Evaluation Report on MCP Servers

Zhiling Luo[*], Xiaorong Shi, Xuanrui Lin, Jinyang Gao

2025-04-13

Executive Summary

1. There are significant differences in effectiveness and efficiency among MCP servers; using MCPs does not demonstrate a noticeable improvement compared to function call.
2. The effectiveness of the MCP server can be enhanced by optimizing the parameters that need to be constructed by the LLM.

1 Introduction

Model Context Protocol (MCP) [1] is an open protocol that enables AI models to securely interact with local and remote resources through standardized server implementations. Thousands of MCPs have been proposed in recent months. At the same time, several model platforms, e.g. OpenAI and Alibaba-cloud announced the support of MCP in their LLM products. The outbreak of the MCP protocol has become a reality. To study the effectiveness and efficiency of MCP servers, we selected several widely used MCP servers and conducted an experiment to evaluate them using MCPBench on their accuracy, time, and token usage. We focused on two tasks: web search and database search. The former involves searching the internet to answer questions, while the latter entails fetching data from a database. All MCP servers were compared using the same LLM and prompt in a controlled environment. We aimed to answer the following questions:

- Question 1: Are MCP servers effective and efficient in practice?
- Question 2: Does using MCP provide higher accuracy compared to function calls?
- Question 3: How to enhance the performance?

To study these questions, we propose an evaluation framework, called MCPBench, which is released at `https://github.com/modelscope/MCPBench`. Besides, we provide the dataset of web search and database search at the same time.

2 Tasks and Dataset

MCPs encompass several tasks, including computing, memory management, web searching, and database interaction. They can be categorized into two groups:

- **Data Fetching**: This group retrieves data from various sources to assist LLMs in completing tasks. The effectiveness of these MCPs is determined by the accuracy of the data fetched. For example, web search MCPs utilize search engines to obtain content from websites.
- **World Changing**: These MCPs alter the state of the world. Their effectiveness hinges on whether the state has been successfully modified. For instance, GitHub MCPs can commit code that changes the data in the GitHub dataset.

Evaluating world-changing MCPs presents challenges due to the difficulty in accessing the underlying status of the data sources (e.g., GitHub dataset). Therefore, our focus will be on data fetching tasks. Specifically, the evaluation report will encompass two tasks:

[*]godot.lzl@alibaba-inc.com

Table 1: An example of web search task

Input	Tool	Result
What is the middle name of Barack Obama	brave_web_search {"query":"middle name of Barack Obama"}	Hussein

Table 3: An example of database search task

Input	Tool	Result
Fetch the sales of Tesla Model S since 2025-01	execute_sql {"query":"Select sum(sales) from sales where series='Tesla Model S' and datetime¿ 2025-01"}	13402

1) Web Search

This task takes a question as input. The LLM rewrites it into keywords or some short sentences, involving a tool that typically searches the internet and returns results to the LLM. Here is an example involving the Brave Search MCP, see Tab. 1.

To eliminate biases in the dataset, we introduce multiple data sources encompassing both Chinese and English languages across various fields.

Table 2: The datasource of web search task

Datasource	Details	Volume	Case
Frames [7]	Open-source dataset	100	"Prompt": "As of August 1, 2024, which country hosted the FIFA World Cup the last time the UEFA Champions League was won by a club from London?", "Answer": "France"
News (Chinese)	Collected by cleaning data from daily Xinwen Lianbo transcripts over the past three months and processing it using reverse engineering techniques.	100	"Prompt": "In which city did Tesla's first energy storage super factory outside of the United States officially start production on February 11, 2025?", "Answer": "Shanghai"
Knowledge (Chinese)	Collected by cleaning data from knowledge-intensive websites like Wikipedia and science and technology reports, and processing it using reverse engineering techniques.	100	"Prompt": "What type of fish might the family Cyprinidae and the family Dace represent as their primitive types?", "Answer": "armored fish"

2) Database Search

Database search, or database interaction, is the data retrieval task with database. This task takes a question as input. The LLM retrieves data from the database through a database MCP server. Table 3 shows an example involving the MySQL MCP. We gathered datasets from various sources, see Tab 4.

