MCP-Zero: Active Tool Discovery for Autonomous LLM Agents

Xiang Fei

Xiawu Zheng*

Hao Feng

xiangf@stu.xmu.edu.cn

zhengxiawu@xmu.edu.cn

haof@mail.ustc.edu.cn

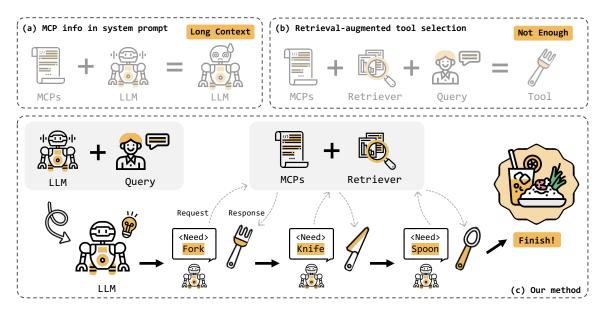


Figure 1. Comparison of tool selection paradigms for LLM agents. (a) System-prompt-based methods inject all MCP tool schemas into the context, resulting in excessive prompt length and inefficiency. (b) Retrieval-augmented approaches select tools by matching the user query once, which may lead to inaccurate or insufficient tool selection, especially in multi-turn scenarios. (c) MCP-Zero enables the LLM to actively analyze the task, iteratively request the most relevant tools as needed, and dynamically construct a multi-step toolchain with minimal context overhead and high accuracy.

Abstract

True intelligence requires active capability acquisition, yet current LLM agents inject pre-defined tool schemas into prompts, reducing models to passive selectors and falling short of robust general-purpose agency. We introduce MCP-Zero, an active agent framework that restores tool discovery autonomy to LLMs themselves. Instead of overwhelming models with all available tools, MCP-Zero enables agents to actively identify capability gaps, and request specific tools on-demand, transforming them from large-scale retrievers into genuine autonomous agents. The framework operates through three core mechanisms: (1) Active Tool Request, where models autonomously generate structured requests specifying their exact tool requirements; (2) Hierarchical Semantic Routing, a two-stage al-

gorithm that matches requests to relevant servers and tools through improved semantic alignment; (3) Iterative Capability Extension, enabling agents to progressively build cross-domain toolchains while maintaining minimal context footprint. We construct MCP-tools, a comprehensive dataset of 308 MCP servers and 2,797 tools from the official Model-Context-Protocol repository. Experiments demonstrate that MCP-Zero preserves agent autonomy while achieving substantial efficiency gains: (i) accurate tool selection from nearly 3k candidates across 248.1k tokens; (ii) 98% reduction in token consumption on APIBank while maintaining high accuracy; and (iii) consistent multi-turn performance that scales with tool ecosystem growth. This work establishes active tool discovery as a fundamental design pattern for scalable autonomous agent systems. The code and dataset is released at https://github.com/ xfey/MCP-Zero.

^{*}Corresponding author

1. Introduction

We only think when we are confronted with a problem.

John Dewey

True thinking emerges not from passive consumption but from active engagement with challenges. Intelligence manifests through the capacity to actively seek solutions rather than merely selecting from predetermined options.

Large language models (LLMs) have evolved rapidly, demonstrating increasingly sophisticated reasoning and problem-solving capabilities [1, 8, 30]. The emergence of function-calling mechanisms has enabled large language models (LLMs) to transcend their parametric boundaries, leveraging external tools, APIs, and execution environments to accomplish complex real-world tasks [14, 18, 21, 26].

However, current tool integration architectures fundamentally compromise agent autonomy. The dominant paradigm injects comprehensive tool schemas into system prompts, forcing agents into a passive role where they select from pre-defined options rather than actively discovering capabilities as needed (Figure 1a). This approach creates two critical problems: **massive context overhead** and **constrained decision autonomy**. For instance, the GitHub MCP server requires over 4,600 tokens for 26 tools, while comprehensive tool ecosystems can exceed 248k tokens—effectively transforming capable reasoning models into overwhelmed database query systems.

Recent retrieval-based approaches attempt to reduce context overhead by pre-selecting relevant tools based on semantic similarity with user queries [7, 15]. While these methods address the scaling problem, they perpetuate the fundamental issue of passive tool consumption. Query like "Debug the file" requires filesystem access, code analysis, and command execution (Figure 1b): static retrieval based on initial queries cannot anticipate the evolving tool needs that emerge during task execution. More critically, this paradigm violates a core principle of autonomous agents—the ability to actively shape their environment based on dynamic assessment of their own capabilities.

The limitations of current approaches stem from three architectural constraints that prevent genuine agent autonomy: (1) external decision authority—tool selection is delegated to retrieval systems rather than the agent itself; (2) semantic distribution gaps—user queries and formal tool specifications exist in different semantic spaces, reducing matching precision; and (3) static capability assumptions—tools are selected once rather than discovered iteratively as task understanding evolves.

