EXECUTION-AWARE SELF-REFINING TEXT-TO-SQL GENERATION USING LARGE LANGUAGE MODELS

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE OF

MASTER OF TECHNOLOGY
IN
ARTIFICIAL INTELLIGENCE

Submitted by

ANKIT SHARMA

(24/AFI/05)

Under the supervision of

PROF. VINOD KUMAR

Department of Computer Science and Engineering

To the

Department of Computer Science and Engineering

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi 110042

JUNE, 2026

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, **Ankit Sharma, Roll No.2K24/AFI/05** student of M.Tech (Artificial Intelligence), hereby certify that the work which is being presented in the thesis entitled **"Execution-Aware Self-Refining Text-to-SQL Generation using Large Language Models"** in partial fulfillment of the requirements for the award of the Degree of Master of Technology in Artificial Intelligence in the Department of Computer Science and Engineering, Delhi Technological University is an authentic record of my own work carried out during the period from August 2024 to June 2026 under the supervision of **Prof. Vinod Kumar**. The matter presented in the thesis has not been submitted by me for the award of any other degree of this or any other Institute.

Date: 31.05.2026 Ankit Sharma

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Bawana Road, Delhi-110042

CERTIFICATE

It is hereby certified that **Ankit Sharma (Roll No. 2K24/AFI/05)** has performed the research work described in the thesis entitled **"Execution-Aware Self-Refining Text-to-SQL Generation Using Large Language Models"** for the purpose of obtaining the degree of Master of Technology from the Department of Computer Science and Engineering, Delhi Technological University, New Delhi under my supervision. The thesis includes the results of original research carried out by the student himself, and the material of the thesis does not form the basis for any other award of degree elsewhere from any other University / Institution.

Place: Delhi **Prof. Vinod Kumar**

Date: 31.05.2026 **SUPERVISOR**

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Bawana Road, Delhi-110042

ACKNOWLEDGMENTS

I would like to express my deep gratitude to my project guide **Prof. Vinod Kumar**, Professor, Department of Computer Science and Engineering, Delhi Technological University, for his guidance with unsurpassed knowledge and immense encouragement. I am also grateful to **Prof. Anil Singh Parihar**, Head of the Department, Computer Science and Engineering, for providing us with the required facilities for the completion of the thesis. I would like to thank my parents, friends, and classmates for their encouragement throughout this period.

Date: 31.05.2026 Ankit Sharma

Abstract

The increasing need for data-driven systems has resulted in the growth of user-friendly database interaction methods, especially for non-technical users. Natural Language to SQL, also known as Text-to-SQL is hence required for converting natural language queries from users into executable SQL statements, allowing users to fetch data from databases more efficiently. Although, it is a challenging task to generate accurate SQL queries due to the complexity of natural language understanding, schema linking and variety of database structures. Recent progress in Large Language Models (LLMs) have significantly improved the traditional Text-to-SQL methods by enhancing their semantic understanding for generalized SQL generation. This thesis proposes an Execution-Aware Self-Refining Text-to-SQL Generation framework using Large Language Models, where generated SQL queries are refined iteratively through execution feedback and context awareness. This framework integrates prompt engineering, RAG based architectures and execution validation to identify syntax errors, logical disparities and failed query results. The study also presents an overview of benchmark Text-to-SQL datasets such as Spider, WikiSQL, BIRD, and CoSQL representing that execution-aware self-refinement enhances SQL generation performance compared to conventional approaches. Moreover, it highlights challenges related to scalability, explainability and computational efficiency towards making a domain-centric Text-to-SQL systems for a broad range of technical and non-technical users.

Contents

List of Tables

List of Figures

List of Symbols

DL	Deep Learning
LLM	Large Language Model
AI	Artificial Intelligence
ML	Machine Learning
DNN	Deep Neural Network
PLM	Pre-trained Language Model
NLP	Natural Language Processing
SQL	Structured Query Language
NL	Natural Language
$LSTM$	Structured Query Language
$BERT$	Bidirectional Encoder Representations from Transformers

Chapter 1

INTRODUCTION

1.1 Background Overview

Natural Language to SQL (NL-to-SQL) also known as Text-to-SQL is a study focused on conversion of natural language queries into executable SQL result statements. With ever increasing use of structured databases across the sectors, there is a growing demand for these smart systems that allow users to fetch data without getting into the expertise of SQL or database management systems. Where traditional database query methods needed users to learn relational schema, table joins and SQL syntax, making database interaction more challenging especially for general users.

Traditional Text-to-SQL systems heavily relied towards rules-based models and pre-trained language approaches. Although these systems worked very efficiently in controlled environments, they simply lacked the scalability and ability to generalize the SQL statements for a variety of databases and complex SQL queries. Later, deep learning models and transformer-based architectures enhanced the capability of these systems to have semantic understanding between the natural language and SQL statements. Pre-trained Language Models (PLMs) such as BERT, Llama further refined the contextual understanding and schema linking within the databases.

In recent times, Large Language Models (LLMs) such as GPT, Llama, and BERT have significantly enhanced the functionality of Text-to-SQL systems because of their enhanced analytical capability in generating SQL queries. LLMs can generate SQL query statements with enhanced semantic understanding and domain-centric capability. Most mod-

ern LLM-based Text-to-SQL systems have demonstrated robust performance on benchmark datasets such as Spider, WikiSQL, BIRD, CoSQL. However, there still remains a problem in producing reliable, robust and valid SQL queries for complex database schema.

1.2 Identification of the Problem

Regardless of the recent progress in the areas of LLM-based Text-to-SQL systems, there exists several constraints in real-world database environments. One of the biggest challenges is that of generated SQL queries can show as syntactically correct but they fail during their execution due to incorrect joins, invalid database schema, and missing conditions. These systems only focus on query generation without properly checking whether the generated query actually executes successfully on the database.

Another big issue is the absence of iterative refinement in Text-to-SQL frameworks due to which approximately all these traditional systems generate SQL queries within a single iteration only and are not able to utilize the execution feedback mechanism to correct logical errors or improve their query accuracies. This is a very major shortcoming for complex multi-table databases with nested SQL queries, and complex natural language inputs from user. Also, semantic understanding and domain-centric capability is an issue for these systems.

Hence, there is a requirement for an execution-aware and self-refining Text-to-SQL framework that can validate generated SQL queries automatically using execution feedback and iteratively refine invalid SQL queries to improve their reliability and accuracy.

1.3 LLM Based Text-to-SQL Framework

This proposed framework utilizes these Large Language Models (LLMs) for creating SQL queries from natural language prompts provided by user while ensuring execution-aware validation and self-refinement in these systems. This framework begins when user supplies a natural language prompt to system, which is then used to extract relevant relevant schema information and contextual data to the database schema.

