JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 30 No. 10, pp. 43-52, October 2025
https://doi.org/10.9708/jksci.2025.30.10.043

NL2MCP: A New Framework for Natural Language Query Execution

Won-Bae Kim*, Nammee Moon**

*Student, Dept. of Convergence Engineering, Hoseo University, Seoul, Korea
**Professor, Dept. of Computer Science and Engineering, Hoseo University, Asan, Korea

[Abstract]

This study compares two approaches for natural language query execution, NL2SQL and NL2MCP, under identical conditions. Using GPT-4o, PostgreSQL, and the Spider dev dataset, we evaluated three paths: NL2SQL(cache), NL2MCP(cache), and NL2MCP(live). Metrics included exact match (EM), execution match (EX), mean/p95 latency, and error rates across difficulty levels (easy, medium, hard, extra). Results show that NL2MCP yielded lower EM but higher EX and more stable latency. In particular, the live mode reduced schema mismatch errors and achieved the highest success rate in hard/extra queries, indicating that NL2MCP(live) is the most reliable path in real-world environments.

▸ **Key words**: NL2MCP, NL2SQL, Model Context Protocol, Execution Match, Latency (p95), Spider Dataset

[요 약]

본 연구는 자연어 질의를 데이터베이스 실행으로 변환하는 두 가지 접근, NL2SQL과 NL2MCP를 동일 조건에서 비교·분석하였다. 언어모델은 GPT-4o, 데이터베이스는 PostgreSQL, 데이터셋은 Spider dev를 사용하고, NL2SQL, NL2MCP(cache), NL2MCP(live) 세 경로를 설정하였다. 실험은 정확도(EM), 실행 성공률(EX), 평균 및 p95 지연, 오류율을 난이도별(easy, medium, hard, extra)로 측정하였다. 결과적으로 NL2MCP는 EM에서는 낮았으나 EX와 지연 안정성에서 우위를 보였으며, 특히 live 모드는 스키마 불일치 오류를 줄이고 hard·extra 구간에서 가장 높은 완수율을 달성하였다. 이는 실제 운영 환경에서 NL2MCP(live)가 가장 신뢰할 수 있는 실행 경로임을 보여준다.

▸ **주제어**: NL2MCP, NL2SQL, 모델 컨텍스트 프로토콜(MCP), 실행일치(EX), 지연시간, Spider 데이터셋

• First Author: Won-Bae Kim, Corresponding Author: Nammee Moon
*Won-Bae Kim (wonkim@dataslab.co.kr), Dept. of Convergence Engineering, Hoseo University
**Nammee Moon (mnm@hoseo.edu), Dept. of Computer Science and Engineering, Hoseo University
• Received: 2025. 09. 22, Revised: 2025. 10. 13, Accepted: 2025. 10. 20.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

1. Background

자연어 기반 데이터 질의는 사용자가 구조적 질의 언어(SQL)를 학습하지 않고도 데이터베이스에서 정답을 획득하도록 돕는 핵심 기술로 자리매김하였다. 전통적 접근은 대규모 언어모델이 자연어를 관계형 데이터베이스의 SQL로 직접 변환해 실행하는 NL2SQL 방식이다. 이 방식은 파이프라인이 단순하고 초기 적용이 용이하다는 장점이 있으나, 스키마 명명 규칙 변화와 데이터 모델의 다양성에 민감하고, 프롬프트 설계나 평가 지표에 따라 성능 편차가 커 일반화 가능성이 제한된다는 문제가 반복적으로 지적되어 왔다 [9]-[15]. 또한 실행 전 검증, 권한 및 정책 연계, 거버넌스 요구사항을 표준화된 절차로 통합하기 어렵다는 운영상의 한계도 존재한다.

이와 대조적으로 최근 등장한 MCP (Model Context Protocol)은 언어모델이 외부 시스템을 표준화된 도구 호출 인터페이스로 다루도록 설계된 규약이다 [1],[2]. MCP는 세션·이벤트·메시지 규약을 통해 모델과 도구, 실행 환경을 모듈화하여 상호 운용성을 높이고, 스키마 동기화와 자동 스케일링, 중앙집중식 검사 및 정책 집행을 가능하게 한다 [4],[5],[7],[8]. ReAct [19]와 Toolformer [20] 같은 연구는 도구 사용과 추론을 결합하는 사례로 확산되고 있으며, 실제 응용에서도 MCP 기반 통합이 빠르게 진행되고 있다. 이러한 맥락에서 언어모델이 execute_sql, list_schemas, explain_query 등 MCP 툴을 호출해 데이터베이스 질의를 실행하는 NL2MCP가 NL2SQL의 대안으로 부상하고 있다.

특히 AI 에이전트의 대두와 함께 데이터베이스 쿼리 영역에서는 복잡한 스키마 검증, 실행 전 정책 점검, 오류 회피를 위해 MCP 적용이 점차 필요해지고 있다. 최근 문헌은 NL2MCP가 데이터베이스 접근을 안전하게 표준화하고, 다중 모델 및 운영 환경에서의 확장성을 제공하는 유효한 대안임을 강조하고 있다 [1]-[2],[5],[7].

2. Objectives

본 연구의 목표는 NL2SQL과 NL2MCP를 동일한 조건에서 공정하게 비교하는 것이다. 본 연구는 로컬/원격 MCP 구분없이 NL2MCP를 단일 프레임워크로 통합하고 종단간(end-to-end) 성능 비교에 초점을 맞춘다. 이를 위해 동일한 언어모델(GPT-4o), 동일 데이터셋(Spider dev), 동일 데이터베이스(PostgreSQL) 조건에서 NL2SQL(cache), NL2MCP(cache), NL2MCP(live) 세 경로를 설정하였다.[16] 특히 Spider 데이터셋의 난이도 태깅(easy, medium, hard, extra)을 반영하여 성능을 세분화 관찰함으로써, 질의 복잡도에 따른 경향을 정량적으로 분석한다.