Toward Active Tool Discovery: True autonomous agents must retain authority over their capability acquisition. Modern LLMs possess sophisticated reasoning, self-reflection, and planning capabilities that enable them to assess their

```
<function>
{"description": "Search for GitHub repositories",
  "name": "mcp_github_search_repositories",
  "parameters": {"$schema":
  "http://json-schema.org/draft-07/schema#",
  "additionalProperties": false, "properties":
  {"page": {"description": "Page number for
  pagination (default: 1)", "type": "number"},
  "perPage": {"description": "Number of results
  per page (default: 30, max: 100)", "type":
  "number"}, "query": {"description": "Search query
  (see GitHub search syntax)", "type": "string"}},
  "required": ["query"], "type": "object"}}
</function>
```

Figure 2. Example of a single MCP tool definition from the GitHub MCP server. This tool requires 143 tokens, while the complete server requires over 4,600 tokens.

own limitations and articulate specific tool requirements. Rather than constraining agents to pre-selected tool sets, we propose active tool discovery—a paradigm where agents autonomously identify capability gaps and request appropriate tools on-demand. Based on this principle, we introduce MCP-Zero (Figure 1c), an active agent framework that restores tool discovery autonomy to LLMs through three core mechanisms:

Active Tool Request. Instead of passive tool consumption, agents generate structured requests that specify their exact requirements:

```
<tool_assistant>
server: ... # Platform/permission domain
tool: ... # Operation type + target
</tool_assistant>
```

This approach ensures semantic alignment between agent needs and tool documentation while preserving decision autonomy.

Hierarchical Semantic Routing. A two-stage matching algorithm first filters candidate servers by platform requirements, then ranks tools within selected servers based on semantic similarity. This hierarchical approach reduces search complexity while maintaining precision.

Iterative Capability Extension. Agents can discover and integrate tools throughout task execution, building cross-domain capabilities dynamically. When initial tools prove insufficient, agents can refine their requests and discover alternatives, providing natural fault tolerance.

To support systematic evaluation, we also construct MCP-tools, a comprehensive dataset comprising 308 servers and 2,797 tools from the official Model-Context-Protocol repository¹. Experiments demonstrate that MCP-Zero maintains agent autonomy while achieving substantial efficiency gains: 98% reduction in token consumption with preserved accuracy across multi-turn conversations and large-scale tool ecosystems.

¹github.com/modelcontextprotocol/servers

Our main contributions establish active tool discovery as a fundamental design pattern for autonomous agent systems:

- We propose MCP-Zero, enabling agents to maintain decision autonomy through active tool discovery, transforming them from passive selectors to autonomous capability architects.
- We design hierarchical semantic routing that preserves semantic alignment between agent requests and tool specifications while reducing computational complexity.
- We construct and release MCP-tools, providing the research community with a comprehensive evaluation framework for tool discovery systems.
- We demonstrate that active discovery paradigms scale effectively with tool ecosystem growth while maintaining the autonomous decision-making that defines genuine agent systems.

2. Related Work

2.1. Tool-Augmented LLMs

The evolution of tool-augmented large language models has progressed through distinct paradigms, each addressing different aspects of the fundamental challenge: how to effectively integrate external capabilities with language model reasoning.

Early Task-Specific Integration. Initial approaches focused on hard-coding specific tools into language models. MRKL [10] introduced modular reasoning by combining neural networks with symbolic modules like calculators, while WebGPT [16] demonstrated web browsing capabilities for information retrieval tasks. Though effective within their domains, these systems suffered from limited scalability and poor generalization to new tool types.

Universal Agent Protocols. The introduction of Re-Act [31] marked a paradigm shift by establishing the "observation-action-thought" pattern, creating a universal protocol for tool-augmented reasoning. This framework became the foundation for modern agent systems including LangChain Agents [4] and AutoGen [29], enabling flexible and extensible tool integration architectures.

Training-Based Tool Learning. Parallel developments explored learning tool usage through model training. Toolformer [26] pioneered self-supervised learning for API call generation, teaching models to insert tool invocation markers within natural text generation. Gorilla [18] extended this approach by constructing large-scale instruction datasets with tool calls, enabling supervised learning of query-to-tool mappings across diverse API collections.

Context-Based Tool Injection. The emergence of Chat-GPT Function Calling and systems like HuggingGPT [27] introduced context-based approaches, which inject JSON-Schema tool descriptions into system prompts, eliminating the need for specialized training. ART [17] further refined this paradigm by constructing demonstration libraries in chain-of-thought format, leveraging in-context learning for tool selection and usage.

Fundamental Limitations. Despite these advances, existing approaches face critical scalability challenges. Training-based methods require expensive retraining for each toolset update, limiting their adaptability to evolving tool ecosystems. Context-based methods, while more flexible, suffer from prohibitive context overhead when injecting comprehensive tool descriptions—a problem that becomes acute as tool collections scale to thousands of APIs. Most critically, when relevant tools are absent from the predefined context, task completion becomes impossible, highlighting the need for dynamic, on-demand tool discovery mechanisms that can adapt to diverse and evolving task requirements.

2.2. Tool Retrieval for LLMs

The success of Retrieval-Augmented Generation (RAG) in addressing knowledge limitations through the "retrieve-insert-generate" paradigm has inspired its adaptation to tool selection for LLMs [11]. Classical RAG frameworks like REALM [9], RETRO [3] and In-Context RALM [25] demonstrated the effectiveness of dynamically incorporating external knowledge into generation processes. Recent work has extended this paradigm to tool selection, aiming to reduce context overhead by retrieving only the most relevant tools for a given query.

