A Large-Scale Evolvable Dataset for Model Context Protocol Ecosystem and Security Analysis

Zhiwei Lin Bonan Ruan Jiahao Liu* Weibo Zhao
National University of Singapore
{zhiweil, r-bonan, jiahao99, weibo}@comp.nus.edu.sg

***Abstract*—The Model Context Protocol (MCP) has recently emerged as a standardized interface for connecting language models with external tools and data. As the ecosystem rapidly expands, the lack of a structured, comprehensive view of existing MCP artifacts presents challenges for research. To bridge this gap, we introduce MCPCorpus, a large-scale dataset containing around 14K MCP servers and 300 MCP clients. Each artifact is annotated with 20+ normalized attributes capturing its identity, interface configuration, GitHub activity, and metadata. MCPCorpus provides a reproducible snapshot of the real-world MCP ecosystem, enabling studies of adoption trends, ecosystem health, and implementation diversity. To keep pace with the rapid evolution of the MCP ecosystem, we provide utility tools for automated data synchronization, normalization, and inspection. Furthermore, to support efficient exploration and exploitation, we release a lightweight web-based search interface. MCPCorpus is publicly available at: https://github.com/Snakinya/MCPCorpus. The video is at https://youtu.be/2a9WrHMcfxU.**

I. Introduction

Given their strong capabilities in language understanding, reasoning, and decision-making, large language models (LLMs) have been increasingly integrated with diverse tools and services to collaboratively perform a wide range of tasks [1], [2].

However, due to the heterogeneity of tool interfaces and the lack of standardized integration protocols, swiftly integrating them remains challenging, which significantly hinders the development of LLM-based applications [3]. To address this challenge, Model Context Protocol (MCP) has been introduced as a lightweight, standardized interface that connects LLMs with diverse tools [4]. Specifically, MCP defines tools as callable endpoints with structured JSON schemas, allowing developers to expose services in a consistent and interoperable manner across different programming environments and platforms. Additionally, MCP employs a modular client-server architecture, where the MCP server registers tool capabilities and executes them upon request, while the MCP client discovers these capabilities and invokes the desired tools through a standardized protocol [5].

Although standardization has driven the rapid growth of MCP, fostering an active and expansive ecosystem, it has also exposed a range of issues, including inconsistent implementations, malicious server behaviors, and weak protocol compliance [6], [7]. This phenomenon has shifted the focus of research from merely exploring new applications of MCP to examining its reliability, security, and ecosystem dynamics [8], [9]. However, we identify several challenges in existing studies: (a) *Limited-source investigation*. Most works focus on MCP artifact information from a single source, ignoring the rich metadata available across different platforms such as code hosting sites (e.g., GitHub), community hubs (e.g., `MCP.so` [10]), and package managers. (b) *Unscalable analysis*. These studies often rely on hand-picked examples, lacking the scalability needed for a comprehensive assessment of the MCP ecosystem's current state. The underlying reason is the absence of a large-scale, well-archived MCP dataset that consolidates MCP servers from diverse sources and unifies them for ease of use.

To bridge this gap and better support future research on the MCP ecosystem, we present MCPCorpus, a large-scale dataset accompanied by utility scripts for maintaining and continuously updating the dataset. MCPCorpus provides a comprehensive view of real-world MCP artifacts, currently including around 14K MCP servers and 300 MCP clients. For a clear overview and ease of analysis, each MCP artifact in MCPCorpus is annotated with over 20 metadata attributes, including *type*, *tool list*, *programming language*, *license*, *author name*, *server configuration*, and *GitHub activity indicators*. Take *tool list* as an example: it allows users to quickly understand the capabilities provided by an MCP server and how to interact with it, helping them efficiently determine whether the server is relevant for a given investigation. It is important to note that these attributes are extracted through static inspection of public repositories, without requiring runtime access or live service interaction.

With this dataset, we envision a range of potential applications, including (1) interoperability benchmarking of tool-augmented LLM agents, (2) security auditing of MCP implementations via linked source code analysis, and (3) schema conformance checking across diverse MCP artifacts. For example, researchers can locate implementations through metadata and inspect code to identify issues like missing authentication or unsafe endpoint exposure.