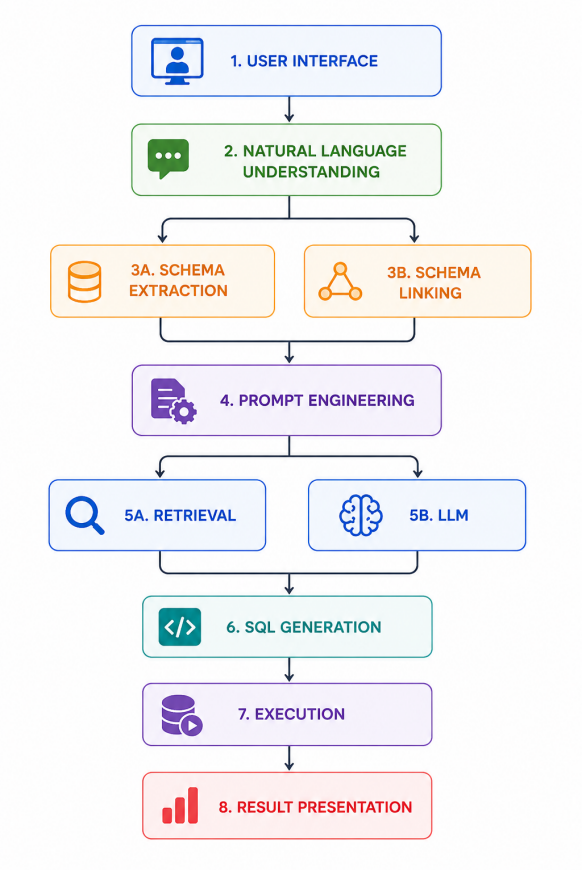

Figure 1.1: Text-to-SQL Framework

To improve the sematic understanding of the schema, Retrieval-Augmented Generation (RAG) based methods are applied on the framework. It uses relevant schema data, sample SQL queries and database metadata to retrieve and provide input to the LLM as contextual prompts. These LLMs then uses the prompt data for generating an initial SQL query.

Unlike conventional Text-to-SQL systems, this generated initial query is then executed on targeted database for execution validation. If error occurs or incorrect outputs are detected on applying the initial SQL query, then feedback from execution feedback is analyzed and provided back to the LLM for iterative query refinement. This self-refinement is an iterative process that continues until a valid and semantically accurate SQL is generated.

The integration of semantic-aware prompts, execution validation and iterative refinement of SQL queries improves the overall accuracy and reliability of the Text-to-SQL pipeline and reduces execution errors across a variety of database schema.

1.4 Motivation

The rationale for this study comes with ever increasing requirement for smart database query systems that have the ability to assist users without having an extensive knowledge of SQL systems. Although recent LLM-based systems have improved SQL generation capabilities, they still face challenges with execution errors, schema validity and complex query generation.

Current Text-to-SQL systems generally lack techniques that can validate the generated queries using executable results due to which many generated SQL queries might fail during runtime or generate semantically invalid results. Applying these systems on real-world applications like healthcare, finance, education can result in incorrect decision making and inefficient data retrieval.

The growing importance of these Large Language Models provides an opportunity to create more innovative and proficient Text-to-SQL systems. By integration of execution-aware validation and self-refinement techniques, the proposed framework can generate semantically accurate SQL query results. It can also improve the overall usability of the database systems for natural language interaction for both professional and non-professional users.

1.5 Objectives

The primary aims of this study are:

- To identify shortcomings in the current LLM-based Text-to-SQL frameworks due to execution-based runtime errors, schema linking, complex queries.
- To analyze the proposed framework on benchmark datasets including Spider, WikiSQL, BIRD and evaluate their performance metrics like execution and exact match accuracy.
- To study existing Text-to-SQL frameworks and evaluate role of LLM-based systems in SQL query generation.

- To create execution-aware Text-to-SQL framework analyzing intiated SQL queries of invalid or inaccurate results.
- To create an execution-aware Text-to-SQL framework validating initiated SQL query results with the database execution feedback.
- To improve the accuracy, reliability and usability of the natural language prompt with the Large Language Models.
- To integrate retrieval-augmented architecture based systems for enhanced semantic and domain-centric understanding.

Chapter 2

LITERATURE REVIEW

Text-to-SQL frameworks focus on translating natural language queries to executable SQL commands that permit users to perform interaction with the databases with help of natural language instead of having extensive SQL knowledge. Text-to-SQL frameworks have evolved over the years from rule-based systems to large language model based frameworks. This development has further led to transformer-based frameworks, pre-trained language frameworks and retrieval-augmented based frameworks resulting in enhanced query generation and semantic understanding. However, there are certain challenges in areas like execution errors, complex queries, schema linking and domain-centric capability.

This chapter provides a literature review of these major developments in Text-to-SQL frameworks beginning from traditional systems, deep learning based systems, trained language models, LLM-based frameworks and the benchmark datasets commonly used for evaluation.

2.1 Evolution of Text-to-SQL Approaches Before LLMs

Earlier, Text-to-SQL frameworks heavily relied on rules-based methods for creating SQL statements from natural language queries. These systems used basic grammatical rules, heuristic data to generate SQL queries. These systems were efficient in constrained environments having fixed database schema and limit amount of query patterns, due to which they lacked scalability and adaptability for real-world applications.

They faced several shortcomings such as challenges in handling complex queries and poor

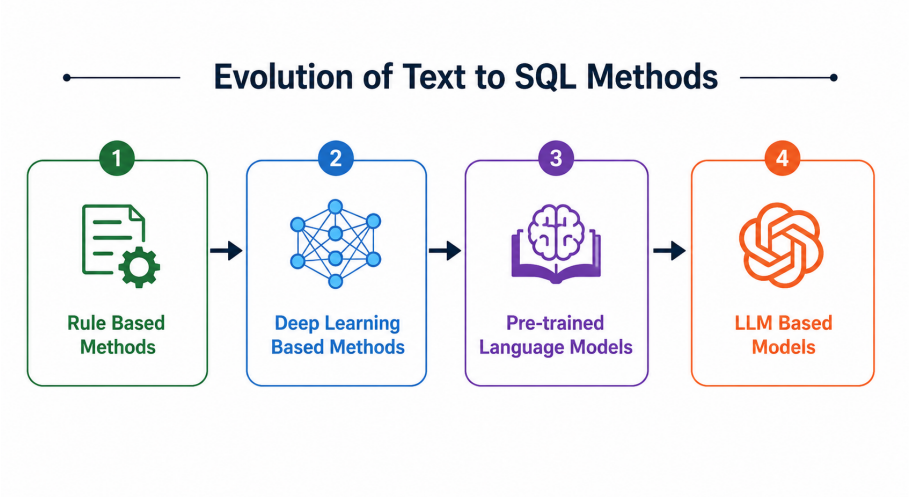

Figure 2.1: Evolution of Text-to-SQL Methods

generalization capability across domain-centric schema. They also suffered challenges in complex SQL operations like nested queries and multi-table joins. The main reason for these limitations was due to the need of manually creating rules for each schema and domain which costed a lot of time and effort.

