptive RAG, and the proposed PA-RAG across a multi-server query.