In summary, we make the following contributions:

- We curate a large-scale dataset named MCPCorpus, covering around 14K MCP servers and 300 clients from multiple sources. These artifacts span diverse domains and programming languages, and are annotated with 20+ normalized attributes to support ecosystem-level analysis.

*Corresponding author

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00356

- We develop a set of utility tools to support data synchronization, normalization, and inspection, as well as a public-facing web interface to explore the dataset.

II. Dataset Construction

To ensure that MCPCorpus can comprehensively profile MCP-related artifacts and support various research and engineering tasks, we conduct a systematic investigation of the current MCP ecosystem and integrate information from multiple online sources. In particular, we focus on metadata-rich, actively maintained platforms that cover a wide range of MCP servers and clients.

In the subsequent sections, we first introduce the data sources selected for MCPCorpus and then elaborate on the data collection process, including our crawling strategy, metadata normalization, and attribute synthesis.

A. Data Sources

To construct a large-scale and high-quality dataset of MCP implementations, we combine two complementary sources: MCP.so, which hosts the largest and most information-rich collection of MCP tools, and GitHub, which provides additional implementation and maintenance metadata.

MCP.so is currently the most extensive registry dedicated to MCP servers and clients. As of 3 June 2025, `MCP.so` hosts over 14,000 MCP servers and 300 client implementations, and continues to update its database daily. Compared to other MCP hosting platforms, `MCP.so` offers the most comprehensive artifact-level information, including textual descriptions, domain tags, author identities, publication dates, and deployment statuses. To enable reliable analysis of the MCP ecosystem, we developed a unified metadata schema and systematically extracted and normalized over ten key attributes from `MCP.so`'s unstructured content.

GitHub complements `MCP.so` by providing rich technical metadata for each associated repository. We extract statistics such as star count, forks, open issues, and contributors, as well as signals of activity and maintenance (*e.g.,* last commit time and license type). We also detect technical traits like programming language composition, and the presence of Dockerfile and README files, which help enrich the static MCP registry entries with development context.

B. Data Collection

The construction of MCPCorpus involves a five-stage pipeline to extract, enrich, normalize, and classify MCP artifacts from both centralized registries and decentralized code repositories. As shown in Figure 1, this process is designed to ensure the dataset is comprehensive, clean, and useful for both research and engineering tasks. The key stages are detailed as follows.

Registry Crawling. The collection process begins with crawling `MCP.so`. Our crawler systematically traverses its paginated listings and artifact detail pages to extract structured metadata, including artifact name, description, domain tags, category, tool schema, author information, deployment

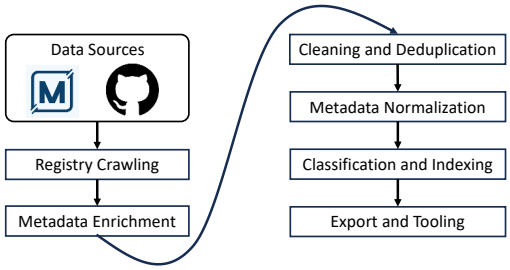

Fig. 1: Approach Overview

interface (*e.g.,* REST APIs), creation/update timestamps, and GitHub URLs.

A total of around 13.9K MCP servers and 300 clients were extracted at the time of collection. For example, one of the MCP server projects, *edgeone-pages-mcp*, exposes a tool named `deploy-html` and is tagged under the `cloud-platforms` category, with a GitHub repository maintained by TencentEdgeOne.

GitHub Metadata Enrichment. For each MCP artifact with a valid GitHub link, we query the GitHub REST API to collect its repository-level attributes. These include popularity metrics (*e.g., stars*, *forks*), maintenance signals (*e.g., last commit time*, *contributor count*), *license type*, and the composition of programming languages (*e.g.,* TypeScript, JavaScript). We also analyze the file structure to detect critical assets like *README.md*, *Dockerfile*, and other deployment-related files. For the aforementioned example, *edgeone-pages-mcp*, this phase confirms 125 stars, 16 forks, an MIT license, and recent activity (last commit in May 2025).

