MCP-CAT: MCP 버전 및 전송 방식을 고려한 서버 적합성 테스트 자동화

김아인[1], 이승수[2]

인천대학교 (대학원생[1], 교수[2])

MCP-CAT: A Version- and Transport-Aware Conformance Automation Toolkit for MCP Servers

Ein Kim[1], Seungsoo Lee[2],

Incheon National University (Graduate student[1], Professor[2])

요 약

LLM 애플리케이션과 외부 도구를 잇는 Model Context Protocol(MCP)은 10개 언어의 공식 SDK를 제공하며, 같은 요청이라도 서버가 지켜야 할 동작이 명세 버전과 전송 방식에 따라 달라져 각 구현의 명세 준수를 일관되게 검증하기 어렵다. 본 논문에서는 명세 버전과 전송 방식을 함께 고려하여 MCP 서버의 적합성 테스트를 자동화하는 MCP-CAT(MCP Conformance Automation Toolkit)을 제안한다. MCP-CAT은 명세 본문과 JSON schema에서 적용 조건이 명시된 판정 기준을 만들고, 대상 SDK가 지원하는 기능을 반영한 단일 조건 테스트를 생성한 뒤, 실행 기록을 근거로 위반 여부와 실패가 처음 드러난 단계를 판정한다. 9개 공식 SDK의 23개 실행 구성을 평가한 결과 테스트 구성 정확도 99%와 위반 판정 정밀도 91%를 확인하였으며, 유지관리자에게 보고한 35건 가운데 29건이 실제 수정으로 이어졌다.

I. Introduction

MCP는 LLM 애플리케이션이 외부 도구와 데이터를 탐색하고 이용하는 절차를 표준화한 프로토콜로, 2024년 11월 공개 이후 10개 언어의 공식 SDK가 제공되고 있다[1]. 외부 도구와 데이터를 제공하는 MCP 서버는 LLM 애플리케이션과 외부 자원 사이의 신뢰 경계에 놓이므로, 서버의 이상 동작은 LLM 애플리케이션에도 영향을 미친다. 예를 들어, 입력을 처리하지 못한 서버 프로세스가 종료되거나, 서버가 유효한 요청에 응답하지 않아 작업이 완료되지 않을 수 있다.

서버의 명세 준수 여부는 개별 메시지의 형식뿐 아니라 초기화 등 선행 절차와 메시지 순서에도 좌우된다. 적용되는 요구사항은 명세 버전마다 달라지며, 같은 요청도 stdio와 Streamable HTTP에서서로 다른 방식 및 연결 관리 절차를 가진다. 따라서 테스트에는 명세 버전과 전송 방식에 맞는 실행 조건이 반영되어야 한다. 또한 통과/실패 결과만으로는 실패가 어느 단계에서 드러났고, 이후 통신에 어떤 영향을 주었는지 파악하기 어렵다.

따라서 본 연구는 명세 분석, 테스트 생성, 실행 기록에 기반한 판정을 하나의 흐름으로 연결한 MCP-CAT을 제안한다. MCP-CAT은 자연어 명세와 schema 파일에서 적용 조건이 명시된 판정 기준을 만들고, 대상 서버를 기동해 얻은 런타임 정보를 반영해 조건 하나만 위반하는 테스트를 생성한 뒤, 실행 기록을 근거로 위반 여부와 실패 단계를 판정한다.

본 논문의 기여는 다음과 같다.

- 명세와 schema를 버전, 전송에 맞는 판정 기준으로 구조화하고, 참조 유효성과 논리적 일관성을 검사한다.
- 대상 서버에서 수집한 런타임 정보를 반영하여 두 전송에서 비교 가능한 단일 조건 테스트를 구성한다.
- 9개 공식 SDK의 23개 실행 구성에서 발견한 35건을 유지관리자에게 보고했고, 이 가운데 29건이 수정되었다.

II. Background

MCP Overview

기본 구성. MCP에는 외부 도구나 데이터를 제공하는 서버와 LLM 애플리케이션 내부에서 서버와의 연결을 담당하는 클라이언트가 존재

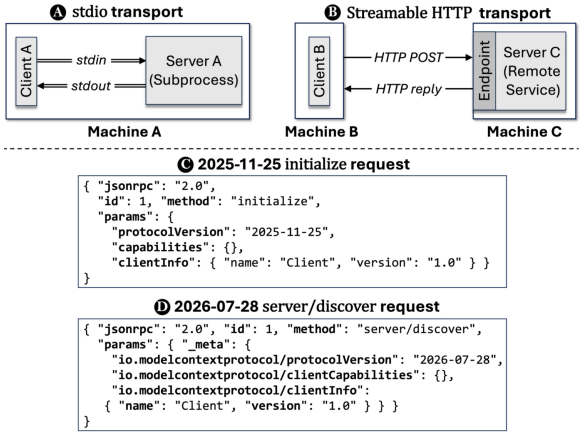


Figure 1. MCP transports and version-specific requests.

