MAC-SQL

기존 LLM 기반 Text-to-SQL 방법은 **성능 저하**를 겪음
✅ 제안 방법 (MAC-SQL)

- **다중 에이전트 기반 프레임워크** 구성:
  1. **Decomposer Agent**:
     - few-shot chain-of-thought로 SQL 쿼리를 생성
  2. **Auxiliary Agents (2명)**:
     - 필요한 경우 호출되어 하위 데이터베이스 추출 및 오류 쿼리 수정 수행
     - 외부 도구나 모델 사용 가능
     - 기능 확장이 쉬움
- **GPT-4**로 프레임워크의 **상한 성능** 측정
- **CodeLlama-7B**를 활용해 **SQL-Llama** 모델을 파인튜닝하여 오픈소스 대안 제시

✅ 성능

- SQL-Llama: 실행 정확도 **43.94**
- GPT-4: 실행 정확도 **46.35**
- MAC-SQL + GPT-4: BIRD 벤치마크에서 **59.59**, **SOTA 성능 달성**

---

1. Introduction

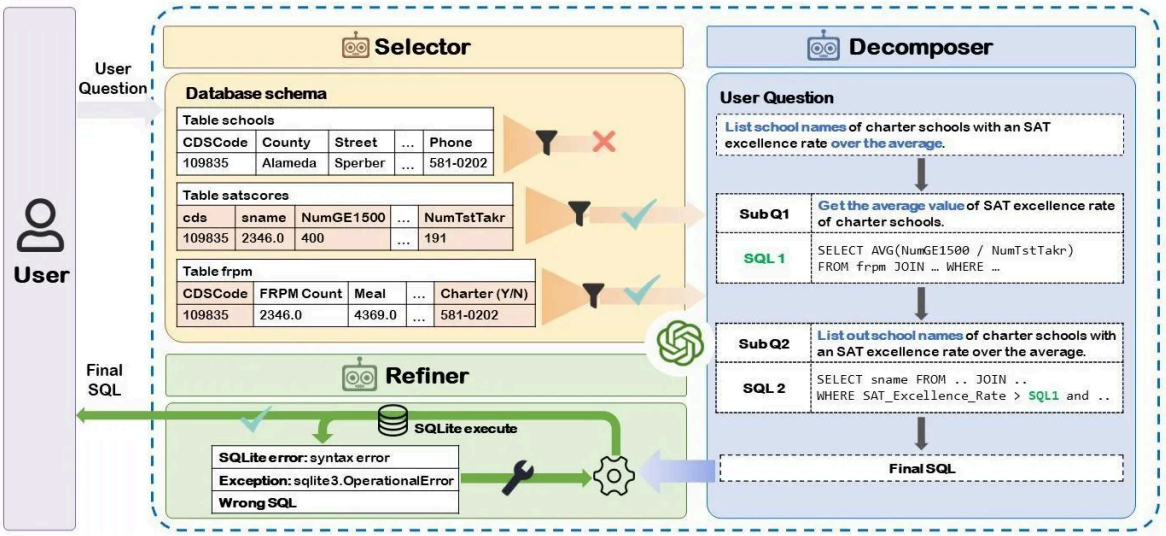
a. 초기 단계에서는 입력 시퀀스를 사전학습(pre-trained) 모델을 사용해 인코딩, SQL 쿼리는 추상 구문 트리 또는 사전 정의된 스케치를 통해 디코딩하는 방식이 주로 사용됨.

b. 시퀀스-투-시퀀스(sequence-to-sequence) 방식이 주류로 자리잡음.

c. 최근의 LLM 기반 Text-to-SQL 연구들은 대부분 In-Context Learning 프롬프트 전략이나, 대상 도메인으로부터 수집한 데이터에 대한 지도학습(supervised fine-tuning)에 집중되어 있음.

⇒ “대규모 데이터베이스”나 복잡하고 다단계 추론을 요구하는 사용자 질문에서 심각한 성능 저하를 겪는 경우가 많음.