Cleaning and Deduplication. To ensure data quality, we remove artifacts with invalid GitHub links, unreachable repositories, or insufficient metadata—defined as missing critical fields such as repository URLs. Duplicate detection is based on GitHub repository canonicalized URLs. If multiple entries point to the same repository (e.g., forks, mirrors, or renamed variants), we retain the version with the highest activity level (e.g., stars, recent commits) or the earliest creation timestamp as a proxy for originality. This process significantly reduces noise and ensures dataset consistency.

Metadata Normalization. Each artifact's metadata is parsed and normalized into a unified schema with 20+ attributes. These attributes include basic descriptors (*e.g.,* category, interface type), technical metadata (*e.g.,* programming language, license), and derived indicators (*e.g.,* maintenance status, tool count). Fields originally encoded as JSON within the registry (*e.g.,* tool definitions, custom metadata blocks) are parsed and flattened into structured representations, enabling efficient downstream querying, filtering, and analysis. As not all attributes are applicable to every artifact type (*e.g.,* clients do not define tools), missing or inapplicable fields are filled with `null`.

Classification and Indexing. To facilitate structured exploration, each artifact is categorized along three core dimensions that are currently implemented: (i) artifact type (server or

TABLE I: Refined field schema of the MCPCORPUS dataset.

Field	Description
id	Unique identifier of the record
name	Short name of the MCP artifact
title	Display title of the project
description	Human-readable summary
author_name	GitHub username or organization
url	Repository URL
category	Domain category (*e.g.,* AI, tools)
tags	Comma-separated topic labels
type	Whether the artifact is a server or client
tools	MCP tools or functions exposed (JSON)
sse_url	Callable endpoint if applicable
server_command	Execution command or entry point
server_config	Tool config and environment variables
stargazers_count	GitHub stars
forks_count	GitHub forks
open_issues_count	Number of open issues
contributors_count	Number of unique contributors
last_commit	Timestamp of the most recent commit
full_name	Full GitHub repo name (*e.g.,* user/project)
language	Primary implementation language
languages	Byte-level language breakdown
license	License type (*e.g.,* MIT)
archived	Whether the repository is archived
has_docker	Presence of a Dockerfile
has_readme	Presence of README.md
has_requirements	Presence of dependency declarations

Basic Info — Interface & Config — GitHub { Signals, Metadata }

client), (ii) application category (e.g., cloud, AI, data), and (iii) programming language. These basic labels enable queries such as "Python-based AI servers" or "Go-written data clients." Additional dimensions such as interface type, officiality, and maintenance status are part of our design and will be integrated in future versions to support more fine-grained analysis.

Export and Tooling. The final dataset is stored in newline-delimited JSON (JSONL) format. Each record contains merged registry and GitHub metadata along with classification tags. We also provide auxiliary tools for data synchronization, statistical analysis, and web-based exploration. These utilities enable reproducible updates and promote ease of integration into research pipelines.

III. DATASET DESCRIPTION

A. Basic Composition

The MCPCORPUS dataset consists of around 14K MCP artifacts, including 13,875 servers and 300 clients. Each artifact is represented as a JSON object with up to 26 structured attributes, combining metadata from `MCP.so` and enriched signals from GitHub repositories. These attributes describe the artifact's identity, deployment interface, functionality, language stack, license, and development activity. All entries conform to a unified schema, supporting consistent parsing and downstream processing.

B. Attribute Categorization

To support downstream analysis and improve the interpretability of the dataset, we group the refined fields into four functional categories based on their semantic roles and data sources. This categorization helps researchers and developers better understand the structure of each artifact and selectively utilize relevant attributes for specific tasks such as filtering, compatibility checking, and repository quality assessment. Table I provides an overview of all retained fields and their corresponding descriptions.