한다. 두 주체는 JSON-RPC 2.0[2] 위에서 기능을 호출하는 request와 결과를 반환하는 response, 응답이 필요 없는 notification을 교환한다. 명세는 서버가 지켜야 할 동작을 MUST나 SHOULD를 포함한 자연어로 서술하고, 각 필드와 타입은 JSON schema로 규정한다.

표준 전송. stdio와 Streamable HTTP 두 가지이다. stdio에서는 Figure 1-Ⓐ와 같이 클라이언트가 서버를 하위 프로세스로 실행하고 표준 입출력 스트림으로 통신하므로, 메시지 처리 오류가 이후 스트림 처리에도 영향을 줄 수 있다. 반면 Figure 1-Ⓑ의 Streamable HTTP에서는 HTTP 요청별로 메시지를 전달하며, JSON-RPC 메시지 외에 HTTP 상태 코드와 헤더까지 검사 대상이 된다.

프로토콜 버전. 명세는 2025-11-25와 그 이전은 legacy로, 2026-07-28과 그 이후는 modern으로 구분한다. Legacy는 Figure 1-Ⓒ의 초기화 요청으로 시작하는 핸드셰이크를 통해 버전과 지원 기능을 협상한다. Streamable HTTP에서는 서버가 초기화 시 발급한 세션 ID를 이후 요청에 사용한다. Modern은 초기화 절차와 프로토콜 수준의 세션을 제거하고, 각 요청의 _meta에 버전과 클라이언트 기능 정보를 포함한다. Figure 1-Ⓓ는 이러한 정보를 포함하는 선택적인 server/discover 요청의 예이다. 따라서 같은 method라도 버전과 전송 방식에 따라 적용되는 요구사항과 선행 절차가 달라진다.

2.2 Challenges

Figure 2는 Go SDK 서버에 id가 null인 요청을 보낸 사례이다. MCP 명세는 요청의 id를 문자열이나 정수로 제한하며, id 필드가 포함된 모든 요청에 응답을 요구하므로, 기대 동작은 잘못된 요청에 대한 오류 응답이다. 그러나 Legacy에서는 요청이 응답 없이 폐기된 뒤 후속 요청은 처리된 반면, Modern에서는 서버 프로세스가 종료되었다. 즉, 같은 입력이 버전별로 다른 결과를 냈고, 응답이 없는 원인도 서로 달

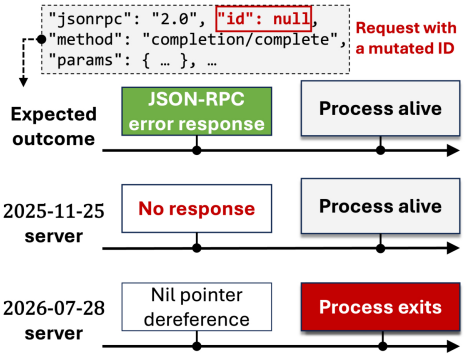


Figure 2. Go SDK outcomes for one stdio request.

랐다. 이렇게 판정을 자동화하는 데는 세 가지 어려움이 있다.

(1) 요구사항은 자연어 규정과 schema에 나뉘어 있고, 버전/전송별로 적용 여부가 다르다.

(2) 선행 절차, 지원 기능 및 런타임 정보가 시험 대상마다 달라, 이를 반영하지 못한 테스트는 목표 지점에 도달하지 못한다.

(3) 응답 유무 관찰만으로는 영향 범위가 드러나지 않아, 위 사례처럼 원인이 다른 실패를 구분하기 힘들다.

2.3 Related Work

ChatAFL은 LLM의 프로토콜 지식을 이용해 메시지 문법과 순서를 생성하고 fuzzing을 유도한다[3]. RFCAudit은 RFC와 소스 코드의 의미를 대조하여 구현의 기능적 오류를 탐지한다[4]. 공식 MCP Conformance Test Framework는 미리 정의된 시나리오로 명세 준수 여부를 검사한다[5]. MCP-CAT은 서버 소스 코드 분석 없이 명세 요구사항과 런타임 정보를 바탕으로 테스트를 구성하고, 버전·전송별 실행 결과의 판정과 비교를 자동화한다.