Later on, as semantic understanding techniques came to existence, it naturally improved the association between natural language and SQL statements. These methods represented the intermediate logic schema before generating the SQL queries. Although these systems improves the flexibility but their performance was still limited for complex and large datasets.

Following the introduction of neural networks, it lead to significant improvement in Text-to-SQL systems. Sequence-to-Sequence (Seq2Seq) frameworks allowed generation of SQL statements directly by mapping of natural language input with SQL queries. With models like Seq2SQL and SQLNet, they have shown enhanced performance by allowing semantic understanding from training data. Despite all these advancements, they still struggle with execution consistency, and generalization issues across variety of databases.

2.2 Deep Learning Techniques

Following the introduction of deep learning methods for Text-to-SQL frameworks significantly enhanced their performance. These approaches used Long Short-Term Memory

(LSTM), encoder-decoder architectures, and attention mechanisms to map semantic relationships between natural language user prompts and executable SQL commands.

Seq2SQL was among first deep learning models used for Text-to-SQL tasks. It has enforced reinforcement learning and sequence generation techniques to enhance the execution accuracy on the WikiSQL dataset. SQLNet later improved upon the existing Seq2SQL model by reducing its dependency on reinforcement learning and instead introducing sketch-based query generation methods.

Simultaneously, SyntaxSQLNet was used for complex SQL queries for nested operations and multi-table joins as it allowed syntax-aware methods to generate hierarchical SQL structures. TypeSQL later improved query generation by allowing type-aware schema mapping techniques for semantic understanding.

Later on, transformer-based architectures substituted traditional models due to their superior contextual awareness. Attention mechanisms allows these models to capture the semantic understanding between the queries and database schema, These techniques enhanced the creation of complex SQL queries and large database tables. Although deep learning significantly improved the performance of query generation, they still face restrictions in contextual understanding and generalization across different database schema,

2.3 Pre-trained Language Models (PLMs)

Pre-trained Language Models (PLMs) later emerged as a major advancement in Text-to-SQL frameworks. Models like BERT, RoBERTa, T5 are essentially pre-trained language models trained on large text corpus which are then further finetuned for SQL query generation for large database schema.

PLMs significantly enhanced the semantic and contextual understanding in comparison to traditional deep learning models. The ability of these models to capture the semantic understanding allowed more accurate association between natural language and database schema.

BERT-based Text-to-SQL systems allows schema-aware encoding methods to enhance table mapping and column selection. T5-based architectures treated Text-to-SQL systems

as a text generation tool representing strong performance on benchmark datasets. RAT-SQL later introduced relation-aware schema with graph-based attention mechanisms to enhance semantic understanding and schema linking.

These pre-trained language models enhanced the exact match and execution accuracy of datasets like Spider and WikiSQL. Although they still need extensive fine-tuning and large database schema for efficient performance as they struggle with complex queries and schema.

2.4 Text-to-SQL Techniques Using Large Language Models (LLMs)

Following the introduction of large language models like GPT, Llama, and Gemini, the functionality and semantic understanding of Text-to-SQL frameworks have been significantly upgraded. Since LLMs are extensively pre-trained on large volumes of textual and code data, they can reason effectively, understand context, and generate code.

Unlike traditional PLM-based approaches, LLMs are capable of performing Text-to-SQL via prompt engineering, in-context learning, and few-shot learning without any specific task-based fine-tuning. The latest LLM-based approaches not only provide state-of-the-art performance but also demonstrate higher domain transfer capabilities on benchmark datasets.

A few sophisticated frameworks based on LLMs have been proposed recently. For example, DIN-SQL uses the decomposition approach for query generation to increase reasoning capabilities. DAIL-SQL makes use of in-context learning and demo selection techniques to enhance query generation accuracy. C3 performs ChatGPT-based zero-shot generation with high execution performance on the Spider dataset.

The retrieval-augmented generation (RAG) paradigm has also been applied for Text-to-SQL. In RAG-based approaches, schema knowledge, query samples, and contextual DB information are retrieved from a database to support query generation by an LLM, helping in improving schema grounding and reducing hallucination.

Despite advancements, however, current state-of-the-art LLMs still face some problems like:

- Generation of erroneous SQL queries
- Execution errors
- Ambiguous schemas
- Poor nested query handling capabilities
- Absence of iterative refinement mechanism.

2.5 Benchmark Datasets for Text-to-SQL Systems

1. **WikiSQL :** WikiSQL is among the earliest large-scale datasets for Text-to-SQL consisting of more than 80,000 natural language utterances along with their respective SQL queries taken from Wikipedia tables. This dataset focuses primarily on simpler single-table queries and has been extensively used in earlier Text-to-SQL schema operations.

2. **Spider :** Spider is another huge multi-domain benchmark dataset with complex SQL queries involving many different databases and schemas. As opposed to WikiSQL, Spider includes SQL operations such as multi-table joins, nested subqueries, etc., hence making it among the hardest and most commonly used benchmarks for current Text-to-SQL operations.

3. **CoSQL :** CoSQL is a dialogue-based large dataset of Text-to-SQL systems testing the model's capability in conversing about database queries. This dataset consists of thousands of conversational query sequences where the system's ability to preserve the contextual meaning through various conversational turns has been tested.

4. **SParC :** SParC is a dataset based on Spider that tests the model's capability to perform contextual understanding based conversational Text-to-SQL operations evaluation by generating SQL queries using previous conversational history.

5. **BIRD :** BIRD is a recently developed large-scale benchmark dataset aimed at analyzing Text-to-SQL operations performed by large language models. The dataset consists of complex schema, large databases, and domain-specific knowledge requirements.

6. **DuSQL and CSpider :** DuSQL and CSpider are the two popular Chinese versions of benchmark datasets used for evaluating multilingual and cross-lingual Text-to-SQL database schema.