Table 4: The datasource of database search task

Datasource	Details	Volume	Case
Car_bi	A synthetic dataset from an automobile manufacturer datasource	355	"Prompt": "What is the total number of orders in the South China region in February 2025?", "Answer": "0"
SQL_EVAL	Sampled from SQL_EVAL[1]. It is based off the schema from the Spider, but with a new set of hand-selected questions and queries grouped by query category.	256	"Prompt": Which authors have written publications in both the domain "Machine Learning" and the domain "Data Science"? "Answer": "Ashish Vaswani"

[1]https://github.com/defog-ai/sql-eval

3 Overview of MCP Servers

We collected the MCP servers from GitHub[2] and Smithary.AI[3]. Due to limitations of time and cost, we selected those having more call records at April 2025.

3.1 Web Search Related MCP

- **Brave Search [4]:** A web and local search utilizing Brave's Search API.
  Source: `https://github.com/modelcontextprotocol/servers/tree/main/src/brave-search`
  Tool Name: brave_web_search
  Developer: erdnax123
- **DuckDuckGo Search Server [11]:** A Model Context Protocol (MCP) server providing web search capabilities through DuckDuckGo, with additional features for content fetching and parsing.
  Source: `https://github.com/nickclyde/duckduckgo-mcp-server`
  Tool Name: search
  Developer: nickclyde
- **Tavily MCP Server [12]:** A search engine for AI agents (search + extract) powered by Tavily.
  Source: `https://github.com/tavily-ai/tavily-mcp`
  Tool Name: tavily-search
  Developer: tavily-ai
- **Exa Search [5]:** A search engine designed for AI by Exa.
  Source: `https://github.com/exa-labs/exa-mcp-server`
  Tool Name: web_search
  Developer: exa-labs
- **Fire Crawl Search [9]:** Extracts web data using Firecrawl.
  Source: `https://github.com/mendableai/firecrawl-mcp-server`
  Tool Name: firecrawl_search
  Developer: mendableai
- **Bing Web Search [8]:** A Model Context Protocol (MCP) server for Microsoft Bing Search API integration, enabling AI assistants to conduct web, news, and image searches.
  Source: `https://github.com/leehanchung/bing-search-mcp`
  Tool Name: bing_web_search
  Developer: leehanchung
- **BochaAI:** A search engine for AI that provides access to high-quality global knowledge from nearly ten billion web pages and ecological content sources across various fields, including weather, news, encyclopedias, healthcare, and travel.
  Source: Alibaba Cloud BaiLian Platform
  Tool Name: bocha_web_search
  Developer: Alibaba Cloud

3.2 Web Search Related Function Calls

For comparison, we included some function calls:

- **Qwen Web Search:** Uses the SDK provided by Qwen-Max-0125 [13] to enable online search with `extra_body={"enable_search": True}`.
- **Quark Search:** A search engine that is particularly useful for searching unknown information such as weather, exchange rates, and current events.
  Source: Official Quark Search Plugin provided by Alibaba Cloud BaiLian Platform
  Tool Name: quark_search
  Developer: Alibaba Cloud

[2]http://github.com
[3]https://smithery.ai

3.3 Database Search Related MCP

- **XiYan MCP Server [14]**: An MCP server that supports data retrieval from a database using natural language queries, powered by XiyanSQL [6] as the text-to-SQL LLM.
  Source: `https://github.com/XGenerationLab/xiyan_mcp_server`
  Tool Name: get_data
  Developer: XGenerationLab

- **MySQL MCP Server [3]**: An implementation that facilitates secure interaction with MySQL databases.
  Source: `https://github.com/designcomputer/mysql_mcp_server`
  Tool Name: execute_sql
  Developer: designcomputer