III. Design

Figure 3과 같이 MCP-CAT은 Conformance Knowledge Builder, Profile-Aware Test Generator, Conformance Evaluator로 구성되며, 세 구성 요소는 Conformance Knowledge Base를 통해 명세 요구사항과 시험 대상의 정보, 테스트 기록을 공유한다. 명세 버전과 SDK commit, 대상 서버의 기동 방법을 담은 test configuration 지정 이후의 과정은 자동으로 진행된다.

3.1 Conformance Knowledge Construction

Spec Analyzer는 명세에서 MUST나 SHOULD가 포함된 문장과 주변 맥락을 모으고, 해당 버전의 schema에서 필드 제약을 추출한다. 이를 Conformance Synthesizer가 LLM에 입력하여 동작의 주체, 대상 메시지, 성립 조건과 기대 결과를 구조화한다. 이때 단일 메시지에 적용되는 항목은 rule로, 여러 메시지의 순서 관계

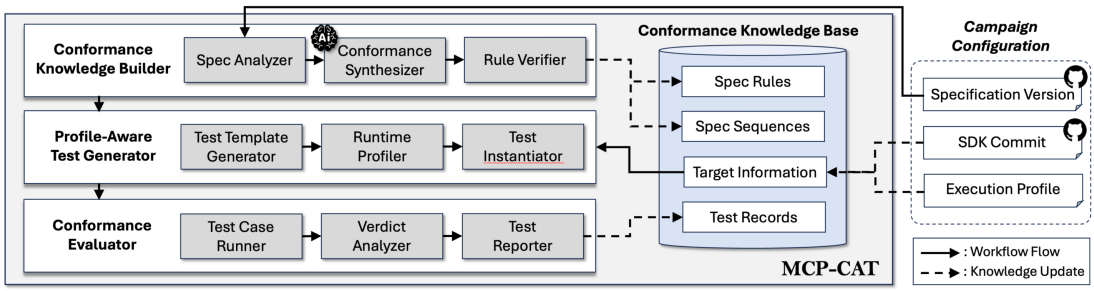


Figure 3. Architecture and workflow of MCP-CAT.

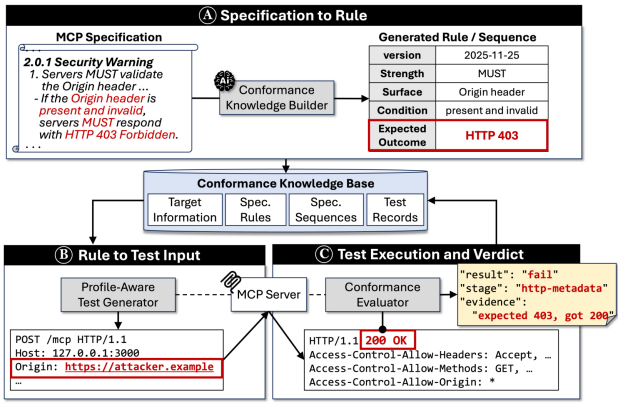


Figure 4. MCP-CAT test generation and evaluation.

에 대한 내용은 sequence로 나누고, 원문, 명세 버전, 전송 및 lifecycle 상태를 연결한다(Figure 4-Ⓐ). Rule Verifier는 참조한 method와 필드가 해당 버전의 schema나 전송 규정에 정의되어 있는지 확인하고, 동일한 응답에 result와 error를 동시에 요구하는 등 제약 간 충돌을 검사한다. 검사를 통과하지 못한 항목은 사유와 함께 기록하고 테스트 생성에서 제외한다. 이 검사는 참조의 유효성과 제약의 일관성을 확인하며, 명세 해석 자체의 정확성을 보장하지는 않는다.

3.2 Profile-aware Test Generation

Test Template Generator는 필수 parameter와 초기화 및 선행 메시지를 조합해 rule과 sequence를 만족하는 baseline template을 만든다. Runtime Profiler는 대상 서버를 실행하여 등록된 tool과 resource 등의 호출 정보를 수집한다. 이는 런타임에 수집되므로 소스가 공개되지 않은 서버에도 적용할 수 있다. Test Instantiator는 호출 정보를 baseline template에 사용하여 실행 가능한 test input을 구성한다(Figure 4-Ⓑ). 정상 실행이 확인된 경우 시험하려는 조건 하나를 활성화하도록 mutation을 적용하고, 나머지 조건과 런타임 의존성이 유지되는지 확인한다. 두 전송에 공통으로 적용되는 기준은 조건과 기대 결과를 유지하면서 메시지 형식, HTTP 메타데이터 등을 각 전송에 맞게 구성하여 전송 간 결과를 비교할 수 있도록 한다.