Dataset	**Examples**	**Databases (DB)**	**Tables**	**Rows**	**Release Date**
WikiSQL	80,654	26,521	1	17	Aug-2017
Spider	10,181	200	5.1	2K	Sep-2018
SParC	-	Based on Spider	-	-	Jun-2019
CoSQL	15,598	200	-	-	Sep-2019
SQUALL	11,468	1,679	1	-	Oct-2020
DuSQL	23,797	200	4.1	-	Nov-2020
KaggleDB QA	272	8	2.3	280K	Jun-2021
ADVETA	-	Based on Spider and WikiSQL	Adversarial table	-	Dec-2022
BIRD	12,751	95	7.3	549K	May-2023

Table 2.1: Comparative Analysis of Datasets

Chapter 3

PROPOSED FRAMEWORK

The framework aims at translating natural language user prompts into executable SQL queries that contain proper semantic meaning. It involves schema-aware prompts, retrieval-augmented context, execution validation, and self-refinement techniques. Unlike existing Text-to-SQL systems where SQL queries are generated without any validation process, the approach described herein includes an iteration of SQL query generation and feedback-driven refinement. Any error during execution will be corrected based on the feedback received from the execution process. This way, the reliability of the query generation process is enhanced.

The overall framework consists of the following major stages:

1. Natural Language Query Processing
2. Schema Retrieval and Context Augmentation
3. Prompt Construction
4. Initial SQL Query Generation using LLM
5. Execution-Aware Validation
6. Self-Refinement and Error Correction
7. Final SQL Query Generation and Output

3.1 Problem Formulation

The objective of this proposed framework is generating a valid SQL query from a natural language prompts while ensuring that both syntax correctness and semantic accuracy is maintained.

Let:

Q = set of natural language prompts

S = set of SQL queries

D = database schema and contextual information

The goal is to create an associating mapping function between natural language prompts and SQL queries:

$$f : \mathbb{Q} \rightarrow S \tag{3.1}$$

such that for every natural language prompt $q \in Q$, the resultant generated SQL query $s \in S$. is both executable valid and semantically accurate to user's understanding. This proposed framework extends beyond traditional Text-to-SQL generation systems by incorporating valid execution feedback into the SQL generation pipeline. The framework iteratively refines each generated SQL queries until execution accuracy is met.

The complete mapping process for the SQL query generation as follows:

$$f(q) = R\big(E\big(G(q, D)\big)\big) \tag{3.2}$$

Where:

G = SQL generation sets using LLM

E = execution-aware validation sets

R = self-refinement sets

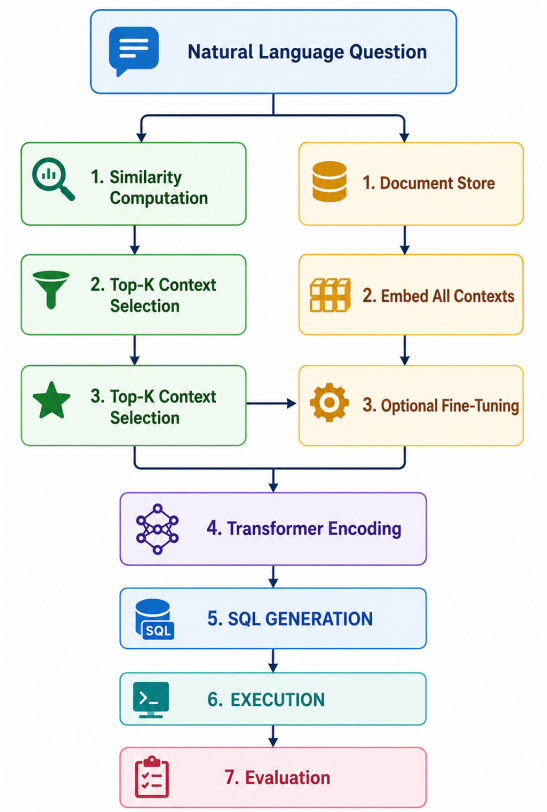

Figure 3.1: Flowchart of Proposed Framework

3.2 Methodology

The proposed framework integrates the large language models with retrieval augmentation based architecture (RAG) and execution-aware refinement systems. These models consists of the following major components:

1. **User Query Interface :** Accepts natural language input from the user.
2. **Schema Retrieval Module :** Retrieves relevant schema information, table descriptions, column names, and sample queries from the database.
3. **Context Augmentation Layer :** Enhances the prompt using retrieved schema context and example SQL patterns.
4. **LLM-Based SQL Generator :** Generating the initial SQL query using a Large Language Model.
5. **Execution Engine :** Executes the generated SQL query on the target database.

6. **Execution Feedback Analyzer :** Analyzes execution errors, runtime failures, and incorrect outputs.

7. **Self-Refinement Module :** Uses execution feedback to iteratively refine and regenerate improved SQL queries.

8. **Final Output Module :** Returns the final executable SQL query and corresponding database results.

3.3 Natural Language Query Processing

The first stage of this proposed framework involves processing the natural language prompt provided by the user.
This user prompt undergoes several preprocessing steps:

- Tokenization
- Noise removal
- Text normalization
- Dependency parsing
- Named entity recognition
- Intent extraction

This stage helps the system identify important entities such as:

- table names,
- column references,
- filtering conditions,
- aggregation operations,
- sorting instructions.

Semantic understanding of the user query is essential for generating accurate SQL statements.

3.4 Schema Retrieval and Context Augmentation

The biggest challenge in Text-to-SQL frameworks is schema understanding. Since large databases often contain multiple tables, relationships, and ambiguous column names, retrieval-based contextual augmentation (RAG) is integrated into the framework.

The schema retrieval module extracts:

- table schemas,
- column descriptions,
- foreign key relations,
- sample SQL queries,
- metadata information.

Embedding-based similarity search is used to retrieve relevant schema components related to prompt given by user.
Let:

- Vq = embedding vector of query
- Vs = embedding vector of schema context

Similarity is computed using cosine similarity:

$$cosinesimilarity(v_q, v_s) = \frac{v_q \cdot v_s}{\|v_q\|\|v_s\|} \tag{3.3}$$

Top-k relevant schema contexts are then retrieved and appended to the LLM prompt.This retrieval-augmented approach improves:

- schema grounding,

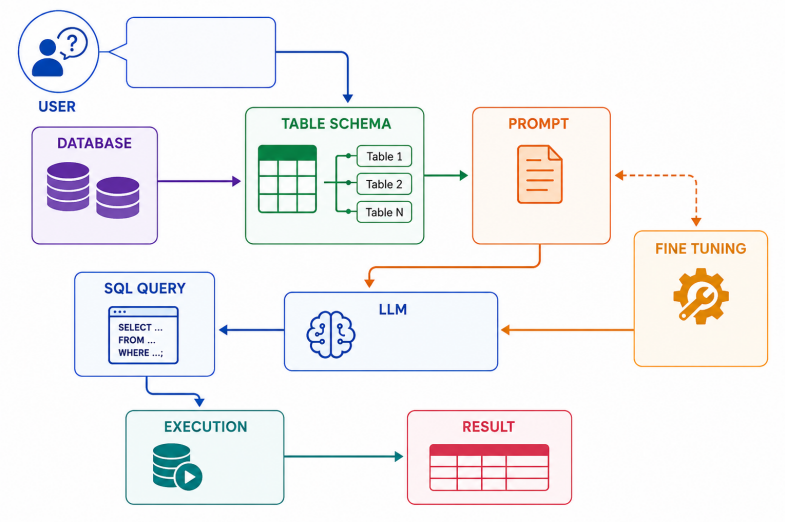

Figure 3.2: Prompt Generation for Table Schema

- contextual understanding,
- cross-domain adaptability,
- reduction of hallucinated SQL generation.