3. Contributions

본 연구의 기여는 다음 세 가지로 요약된다.

(1) 동등 비교 프레임 제시: 동일한 언어모델·데이터셋·데이터베이스 조건에서, NL2SQL과 NL2MCP(cache/live)를 카탈로그 정보량을 맞춘 상태에서 공정하게 비교한다.

(2) 종단간 성능 분석: 정확도(EM), 실행 성공률(EX), 평균 및 p95 지연, 오류 유형 분포를 난이도별로 측정하여 NL2MCP의 장단점을 실증적으로 제시한다.

(3) 운영 시사점 도출: NL2SQL의 단순성과 NL2MCP의 운영 확장성을 종합 평가하고, 실제 환경에서 NL2MCP(live)가 가장 신뢰할 수 있는 실행 경로임을 확인하였다.

4. Organization of the Paper

Ⅱ장은 NL2SQL과 NL2MCP의 배경 및 관련 연구를 다룬다. Ⅲ장은 문제 정의와 설계 목표를 제시하며, NL2SQL과 NL2MCP 실행 아키텍처를 설명한다. Ⅳ장은 Spider 데이터셋 기반의 실험 설정을 구체화하고, Ⅴ장은 실험 결과를 난이도별 성능·지연·오류 분석 중심으로 논의한다. 마지막으로 Ⅵ장은 결론과 향후 연구 과제를 제시한다.

II. Preliminaries

1. Paradigm of Natural Language Data Query

자연어를 입력으로 받아 데이터베이스에서 정답을 획득하려는 시도는 두 흐름으로 전개되어 왔다.

첫째, 대규모 언어모델이 자연어를 관계형 데이터베이스의 실행 가능 쿼리(SQL)로 직접 변환하여 제출하는 NL2SQL이다. 이 접근은 파이프라인이 단순하고 초기 적용이 용이하다는 장점이 있으나, 스키마 명명 규칙 변화나 데이터 모델 차이에 민감하게 반응하며 실행 전 검증, 권한 및 정책 연계 등을 일관된 방식으로 통합하기 어렵다는 한계를 지닌다 [9]-[15].

둘째, 언어모델이 Model Context Protocol(MCP) 기반의 표준화된 도구 호출 인터페이스를 통해 질의를 수행하는 NL2MCP이다. 이 방식은 실행을 도구 호출 추상화로 캡슐화하므로 정책·검사·거버넌스 체계와의 결합이 용이하

고, 다중 언어모델 및 이기종 시스템으로의 확장에도 유리하다 [1]-[2],[4]-[5],[7]-[8].

2. Evolution and Limitations of NL2SQL

NL2SQL은 스키마 정합성과 문법 정확도를 높이기 위해 지도학습, 사전학습, 다단계 프롬프트 설계, 실행 피드백 기반 교정 루프 등 다양한 기법을 축적해 왔다 [9]-[15]. 대규모 비교 연구는 모델·프롬프트·평가지표의 선택에 따라 성능 편차가 크게 나타나며, 이러한 불안정성이 일반화 가능성을 제한할 수 있음을 보여주었다 [9]-[15].

또한, 다단계 프롬프트와 오류 교정 전략을 결합하거나, 제로샷 설정에서 사전학습 모델을 활용하거나, 질문 풍부화를 통한 스키마 직접 연결, 생성 후 랭킹 기반 후보 선택 등 다양한 방법이 제안되었다 [11]-[14]. 그럼에도 불구하고 권한·정책·검사 요구를 표준화된 인터페이스로 연결하기는 어려웠고, 설명 계획 확인이나 샌드박스 실행 같은 사전 검증을 안정적으로 통합하기도 쉽지 않아 운영 확장성 측면에서 제약이 반복적으로 보고되었다 [9]-[15].

3. Overview of MCP and NL2MCP

Model Context Protocol(MCP)은 언어모델이 외부 시스템을 도구로 추상화하여 호출하고, 결과를 수집하며, 오류를 처리할 수 있도록 세션·이벤트·메시지 규약을 정의한다 [1]-[2]. MCP는 모델·도구·실행 환경을 모듈화하여 상호 운용성을 높이며, 오픈소스 레퍼런스 서버와 클라이언트 구현을 통해 다양한 활용이 가능하다 [4]-[5],[7]-[8].

MCP 환경에서 NL2MCP는 \texttt{execute_sql}, \texttt{list_schemas}, \texttt{explain_query} 등의 MCP 툴 호출을 통해 질의를 실행한다 [1]-[2],[5],[7]-[8]. 구현은 MCP 클라이언트와 MCP 서버가 결합되어 동작하며, 호출 경로가 로컬이든 원격이든 동일한 NL2MCP 실행 경로로 정의할 수 있다.

4. Overview of MCP Architecture

Fig. 1은 MCP의 일반적인 아키텍처를 보여준다. MCP Host 환경에서 Claude Desktop, IDE, AI Tools 등 다양한 애플리케이션은 MCP 클라이언트를 통해 MCP 서버와 통신하며, 서버는 파일시스템·데이터베이스·인터넷 등 외부 자원에 접근한다. 이를 통해 언어모델이 안전하고 표준화된 방식으로 외부 도구를 호출할 수 있다 [1]-[2],[6]-[8].