3.3 Evidence-based Conformance Evaluation

Test Case Runner는 테스트를 실행하고 응답 메시지, HTTP 상태 코드와 헤더, 타임아웃 및 연결·프로세스 상태를 기록한다. 응답이 없다면 후속 요청을 전송하여, 실패가 해당 요청에 국한되는지 혹은 연결이나 프로세스 상태에 영향을 주었는지 확인한다. Verdict Analyzer는 이 기록에서 응답 수신을 막은 상태가 있었는지 확인하고, 메시지의 schema 준수 여부, rule의 기대 결과, sequence의 순서 제약을 차례로 검사하여 최초 실패 단계를 찾는다. 위반은 근거가 충족되었을 때만 확정하고, 명세가 기대 동작을 하나로 정하지 않았거나 근거가 부족하면 inconclusive로 남긴다. Test Reporter는 판정, 실패 단계, 영향 범위와 적용된 rule 및 근거를 test record로 저장한다(Figure 4-Ⓒ).

IV. Evaluation

Implementation. MCP-CAT은 TypeScript로 구현했으며, 명세 요구사항을 구조화하기 위해 QuantTrio/Qwen3.6-27B-AWQ 모델을 사용했다. 재현성을 위해 2025-11-25와 2026-07-28 명세 및 각 SDK의 commit hash가 고정하였다. Swift를 제외한 9개 공식 SDK에서 명세 버전·SDK·전송 방식의 조합을 하나의 실행 구성으로 정의하고, 공식 레퍼런스 서버가 없거나 해당 전송을 지원하지 않는 조합을 제외한 23개 구성을 평가했다(Table 1). 따라서 평가 범위는 각 레퍼런스 서버의 실행 경로만에 한정되어, SDK의 전체 API나 클라이언트 구현을 포괄하지 않는다.

Table 1. Evaluated MCP SDK versions and transports

SDK	2025-11-25		2026-07-28	
	stdio	HTTP	stdio	HTTP
TypeScript	v1.29.0	v1.29.0	N/A	`95d28cb`
Python	v1.27.1	v1.27.1	N/A	`0da9092`
Go	v1.6.0	v1.6.0	`65ece59`	`65ece59`
Rust	v1.7.0	v1.7.0	N/A	N/A
Ruby	v0.16.0	v0.16.0	N/A	N/A
Kotlin	0.12.0	0.12.0	N/A	N/A
PHP	v0.5.0	v0.5.0	N/A	N/A
C#	v1.3.0	v1.3.0	`d6b1615`	`d6b1615`
Java	N/A	v1.1.2	N/A	N/A
Total	**8**	**9**	**2**	**4**

2025-11-25: release tags.
2026-07-28: seven-character pinned `main` commits.
N/A: no official reference server was available or the transport was unsupported.

Table 2. Quantitative evaluation results.

Metric	2025-11-25	2026-07-28
Valid test construction	481/484 (99.4%)	1,588/1,592 (99.7%)
Violation precision	91/99 (91.9%)	53/57 (93.0%)
Inconclusive precision	163/167 (97.6%)	300/304 (98.7%)
Failure-stage accuracy	92/99 (92.9%)	54/57 (94.7%)
Impact-scope accuracy	98/99 (99.0%)	55/57 (96.5%)
Cross-transport divergence	54/336 (16.1%)	6/35 (17.1%)

Verdict Accuracy. 판정 결과는 연구진 3명이 명세 조항과 실행 기록을 대조하여 검토했다. 위반 여부와 실패 단계, 영향 범위는 각각 독립적으로 평가되었다. 검증을 통과한 판정 기준으로부터 대상별 구체화 이전에 생성한 고유 테스트는 Legacy 484개, Modern 1,592개이다. 테스트 구성 정확도는 이 가운데 목표 조건과 기대 결과를 보존하고 나머지 조건 및 baseline 상태를 유지한 비율이다. 위반과 inconclusive 판정 정밀도는 각 판정 집합에서 수동 검토와 일치한 비율로, 각각 91.9–93.0%와 97.6–98.7%였다. 실패 단계와 영향 범위 정확도는 92.9–94.7%와 96.5–99.0%이었다. 위반으로 잘못 판정한 사례는 주로 서버가 클라이언트에 요청을 보내거나 응답이 오래 걸리는 요청을 처리하는 과정에서 발생했다. 동일한 판정 항목·SDK·명세 버전에 대해 양쪽 전송 모두 실행된 336개와 35개의 테스트 쌍 중, 각 16.1%와 17.1%에서 결과가 달랐다. 두 전송의 판정이 다르거나, 모두 실패하더라도 실패 단계·유형·영향 범위 중 하나 이상이 다르면 전송 간 결과 차이로 보았다.