Semantic Similarity-Based Retrieval. Early approaches focused on direct semantic matching between user queries and tool descriptions. Gorilla [18] constructed vector databases from API documentation and usage examples, employing semantic similarity to retrieve relevant tools. Tool2Vec [15] addressed the semantic gap between user requests and formal API descriptions by pre-collecting diverse user invocation patterns and computing averaged embeddings, though this approach requires extensive user interaction datasets for training. RAG-MCP [7] performs server-level matching between user queries and documentation from MCPBench [13], returning all tools from the most similar server as the context of LLMs.

Hierarchical Tool Retrieval. Recognizing the limitations of flat retrieval, several works have explored coarse-to-fine approaches. AnyTool [6] implemented a multi-level retrieval system based on the RapidAPI dataset, constructing separate retrievers for "category-tool-API" hierarchies. ToolRerank [32] leveraged pre-trained BERT models for semantic matching, while COLT [23] employed specialized language models for tool selection. Re-Invoke [5] introduced key information extraction from user queries before tool matching.

Limitations of Current Approaches. Despite reduc-

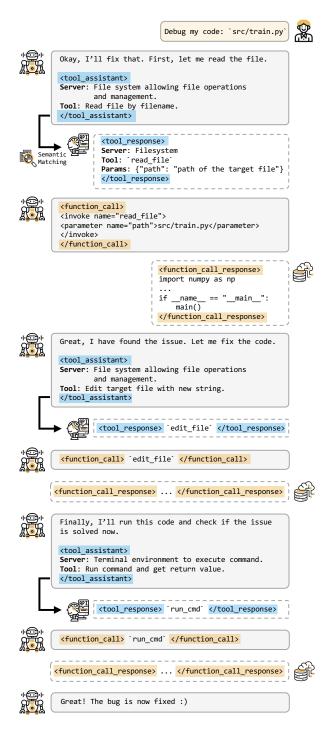


Figure 3. Illustrative example of MCP-Zero's iterative active invocation. The model progressively identifies capability gaps and requests tools across three domains on-demand.

ing context overhead, existing retrieval-based methods face fundamental limitations. First, they usually rely on singleround matching based on initial user queries: if the retrieval fails, task completion becomes impossible. Second, and more critically, user requests are often broad and composite in nature (e.g., "write and run code to crawl AI repositories from GitHub"), requiring decomposition into multiple subtasks that cannot be solved by any single tool. Such scenarios demand progressive task breakdown where the model iteratively identifies capability gaps and requests appropriate tools on-demand, a challenge that current "query-once, retrieve-once" paradigms cannot adequately address.

2.3. Model Context Protocol

The Model Context Protocol (MCP) is an open standard introduced in 2024 to enable secure and uniform access to external tools and services for large language models [2]. By defining a standardized interface through JSON-RPC message exchange, MCP addresses the fragmentation problem in AI tool integration, where different platforms previously required custom connectors and proprietary protocols. This standardization has led to rapid adoption across major AI platforms and development environments, facilitating the creation of hundreds of MCP servers spanning diverse domains including file systems, databases, web services, and specialized APIs.

However, this expansion creates significant context overhead, as traditional approaches inject all server JSON-Schemas into system prompts simultaneously. Current solutions for MCP context optimization remain limited and follow conventional retrieval paradigms [7]. Given the enhanced capabilities of modern LLMs and MCP's potential for agentic AI, we argue that existing methods fail to fully leverage contemporary language models' tool-calling capabilities, motivating our MCP-Zero framework for dynamic, on-demand tool discovery.

3. Method: MCP-Zero

We introduce MCP-Zero through two key aspects: the comprehensive framework design (Section 3.1) and theoretical analysis of active retrieval (Section 3.2). Our approach fundamentally shifts from passive tool injection to active tool discovery, enabling LLMs to dynamically construct task-specific toolchains across diverse domains with minimal context overhead.

3.1. Active Tool Discovery Framework

MCP-Zero is a active agent framework that enables LLMs to dynamically construct task-specific toolchains through on-demand tool retrieval. The framework operates through three core components that work synergistically to address the context overhead and multi-domain coordination challenges of existing approaches.

Overall Workflow. Given a user query such as "Debug my code: src/train.py", the LLM analyzes the task and autonomously determines when external tool assistance is needed. As shown in Figure 3, the model pro-

gressively breaks down the complex debugging task into subtasks: reading the file, analyzing and fixing the code, and validating the fix through execution. At each step, instead of relying on pre-injected tool schemas, the model actively generates structured tool requests. These requests are processed through hierarchical vector routing to retrieve the most relevant tools, which are then injected into the context for immediate use. The model iteratively repeats this process throughout the conversation, constructing a cross-domain toolchain spanning filesystem operations, code editing, and command execution.

Active Tool Request. The foundation of our framework lies in returning tool requirement specification authority to the LLM itself. When the model identifies a capability gap that requires external assistance, it generates a structured request block:

```
<toolassistant>
server: ... # Platform/permission domain
tool: ... # Operation type + target
</toolassistant>
```

This mechanism enables the model to express tool needs spontaneously as they arise during task execution. Compared to raw user queries, the model-generated requests ensures better semantic alignment with tool documentation. The server field specifies the platform or permission domain requirements, while the tool field describes the desired operation type and target.