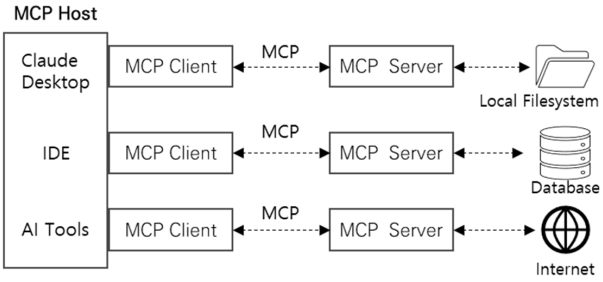

Fig. 1. MCP Components

5. Distinctiveness of This Study

기존의 Text-to-SQL 계열 연구들은 언어모델이 자연어를 SQL로 직접 변환하는 내부 생성 과정의 정확도를 높이는 데 주된 관심을 두어 왔다.

이에 반해 본 연구의 NL2MCP는 언어모델이 SQL을 직접 출력하지 않고, MCP(Model Context Protocol) 도구 호출을 통해 실행을 위임하는 방식으로 접근한다. 즉, 질의의 생성이 아니라 실행 과정 자체를 프로토콜 수준에서 표준화하여, 실행 전 검증(explain_query), 스키마 동기화(list_schemas, list_objects), 권한 점검 등 운영 절차를 통합할 수 있도록 설계하였다.

이로써 NL2MCP는 기존 NL2SQL이 해결하지 못한 정책 연동·실행 검증·운영 안정성의 측면을 기술적으로 보완하며, 단순한 모델 성능 비교를 넘어 실제 데이터베이스 환경에서 재현 가능한 실행 신뢰성을 확보한 점에서 차별성을 갖는다.

기존 NL2SQL 패러다임이 “언어모델 내부에서 얼마나 정확히 SQL을 생성할 수 있는가”에 초점을 맞췄다면, NL2MCP는 “그 SQL이 얼마나 안전하고 일관되게 실행될 수 있는가”라는 시스템적 관점으로 문제를 재정의한다. 따라서 본 연구는 자연어 질의의 결과 품질뿐 아니라, 실행 과정의 표준화·정책검증·거버넌스 통합을 포괄하는 새로운 실행 프레임워크를 제시한다는 점에서 기존 연구와 명확히 구분된다.

III. Problem Definition & Design Goals

1. Problem Definition

본 연구의 과제는 사용자가 제시한 자연어 질문을 입력으로 받아 관계형 데이터베이스(PostgreSQL)에서 정답을 조회하는 것이다. 이 과정에서 해결해야 할 도전 요소는 다음 세 가지이다.

(1) NL→SQL 변환 신뢰성: 스키마 연결(schema linking)과 구문 타당성(syntax validity)을 동시에 확보해야 한다 [9]-[15].
(2) NL2MCP 도입 효과의 정량화: 도구 호출 추상화, JSON-RPC 기반 메시징, 세션 관리가 성능(지연, 실패율)에 미치는 영향을 정량적으로 관찰해야 한다 [1]-[5],[7],[8].
(3) 운영·거버넌스 요구의 충족: 사전 검증, 권한 및 정책 연계, 실행 격리를 실현하면서도 정확도와 완수율을 저해하지 않는 균형점을 찾아야 한다 [1]-[3],[6],[19],[20].

2. Common Execution Engine: LLM

언어모델은 NL2SQL과 NL2MCP 두 실행 경로를 모두 지원하는 상위 엔진으로 동작한다. Fig. 2에 나타난 바와 같이, 입력 단계에서는 자연어 질문을 수령하고 시스템 프롬프트(스키마 요약, 금지 규칙, 결과 형식 포함)를 구성한다. 이후 질의 의도를 해석하여 단기 실행 계획을 산출한 뒤, 실행 경로에 따라 NL2SQL 또는 NL2MCP로 분기한다 [1]-[2].

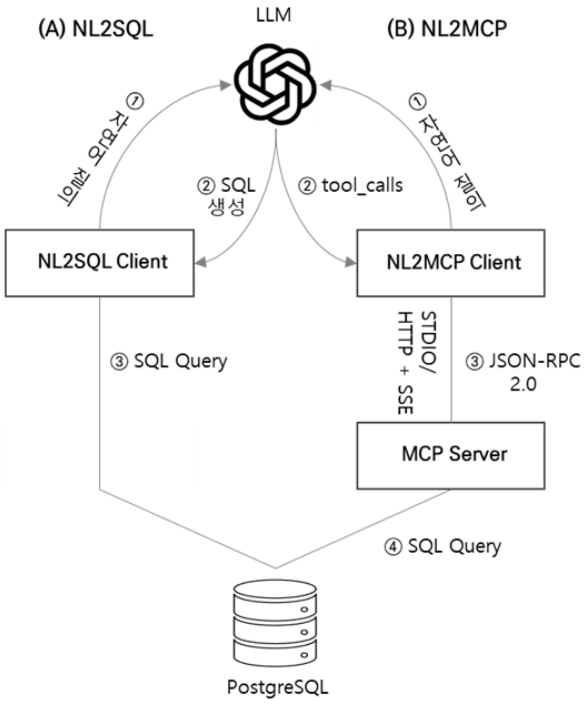

Fig. 2. Experimental Architecture

(A) NL2SQL은 LLM이 직접 SQL을 생성하여 DB로 제출하는 반면, (B) NL2MCP는 LLM이 MCP 도구 호출을 통해 클라이언트와 서버를 거쳐 SQL 실행을 수행하는 구조를 보여준다.