3.5 Prompt Construction

After retrieval augmentation framework in the pipeline, a structured prompt is constructed for the large language model. This prompt includes user natural language query, database schema information, table, relationships and SQL generation instructions. Prompt engineering plays an important role in improving SQL generation accuracy and reducing ambiguity within Text-to-SQL systems. Few-shot prompting and chain-of-thought validation strategies are also incorporated to improve complex SQL query generation.

3.6 Initial SQL Query Generation using LLM

This enriched prompt is then passed on to the large language model for SQL query generation. The generated query is then forwarded to the execution-aware validation stage. The LLM then generates an initial SQL query based on semantic interpretation, schema context, prompt instructions, example patterns in the knowledge base.

3.7 Execution-Aware Validation

Execution-aware validation is the most important component of the proposed framework. Unlike traditional Text-to-SQL systems that stop after query generation, this proposed framework executes generated SQL query on the target database to verify its accuracy. Execution feedback is then captured and passed on to the self-refinement module for its validation.

The execution engine checks:

- syntax validity,
- schema consistency,
- runtime execution,
- logical correctness,
- output validity.

Possible execution outcomes include:

1. Successful Execution : The query executes successfully and produces the intended result.
2. Syntax Errors : The query contains invalid SQL syntax.
3. Schema Errors : Incorrect table or column references are detected.
4. Logical Errors : The query executes successfully but produces incorrect results.

3.8 Self-Refinement Mechanism

The self-refinement mechanism is required for correcting the SQL queries using execution validation that are erroneous automatically. Instead of regenerating queries from scratch, the framework uses iterative refinement methods where the LLM analyzes:

- execution errors

- invalid schema references
- missing conditions
- incorrect joins
- aggregation mistakes

The refinement loop can be represented as:

$$SQL_{t+1} = Refine(SQL_t, Feedback_t) \tag{3.4}$$

Where:

SQL(t)= generated query at iteration t

Feedback(t)= execution feedback

SQL(t+1)= refined query

This iterative process continues until the query executes successfully and semantic correctness is achieved or maximum refinement iterations are reached. This mechanism significantly improves execution accuracy, robustness, complex query handling, reliability in real-world databases.

3.9 Retrieval-Augmented Self-Refinement

Retrieval-Augmented architecture is used to enhance the refinement of SQL queries during correction stages. This contextual understanding helps the LLM to generate more accurate refinements during iterative corrections. The integration of retrieval augmentation with self-refinement reduces the overall hallucinations and improves SQL query consistency.

When execution failures occur, the system retrieves:

- relevant schema corrections
- similar query examples

- execution patterns
- related SQL templates

3.10 Final SQL Generation

Once the refined query successfully passes execution validation, the final SQL query is returned as output where the final output ensures syntactic correctness, semantic accuracy, execution validation.
The system provides:

- executable SQL query,
- query execution status,
- database results,
- optional natural language explanation.

3.11 Datasets Used

These datasets provide diverse schemas and query complexities for evaluation. The proposed framework can be evaluated using the benchmark datasets such as:

1. Spider: Complex cross-domain dataset with multi-table queries.
2. WikiSQL: Large-scale single-table SQL generation dataset.
3. BIRD:Enterprise-level benchmark for LLM-based Text-to-SQL evaluation.
4. CoSQL: Conversational Text-to-SQL dataset.
5. SParC: Context-dependent SQL generation benchmark.

Chapter 4

EXPERIMENTAL SETUP

This project was implemented using Google Colab, using Python programming language with GPU support. It also allowed for interaction with Gemini as the large language model for prompt engineering and retrieval of queries. For applications in machine learning, database interaction, and natural language processing, Python programming language was used for them.

NumPy library was used for numerical computation and array-based operations for data manipulation. Data preparation and preprocessing, and structured data manipulation were implemented using Pandas package. In particular, Pandas DataFrames were used for processing benchmark datasets such as Spider, WikiSQL, BIRD when generating and validating queries according to the mapped schemas.

The backbone for generating SQL queries with self-refinement is the Gemini API. This model was chosen for generating SQL requests due to excellent reasoning and ability to understand contexts in addition to being able to generate well-written code. Our pipeline uses an API key to call prompts for generating SQL queries, as well as doing a self-refinement of the results based on execution feedback validation

Execution of SQL queries and their validation were done using SQLite as the database engine. SQLite was selected as the RDBMS due to its simplicity of execution and easy integration with Python libraries, in addition to being suitable for our experimentation with Text-to-SQL tasks.

Chapter 5

RESULTS and DISCUSSION

Performance evaluation for the proposed Execution-Aware Self-Refining Text-to-SQL model is carried out in order to gauge the effectiveness of the proposed mechanisms for execution-aware validation and iterative self-refinement as means of improving SQL generation precision, reliability, schema alignment, and adaptability to various types of queries. Evaluation was performed with benchmark datasets like Spider, WikiSQL, and BIRD. Performance Metrics that are used during performance analysis include Execution Accuracy (EX), Syntax Validity Rate, Refinement Success Rate, and Query Execution Time.

5.1 Experimental Evaluation

The experimental environment comprised an SQLite relational database, containing several related tables, namely employees, departments, and projects. Employees table had fields like employee_id, employee_name, salary, department_id, and project_id. Department table included department_id, department_name, and location. Projects included attributes like project_id, project_name, and project_status.

A variety of natural language queries, ranging from simple SELECT operations to complex queries involving nesting, multiple tables joining, and other elements, were provided to the model as a means of evaluating its SQL generation capability for large language model with execution-aware refinement. Specifically, the experiment involved Gemini Pro with retrieval-augmented prompting and schema-aware contextual embeddings for gener-

ating SQL commands.

In addition, some query augmentation mechanisms based on semantic transformations using BERT were introduced in order to further boost robustness and generality of the model. Specifically, the concept of query augmentation refers to creation of semantically equivalent but lexically varied instances of user's natural language input queries.