We design our framework around these two fields because they align naturally with the MCP specification, which mandates that all servers and tools provide descriptive documentation. This inherent requirement ensures consistent semantic information availability across the entire MCP ecosystem, making our retrieval approach universally applicable without additional metadata engineering.

Importantly, the model can generate multiple such requests throughout a single conversation, with each request triggering an independent retrieval process.

Hierarchical Vector Routing. To efficiently locate relevant tools from thousands of candidates, we employ a two-stage coarse-to-fine retrieval algorithm, using OpenAI text-embedding-3-large embeddings for semantic similarity matching.

The system first filters candidate MCP servers by matching the server field against server descriptions. Since server descriptions are typically brief single sentences, we construct extended summaries that include comprehensive usage examples essential for accurate matching (detailed in Section 4). We then perform matching against both the original descriptions and these enhanced summaries, taking the higher similarity score between the two approaches. This dual-matching strategy leverages both original MCP documentation and enhanced summaries to improve retrieval precision.

Subsequently, tools within selected servers are ranked based on semantic similarity between the tool field and tool descriptions. The final ranking score combines both server-level and tool-level similarities:

$$score = (s_{server} \times s_{tool}) \times \max(s_{server}, s_{tool})$$
 (1)

where $s_{\rm server}$ and $s_{\rm tool}$ represent cosine similarities at server and tool levels respectively. This scoring mechanism ensures that high similarity in either dimension contributes significantly to the final ranking, enabling effective recall of highly relevant tools. The system returns the top-k tools, with dynamic adjustment possible when similarity scores are closely clustered. In our experiments, we achieve high accuracy with top-1 retrieval, though we can configure larger k values to enhance fault tolerance when needed.

Iterative Active Invocation. Unlike traditional single-round retrieval approaches, MCP-Zero supports iterative tool discovery throughout task execution. After receiving retrieved tools, the model autonomously evaluates their adequacy for the current subtask. If the returned tools are insufficient or inappropriate, the model can refine its request specification and reinitiate retrieval, providing natural fault tolerance and self-correction capabilities. This iterative process continues until the model determines that either (1) suitable tools have been found and the task can proceed, or (2) no appropriate tools exist and the task should rely on the model's parametric knowledge.

The framework's key advantage lies in its ability to construct cross-domain toolchains dynamically. For complex tasks requiring coordination across multiple domains (e.g., filesystem access, code generation, and command execution in Figure 3), the model can progressively identify and request tools from different servers as subtask requirements become clear, avoiding the context overhead of pre-loading comprehensive tool collections while maintaining high task completion accuracy. MCP-Zero represents a fundamental shift from "predefined toolset" to "dynamic on-demand tool discovery" for the community.

3.2. Theoretical Analysis

We provide a theoretical analysis of MCP-Zero's advantages over traditional approaches through formal modeling that connects to established theories in active learning and information acquisition.

Problem Formulation. Let $T = \{t_1, t_2, ..., t_n\}$ denote the complete tool collection, q represent the user query, s_t the current conversation state, and t^* the optimal tool selection. Traditional approaches require simultaneous evaluation over the entire collection:

$$P_{\text{passive}}(t^*|q,T) = \frac{P(q|t^*,T)P(t^*|T)}{\sum_{t_i \in T} P(q|t_i,T)P(t_i|T)}$$
 (2)

MCP-Zero employs active information acquisition where agents generate requests r based on their current state

and capability assessment:

$$P_{\text{active}}(t^*|s_t) = \sum_r P(t^*|r)P(r|s_t)$$
 (3)

where $P(r|s_t)$ represents the agent's ability to articulate its needs given current understanding.

Active Information Acquisition. Tool request generation can be modeled as an active learning process where agents select actions to maximize information gain about task completion:

$$r^* = \arg\max_{r} I(T^*; r|s_t)$$

= $\arg\max_{r} [H(T^*|s_t) - H(T^*|r, s_t)]$ (4)

where $I(T^*;r|s_t)$ represents the mutual information between the optimal tool set and the request. This formulation captures the essence of active discovery: agents actively reduce uncertainty about their tool requirements.

Scalability Analysis. Traditional methods face fundamental scalability barriers:

Search Space Complexity: Passive approaches must process all n tools with O(n) complexity. Active approaches first filter among m servers $(m \ll n)$, then match within filtered subsets, reducing complexity to O(m+k) where k is the average tools per selected server.

Attention Distribution: With finite cognitive resources, passive methods distribute attention as $\frac{1}{n}$ per tool, degraded by noise factor $\eta(n) \propto \log(n)$. Active methods concentrate attention on relevant subsets, maintaining effectiveness as $\frac{1}{k}$ where $k \ll n$.

Semantic Alignment Advantage. The core advantage of active requests lies in improved semantic consistency. Agent-generated requests r exhibit stronger alignment with tool documentation compared to raw user queries q:

Alignment
$$(r, t) = \cos(\mathbf{e}_r, \mathbf{e}_t) > \cos(\mathbf{e}_q, \mathbf{e}_t)$$
 (5)

where \mathbf{e}_r , \mathbf{e}_q , \mathbf{e}_t represent embeddings of request, query, and tool description respectively. This improvement stems from agents operating in the same semantic space as tool documentation.