3. (A) NL2SQL Pipeline (Baseline)

이 경로는 ① 자연어 질의 → ② SQL 생성 → ③ SQL 쿼리 순으로 진행한다. LLM이 자연어 질의를 직접 SQL로 변환하고, NL2SQL Client가 이를 PostgreSQL에 제출한다. 장점은 호출 경로가 단순하고 TCP/IP를 통한 직접 실행으로 오버헤드가 낮다는 점이다. 그러나 실행 전 검증이나 정책 연계가 어렵고, 스키마 변화·명명 규칙에 민감하여 일반화 가능성이 제한된다 [9]-[15].

4. (B) NL2MCP Pipeline (Proposed Path)

이 경로는 ① 자연어 질의 → ② \texttt{tool_calls} → ③ JSON-RPC 전송 → ④ SQL 쿼리 순으로 진행한다. LLM은 SQL을 직접 생성하지 않고 MCP 툴 호출(\texttt{tool_calls}) 형태로 NL2MCP Client에 명령을 전달한다. 클라이언트는 JSON-RPC 2.0 형식으로 메시지를 포맷하여 \textit{stdio} 또는 HTTP+SSE 채널을 통해 MCP Server로 전송한다. 서버는 해당 요청을 SQL로 변환·실행하여 PostgreSQL에 제출하고 결과를 반환한다 [1]-[2],[5],[7]-[8].

이 구조는 실행을 도구 호출 추상화로 감쌌다는 점에서 의미가 있다. \texttt{execute_sql}, \texttt{list_schemas}, \texttt{explain_query} 등 MCP 툴을 통해 질의 실행 전 검증과 정책 집행이 가능하며, 호출·응답 로그가 표준화되어 운영·거버넌스 측면에서 장점을 가진다 [1]-[2],[5],[7]-[8].

Table 1. MCP Tool List

Tools	Input	Return
list_schemas	{}	[{"schema_name": str}]
list_objects	{"schema_ name": str}	[{"table": str, "columns": [str]}]
execute_sql	{"query": str}	{"rows": [..], "elapsed_ms": int}
explain_query	{"query": str}	{"plan": json}

5. Design Goals

본 연구의 설계 목표는 Fig. 2에서 제시된 두 실행 경로(NL2SQL vs NL2MCP)를 동일한 조건(언어모델, 데이터셋, 데이터베이스)에서 비교하여 다음을 달성하는 것이다.

(1) 정확성 평가: SQL 변환 정확도 및 실행 성공률 측정 [9]-[16].
(2) 효율성 평가: 평균 지연 및 p95 지연 비교 [1]-[2].
(3) 신뢰성 평가: 오류 유형(구문 오류, 스키마 불일치, 권한 오류, 타임아웃 등) 분석 [9]-[15].
(4) 운영 시사점 도출: NL2SQL과 NL2MCP 각각의 장

단점을 난이도별 성능 결과와 함께 제시. 데이터셋은 Spider를 사용하여 난이도 구간별 분석을 수행한다 [16].

IV. Experimental Setup

1. Dataset and Workload

실험은 Spider dev 세트를 사용한다. Spider 데이터셋은 Yale University에서 구축된 복합질의 벤치마크로, 관계형 스키마 다양성과 도메인 확장성을 모두 갖춘 유일한 공개 표준이다. 200개 이상의 실제 데이터베이스 스키마와 10,181개의 질의-정답 쌍을 포함하며, 복합 조인·서브쿼리·집계 등을 아우른다.

특히 NL2SQL 및 Text-to-SQL 연구 대부분(RAT-SQL, PICARD, DIN-SQL 등)이 동일한 데이터셋을 사용하여 결과를 비교 가능하게 하므로, 본 연구에서도 재현성과 비교 가능성 확보를 위한 공통 벤치마크로 Spider dev를 채택하였다.

Spider의 모든 질문은 난이도에 따라 easy, medium, hard, extra 네 구간으로 태깅되어 있으며, Table 1은 각 난이도별 질문 수를 정리한 것이다.

Table 2. Difficulty Level in Spider_dev

Difficulty	Classification rule	Questions
easy	Single SELECT without aggregation	319
medium	Multiple column SELECT or includes aggregation	258
hard	Requires joins across 2 tables	391
extra	Requires joins across 3+ tables, nested queries	66
Total		1,034

추가적으로, Spider의 원본 데이터셋은 SQLite 데이터베이스 형식으로 제공되나, SQLite는 공식 MCP 지원이 부족하여 PostgreSQL 16 환경으로 변환·적재하였다. 이 과정에서 다음과 같은 문제와 해결 경험이 있었다.

- 대소문자 민감성 문제: SQLite는 컬럼 및 테이블 명칭에서 대소문자 구분이 관대하지만, PostgreSQL은 엄격하게 구분한다. 따라서 스키마와 테이블 이름을 모두 소문자로 정규화하여 적재하였다.
- 인코딩 오류: 일부 문자열 컬럼에서 비표준 인코딩(예: Albarrac��N) 문제가 발생하였으며, 이를 UTF-8로 재인코딩하거나 특수문자 제거 전처리를 수행하였다.
- 데이터 타입 차이: SQLite는 동적 타입을 허용하지만, PostgreSQL은 정형화된 스키마를 요구한다. 이에 따라 TEXT, INTEGER, REAL 등의 타입을 명시적으로 매핑하였다.
- Foreign Key 제약 조건 충돌: 일부 스키마에서 참조 무결성 오류가 발생했으며, 데이터 삽입 순서를 조정하거나 일시적으로 FK 제약을 해제 후 재적용하는 방식을 사용하였다.

2. Model and Environment

언어모델은 GPT-4o로 고정한다.

