DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

NL2SQL using LLM Agents, DB Execution Feedback, RAG/TAG, AQP, and Finetuning (Large Umbrella Project)

Keywords: LLM, agent, query execution, schema linking, RAG, TAG, AQP, finetuning, GRPO, curriculum learning, tabular foundation models

Problem: NL2SQL, the problem of automatically translating natural language (NL) questions into SQLs that run on database systems, has become more and more 1) 'performant' with the advance of LLMs, agentic frameworks, reasoning skills, and finetuning methods, and 2) 'important' as real-world, production SQL queries become extremely complex (spanning over 100 tables [1]) and only expert DBAs can write such queries. Industries, therefore, are actively trying to offer an NL2SQL interface on top of their data systems to allow non-DBAs to easily query their data.

Popular NL2SQL benchmarks such as BIRD (https://bird-bench.github.io) enable comparison of various NL2SQL methods and development set to evaluate one's own method. Evaluation on the test set is based on the execution accuracy (EX), comparing the "output tables" of generated SQLs to the ground truths, as this is more reliable than comparing SQLs; an NL question can be translated into multiple different SQLs with the exact same meaning. As NL2SQL is a highly active area, you can see new methods released every month.

Still, we believe that there's a large room for improvement. While human experts achieve 93% of EX, the rank #1 method as of December 10th, gives 82%. Furthermore, the benchmark does not contain very large or complex queries in production scenarios, and this is why new advanced benchmarks are kept introduced.

Developing a new, performant NL2SQL method as our final goal, we are seeking to apply various proven and unproven techniques to this problem.

1. LLM Agents and Multi-Path Reasoning

The scaling laws of LLMs have moved from pretraining to finetuning to inference phases. That is, people have first increased the data and model size in pretraining, then focused on finetuning with high-quality data, then moved to increase the reasoning steps backed by the test-time scaling laws. OpenAI and DeepSeek have proven its power over a single LLM inference. Agents and tool-use allow LLMs interact with environments and get unsupervised feedback. The pipelines around LLMs are not static (predetermined) but dynamically change based on the feedback. Here, a common misunderstanding is that chain-of-thought is an instance of multi-step reasoning, but its first appearance was for a single LLM inference, specifying that the LLM generates a "single" output text that involves "multiple" sub-steps to generate that output. But this doesn't mean that the LLM is actually called multiple steps (each step based on previous steps' outputs) as in multi-step reasoning. Chain-of-thought "reasoning" is a multi-step reasoning. Multi-"path" reasoning [2] transforms a simple path-like reasoning steps into a tree shape. It allows going back to the previous step if the current step seems a failure, and makes a different attempt as a new branch.

Due to its effectiveness, multi-step reasoning has become a prevalent technique in NL2SQL task. CHASE-SQL [3] uses "divide-and-conquer" approach to divide the problem into easier subtasks, which well aligns with the idea of multi-step reasoning. Still, it's not fully adaptive in the sense that each subtask is solved independent to each other. Defining a general & performant agentic framework, with clear definitions of agents, rewards, states, and actions, is still an important and interesting challenge.