To understand an example of semantic paraphrasing, the original query written in natural language is transformed to other sentences without changing its semantic meaning. This helps the model to learn different ways to represent the same SQL meaning. Hence, it helps in improving the overall model's generalization capability for different SQL queries.

Examples:

- Original User Query: "Show employees having salaries more than 70000."
- Refined Query: "Show employee records with salary more than 70000."

Second Example:

- Original User Query: "Show departments having their offices in Delhi."
- Refined Query: "Return department names having their office in Delhi."

5.2 Evaluation Metrics

Evaluation criteria play an essential role in measuring effectiveness of Text-to-SQL systems. Evaluation methods for the Execution-Aware Self-Refining Text-to-SQL framework are different from those used when comparing a generated SQL query only with respect to its syntax. In addition to syntax, the quality of an output query is also estimated based on successful execution on a given database, semantic accuracy, and efficient performance of the query.

5.2.1 Content-Matching-Based Metrics

Content-matching criteria include benchmark indicators designed to compare semantic similarity of an initial generated SQL query with a gold standard query. Such criteria are

focused on generating correct SQL clauses, including SELECT clauses, filtering conditions, aggregation, and JOINs.

a. Component Matching (CM)

Component Matching criteria measure if a generated SQL query contains all necessary logical components that the gold standard includes, disregarding the format and ordering. For example:

- **Gold Query:** SELECT book_title FROM books WHERE publish_year > 2018 AND category = 'Science';
- **Generated Query:** SELECT book_title FROM books WHERE category = 'Science' AND publish_year > 2018;

Despite possible order differences, the two queries may have the same logical components, generate the same set of filtering conditions, but differ in format. Such queries are considered to be valid according to component matching criterion. It should be mentioned that a single SQL query may have many alternative formats without changing the logic of the query and ensuring semantic accuracy.

b. Exact Match Accuracy (EM)

A more strict content-matching criteria that compares queries in terms of identical clauses' order is Exact Match Accuracy criterion. According to this criterion, a generated query must have the exact same format and the identical order of clauses to be regarded as valid. Thus, although the queries are logically equivalent, they are different in terms of their structure and cannot be evaluated according to this criterion.

Using the previous example:

- **Gold Query:** SELECT book_title FROM books WHERE publish_year > 2018 AND category = 'Science';
- **Generated Query:** SELECT book_title FROM books WHERE category = 'Science' AND publish_year > 2018;

Table 5.1: Comparison of Exact Match Accuracy and Execution Accuracy on Different SQL Queries using the Proposed Execution-Aware Self-Refining Framework

Query Type	**Expected SQL**	**Generated SQL after Self-Refinement**	**EM (%)**	**EX (%)**
Simple SELECT	`SELECT * FROM books;`	`SELECT * FROM books;`	100	100
Conditional SELECT (WHERE)	`SELECT patient_name FROM patients WHERE age > 60;`	`SELECT patient_name FROM patients WHERE age > 60;`	100	100
Equality Filter	`SELECT course_name FROM courses WHERE semester = 5;`	`SELECT course_name FROM courses WHERE semester = 5;`	100	100
Aggregate with COUNT	`SELECT COUNT(*) FROM orders WHERE order_status = 'Delivered';`	`SELECT COUNT(*) FROM orders WHERE order_status = 'Delivered';`	100	100
Aggregation with GROUP BY	`SELECT category, AVG(price) FROM products GROUP BY category;`	`SELECT category, AVG(price) FROM products GROUP BY category;`	95	98
SELECT with ORDER BY	`SELECT movie_title FROM movies ORDER BY rating DESC;`	`SELECT movie_title FROM movies ORDER BY rating DESC;`	93	97
Nested Query	`SELECT student_name FROM students WHERE marks > (SELECT AVG(marks) FROM students);`	`SELECT student_name FROM students WHERE marks > AVG(marks);` → Refined to correct nested query	88	94
Join with Another Table	`SELECT s.student_name, d.department_name FROM students s JOIN departments d ON s.department_id = d.department_id;`	`SELECT student_name, department_name FROM students JOIN departments ON students.department_id = departments.department_id;`	90	96
Schema Correction Query	`SELECT hospital_name FROM hospitals WHERE city = 'Mumbai';`	Initial Query: `WHERE location = 'Mumbai'` → Refined using execution feedback	85	95
Aggregate with HAVING Clause	`SELECT genre, COUNT(*) FROM songs GROUP BY genre HAVING COUNT(*) > 10;`	`SELECT genre, COUNT(*) FROM songs GROUP BY genre HAVING COUNT(*) > 10;`	92	97
Ambiguous Query	`SELECT product_name FROM products ORDER BY sales_count DESC LIMIT 5;`	Initial Query generated incorrect sorting column → Refined after execution feedback	82	91
Invalid Table Reference	`SELECT AVG(fee_amount) FROM courses;`	Initial Query: `FROM course` → Corrected to `courses` after refinement	87	96
Synonym-Based Query	``Show books published after 2020'' → `SELECT book_title FROM books WHERE publish_year > 2020;`	`SELECT book_title FROM books WHERE publish_year > 2020;`	96	99
Ambiguous Natural Language Query	``Top rated restaurants'' → `SELECT restaurant_name FROM restaurants ORDER BY rating DESC LIMIT 5;`	Initial SQL missing LIMIT clause → Refined after execution validation	84	92

5.2.2 Execution-Based Metrics

Execution-based criteria measure the quality of execution of a generated SQL query on the target database.

a. Execution Accuracy (EX)

Accuracy of Execution determines whether the SQL statement created by the model produces the same output as the SQL gold standard when executed on the database.
For example:

- **Gold Query:** SELECT course_name FROM courses WHERE semester = 6;
- **Generated Query:** SELECT course_name FROM courses WHERE semester = 6;

If the two statements yield exactly the same output after being executed, then the created SQL statement is considered to be semantically correct. It should be noted that the accuracy of execution plays a significant role in the presented method since execution-based validation is the focus of this approach.

b. Valid Efficiency Score (VES)

Valid Efficiency Score (VES) is used as benchmark in measuring the efficiency of the created SQL statement as compared to the gold standard. While the created SQL statement may produce the desired output, it may contain some unnecessary features such as unnecessary join operations, redundant conditions, or subqueries that may affect its execution speed negatively.
For example:

- **Gold Query:** SELECT patient_name FROM patients WHERE age > 65;
- **Generated Query:** SELECT patient_name FROM patients WHERE age > 65 AND patient_id IN (SELECT patient_id FROM patients);

Even though the created SQL statement produces the desired output, there is a possibility that it contains a redundant subquery, which may increase its execution time. VES shows how efficient the created SQL statement is as compared to the idealized gold standard.

```sql
-- Predicted SQL Query (Initial Generation):
SELECT employee_name
FROM employees
WHERE salary > 50000
GROUP BY employee_name;
```