- **Basic Information**: Fields such as *id*, *name*, *title*, *description*, *author_name*, *url*, *category*, *tags*, and *type* provide fundamental metadata that describes the identity, content, and domain classification of the MCP artifact.
- **Interface and Configuration**: Fields such as *tools*, *sse_url*, *server_command*, and *server_config* capture how an MCP artifact exposes its functionality, including runtime interface details and tool execution configuration. These fields are critical for tool compatibility analysis and endpoint validation.
- **GitHub Signals**: Fields such as *stargazers_count*, *forks_count*, *open_issues_count*, *contributors_count*, and *last_commit* represent quantitative signals extracted from GitHub that reflect project popularity, development activity, and community engagement. They are useful for analyzing development characteristics and identifying active or well-supported artifacts.
- **GitHub Metadata**: Fields such as *full_name*, *language*, *languages*, *license*, *archived*, *has_docker* and *has_readme* capture the technical structure, licensing, and documentation quality of each artifact. These attributes are valuable for security auditing, codebase profiling, and integration feasibility studies.

C. Data Sample

To illustrate the structure and metadata richness of individual entries in MCPCORPUS, we present a representative MCP server named *playwright-mcp*, developed and maintained by Microsoft [11]. This artifact belongs to the *browser-automation* category and is implemented in TypeScript. It exposes a command-line interface for headless browser control via Playwright, with metadata drawn from both the `MCP.so` registry and the corresponding GitHub repository. A simplified version of the entry is shown in 1. This sample contains all 26 structured fields defined in MCPCORPUS's schema, including both registry metadata (*e.g.,* category, tags, server command) and GitHub-derived indicators (*e.g.,* stars, forks, contributors). Even if some values are absent (*e.g.,* `sse_url`), the corresponding keys are retained to preserve consistency across records.

D. Insights and Characteristics

To further understand the structure and dynamics of the MCP ecosystem, we analyze two critical characteristics from the MCPCORPUS dataset: GitHub stargazer count and primary implementation language. These two features not only reflect project popularity and technical preference but also exhibit diverse distributions that support meaningful ecosystem-level insights.

```
{ "id": 4493,
  "name": "playwright-mcp",
  "title": "Playwright Mcp",
  "url":
  ↪  "https://github.com/microsoft/playwright-mcp",
  "description": "Playwright MCP server",
  "author_name": "microsoft",
  "tags": "mcp,playwright",
  "category": "browser-automation",
  "type": "server",
  "tools": "",
  "sse_url": null,
  "server_command": "docker exec -i mcp-node bash -c
  ↪  \"npx @playwright/mcp@latest --headless\"",
  "server_config": {"mcpServers": {"playwright":
  ↪  {"command": "npx","args":
  ↪  ["@playwright/mcp@latest","--headless"]}}},
  "github": {
    "full_name": "microsoft/playwright-mcp",
    "forks_count": 727,
    "stargazers_count": 11162,
    "open_issues_count": 22,
    "contributors_count": 24,
    "language": "TypeScript",
    "languages": {
      "TypeScript": 188386,
      "JavaScript": 13482,
      "Dockerfile": 2210},
    "license": "Apache License 2.0",
    "archived": false,
    "has_docker": true,
    "has_readme": true,
    "has_requirements": false,
    "last_commit": "2025-06-03T18:10:47Z"}}
```


Listing 1: Full MCP server entry covering all schema fields.

Popularity Distribution. Figure 2 illustrates the distribution of GitHub stars across MCP artifacts. We observe a highly skewed long-tail pattern: the majority of projects (over 10,000) have fewer than 10 stars, while only a small fraction exceed 500 or 1,000 stars. This suggests that although the ecosystem is rapidly growing, most MCP projects remain in early stages of development or adoption. Notably, some highly starred projects surpass 5,000 stars, indicating early consolidation of a few popular tools.

Language Ecosystem. Figure 3 shows the distribution of primary programming languages among MCP artifacts. Specifically, Python, TypeScript, and JavaScript are the most commonly used languages, accounting for the majority of implementations. Languages such as Go, Rust, Java, and C# also appear but with lower frequency. Interestingly, the distribution is consistent even when filtered to the top-starred projects, suggesting that these languages are not only popular but also dominant among well-maintained, widely adopted MCP artifacts.