Iterative Information Gain. Unlike single-shot retrieval, active discovery enables cumulative information acquisition over *k* iterations:

$$I_{\text{total}} = \sum_{i=1}^{k} I(T^*; r_i | s_{i-1}) - \lambda \cdot \text{Cost}(r_i)$$
 (6)

where λ represents the context overhead per request. This formulation captures the trade-off between information gain and computational efficiency that MCP-Zero optimizes.

In summary, the active paradigm provides measurable advantages:

Figure 4. The schema of the MCP-tools dataset.

- Complexity Reduction: From O(n) to O(m+k) where $m+k \ll n$.
- Semantic Consistency: Direct embedding alignment in tool description space.
- Information Efficiency: Targeted uncertainty reduction rather than exhaustive search.
- Adaptive Capability: State-dependent tool discovery that evolves with task understanding.

These theoretical foundations explain the empirical performance gains observed in our experiments: 98% token reduction with maintained accuracy reflects the efficiency of targeted information acquisition over exhaustive tool enumeration.

4. Dataset: MCP-Tools

To support our active tool discovery framework and enable comprehensive evaluation, we construct **MCP-tools**, the first retrieval-oriented dataset in the MCP domain. Unlike existing MCP evaluation frameworks such as MCP-Bench [13] that focus on server availability and latency testing, our dataset is specifically designed to facilitate semantic tool discovery and retrieval for large language models.

4.1. Dataset Construction

Our dataset is constructed with the following steps:

Data Collection. We systematically collected MCP server information from the official Model Context Protocol repository² (Tag: 2025.4.28, Commit: ad2d4e6), encompassing 396 servers across three categories: 20 reference implementations, 114 third-party official servers, and 262 community contributions. This collection represents the current state of the MCP ecosystem and its diversity of tool capabilities.

Quality Assurance and Filtering. We implemented rigorous filtering to ensure dataset completeness and semantic richness. Servers were retained only if they provided: (1)

²github.com/modelcontextprotocol/servers

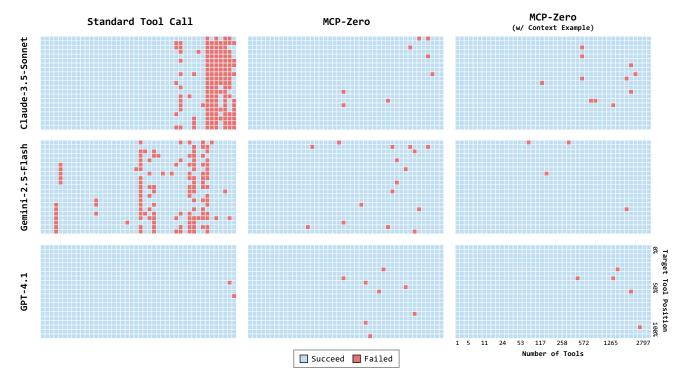


Figure 5. Needle-in-a-haystack test results demonstrating MCP-Zero's performance under extreme scale conditions with varying tool collection sizes. The left column shows baseline methods with standard tool call schemas; the middle column shows our MCP-Zero approach; the right column shows MCP-Zero enhanced with one ICL example to guide the model's description generation. Our method shows significant gains on Claude-3.5-Sonnet and Gemini-2.5-Flash, while GPT-4.1 shows no improvement due to its already strong baseline performance.

MCP-compliant tool definitions, (2) comprehensive documentation enabling semantic understanding, and (3) sufficient detail for meaningful retrieval evaluation. This process yielded 308 high-quality servers representing the core of the MCP ecosystem.

Structured Information Extraction. We employed Qwen2.5-72B-Instruct [24] with carefully designed fewshot examples to extract structured information following a standardized schema, which is demonstrated in Figure 4. The server_summary field represents a key innovation: LLM-generated comprehensive summaries that distill server capabilities while excluding operational details. This design enables effective semantic matching across diverse query formulations, from formal API specifications to natural language capability descriptions.

4.2. Dataset Characteristics

After the above process, we have completed the construction of the dataset as follows:

Scale and Diversity. The final dataset comprises 308 servers and 2,797 tools, providing comprehensive coverage of the MCP ecosystem. Tool distribution shows significant variance (mean: 9.08, median: 5.0, σ : 11.40), reflecting the

ecosystem's diversity from lightweight utilities to comprehensive API suites. Over half the servers (162) contain \leq 5 tools, while specialized servers include 60+ tools.

Semantic Infrastructure. To support efficient retrieval evaluation, we pre-compute embeddings for all textual content using OpenAI text-embedding-3-large, creating a searchable vector index that enables rapid semantic matching. These embeddings are included in the dataset release to ensure reproducible evaluation.

Research Impact. MCP-tools represents the first dataset specifically designed for semantic tool discovery in the MCP domain, complementing performance-focused frameworks like MCPBench [13]. The dataset's design supports not only our active discovery framework, but provides a general foundation for evaluating retrieval-based tool selection approaches across the research community.