Figure 5.1: Predicted SQL Query

```
-- Execution Feedback:
Error: Invalid GROUP BY usage without aggregation function
```

Figure 5.2: Execution Feedback

```sql
-- Refined SQL Query:
SELECT employee_name
FROM employees
WHERE salary > 50000;
```

Figure 5.3: Refined SQL Query

```sql
-- Gold Standard SQL Query:
SELECT employee_name
FROM employees
WHERE salary > 50000;
```

Figure 5.4: Gold Standard SQL Query

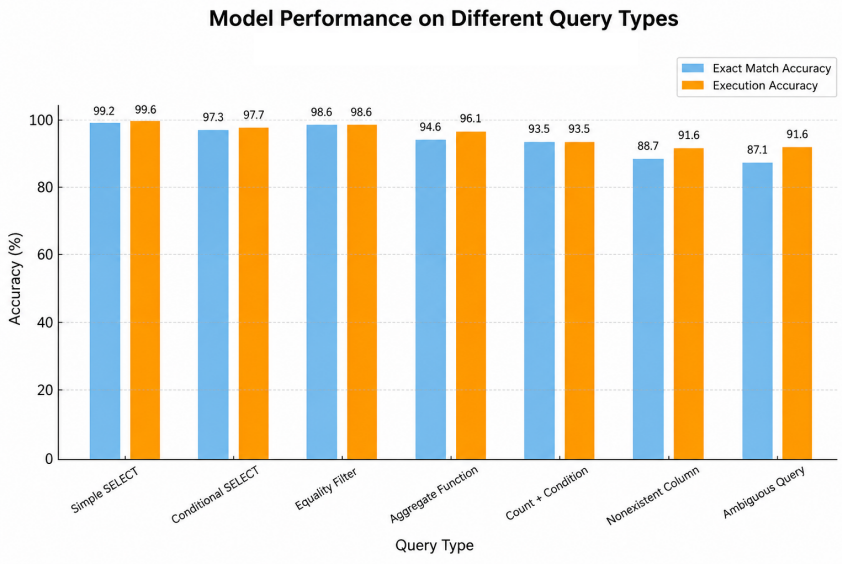

Figure 5.5: Performance on Different Queries

5.3 Result Analysis

A SQL query initially generated had a GROUP BY statement without an aggregate function, which caused inconsistent execution. In such a situation, the execution-aware component recognized the failure at the query execution stage and notified the self-refinement unit about the issue. Then the incorrect grouping was excluded from the SQL code, resulting in a revised SQL statement.

The experiment involved testing the RAG model built on the Gemini Pro model for simple selection, conditional, and aggregation queries on a natural language corpus. SQL codes generated by the model were verified through execution-aware feedback validation and self-refinement methods. It was found that combining the execution-aware feedback and self-refinement techniques considerably improved the generation results, including accuracy, schema-awareness, and execution reliability. The tested framework showed capability in detecting failed executions, correcting generated SQL statements and producing new correct queries.

5.4 Effect of Prompt Engineering on Results

It is extremely important to ensure clarity within the prompts as input. The framework ensures the schema mention in prompt is significantly minimized allowing the model to have accurate data and columns. In absence of prompt engineering, it can lead to generation of hallucinated tables or truncated SQL query results.

5.5 Effect of Execution Awareness on Results

The execution-aware validation generated SQL queries through actual database execution. Queries producing runtime errors or invalid outputs were identified and forwarded for correction for accurate results. This significantly reduced runtime failures compared to conventional single-pass Text-to-SQL systems.

Execution-aware validation enabled the framework to:

- detect schema mismatches,
- identify logical inconsistencies,
- capture execution failures,
- validate query outputs.

5.6 Effect of Self-Refinement on Results

The self-refinement module further enhanced system performance by introducing iterative accuracy enhancement. Unlike conventional systems that stop after generating an incorrect query, the proposed framework utilized execution feedback validation to iteratively refine SQL outputs.

The self-refinement mechanism improved:

- handling of ambiguous queries,
- correction of schema mismatches,
- nested query generation,
- complex join operations.

5.7 Analysis of Complex Query Handling

The proposed framework showed particularly strong performance for nested SQL queries, multi-table joins, aggregation operations, context-dependent queries.

Execution-aware validation particularly improved:

- join consistency,
- filtering correctness,
- aggregation accuracy.

The self-refinement mechanism successfully corrected many errors involving:

- missing WHERE clauses,
- incorrect GROUP BY operations,
- invalid table aliases,
- foreign key mismatches.

Chapter 6

CONCLUSION AND FUTURE SCOPE

Firstly, this proposed framework has overcome several shortcomings that hinder the practicality of previous Text-to-SQL frameworks. Existing Text-to-SQL frameworks do not consider execution feedback to check the execution validity of generated SQL queries and iteratively refine output accordingly. While existing LLM-based Text-to-SQL models aim at creating syntactically valid SQL queries, many queries generated by these systems fail at the runtime due to various reasons including schema mismatching, wrong join operation, lack of conditionality in the generated code, among others.

The proposed approach combines retrieval-augmented schema understanding, schema-aware prompt engineering, execution-aware validation, and self-refinement mechanisms into a comprehensive Text-to-SQL system. The key differentiator from traditional models is the ability of the proposed approach to execute generated queries and provide runtime execution feedback for further refinement of the query if required. Besides, retrieval augmentation technique has been used in order to better understand schema and context so that hallucinated SQL generation can be avoided.

Experimental evaluations show that the proposed framework is capable of performing well in a number of benchmark databases including Spider, WikiSQL, BIRD, CoSQL, and SParC. At first, a baseline framework without execution awareness was implemented, which gave an EM of 71.4%, EX of 74.8%, and SVR of 88.2%. The retrieval model helped connect the schema and comprehend the context in a proper way by providing relevant schema descriptions, metadata, and example SQL codes within the prompt construction. Execution-aware validation and contributed to increasing the robustness of our frame-

work. Therefore, one could conclude that the execution of SQL code in order to validate its correctness proved to be extremely beneficial in terms of achieving a greater semantic accuracy and avoiding referencing of invalid tables.

Finally, the self-refinement module caused the greatest improvement in the performance of our system. This allowed resolving numerous issues associated with generated SQL including incorrect nesting of subqueries, inappropriate use of joins, erroneous aggregations, lack of proper filters, and invalid table references, among others.

In the future, conversation-based frameworks can incorporate techniques such as memory-augmented architecture, dialogue awareness, and adaptive context windowing to enhance query generation efficiency. Though the suggested system significantly increased the accuracy of execution, very challenging nested SQL queries and reasoning-based problems require extra efforts. In future developments, one can employ such techniques as chain of thought reasoning, graph-based schema reasoning, symbolic reasoning, and agent-based LLM architecture to handle nested queries effectively.