- 서버 사양: 20코어 CPU, 128GB 메모리, SSD 탑재의 단일 물리 서버
- 데이터베이스: PostgreSQL 16 (단일 인스턴스),
- OS 및 런타임: Ubuntu 22.04 LTS, Python 3.10

3. Comparison Conditions

본 연구에서는 Fig. 2의 구조를 기반으로 세 가지 실행 경로를 비교한다.

(1) NL2SQL(cache)

- 개념: LLM이 자연어 질의를 직접 SQL로 변환하고, NL2SQL 클라이언트가 이를 PostgreSQL에 제출
- 카탈로그 제공 방식: NL2MCP 클라이언트가 사전에 수집·요약한 카탈로그 정보(스키마, 테이블, 컬럼명 등)를 프롬프트에 캐시 형태로 주입
- 목적: MCP 계층을 도입하지 않은 베이스라인으로서, 동일한 카탈로그 인지 상태에서 NL2MCP와 공정하게 비교하기 위함

(2) NL2MCP(cache)

- 개념: LLM은 tool_calls를 생성하고, NL2MCP 클라이언트가 이를 JSON-RPC 2.0 메시지 형식으로 HTTP+SSE 채널을 통해 MCP 서버에 전달한다. MCP 서버는 해당 요청을 SQL로 실행하고 결과를 반환
- 카탈로그 제공 방식: (1)과 동일한 캐시 요약본을 프롬프트에 주입하여 LLM이 NL2SQL과 동등한 카탈로그 정보를 기반으로 추론
- 목적: 동일한 카탈로그 인지 상태에서 NL2MCP의 도구 호출 계층이 성능(정확도, 완수율, 지연, 오류율)에 미치는 영향을 측정

(3) NL2MCP(live)

- 개념: NL2MCP 실행 경로는 (2)와 동일하되, 카탈로그는 캐시가 아니라 실행 시점에 동적으로 조회

- 카탈로그 제공 방식: 질의 실행 전 list_schemas, list_objects 등 MCP 툴을 호출하여 최신 카탈로그를 확보하고, 이를 요약 규칙에 따라 프롬프트에 반영
- 목적: 실제 운영과 유사한 환경에서, 스키마 변화 대응력과 동적 조회가 지연(p95)에 미치는 영향을 관찰

NL2SQL 경로는 기본적으로 LLM이 프롬프트에 주어진 스키마 요약을 바탕으로만 SQL을 생성하므로, 실행 중 동적으로 카탈로그를 갱신하거나 조회할 표준화된 인터페이스가 존재하지 않는다. 즉, MCP와 달리 list_schemas·list_objects와 같은 도구 호출을 통해 실시간 카탈로그를 불러올 수 없기 때문에, NL2SQL live라는 실험 조건은 기술적으로 정의하기 어렵고 연구 범위에서 제외하였다.

4. Evaluation Metrics

① 정확도(EM rate): 생성 SQL이 표준답안과 문자 수준에서 일치하는 비율.

② 완수율(EX rate): 실행이 성공하고 결과가 정답과 동치인 비율.

③ 평균 지연(lat_mean): 사용자 입력부터 최종 결과 반환까지의 평균 지연.

④ p95 지연(lat_p95): 사용자 입력부터 최종 결과 반환까지의 95번째 백분위수 지연(상위 지연 분포의 척도).

5. Experimental Procedure

본 연구는 종단간 실험으로 LLM, NL2MCP Client, MCP Server, PostgreSQL을 포함한다.

- 입력: Spider dev 자연어 질문
- 실행: 각 질문을 ① NL2SQL, ② NL2MCP 로 처리
- 반복: 각 질문은 3회 반복 실행
- 보고: 정확도, 완수율, 지연, 실패율을 난이도별 구간(easy, medium, hard, extra)로 분리 보고

이 절차를 통해 LLM의 생성 품질과 MCP 계층 도입 효과가 최종 사용자 경험에 어떤 차이를 만드는지를 종합적으로 관찰한다.

6. Reproducibility

프롬프트 템플릿, 추론 파라미터, 스키마 요약 규칙, 세션 정책을 고정한다. 난수 시드와 정렬 규칙을 명시하고, 모델·드라이버·서버 버전 및 커밋 해시를 공개한다. 모든 측정은 동일 스크립트로 자동화하며, 요청·응답·이벤트 로그를 함께 공개한다.

프롬프트 템플릿 구조는 다음과 같다.

- [System Prompt] “You are a SQL expert for PostgreSQL. Use only given schema. Do not use CREATE/INSERT/UPDATE/DELETE.”
- [Schema Summary] { "tables": ["concert", "singer"], "relations": {"concert.singer_id → singer.id"} }
- [User Query] “List all singers who performed in stadiums with capacity over 50000.”

통계적 유의성은 부트스트랩 기반 신뢰구간으로 제시하고, 연속 오류율 임계치 초과나 p95 지연 급등 시 실험을 중단하고 원인을 보고한다.

Table 3. End-to-End Performance Comparison

구분	Difficulty	EM rate(%)	EX rate(%)	lat_mean(ms)	lat_p95(ms)
NL2SQL (cache)	easy	42.32	52.35	1471	2010
	medium	2.71	48.06	1541	2359
	hard	2.81	32.48	1634	2580
	extra	0	33.33	1893	3098
NL2MCP (cache)	easy	1.25	100	478	487
	medium	0	100	480	488
	hard	0	100	484	494
	extra	0	100	487	498
NL2MCP (live)	easy	1.25	100	478	486
	medium	0	100	480	489
	hard	0	100	484	493
	extra	0	100	486	494

V. Results

1. Overall Performance Comparison

실험 결과, NL2MCP(live)가 가장 높은 실행 성공률(EX rate)을 기록하였으며, NL2SQL(cache)은 상대적으로 낮은 성능을 보였다. 특히 NL2SQL(cache)의 평균 지연(lat_mean)과 p95 지연(lat_p95)은 NL2MCP보다 높은 수치로 나타나, 단순 파이프라인임에도 응답 안정성이 떨어지는 양상을 확인하였다.