5. Experiments

5.1. Existing Datasets and Their Limitations

Before presenting our experimental evaluation, we analyze existing tool-calling datasets and explain why they are insufficient for evaluating our MCP-Zero framework.

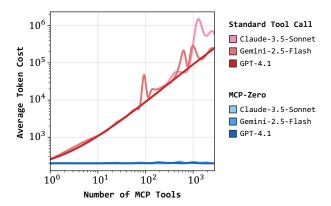


Figure 6. Token efficiency comparison in needle-in-a-haystack experiments. The graph shows the average token cost per successful retrieval across different collection sizes.

API-Bank [12] provides tool information with multiturn conversations and human-annotated test sets, making it the most suitable existing dataset for our evaluation. ToolBench [20] collected 16,464 REST APIs from RapidAPI Hub but many APIs lack essential descriptions. ToolBank [15] improved upon ToolBench but relies on modelgenerated data. ToolAlpaca [28] synthesized 3,938 instances from 400 APIs but targets scenarios mismatched with contemporary use cases. The APIBench1 from Gorilla [18] contains 1,645 APIs but uses entirely GPT-synthesized conversations. Another APIBench2 [19] is proposed, but its application scenarios have low relevance to MCP tool calls.

Existing datasets have limitations for evaluating MCP-Zero: API scenarios differ from contemporary MCP use cases, lack hierarchical server-tool organization, or miss critical evaluation fields. Therefore, we use API-Bank as a reference and create the MCP-tools dataset for comprehensive evaluation. We are continuing to investigate other available datasets and conducting further validation.

5.2. Needle-in-a-Haystack Evaluation

To evaluate MCP-Zero's ability to accurately retrieve tools from large-scale collections under extreme context conditions, we conduct needle-in-a-haystack experiments on our MCP-tools dataset.

Experimental Setup. We construct test scenarios by injecting 1 to 2,797 tools into the environment, selecting task descriptions from various positions as queries, and requiring models to retrieve the target tool. This setup simulates the challenging scenario where relevant tools are buried within massive tool collections. We compare three approaches:

• Baseline: Standard tool call schemas with all tools injected into context

Collection	Method	Claude-3.5	GPT-4.1	Gemini-2.5	Avg. Tokens↓
Single-turn Conversation					
	Q.Retrieval		71.63		_
Domain	Standard	97.60	98.08	92.79	312.4
	MCP-Zero	96.15	96.62	97.12	111.0 (-64.47%)
	Q.Retrieval		71.63		_
Full	Standard	69.23	94.71	94.23	6308.2
	MCP-Zero	95.19	95.19	96.63	111.0 (-98.24%)
Multi-turn Conversation					
	Q.Retrieval		65.05		_
Domain	Standard	100.00	99.46	91.40	406.4
	MCP-Zero	91.40	93.01	93.01	159.0 (-60.84%)
Full	Q.Retrieval		65.05		-
	Standard	60.22	93.01	92.47	6402.2
	MCP-Zero	90.32	92.47	94.62	159.0 (-97.52%)

Table 1. APIBank evaluation results comparing MCP-Zero with standard tool calling methods across different scenarios. The accuracy is based on the top-1 result. Results show accuracy (%) for three LLMs and average token consumption. "Q.Retriaval" indicates retrieval based on user query.

- MCP-Zero: Our active retrieval approach
- MCP-Zero + ICL: MCP-Zero enhanced with one incontext learning example to guide description generation

Results Analysis. As shown in Figure 5, our method demonstrates significant performance gains on Claude-3.5-Sonnet and Gemini-2.5-Flash, while GPT-4.1 shows no improvement due to its already strong baseline performance across all tool collection sizes. Figure 6 illustrates the dramatic difference in token consumption between traditional approaches and MCP-Zero. While standard tool call methods exhibit exponential growth in token costs as the number of MCP tools increases, MCP-Zero maintains consistently low token usage.

5.3. APIBank Evaluation

To validate MCP-Zero's effectiveness in realistic conversational tool retrieval scenarios, we conduct comprehensive experiments on the APIBank dataset.

Experimental Setup. We extract description information from the APIBank level-1 dataset for retrieval tasks, get 48 unique tools in total, and process it for our evaluation framework. Since APIBank organizes data by individual APIs without server-level hierarchy, we directly retrieve tools without the server filtering stage.

We evaluate across two key dimensions: conversation context and tool collection scope. For conversation context, we test both single-turn scenarios (one user query for one response) and multi-turn scenarios (extended conversations with multiple exchanges). For tool collection scope, we examine both domain collections (using a curated subset of tools relevant to the specific domain) and full collections (retrieving from the complete set of all available tools).

Results Analysis. As shown in Table 1, we can conclude the following findings:

- Extreme Context Efficiency. MCP-Zero cuts prompt length by 60–98% across all settings (e.g. 111 vs. 6.3k tokens in the full single-turn case), validating its ability to "pay for tools only when they are needed".
- **Robust Scalability.** When moving from a hand-curated *Domain* subset to the *Full* tool pool (40x more APIs), standard schema-injection accuracy on Claude-3.5 plummets from 97.60 to 69.23 (single-turn) and 100.00 to 60.22 (multi-turn); MCP-Zero instead keeps accuracy at 95.19 / 90.32 respectively, demonstrating strong resilience to attention dilution.
- Multi-turn Consistency. MCP-Zero maintains high accuracy over conversation rounds (≤3% drop from single-to multi-turn), whereas standard methods degrade sharply once the context accumulates previous calls and larger tool sets.
- Necessity of Active Requests. Pure query-retrieval baselines stall at 65–72 % accuracy, confirming that letting the model *author* semantically aligned requests is crucial.