While most benchmarking datasets feature fairly clean academic databases, practical implementation will require dealing with noisy, partially labeled schemas with interconnected tables in enterprises. It would be useful to conduct experiments using fine-tuning and adaptation to healthcare, financial, cybersecurity, law, and science domains to improve application prospects.One can also integrate other systems such as knowledge graphs, APIs, and semantic retrievals into future frameworks to improve contextual capabilities and handle domain-specific SQL query generation.

Bibliography

[1] Albert Wong, Lien Pham, Young Lee, Shek Chan, Razel Sadaya, Youry Khmelevsky, Mathias Clement, Florence Wing Yau Cheng, Joe Mahony, and Michael Ferri. Translating natural language queries to sql using the t5 model. In 2024 IEEE International Systems Conference (SysCon), pages 1–7. IEEE, 2024.

[2] TZhang, Bin, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. "Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation." arXiv preprint arXiv:2403.02951 (2024).

[3] Wuzhenghong, Wen, Zhang Yongpan, Pan Su, Sun Yuwei, Lu Pengwei, and Ding Cheng. "LR-SQL: A Supervised Fine-Tuning Method for Text2SQL Tasks under LowResource Scenarios." arXiv preprint arXiv:2410.11457 (2024).

[4] Gao, Dawei, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. "Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation." arXiv preprint arXiv:2308.15363 (2023).

[5] Xuan-Bang Nguyen, Xuan-Hieu Phan, Massimo Piccardi, Fine-tuning text-to-SQL models with reinforcement-learning training objectives, Natural Language Processing Journal, Volume 10, 2025, 100135, ISSN 2949-7191, https://doi.org/10.1016/j.nlp.2025.100135.

[6] Hong, Zijin, Zheng Yuan, Hao Chen, Qinggang Zhang, Feiran Huang, and Xiao Huang. "Knowledge-to-sql: Enhancing sql generation with data expert llm." arXiv preprint arXiv:2402.11517 (2024).

[7] Yingwen Fu, Wenjie Ou, Zhou Yu, and Yue Lin. Miga: A unified multi-task generation framework for conversational text-to-sql. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 37, pages 12790–12798, 2023.

[8] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. Next-generation database interfaces: A survey of llm-based text-to-sql. arXiv preprint arXiv:2406.08426, 2024.

[9] Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. arXiv preprint arXiv:2405.07467, 2024.

[10] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 37, pages 13067–13075, 2023.

[11] Zhao, Siyun, Yuqing Yang, Zilong Wang, Zhiyuan He, Luna K. Qiu, and Lili Qiu. "Retrieval Augmented Generation (RAG) and Beyond: A Comprehensive Survey on How to Make your LLMs use External Data More Wisely." arXiv preprint arXiv:2409.14924 (2024).

[12] Xie, Yuanzhen, et al. "Decomposition for enhancing attention: Improving llm-based text-to-sql through workflow paradigm." Findings of the Association for Computational Linguistics: ACL 2024. 2024.

[13] Mao, Wenxin, et al. "Enhancing text-to-SQL parsing through question rewriting and execution-guided refinement." Findings of the Association for Computational Linguistics: ACL 2024. 2024.

[14] Yang, Yicun, et al. "Automated validating and fixing of text-to-sql translation with execution consistency." Proceedings of the ACM on Management of Data 3.3 (2025): 1-28.

[15] Su, Xiaodong, et al. "A robust natural language text-to-SQL generation framework with dynamic strategies based on LLMs." Scientific Reports (2026).

[16] Liu, Xinyu, et al. "A Survey of Text-to-SQL in the Era of LLMs: Where are we, and where are we going?." IEEE Transactions on Knowledge and Data Engineering (2025).

[17] Li, You, et al. "Mitigating LLM hallucinations in Text-to-SQL parsing with a self-refinement feedback loop and memory augmentation." Journal of Intelligent Information Systems (2026): 1-26.

[18] Deng, Minghang, et al. "ReFoRCE: a text-to-SQL agent with self-refinement, consensus enforcement, and column exploration." arXiv preprint arXiv:2502.00675 (2025).

[19] Hong, Zijin, et al. "ErrorLLM: Modeling SQL Errors for Text-to-SQL Refinement." arXiv preprint arXiv:2603.03742 (2026).

[20] Shi, Liang, et al. "A survey on employing large language models for text-to-sql tasks." ACM Computing Surveys 58.2 (2025): 1-37.

[21] Zhang, Bin, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. "Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation." arXiv preprint arXiv:2403.02951 (2024).

[22] L. Wang, A. Zhang, K. Wu, K. Sun, Z. Li, H. Wu, M. Zhang, and H. Wang, "Dusql: A large-scale and pragmatic chinese text-to-sql dataset," in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2020

[23] Wuzhenghong, Wen, Zhang Yongpan, Pan Su, Sun Yuwei, Lu Pengwei, and Ding Cheng. "LR-SQL: A Supervised Fine-Tuning Method for Text2SQL Tasks under LowResource Scenarios." arXiv preprint arXiv:2410.11457 (2024)

[24] Yang, Yu-Jie, Hung-Fu Chang, and Po-An Chen. "LLM-Based SQL Generation: Prompting, Self-Refinement, and Adaptive Weighted Majority Voting." arXiv preprint arXiv:2601.17942 (2026).

Vinod kumar

Ankit Sharma thesis_removed

Ankit_Sharma_mtech1

Document Details

Submission ID
trn:oid:::27535:140821853

Submission Date
May 28, 2026, 9:44 PM GMT+5:30

Download Date
May 28, 2026, 9:48 PM GMT+5:30

File Name
Ankit Sharma thesis_removed.pdf

File Size
5.1 MB

32 Pages

5,950 Words

36,296 Characters

8% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

55 Not Cited or Quoted 8%
Matches with neither in-text citation nor quotation marks

0 Missing Quotations 0%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

2% Internet sources
3% Publications
7% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Vinod kumar

Ankit Sharma thesis_removed

Ankit_Sharma_mtech1

Document Details

Submission ID
trn:oid:::27535:140821853

Submission Date
May 28, 2026, 9:44 PM GMT+5:30

Download Date
May 28, 2026, 9:48 PM GMT+5:30

File Name
Ankit Sharma thesis_removed.pdf

File Size
5.1 MB

32 Pages

5,950 Words

36,296 Characters

*% detected as AI

AI detection includes the possibility of false positives. Although some text in this submission is likely AI generated, scores below the 20% threshold are not surfaced because they have a higher likelihood of false positives.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.