These observations confirm that MCPCorpus captures both the breadth and structural concentration of the current MCP ecosystem, enabling downstream studies on language-specific adoption, sustainability, and integration practices.

IV. Conclusion

In this paper, we introduce MCPCorpus, a large-scale and evolvable dataset capturing the structure and dynamics of the MCP ecosystem. By integrating metadata from diverse

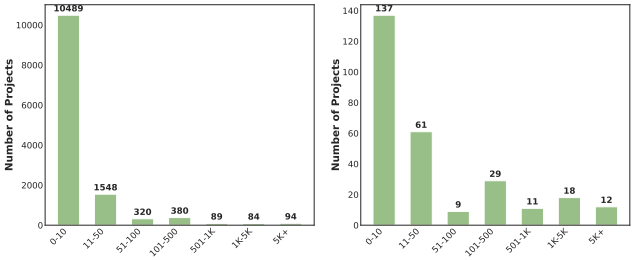

Fig. 2: Distribution of GitHub stars for all MCP servers (left) and MCP clients (right).

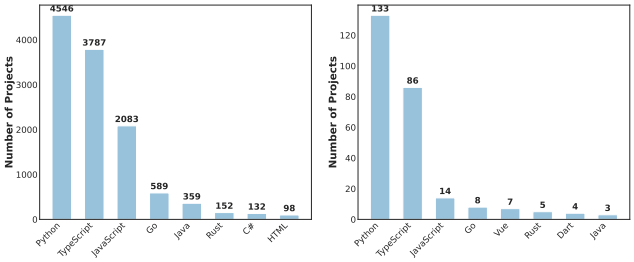

Fig. 3: Primary programming language distribution for all MCP servers (left) and MCP clients (right).

sources, MCPCorpus offers a unified view of MCP artifacts, supporting analyses of interoperability, security, and ecosystem health. The dataset, along with utility tools, is publicly released to foster reproducible research and inform the development of robust, tool-augmented LLM systems.

References

[1] Z. Shen, "Llm with tools: A survey," *arXiv preprint arXiv:2409.18807*, 2024.

[2] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models," *arXiv preprint arXiv:2307.06435*, 2023.

[3] N. Nahar, C. Kästner, J. Butler, C. Parnin, T. Zimmermann, and C. Bird, "Beyond the comfort zone: Emerging solutions to overcome challenges in integrating llms into software products," *arXiv preprint arXiv:2410.12071*, 2024.

[4] "Model context protocol specification," https://modelcontextprotocol.io/specification/2025-03-26, 2025, accessed 2025-06-14.

[5] A. Singh, A. Ehtesham, S. Kumar, and T. T. Khoei, "A survey of the model context protocol (mcp): Standardizing context to enhance large language models (llms)," 2025.

[6] "Whatsapp mcp exploited: Exfiltrating your message history via mcp," https://invariantlabs.ai/blog/whatsapp-mcp-exploited, 2025, accessed 2025-06-23.

[7] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model context protocol (mcp): Landscape, security threats, and future research directions," *arXiv preprint arXiv:2503.23278*, 2025.

[8] J. Fang, Z. Yao, R. Wang, H. Ma, X. Wang, and T.-S. Chua, "We should identify and mitigate third-party safety risks in mcp-powered agent systems," *arXiv preprint arXiv:2506.13666*, 2025.

[9] H. Song, Y. Shen, W. Luo, L. Guo, T. Chen, J. Wang, B. Li, X. Zhang, and J. Chen, "Beyond the protocol: Unveiling attack vectors in the model context protocol ecosystem," 2025. [Online]. Available: https://arxiv.org/abs/2506.02040

[10] mcp.so, "MCP Servers — mcp.so," https://mcp.so/, 2025.

[11] Microsoft, "GitHub - microsoft/playwright-mcp: Playwright MCP server," https://github.com/microsoft/playwright-mcp, 2025.