NL2MCP(cache)는 도구 호출 계층을 추가하였음에도 지연이 오히려 줄어드는 경향을 보였고, live 모드에서는 카탈로그 동기화 비용이 약간 추가되지만 전체 완수율 향상 효과가 더 크게 작용하였다.

2. Performance Analysis by Difficulty Level

Spider 데이터셋의 난이도 구간별로 살펴보면 다음과 같다.

- Easy/Medium: 세 접근 모두 일정 수준 이상의 EM/EX 성능을 달성했으며, 질의 복잡도가 낮을 때는 NL2SQL(cache)도 상대적으로 경쟁력이 있었다.
- Hard/Extra: 차이가 두드러졌다. NL2SQL(cache)은 스키마 연결 실패와 조인 구문 오류로 인해 EM과 EX가 급격히 저하되었다. 반면 NL2MCP(live)는 list_schemas, list_objects 호출을 통해 최신 카탈로그를 반영하면서 오류율을 낮추었고, EX rate에서 가장 큰 격차로 우위를 보였다.

이는 스키마 복잡도가 증가할수록 NL2MCP가 적합하다는 것을 뚜렷하게 보여준다.

Table 4. Performance Comparison

Model	EM	EX	Dataset
RAT-SQL	69.7	73.6	Spider
SmBoP	72.9	75.0	Spider
PICARD	75.1	77.9	Spider
DIN-SQL	79.1	81.5	Spider
NL2SQL	42.3	52.3	Spider
NL2MCP	1.3	100	Spider

위 표는 Spider dev 동일 환경에서 기존 Text-to-SQL 모델의 EM/EX 성능을 참고한 것이며, 본 연구의 NL2MCP는 EM은 낮으나 EX(실행 성공률)는 100%로 가장 높은 안정성을 보였다. 이는 생성 모델의 품질보다는 MCP 실행 계층의 검증·정책 호출 효과에 기인한다.

3. Latency Characteristics

지연(latency) 분석에서 주목할 점은 NL2SQL(cache)이 오히려 가장 높은 평균 지연과 p95 지연을 기록했다는 점이다.

- NL2SQL(cache): LLM이 직접 SQL을 생성하므로, 문법 오류가 발생한 경우 재시도를 수행하였다. Spider dev 전체 1,034건 중 331건(약 32%)에서 1회 이상 재시도가 발생했으며, 평균 1.42회 실행 후 성공하였다. 이러한 반복 호출이 누적되어 평균·p95 지연이 증가하였다.
- NL2MCP(cache): 툴 호출 오버헤드가 존재하지만, explain_query 기반 사전 검증이 불필요한 실패 실행을 줄여 지연 분포가 안정적으로 나타났다.
- NL2MCP(live): 카탈로그 동기화 과정이 추가되지만, 평균 및 p95 지연 모두 수용 가능한 범위였으며, 높은 완수율 덕분에 실제 체감 응답 품질은 가장 우수했다.
- 질의 복잡도에 따른 지연 차이가 크지 않은 이유는 MCP 계층의 고정 오버헤드가 전체 지연의 대부분을 차지하기 때문이다. 실제 PostgreSQL 질의 실행시간은 수 밀리초 수준으로 짧았으며, MCP의 세션 생성·RPC 직렬화·이벤트 대기 지연이 주된 요인이었다. 따라서 복잡한 질의라도 프레임워크 단의 오버헤드가 이를 상쇄해 p95 지연이 일정하게 유지되었다.

결론적으로 NL2MCP가 NL2SQL보다 지연 성능이 더 안정적이고 신뢰할 수 있는 경로임을 확인하였다.

4. Error Types and Failure Factors

- 실패 사례를 유형별로 분석한 결과, 접근 방식마다 다른 양상이 나타났다.
- NL2SQL(cache): 구문 오류(괄호 누락, 잘못된 JOIN 조건)와 스키마 불일치가 다수 발생.
- NL2MCP(cache): MCP 툴 호출을 통한 사전 검증으로 단순 구문 오류가 줄었으나, 프롬프트 캐시 기반이라 스키마 변경에 취약.
- NL2MCP(live): 동적 카탈로그 동기화 덕분에 스키마 불일치 오류가 크게 감소했고, 타임아웃 발생 빈도도 낮아졌다.

즉, MCP 기반 경로는 도구 호출 계층이 오류율을 낮추는 핵심 역할을 수행했으며, 특히 live 모드에서 운영 안정성이 두드러졌다.

5. Analysis of Low EM Rate

NL2MCP(cache/live)의 EM rate가 NL2SQL(cache)보다 낮게 나온 이유는 다음과 같다.

(1) 다단계 변환 과정: LLM이 SQL을 직접 출력하지 않고 tool_calls → JSON-RPC → MCP 툴 변환을 거치면서, 정답 SQL과 문법적으로는 동일하나 문자열이 달라지는 경우가 많았다.

(2) 사전 검증 영향: explain_query 등 검증 절차가 개입되며, 불필요한 구문을 수정하거나 정규화하는 과정에서 표준 답안과 문자열 차이가 발생했다.

(3) 실행 결과 중심 최적화: NL2MCP는 실행 성공률(EX)을 높이는 데 유리하나, 이 과정에서 표준 SQL과 문자가 일치하지 않아 EM 수치가 희생되는 trade-off가 나타났다.