- **PostgreSQL MCP Server [10]**: A Model Context Protocol server that provides read-only access to PostgreSQL databases.
  Source: `https://github.com/modelcontextprotocol/servers/tree/main/src/postgres`
  Tool Name: query
  Developer: modelcontextprotocol

4 Criteria for Evaluation

4.1 Accuracy

Accuracy is evaluated to determine the correctness of the answer. It is assessed by an LLM-based grader. We use DeepSeek-v3 [2] as the grader. The prompt is:

> For the following question: {question}
> Please judge whether the predicted answer is correct. It is considered correct if it addresses the key information:
> Predicted Answer: {prediction}
> Correct Answer: {ground_truth}
> Just return True or False.

The accuracy of each sample is 1 if the grader replies True, and 0 otherwise. The overall accuracy is the average of all samples. Due to the instability of the network environment, we consider another criterion in practice: accuracy of valid samples. Valid samples are those returned within a limited time, while invalid samples are those that timed out.

4.2 Time Consumption

We collect the end-to-end time consumption, which includes the latency of both the LLM and the MCP server. This criteria reflects the efficiency of MCP.

4.3 Token Consumption

We record the pre-fill (prompt) and completion tokens used during the experiment. The token consumption will affect the incurred costs.

4.4 Other Setups

The experiment is executed on a server in Singapore with a dual-core CPU and 2GB RAM. The evaluation framework used is MCPBench. All MCP servers (except DuckDuckGo) are launched on the server in SSE mode. The timeout is set to 30 seconds. The system prompt is as follows: We collect the query from `<WebSearch></WebSearch>` and send it to the MCP.

> handles questions that may require web searching.
>
> - Input contains:
>   - The question that needs to be answered.

  - Past search steps and their results.
- Output can be either:
  - If more search is needed: output in the format `<WebSearch>search_query</WebSearch>`
  - If the question can be answered: direct answer.

Example: Input question: ”Who is the current President of the United States?”
- If no search has been conducted: `<WebSearch>current President of United States</WebSearch>`
- If sufficient information is available: ”Joe Biden is the current President of the United States.”

For the database search, we use MySQL database whose version is 9.2, the PostgreSQL version is 15.8.

5 Comparative Analysis

5.1 Question 1: Are MCP Servers Effective and Efficient in Practice?

Table 5: The experiment results on MCPs

MCP Server	Accuracy (%) ↑	Time Consumption (s) ↓	Pre-fill Token Consumption ↓	Completion Token Consumption ↓
Brave Search [4]	46.6	13.98	5802.35	236.26
DuckDuckGo Search Server [11]	13.62	64.17	1718.84	162.25
Tavily MCP Server [12]	47.99	95.52	2441.73	196.03
Exa Search [5]	15.02	231.24	2475.24	190.49
BoChaAI Search	20	35.54	1642.71	189.13
Fire Crawl Search [9]	58.33	15.44	1727.17	179.61
Bing Web Search [8]	64.33	12.4	4060.34	206.87

The experiment results are reported in Tab. 5, in which we can have following observations.

1. The differences in effectiveness are significant. Based on the accuracy of valid samples, the highest accuracy is observed with Bing Web Search (64%), while DuckDuckGo has the lowest at just 10%, representing a difference of 54 percentage points.
2. The differences in efficiency are even more pronounced; regarding the average time consumed for valid samples, the fastest are Bing Web Search and Brave Search, which require less than 15 seconds, while the slowest, Exa Search, takes 231 seconds (note that valid samples are cases of normal returns without timeouts, so this value is unaffected by timeouts).
3. Token consumption is similar; based on the number of output tokens for valid samples, consumption generally falls between 150 and 250 tokens, indicating that the model consistently provides concise answers without unnecessary elaboration on its MCP usage.

