Training LLMs for Natural Language to SQL Translations for Osquery

A THESIS WORK SUBMITTED TO
DEFENCE INSTITUTE OF ADVANCED TECHNOLOGY, PUNE
FOR THE SEMESTER THREE EVALUATION (2024-2026) OF
MASTER OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING
(Artificial Intelligence)

submitted by

Nitish Kumar

(Registration No. 24-26-35)

Under the Guidance of

Dr. Sunita Vikrant Dhavale

Department of Computer Science and Engineering
Defence Institute of Advanced Technology
(Deemed University)
Girinagar, Pune – 411025

Dec 2025

Department of Computer Science and Engineering, DIAT, Pune

M Tech (Computer Science & Engineering – Cyber Security/AI)

Synopsis of Dissertation - Part 1

1.	**Name & Registration No**	:	**Nitish Kumar** (24-26-35)
2.	**Dissertation Title**	:	Training LLMs for Natural Language to SQL Translations for Osquery
3.	**Guide (Name, Designation & Company) Internal**	:	**Dr. Sunita V. Dhavale** (Assistant Prof. DIAT Pune)
4.	**Co-Guide (Name, Designation & Company) External**	:	**Prof. Manjesh K. Hanawal** (Associate Prof. IIT Bombay)
5.	**Associated Lab/Internship Agency and details (Attach snapshots of related email/letters in Annexure B)**	:	TCA2I Vajra Lab, IIT Bombay
6.	**Details of Online courses/certifications done for skill development to carry out research etc. (Attach snapshots of certificates in Annexure C)**	:	Fine Tuning Large Language Models (Infosys Springboard)

7. Problem Statement:

The goal of this project is to develop a Natural Language to SQL (NL2SQL) conversion model that can accurately translate user queries written in plain English into corresponding SQL statements for a fixed database schema. In this work, the fixed database schema used is derived from the osquery tables, which provide a structured and standardized schema representing system-level information in tabular form. This setup allows the model to generate SQL queries specifically tailored to the OSQuery environment, facilitating effective interaction with system data through natural language. The model will be trained and optimized for a predefined database schema, ensuring domain-specific understanding and precise mapping between natural language expressions and SQL components such as SELECT, FROM, WHERE, and JOIN clauses. By restricting the schema, the focus shifts toward achieving high translation accuracy, robust understanding of linguistic variations, and effective handling of complex queries like aggregation, filtering, and nested conditions within that schema.

8. Introduction:

Databases are the backbone of nearly every software system we interact with daily—from mobile apps and websites to enterprise tools. They store, organize, and manage data efficiently. SQL (Structured Query Language) is the standard language used to access and manipulate relational databases, enabling users to perform operations such as selecting, inserting, updating, and deleting data.

However, not everyone is proficient in SQL. It is a specialized skill that requires an understanding of syntax, joins, aggregations, and database schemas. This creates a barrier for non-technical users—such as business analysts, managers, and general users—limiting their ability to retrieve and analyze data independently.

To address this challenge, the concept of querying databases using natural language (e.g., plain English) has emerged. This approach democratizes access to data by making interaction with databases more intuitive and user-friendly. As some tech leaders have pointed out, natural language is rapidly becoming the "new programming language"—thanks to the rise of artificial intelligence (AI) models capable of translating conversational queries into executable code. This significantly reduces the need for traditional programming expertise.

Scope of Work:

The scope of this work is to develop an NL2SQL system specifically tailored for OSQuery to simplify system monitoring and security analysis.

- It focuses on creating a high-quality dataset that maps natural language questions to correct OSQuery SQL queries.
- The work includes fine-tuning large language models such as CodeT5, BART, and Qwen-7B to accurately generate SQL for OSQuery tables.
- It implements a schema-aware retrieval mechanism that identifies the most relevant OSQuery tables based on the user's natural language input.
- The system is evaluated using key metrics like exact match, component match, recall, and SQL validity to measure accuracy and robustness.
- Finally, the system is designed to support real-world use cases, enabling analysts and administrators to query OSQuery databases using natural language without needing SQL expertise.