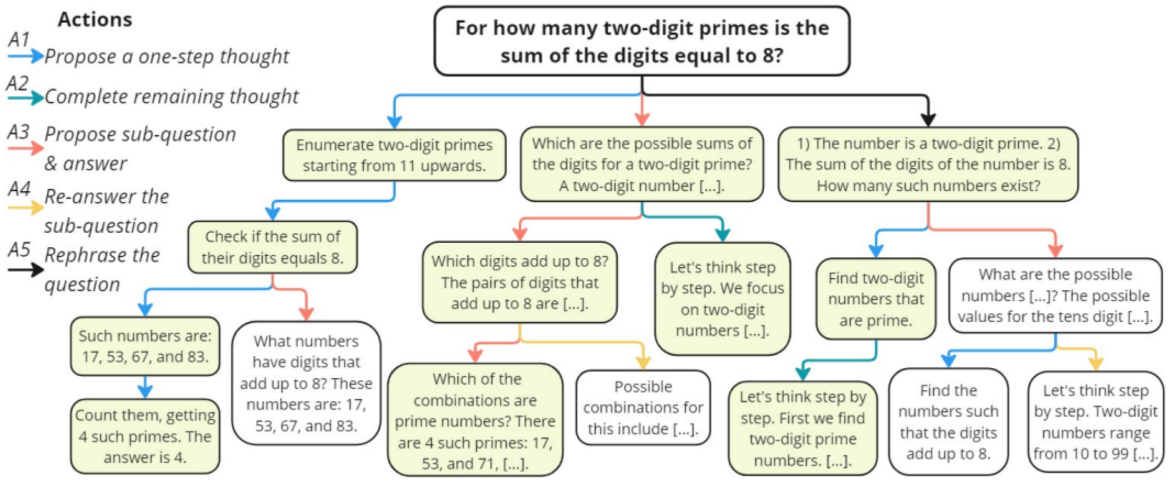
Actions
A1 Propose a one-step thought
A2 Complete remaining thought
A3 Propose sub-question & answer
A4 Re-answer the sub-question
A5 Rephrase the question
For how many two-digit primes is the sum of the digits equal to 8?
Enumerate two-digit primes starting from 11 upwards.
Which are the possible sums of the digits for a two-digit prime? A two-digit number [...].
1) The number is a two-digit prime. 2) The sum of the digits of the number is 8. How many such numbers exist?
Check if the sum of their digits equals 8.
Which digits add up to 8? The pairs of digits that add up to 8 are [...].
Let's think step by step. We focus on two-digit numbers [...].
Find two-digit numbers that are prime.
What are the possible numbers [...]? The possible values for the tens digit [...].
Such numbers are: 17, 53, 67, and 83.
What numbers have digits that add up to 8? These numbers are: 17, 53, 67, and 83.
Which of the combinations are prime numbers? There are 4 such primes: 17, 53, and 71, [...].
Possible combinations for this include [...].
Let's think step by step. First we find two-digit prime numbers. [...].
Find the numbers such that the digits add up to 8.
Let's think step by step. Two-digit numbers range from 10 to 99 [...].
Count them, getting 4 such primes. The answer is 4.

Figure from [2] illustrating multi-path reasoning. Given an NL question, we can apply the same approach with well-defined actions, rewards, and states, to generate candidate SQLs.

2. DB Execution Feedback

Unlike typical LLM-based tasks that require supervised human feedback, NL2SQL allows to use powerful feedback from databases. That is, once a candidate SQL is generated, we can run this SQL over the database and check for 1) syntactic errors (SQL has wrong syntax and cannot be run at all) and 2) semantic errors (SQL has a different meaning from the initial NL question). Of course, we cannot use the ground truth SQL or table to identify those errors but solely rely on the database execution results. Execution-based verification has been an effective component in NL2SQL methods, and how to define an error taxonomy is a critical problem [3, 4].

3. Retrieval-/Table-Augmented Generation (RAG/TAG) and Foundation Models (FMs)

RAG is also a popular technique to make LLMs more trustworthy, by referring to the actual (fragmented) piece of information. Typically, reference documents are chunked, embedded into vectors, stored in the vector database systems, and later similar chunk-vectors and their original chunks are retrieved for an (embedded) NL question. The retrieved chunks are added to the prompt. However, in NL2SQL how should we feed "tabular" data, that are obtained from SQL executions or input databases? While papers embed textual table and column names (i.e., schema information) into vectors and fetch relevant table/column names for an NL question using RAG, with an effective M-schema [5] being more "textual" and human-understandable than the schema written in DDL (data description language), feeding all "attribute values" inside a table can add significant space/search overheads with undesired data noises. We could then follow the principle of TAG (table-augmented generation [6]) to feed the tabular data into LLMs, which was originally proposed to table question answering, slightly different from NL2SQL. TAG can be implemented with tabular foundation models, which are trained over raw tables instead of texts. Foundation models can be an efficient tool (smaller sizes than LLMs) to retrieve relevant data inside the tables using the attention mechanism. How we feed those back to LLMs, in forms of embeddings or raw texts, is a decision to make.