5.2 Question 2: Does MCP Provide Higher Accuracy Compared to Function Calls?

We compared the performance of function call with the above-mentioned MCP servers. The results are shown in Figure 1 and Tab 6. We observe that both function calls (Qwen Web Search) and tool usage (Quark Search) exhibit competitive accuracy and time consumption. The accuracy of Qwen Web Search is 55.52%, surpassing that of Exa Search, DuckDuckGo, Tavily, and Brave Search. There is not much difference of time consumption of function calls (Qwen Web Search and Quark Search) compared to MCP services.

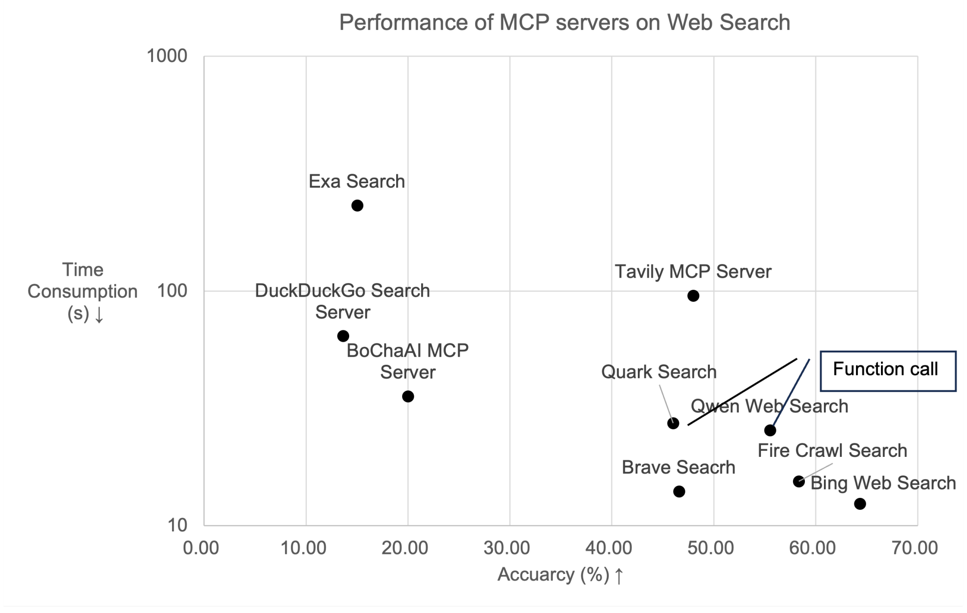

Figure 1: The performance of MCP servers in Web search

Table 6: The experiment results of MCP and Function Call

	Accuracy (%) ↑	**Time Consumption (s) ↓**	**Pre-fill Token Consumption ↓**	**Completion Token Consumption ↓**
Quark Search (Function Call)	46.00	27.31	1142.21	158.68
Qwen Web Search (Function Call)	55.52	25.48	1149.69	183.98
Brave Search (MCP)	46.6	13.98	5802.35	236.26

5.3 Question 3: How to enhance the performance?

Table 7: The performance of declarative interface on MySQL MCP

MCP Server	**Accuracy (%)↑**	**Time Consumption (s) ↓**	**Pre-fill Token Consumption ↓**	**Completion Token Consumption ↓**
MySQL MCP Server [3]	54.73	4.64	2800	64.26
MySQL MCP Server + Declarative Interface (XiYan MCP Server) [14]	56.06	6.38	415.52	44.46

Table 8: The performance of declarative interface on PostgreSQL MCP

MCP Server	**Accuracy (%) ↑**	**Time Consumption (s) ↓**	**Pre-fill Token Consumption ↓**	**Completion Token Consumption ↓**
PostgreSQL MCP Server [10]	58.5	5.85	6896.51	103.22
PostgreSQL MCP Server + Declarative Interface				

(XiYan MCP Server) [14]	80.08	12.87	434.86	97.57

To address this question, we considered the database search task. The MySQL MCP server implements a straightforward encapsulation of the database connection. After configuring the database account, password, and other information, these MCPs establish a persistent connection to the database and expose the execute_sql tool interface. When calling this tool, the model must construct a query parameter that must be an executable SQL statement. Although this simple encapsulation functions as an MCP, it assigns the most challenging part of the process—constructing the SQL query statement—to the LLM. Consequently, the success of the entire tool call heavily relies on the LLM's ability to construct SQL statements.