9. Relevance to defense and Application Area:

The proposed NL2SQL system for OSQuery is highly relevant to defense, security operations, and cyber-forensic environments. Modern defense infrastructures rely on real-time system monitoring, threat detection, and anomaly investigation across thousands of endpoints. OSQuery already provides powerful system-level information, but requires expert knowledge of complex SQL queries. By enabling analysts to retrieve system data using natural language, the system significantly reduces response time, improves situational awareness, and minimizes human error. This technology can assist in incident response, insider-threat analysis, vulnerability assessment, and digital forensics, making it a valuable tool for military, government, and critical infrastructure protection.

- **Faster Threat Investigation:** Converts natural-language queries into OSQuery SQL, allowing defense analysts to quickly retrieve system information during cyber incidents.
- **Reduces Skill Gap:** Eliminates the need for deep SQL knowledge, enabling even junior analysts to perform advanced system investigations.
- **Real-time Endpoint Visibility:** Enhances monitoring of endpoints across defense networks, improving detection of abnormal processes, login activities, and configuration changes.
- **Supports Digital Forensics:** Helps in analyzing system artifacts, user activities, memory information, and processes during forensic analysis.
- **Strengthens Cyber Defense:** Contributes to proactive defense mechanisms such as vulnerability assessment, insider-threat detection, and compliance monitoring.

10. **Requirements (H/W & S/W):**

Hardware Requirements

- **Processor:** Minimum Intel i5 / AMD equivalent (Recommended: i7 or higher for model fine-tuning).
- **RAM:** Minimum 8 GB (Recommended: 16–32 GB for training deep-learning models).
- **Required for fine-tuning Qwen-7B →** NVIDIA GPU with **16–24 GB VRAM** or more.
- **Storage:** Minimum **50 GB free space** (datasets, model checkpoints, logs).
- **System:** Windows/Linux/macOS (Linux recommended for training).

Software Requirements

- **Operating System:** Ubuntu 20.04/22.04 (recommended) or Windows 10/11.
- **Python Version:** Python **3.10 or later**.
- **Libraries / Frameworks:**
  - PyTorch
  - Transformers (HuggingFace)
  - PEFT (for LoRA fine-tuning)
  - Datasets
  - Accelerate
  - Sentence Transformers (for retrieval models)
  - Flask / FastAPI (for API interface)
- **Development Tools:**
  - VS Code / PyCharm
  - Git & GitHub
  - Jupyter Notebook (optional)
- **GPU Drivers:**
  - CUDA Toolkit (11.x or 12.x)

11. **Name of outside Agencies interested:**

Indian defence and cybersecurity agencies such as DRDO, BEL, HAL, BDL, NTRO, CDAC, and the Defence Cyber Crime Coordination Centre can benefit from this NL2SQL system, as it enhances secure data access, improves analytical workflows, and supports mission-critical intelligence operations.

12.	**Publications (If Any)**	N/A
13.	**Name and Signature of Student**	
14.	**Name and Signature of Guide**	
15.	**Name and Signature of Co-Guide**	
16.	**Remarks by HOD (If Any)**	

Annexure A

1. Brief Literature Survey:

The evolution of NL2SQL systems, reflects a progression from rule-based and early neural approaches to modern, language model–driven methods. With the emergence of large language models (LLMs) such as GPT-4 and CodeLlama, zero-shot and few-shot SQL generation has become increasingly feasible. In this section, we discuss how advancements in pre-trained language models (PLMs) and LLMs have shaped the NL2SQL pipeline, which can be broadly divided into three key stages as shown in Figure 1.1: Pre-Processing, NL2SQL Translation Methods, and Post-Processing. This structured pipeline enhances the accuracy, efficiency, and robustness of NL2SQL systems in real-world applications.

1.1.Pre-processing

Pre-processing prepares inputs for the translation stage by identifying and enriching relevant database elements, mitigating issues like token limits and schema complexity.

Schema Linking

Schema linking plays a crucial role in bridging the gap between natural language (NL) queries and database structures. The motivation arises primarily from two factors: the inherent **schema complexity** and the **presence of irrelevant tables or columns** in databases, which often make it challenging to align user intents with corresponding database entities. Additionally, the **input length limitations** of large language models (LLMs), such as the 4096-token cap in ChatGPT, restrict the amount of schema information that can be processed simultaneously.