⭐이러한 문제를 해결하기 위해 우리는 MAC-SQL이라는 새로운 LLM 기반 다중 에이전트 협력 프레임워크를 제안



MAC-SQL 요약

: 다양한 기능을 갖춘 지능형 에이전트로서 LLM을 활용하여 효과적인 Text-to-SQL 파싱을 수행

핵심 에이전트인 **Decomposer**를 중심으로

- 복잡한 질문을 더 단순한 하위 질문들로 분해한 후, chain-of-thought 방식으로 순차적으로 해결

두 개의 보조 에이전트인 **Selector**와 **Refiner**가 함께 작동

- Selector : 방대한 데이터베이스를 더 작은 하위 데이터베이스로 분해하여 불필요한 정보를 제거
- Refiner : 외부 도구를 활용해 SQL을 실행하고, 그 결과를 바탕으로 오류를 정정

CodeLlama 7B를 기반으로 한 instruction-following 모델인
SQL-Llama를 fine-tuning

⇒ MAC-SQL에서 사용되는 에이전트의 지시 데이터를 학습시킴으로써 데이터베이스 단순화, 질문 분해, SQL 생성, SQL 수정 등의 기능을 수행할 수 있도록 함.

MAC-SQL 개요

Decomposer는 Text-to-SQL 쿼리를 생성하는 핵심 역할

Selector는 불필요한 스키마 정보를 제거하여 데이터베이스를 단순화

Refiner는 외부 도구를 사용하여 SQL 실행 결과를 확인하고 잘못된 쿼리를 정제

💡 *Algorithm 1*

Algorithm 1 The algorithm of MAC-SQL

Input: question q, database db, knowledge kg
Output: sql

```
if need simplify to database then
    db = LLM_Selector(q, db, kg)
end if
dbDesc = getDbRepresenation(db, kg)
subQs, subSQLs = LLM_Decomposer(q, dbDesc)
sql = subSQLs[-1]
count = 0
while count < maxTryTimes do
    ok, err = executeAndAnalyze(sql, db)
    if ok then
        return sql
    else
        sql = LLM_Refiner(q, dbDesc, sql, err)
    end if
end while
return sql
```

`질문 q` , `데이터베이스 db` , `외부 지식 kg`

데이터베이스가 복잡하다고 판단되면, `Selector` 에이전트를 통해 간소화된 데이터베이스로 변환

⇒ 데이터베이스의 구조 표현을 얻고(`dbDesc`)

⇒ 질문을 분해하여 하위 질문들과 해당 SQL 쿼리들을 생성

⇒ 이 중 마지막 하위 쿼리를 최종 SQL로 가정하고 실행을 시도

⇒ 쿼리가 성공적으로 실행되면 그대로 반환하고, 그렇지 않다면 Refiner를 호출하여 오류를 반영해 쿼리를 수정

⭐ 이 과정을 최대 시도 횟수만큼 반복하며, 최종적으로 SQL 쿼리를 반환

Selector

Selector를 설계한 주된 동기:

1. 불필요한 스키마 항목이 너무 많이 프롬프트에 포함되면, 모델이 결과 쿼리에서도 불필요한 스키마 항목을 포함할 가능성이 높아짐
2. 전체 데이터베이스 스키마를 그대로 사용하면 프롬프트 길이가 과도하게 길어져 API 비용이 증가하거나, LLM의 최대 컨텍스트 길이를 초과할 수 있음.

데이터베이스 스키마의 길이가 일정 임계값을 초과할 경우에만 작동하고, 그렇지 않은 경우에는 원래의 전체 스키마 S를 그대로 사용

Decomposer

Decomposer 프롬프트 방식

1. **Chain-of-Thought (CoT) prompting**: 전체 질문을 **한 번의 추론 흐름으로** 단계적으로 나눠서 **하위 질문들과 그에 대응되는 SQL**을 생성
2. **Least-to-Most prompting**: 쉬운 질문부터 시작해 점차 복잡한 질문으로 확장 (계산 비용 높음)

Refiner

모델 스스로 오류를 검출하고 수정하는 능력을 갖추게 하여, 전체 프레임워크의 오류 허용성(fault tolerance)과 정확도(accuracy)를 향상시키는 것