Findings. Table 3은 SDK별로 보고한 35건을 17개 동작 유형으로 묶은 것으로, 9개 SDK 중 8개와 양쪽 전송 모두에서 확인되었다. 이 가운데 4개 유형은 명세가 결과 동작을 정하지 않아 inconclusive로, 나머지 13개는 메시지 검증, 에러 처리, lifecycle 및 버전 호환성 요구사항을 위반한 사례로 분류하였다. 예를 들어 중복된 초기화 요청이 세션 상태 덮어쓰기, 새로운 세션 생성 등으로 처리된 경우 명세가 수락/거부 방식을 정하지 않아 inconclusive로 판정했다(I4). 반면 거부 응답에서 필수 id를 누락한 사례는 명시적인 응답 형식 위반이다(V1). Figure 4의 PHP 서버는 임의 출처 헤더가 포함된 요청에 HTTP 403 거부가 아닌 200을 반환하여 위반으로 판정되었다(V5). 보고한 35건 가운데 29건이 수정되었고, 6건은 수정 중이다.

V. Conclusion

본 논문은 명세 버전과 전송을 고려하여 MCP 서버의 적합성 테스트를 자동화하는 MCP-CAT을 제안한다. MCP-CAT은 적용 조건이 명시된 판정 기준을 만들고, 대상 서버의 실행 정보를 반영한 단일 조건 테스트를 생성하며, 실행 기록을 근거로 명세 위반 여부와 실패 단계, 영향 범위를 제시한다. 실험을 통해 9개 공식 SDK의 23개 구성을 평가하여 테스트 구성 정확도 99%와 위반 판정 정밀도 91%을 확인하였고, 보고한 35건 가운데 29건이 수정으로 이어졌다.

Table 3. Finding patterns from 35 disclosure records.

ID	Ver.	Test case	Status by SDK
I1	L	Extreme JSON nesting	Fixed (Rs)
I2	L	Version conflict: header vs. body	Fixed (Rs, Rb, C#), In progress (TS, Py, Kt)
I3	L	Malformed session ID	Fixed (PHP)
I4	L	Duplicate `initialize` with modified params #1	Fixed (Go, Rb, Rs, PHP, Kt), In progress (TS, Py)
V1	L	Duplicate `initialize` with modified params #2	Fixed (Kt)
V2	L	Invalid `jsonrpc`/`method` types	Fixed (Go, Rb)
V3	L	Empty or null `method`/`id`	Fixed (Go), In progress (TS)
V4	L	Invalid MCP version header	Fixed (PHP, Rb)
V5	L	Disallowed `Origin` header	Fixed (PHP)
V6	L	`initialize` missing a required field	Fixed (Go, Rb, Kt)
V7	L, M	Modern version in legacy handshake	Fixed (Go, C#)
V8	L, M	Unknown method or invalid params	Fixed (Go)
V9	L, M	Cancellation notification with an ID	Fixed (Rb)
V10	M	Missing `Mcp-Name` header	Fixed (C#)
V11	M	Required `resultType` omitted	Fixed (Go, C#)
V12	M	`id:null` request	Fixed (Go)
V13	M	Server request with unset `mode`	Fixed (Go)

I: inconclusive, V: violation, L: 2025-11-25, M: 2026-07-28.
Kt: Kotlin, Py: Python, Rb: Ruby, Rs: Rust, TS: TypeScript.

Acknowledgements

이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. RS-2025-16069415).

[참고문헌]

[1] Model Context Protocol, “Model Context Protocol Specification,” https://modelcontextprotocol.io/specification

[2] JSON-RPC Working Group, “JSON-RPC 2.0 Specification,” https://www.jsonrpc.org/specification

[3] R. Meng, M. Mirchev, M. Böhme, and A. Roychoudhury, “Large Language Model Guided Protocol Fuzzing,” Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS), 2024.

[4] M. Zheng, C. Wang, X. Liu, J. Guo, S. Feng, and X. Zhang, “RFCAudit: AI Agent for Auditing Protocol Implementations Against RFC Specifications,” Proceedings of the 40th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1221–1233, 2025.

[5] Model Context Protocol, “Model Context Protocol Conformance Test Framework,” GitHub repository, https://github.com/modelcontextprotocol/conformance