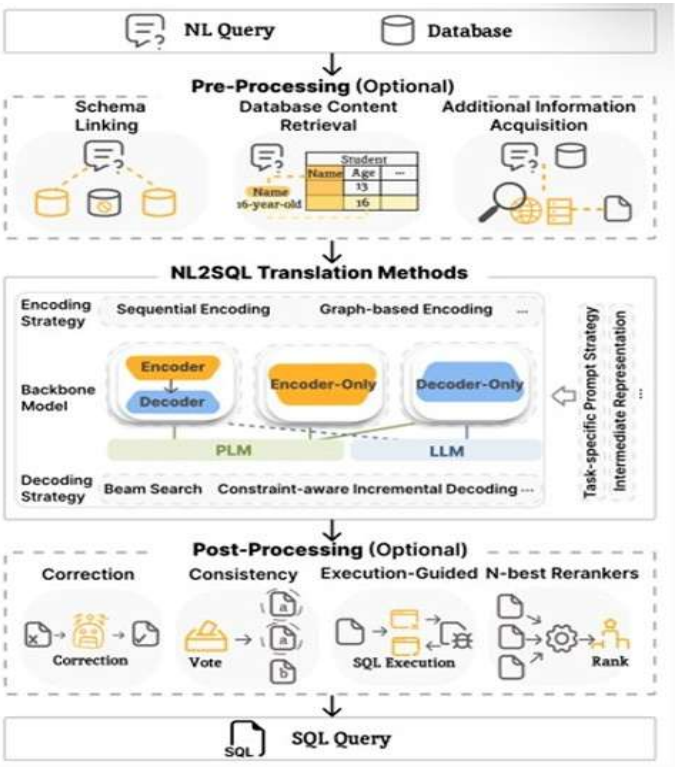


Figure 1.1: An Overview of NL2SQL Methods in the LM Era [11]

1.2. NL2SQL Translation Methods

Encoding Strategy

The encoding strategy is a fundamental component of NL2SQL systems, as it transforms **unstructured or semi-structured data** into representations that can be effectively processed for SQL query generation. The motivation behind encoding lies in its ability to capture both the **semantic meaning** of the natural language (NL) input and the **structural information** of the database schema. By doing so, the model can better comprehend user intent and align it with the appropriate database elements, forming the foundation for accurate SQL synthesis. The overall goal is to convert the NL query and the schema into a structured, machine-understandable format that can be efficiently utilized by downstream models or large language models (LLMs).

Decoding Strategy

The choice of decoding strategy plays a critical role in determining the quality and performance of generated SQL queries. An effective decoding strategy not only ensures syntactically correct SQL queries but also aligns the semantics of the generated SQL with the input natural language (NL), and can even improve query execution efficiency. The primary goal of decoding is to convert the representations generated by the encoder into the target SQL queries. Common decoding strategies include **greedy search**, **beam search**, and **constraint-aware incremental decoding**.

1.3. Post-Processing

Post-processing is a crucial step in NL2SQL systems, aimed at refining the initially generated SQL queries to better meet user expectations. This step enhances the quality and reliability of the output by applying additional strategies to correct errors, improve consistency, and optimize execution. Common post-processing methods include **Correction**, **Consistency**, **Execution-Guided Decoding**, and **N-best Re-rankers**.

Correction involves identifying and fixing syntactic or semantic errors in the generated SQL queries, ensuring that the queries are executable and meaningful. **Consistency** methods use voting or ensemble mechanisms across multiple generated queries to select the most consistent output. **Execution-Guided Decoding** evaluates SQL candidates by executing them against the database and selecting queries that produce valid results, effectively guiding generation with feedback from execution. Finally, **N-best Re-rankers** generate multiple SQL candidates and rank them using learned scoring functions or heuristics to select the most accurate query. Together, these strategies ensure that the final SQL queries are not only syntactically correct but also semantically aligned with the natural language input.

1.4. NL2SQL Evaluation Metrics

To effectively evaluate the performance of Natural Language to SQL (NL2SQL) models, a combination of accuracy, efficiency, and robustness metrics are employed. These metrics provide a comprehensive understanding of how well a model translates natural language into correct, executable, and efficient SQL queries.