Experiments on APIBank corroborate our claims: MCP-Zero delivers near-optimal or superior tool-selection accuracy while slashing context usage by up to two orders of magnitude, remaining robust in both single- and multi-turn conversations and under massive tool-pool scaling. These results highlight active, iterative tool discovery as a practical path toward scalable, cost-efficient agent systems.

6. Conclusion

This work establishes active tool discovery as a fundamental paradigm for autonomous agent systems, enabling models to maintain decision autonomy while addressing critical scalability challenges in tool-calling architectures. MCP-Zero demonstrates that shifting from passive tool consumption to agent-driven capability acquisition achieves substantial efficiency gains—98% token reduction with preserved accuracy—while restoring the core principle of autonomous agency: the ability to assess limitations and actively acquire necessary resources. Our theoretical framework, empirical validation, and the MCP-tools dataset provide both the foundation and infrastructure for advancing autonomous agent architectures as tool ecosystems continue expanding exponentially.

7. Discussion

In this section we reflect on how the MCP-Zero paradigm can be adopted by other researchers (§7.1), analyse the surprisingly gain from a single in–context example (§7.2), and position MCP-Zero with respect to the contemporaneous *Alita* system, outlining a promising path toward self-improving agentic AI (§7.3).

7.1. Cookbook: Integrate MCP-Zero Into Agent

MCP-Zero is fundamentally a simple yet effective approach that we hope will benefit the broader MCP community. The core methodology distills into three straightforward steps: prompting models to actively request tools, maintaining a lightweight tool index with semantic descriptions, and leveraging the improved semantic alignment for high-precision retrieval. Below we provide a practical guide for integrating these ideas into existing agent frameworks.

Step 1 – Prompting the LLM to *ask* **for tools.** Give the model an explicit "permission" to declare missing capabilities. In practice this is a system instruction such as:

```
If the current task cannot be solved with your own knowledge, emit a <tool_assistant> block specifying the server domain and the tool operation you require.
```

In addition, the output structure needs to be specified as we mentioned in Section 3.1. This step aims to stimulate the model's ability to "actively" propose requirements.

Step 2 - Curate a lightweight MCP-style tool index.

Firstly, choose a scope based on your needs: the entire MCP-tools collection, a vertical slice (e.g. databases only), or your in-house APIs. Then, for every server/tool:

- extract the name and description from metadata;
- optionally let a strong LLM generate an *enhanced sum-mary* that emphasises capabilities and usage patterns;
- store all texts in a vector store with pre-computed embeddings such as text-embedding-3-large.

Step 3 – Marry model output and retrieval.

When the agent emits a <tool_assistant> block:

- Match the server field against server descriptions and summaries; take top-m candidates.
- Within each candidate server, rank tools by the tool field with the tool description embeddings.
- Feed the best (or top-k) JSON-schemas back to the LLM.

Because the request text is already semantically aligned with the documents, retrieval precision is higher than "user query \rightarrow API doc" matching, maintaining performances while significantly conserving context.

7.2. Why Does a Single ICL Example Help?

In §5.2 we observed that adding **one** in-context example ("ICL-1") helps lifting needle-in-haystack accuracy marginally. We hypothesise two simple but potent effects:

 Stylistic anchor. Our base prompt merely says "output the server and tool you need", but gives no example of how the sentence should look like. The single in-context sample provides the writing style as the reference, helping the generated requests land much closer to the curated descriptions, thus semantic matching becomes easier. 2. **Semantic grounding.** The example also clarifies the *meaning* of each field, helping the model understand the specific definitions of MCP server and tool, thereby limiting its expression scope. After seeing this, the model reliably emits phrases such as filesystem_read instead of a vague "read the file", sharply reducing semantic mismatch.

In short, a tiny demonstration patch acts as a *schema anchor*; future work could replace ICL with a short grammar-based decoder rule, but the one-shot approach is free and highly effective.

7.3. Synergy with Alita: Using and Making Tools

Concurrently, **Alita** [22] proposes a united manager agent that *creates* its own toolchain: it web-searches for code, clones GitHub repos, builds environments, and executes the resulting programs to accomplish tasks. We were pleasantly surprised by the contribution of this article, and found that the two lines of work are complementary:

- MCP-Zero: efficiently finds and invokes existing tools
- Alita: automatically builds missing tools on-the-fly

MCP-Zero and Alita address complementary halves of the same problem: the former maximises *tool discovery* while the latter maximises *tool creation*. When combined, they form a virtuous loop: an agent first actively discover tools from *all* available resources; if none fits, it switches to Alita's workflow to synthesize a new one, then registers the freshly built tool for the community. We believe such a pipeline is a compelling direction toward self-evolving, cost-aware agentic AI systems.