We introduce a Declarative Interface method to alleviate this challenge. The key idea is replace the structural parameter of MCP by a declarative interface. In other words, we use the natural language as the interface in MCP. In our experiment, we developed an updated version, called XiYan MCP server, using natural language instead of SQL. It utilizes XiYanSQL-QwenCoder-32B[4] [6] to convert natural language queries into SQL before executing comparable to the MySQL MCP server. The experiment results are shown in Table 7. By adding a text-to-SQL model to the MCP server, it improves accuracy by 2 percentage points. In the PostgreSQL experiment, the optimization results is a 22-point increase 8.

Conclusion

The evaluation of various Model Context Protocol (MCP) servers highlights significant differences in both effectiveness and efficiency. While MCPs offer distinct advantages in structuring tool usage, they do not consistently demonstrate marked improvements over non-MCP approaches, such as function calls. Our experiments showed that the most effective MCP, Bing Web Search, achieved an accuracy of 64%, whereas DuckDuckGo lagged at just 10%. Furthermore, performance varied widely in terms of time consumption, with top performers like Bing and Brave Search executing tasks in under 15 seconds, contrasted with significantly slower alternatives like Exa Search.

Importantly, we found that the accuracy of MCP servers can be substantially enhanced by optimizing the parameters that LLMs must construct. For instance, transitioning from SQL-based queries to natural language processing in the XiYan MCP server resulted in a noteworthy increase in accuracy, demonstrating that incorporating a text-to-SQL model can lead to a 22 percentage point improvement.

Overall, while MCPs provide a structured means for AI tools to interact with data, there remains considerable potential for optimization. By addressing the challenges LLMs encounter in parameter construction and enhancing the user-friendliness of tool interfaces, developers can significantly improve the performance and reliability of MCP servers. This research paves the way for further investigations into optimized MCP implementations, ultimately leading to better AI-driven search and data retrieval solutions.

References

[1] Model context protocol. https://modelcontextprotocol.io.

[2] DeepSeek-AI. Deepseek-v3 technical report, 2024.

[3] designcomputer. mysql-mcp-server. https://github.com/designcomputer/mysql_mcp_server.

[4] erdnax123. brave-search. https://github.com/modelcontextprotocol/servers/tree/main/src/brave-search.

[5] exa labs. exa-mcp-server. https://github.com/exa-labs/exa-mcp-server.

[6] Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, Jinyang Gao, Liyu Mou, and Yu Li. A preview of xiyan-sql: A multi-generator ensemble framework for text-to-sql, 2025.

[7] Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananey, Steven Schwarcz, Adam Stambler, Shyam Upadhyay, and Manaal Faruqui. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation, 2025.

[4]https://www.modelscope.cn/models/XGenerationLab/XiYanSQL-QwenCoder-32B-2412

[8] leehanchung. bing-search-mcp. https://github.com/leehanchung/bing-search-mcp.

[9] mendableai. firecrawl-mcp-server. https://github.com/mendableai/firecrawl-mcp-server.

[10] modelcontextprotocol. postgres. https://github.com/modelcontextprotocol/servers/tree/main/src/postgres.

[11] nickclyde. duckduckgo-mcp-server. https://github.com/nickclyde/duckduckgo-mcp-server.

[12] tavily ai. tavily-mcp. https://github.com/tavily-ai/tavily-mcp.

[13] Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[14] XGenerationLab. xiyan-mcp-server. https://github.com/XGenerationLab/xiyan_mcp_server.