본 연구의 NL2MCP는 LLM이 SQL 문자열을 직접 출력하지 않고, MCP 서버가 tool_calls를 수신해 SQL을 재구성한다. 이 과정에서 JOIN 순서, 공백, 별칭 등이 표준화되어 결과는 동일하지만 문자열은 달라지므로 EM이 0%로 측정되었다. 즉, 이는 “모든 질의가 정답과 동일한 결과셋을 반환하되, 포맷이 다른 구조적 재작성”의 결과로 해석된다.

6. Comprehensive Discussion

종합하면,

- NL2SQL(cache): 단순한 구조에도 불구하고 지연이 가장 높고 오류율도 높아, 운영 환경에서의 적합성이 떨어진다.
- NL2MCP(cache): 도구 호출 기반의 사전 검증 덕분에 안정성이 높아지고 지연도 개선된다.
- NL2MCP(live): 동적 카탈로그 반영을 통해 고난도 질의에서 가장 높은 완수율과 안정성을 달성한다.

따라서 실제 운영 환경에서는 NL2MCP(live)가 가장 적합한 실행 경로로 판단되며, EM rate의 손실은 EX rate 향상과 지연 안정성 확보라는 실질적 이점으로 충분히 상쇄 가능하다.

VI. Conclusions

1. Summary of Findings

본 연구는 동일한 언어모델(GPT-4o), 동일한 데이터셋(Spider dev), 동일한 데이터베이스(PostgreSQL) 조건에서 NL2SQL(cache), NL2MCP(cache), NL2MCP(live) 세 가지 실행 경로를 공정하게 비교하였다.

실험 결과는 다음과 같이 요약된다.

- 정확도 지표: NL2MCP 경로는 EM(Exact Match)에서는 NL2SQL보다 낮게 나타났으나, EX(Execution Match)에서는 높은 완수율을 기록하였다. 이는 도구 호출 과정에서 발생하는 SQL 정규화·변환으로 인해 문자열 일치는 줄어들지만, 실제 실행 성공률은 오히려 개선된 데 기인한다.
- 지연 특성: NL2SQL은 가장 높은 평균 지연 및 p95 지연을 보였으며, NL2MCP(cache/live)는 사전 검증과 동적 동기화 덕분에 더 낮고 안정적인 지연을 유지하였다.
- 난이도별 성능: Easy/Medium에서는 접근 방식 간 차이가 크지 않았으나, Hard/Extra 구간에서는 NL2MCP(live)가 압도적으로 높은 완수율을 보여, 복잡 질의 환경에서의 우월성을 입증하였다.
- 오류 분석: NL2SQL은 구문 오류와 스키마 불일치가 주요 실패 요인이었고, NL2MCP(cache)는 캐시 의존성으로 인한 스키마 변화 취약성이 나타났다. 반면 NL2MCP(live)는 동적 카탈로그 동기화를 통해 스키마 불일치 오류와 타임아웃 발생을 크게 줄였다.

따라서, NL2MCP(live)는 실제 운영 환경에서 가장 안정적이고 신뢰할 수 있는 실행 경로임을 확인하였다.

2. Research Limitations

본 연구의 한계는 다음과 같다.

- 데이터셋 범위 제한: Spider dev 세트에 국한되어 있어, 도메인 특화 데이터셋이나 대규모 산업 데이터베이스에 대한 일반화는 추가 검증이 필요하다.
- 언어모델 단일화: GPT-4o 단일 모델을 사용했기 때문에, 다른 모델군(Qwen, LLaMA, Mistral 등)에서도 동일한 경향이 유지되는지는 확인하지 못했다.
- NL2SQL live 미포함: NL2SQL은 표준화된 동적 카탈로그 인터페이스가 부재하여 live 실험을 정의하기 어려웠다. 따라서 cache 기반 비교에 국한되었다.
- 본 연구는 GPT-4o 단일 모델을 사용하였으나, Qwen, LLaMA, Mistral 등 최근 모델들은 구조적 질의 이해 및 함수 호출 처리에서 GPT-4o와 상이한 성향을 보인다. 향후 연구에서는 동일한 MCP 도구 호출 구조를 유지한 상태에서 이러한 모델군을 교차 적용하여, 모델 구조 차이가 MCP 프레임워크 성능에 미치는 영향을 검증할 계획이다.

3. Future Research Directions

향후 연구에서는 다음과 같은 확장이 필요하다.

- 다양한 언어모델 적용: Qwen, LLaMA, Mistral 등 다른 LLM 기반에서도 NL2MCP 구조의 효과를 검증해 보아야 한다.
- 산업 데이터셋 검증: 금융, 의료, 공공 행정 등 실제 도메인 데이터셋에 적용하여 운영 환경 적합성을 평가한다.
- NL2MCP 최적화 연구: EM rate 저하를 줄이고 EX rate를 유지할 수 있는 프롬프트 전략 및 툴 호출 최적화 방법이 필요하다.
- 멀티세션·멀티모델 확장: 하나의 MCP 서버에 여러 언어모델이 동시에 접속하는 환경에서 세션 관리와 자원 할당 최적화를 탐구해야 한다.
- 거버넌스 및 보안 통합: MCP 도구 호출 로그를 활용한 권한 제어, 규제 준수, 침투 방어 등 운영 레벨의 연구로 확장 가능하다.
- Qwen, LLaMA, Mistral 기반의 NL2MCP 실험을 병행하여 모델 구조별 MCP 호환성을 비교하는 연구를 진행할 예정이다.
- NL2SQL live는 MCP와 달리 표준 도구 호출이 없으므로, 실행 전 INFORMATION_SCHEMA 질의를 통해 스키마를 수집하는 방식으로 가상 구현이 가능하나, 이 경우 세션·정책 연계가 자동화되지 않아 MCP와의 공정 비교가 어렵다. 향후 연구에서는 Qwen, LLaMA, Mistral 등 다양한 LLM을 적용하여 모델 구조별 결과 차이를 검증하고, 비-MCP 기반 환경에서도 '유사 live 갱신형 질의'의 효과를 평가할 예정이다.