7.4. Future Work

While MCP-Zero demonstrates significant improvements in tool retrieval efficiency and accuracy, several promising directions warrant further investigation:

Enhanced Experimental Validation. Future work should expand evaluation across diverse domains. We plan to conduct comprehensive experiments on additional datasets to validate generalizability.

Advanced Matching Algorithms. The current semantic similarity approach could be enhanced. We envision incorporating multi-modal descriptions (e.g., code examples, usage patterns, parameter schemas) into the retrieval process, and exploring usage co-occurrence patterns for improved contextual understanding.

MCP Server Implementation. A natural extension involves packaging MCP-Zero as a dedicated MCP server providing tool discovery services. This "meta-server" would expose standardized APIs for active tool retrieval, enabling seamless integration into existing MCP ecosystems and serving as a centralized discovery hub for distributed tool collections.

Multi-Agent Orchestration. MCP-Zero's active discovery approach could enable better multi-agent collaboration. Future work could investigate how different agents can automatically discover and share tools with each other, allowing them to work together more effectively on complex tasks that require diverse capabilities.

References

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023. 2
- [2] Anthropic. Model context protocol. https://docs.anthropic.com/en/docs/agents-and-tools/mcp, 2024. Accessed: June 25, 2025. 4
- [3] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR, 2022. 3
- [4] Harrison Chase. Langchain. https://github.com/langchain-ai/langchain, 2022. Python framework for developing applications powered by language models. 3
- [5] Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammadhossein Bateni, Chen-Yu Lee, and Tomas Pfister. Re-invoke: Tool invocation rewriting for zero-shot tool retrieval. arXiv preprint arXiv:2408.01875, 2024. 3
- [6] Yu Du, Fangyun Wei, and Hongyang Zhang. Anytool: Self-reflective, hierarchical agents for large-scale api calls. arXiv preprint arXiv:2402.04253, 2024. 3
- [7] Tiantian Gan and Qiyao Sun. Rag-mcp: Mitigating prompt bloat in llm tool selection via retrieval-augmented generation. *arXiv preprint arXiv:2505.03275*, 2025. 2, 3, 4
- [8] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024. 2
- [9] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pretraining. In *International conference on machine learning*, pages 3929–3938. PMLR, 2020. 3
- [10] Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, et al. Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. arXiv preprint arXiv:2205.00445, 2022. 3
- [11] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich

- Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020. 3
- [12] Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. arXiv preprint arXiv:2304.08244, 2023. 8
- [13] Zhiling Luo, Xiaorong Shi, Xuanrui Lin, and Jinyang Gao. Evaluation report on mcp servers. *arXiv preprint arXiv:2504.11094*, 2025. 3, 6, 7
- [14] Tula Masterman, Sandi Besen, Mason Sawtell, and Alex Chao. The landscape of emerging ai agent architectures for reasoning, planning, and tool calling: A survey. arXiv preprint arXiv:2404.11584, 2024. 2
- [15] Suhong Moon, Siddharth Jha, Lutfi Eren Erdogan, Sehoon Kim, Woosang Lim, Kurt Keutzer, and Amir Gholami. Efficient and scalable estimation of tool representations in vector space. arXiv preprint arXiv:2409.02141, 2024. 2, 3, 8
- [16] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. arXiv preprint arXiv:2112.09332, 2021. 3
- [17] Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models. arXiv preprint arXiv:2303.09014, 2023. 3
- [18] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565, 2024. 2, 3, 8
- [19] Yun Peng, Shuqing Li, Wenwei Gu, Yichen Li, Wenxuan Wang, Cuiyun Gao, and Michael R Lyu. Revisiting, benchmarking and exploring api recommendation: How far are we? *IEEE Transactions on Software Engineering*, 49(4): 1876–1897, 2022. 8
- [20] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. arXiv preprint arXiv:2307.16789, 2023. 8
- [21] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. Tool learning with foundation models. ACM Computing Surveys, 57(4):1–40, 2024. 2
- [22] Jiahao Qiu, Xuan Qi, Tongcheng Zhang, Xinzhe Juan, Jiacheng Guo, Yifu Lu, Yimin Wang, Zixin Yao, Qihan Ren, Xun Jiang, et al. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution. *arXiv* preprint arXiv:2505.20286, 2025. 10
- [23] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Colt: Towards completeness-oriented tool retrieval for large language models. arXiv e-prints, pages arXiv-2405, 2024. 3
- [24] 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, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.
- [25] Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. Incontext retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331, 2023. 3
- [26] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances* in Neural Information Processing Systems, 36:68539–68551, 2023. 2, 3
- [27] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. Advances in Neural Information Processing Systems, 36:38154–38180, 2023. 3
- [28] Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. arXiv preprint arXiv:2306.05301, 2023. 8
- [29] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023. 3
- [30] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv* preprint arXiv:2505.09388, 2025. 2
- [31] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Con*ference on Learning Representations (ICLR), 2023. 3
- [32] Yuanhang Zheng, Peng Li, Wei Liu, Yang Liu, Jian Luan, and Bin Wang. Toolrerank: Adaptive and hierarchy-aware reranking for tool retrieval. *arXiv preprint arXiv:2403.06551*, 2024. 3