4. Approximate Query Processing (AQP)

AQP is a pure database concept to achieve higher efficiency while giving approximate answers to SQL queries. This is suitable for NL2SQL as LLMs cannot take large data into their context and few examples and partial answers are enough to verify the generated SQLs. The easiest example of AQP is table sampling.

5. Finetuning (GRPO)

So far, we've focused on the LLM inference phase only. But recently (date: December 10) Q-SQL [7] has proven that effectively finetuning a small model (3B) can actually be comparable to using large models. Their core ideas lie in GRPO [8], DAPO with curriculum learning (training from simple to complex queries), GSPO, process reward (intermediate reward for the correctness of schema linking [9]), episodic memory (gather error traces, cluster them and fetch relevant clusters with examples for similar future queries), MoE, self-consistency with many candidates, and varying prompting styles. They emphasize that their finetuning took less than 24 hours! Note that they do not (or merely) use supervised finetuning but rely on GRPO-style unsupervised, RL-based feedback. It'd be an interesting and impactful milestone to implement and reproduce Q-SQL. For implementation of RL-finetuning, we can use HuggingFace TRL, Unsloth, or ModelScope Swift with vLLM backend for efficient inference.

Finetuning can be an effective way to fully utilize the given development set of ground truth SQL queries paired with NL questions, as we cannot give all pairs in the LLM context in the inference phase; it can only be few-shot. Finetuning can embed those pairs and seek for the policy optimized to find ground truth-like SQLs. But of course, inference and finetuning are interleaved; good inference patterns and multi-step reasoning can be and should be embedded into finetuning, so the finetuned LLM can do a better job in the inference phase.

We kindly note that this is a large umbrella project involving many 3-6 month semester projects, and are looking for highly motivated students with background in specific part of the project (e.g., finetuning, RAG, SQL execution as a tool-use of LLMs). NL2SQL is indeed a highly active and challenging area powered by the advance of LLMs [10, 11].

[1] Workload Insights From The Snowflake Data Cloud: What Do Production Analytic Queries Really Look Like?

[2] Mutual Reasoning Makes Smaller LLMs Stronger Problem-Solvers
[3] CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL
[4] The Power of Constraints in Natural Language to SQL
[5] Automatic Database Description Generation for Text-to-SQL
[6] Text2SQL is Not Enough: Unifying AI and Databases with TAG
[7] Q-SQL
[8] DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models
[9] Relation-aware schema encoding and linking for Text-to-SQL
[10] The Dawn of Natural Language to SQL: Are We Fully Ready?
[11] A Survey of NL2SQL with Large Language Models

Project: In this project, the student(s) will 1) get to know about the NL2SQL problem and related work, 2) identify the feasible research direction among the ones mentioned above (which can vary per semester and as we gain more knowledge, expertise, and preliminary results) based on the student's interest and background, 3) implement specific ideas and compare the performance (in terms of EX or other finer-grained metrics) with the baselines. Students can implement close-sourced baselines (e.g., Q-SQL), analyze baseline's success and failure cases, as well as our ideas, ultimately towards building a stable & effective end-to-end NL2SQL framework.

Plan:

1. (Pre-semester) Get to know the NL2SQL problem and understand key related work.
2. (Pre-semester) Pick a research direction based on the student's background, conduct a survey of related work on that direction.
3. Select key baselines.
4. Identify the bottlenecks in the baselines (case analysis – failures and successes) and set these as the research challenge. Both quantitative and qualitative analyses are required.
5. Apply the above techniques to mitigate the bottlenecks.

Supervisor: Prof. Anastasia Ailamaki, anastasia.ailamaki@epfl.ch
Responsible collaborator(s): Kyoungmin Kim, kyoung-min.kim@epfl.ch

Duration: 3-6 months