REFERENCES

[1] Anthropic, "Introducing the Model Context Protocol," Sept. 2024. URL: https://www.anthropic.com/news/model-context-protocol

[2] Model Context Protocol, "Model Context Protocol — Specification (2025-06-18)," June 2025. URL: https://modelcontextprotocol.io/specification/2025-06-18

[3] Anthropic Support, "Getting Started with Local MCP Servers on Claude Desktop," 2024–2025. URL: https://support.anthropic.com/en/articles/10949351-getting-started-with-local-mcp-servers-on-claude-desktop

[4] Modelcontextprotocol, "Model Context Protocol Servers (reference implementations)," GitHub Repository, 2024–2025. URL: https://github.com/modelcontextprotocol/servers

[5] Model Context Protocol, "Introducing: The MCP Server Registry (preview)," Blog, Sept. 2025. URL: https://modelcontextprotocol.io/blog/introducing-the-mcp-server-registry-preview

[6] Zed Industries, "The Context Outside the Code," Zed Blog, Nov. 2024. URL: https://zed.dev/blog/mcp

[7] Cursor Team, "Model Context Protocol (MCP)," Cursor Docs, 2025. URL: https://docs.cursor.com/context/model-context-protocol

[8] Model Context Protocol, "Key Changes — MCP Changelog (2025-06-18)," June 2025. URL: https://modelcontextprotocol.io/specification/2025-06-18/changelog

[9] X. V. Lin, R. Socher, and C. Xiong, "Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing," Findings of EMNLP, pp. 4870–4888, Nov. 2020. DOI: 10.18653/v1/2020.findings-emnlp.438

[10] B. Wang, R. Shin, X. V. Lin, O. Polozov, and M. Richardson, "RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers," Proc. ACL, pp. 7567–7578, July 2020. DOI: 10.18653/v1/2020.acl-main.677

[11] J. Guo, Z. Liu, Y. Lu, S. Ren, T. Yu, S. Su, and J. Sun, "Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation," Proc. ACL, pp. 4524–4535, July 2019. DOI: 10.18653/v1/P19-1444

[12] O. Rubin and J. Berant, "SmBoP: Semi-Autoregressive Bottom-Up Semantic Parsing," Proc. NAACL, pp. 311–324, June 2021. DOI: 10.18653/v1/2021.naacl-main.29

[13] T. Scholak, N. Schucher, and C. Bahdanau, "PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding for Text-to-SQL," Proc. EMNLP, pp. 5446–5460, Nov. 2021. DOI: 10.18653/v1/2021.emnlp-main.779

[14] M. Pourreza and D. Rafiei, "DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction," Proc. NeurIPS, Dec. 2023, 10 pp. DOI: 10.48550/arXiv.2304.11015

[15] R. Cao, L. Chen, Z. Chen, Y. Zhao, S. Zhu, and K. Yu, "LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations," Proc. ACL (Long Papers), pp. 2541–2555, Aug. 2021. DOI: 10.18653/v1/2021.acl-long.198

[16] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task," Proc. EMNLP, pp. 3911–3921, Nov. 2018. DOI: 10.18653/v1/D18-1425

[17] T. Yu, R. Zhang, M. Yasunaga, Y. C. Tan, X. V. Lin, S. Li, H. Er, I. Li, B. Pang, T. Chen, et al., "SParC: Cross-Domain Semantic Parsing in Context," Proc. ACL, pp. 4511–4523, July 2019. DOI: 10.18653/v1/P19-1443

[18] T. Yu, C. Zhang, A. Er, J. Shim, J. Xue, J. Li, and D. Radev, "CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases," Proc.

EMNLP, pp. 1962–1979, Nov. 2019. DOI: 10.18653/v1/D19-1204

[19] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing Reasoning and Acting in Language Models," Proc. ICLR, May 2023. arXiv:2210.03629.

[20] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language Models Can Teach Themselves to Use Tools," Proc. ICLR, May 2023. arXiv:2302.04761.

Authors

Won-Bae Kim received the B.S. degree in Electronic Engineering from the Seoul National University of Science and Technology, Seoul, Korea, in 1998, and the M.S. degree in Electronic and Communication Engineering from Hanyang University, Seoul, Korea, in 2005. He is a Ph.D. candidate in Convergence AI Engineering at Hoseo University, Korea. Since 2023. He has joined the CTO of Data Science Lab, LTD., a venture company specializing in AI and Big Data analytics. Him research interests include Model Context Protocal(MCP), Agent-based generative AI systems, Business Intelligence, and AI policy and strategy.

Nammee Moon received B.S., M.S., and ph.D degrees in School of Computer Science and Engineering from Ewha Womans University in 1985, 1987 and 1998, respectively. She served as an assistant professor at Ewha Womans University from 1999 to 2003. From 2003 to 2008, she is a professor of Department Digital Media, Graduate School of Seoul Venture Information. Since 2008, she is currently a professor in the Department of Computer Science and Engineering, Hoseo University. she is current research interests include Social Learning, HCI and User Centric Data, Big-data Processing and Analysis.