Execution Accuracy (EX) measures the proportion of predicted SQL queries whose execution results exactly match those of the ground-truth queries.

$$\text{EX:} \quad \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\left(V_i = \hat{V}_i\right)$$

2. Methodology

In this section, we discuss each stage of the NL2SQL pipeline as outlined in the previous section. We begin with the schema linking strategy employed to align natural language queries with the corresponding database schema. Next, we present our training experiments using both encoder–decoder and decoder-only models. Finally, we describe the post-processing stage of the pipeline, which incorporates a feedback loop and a beam search mechanism to refine and enhance the generated SQL queries.

2.1. Data Synthesis Framework

Developing a Natural Language to SQL (NL2SQL) model for a fixed osquery schema requires a large, diverse, and schema-accurate dataset of natural language–SQL pairs. Since real-world data of this form is scarce and costly to create manually, synthetic data generation becomes essential. However, generating such data automatically introduces several challenges that must be addressed through careful prompt design and post-processing.

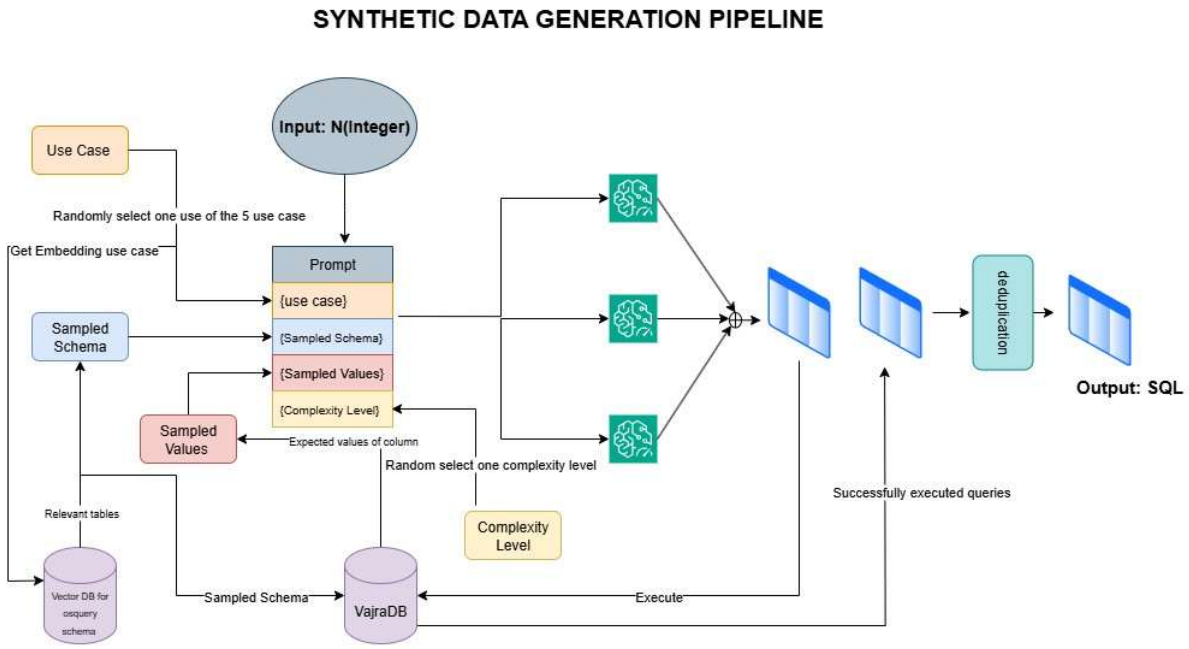


Figure 2.1: Synthetic data generation framework

Generation Pipeline

To overcome the challenges of generating accurate, diverse, and executable SQL queries, a structured and constraint-based prompt was developed. Each component of the prompt plays a specific role in guiding the model toward realistic and schema-consistent outputs.

Use Case

- The prompt begins with a **use case context** that grounds query generation in a real-world operational scenario.

- Since **Osquery** supports multiple cybersecurity and system management applications, the use cases are broadly categorized into five domains:Threat Hunting, Compliance & Audit, IT Asset Inventory, Performance Monitoring and Forensics
- During prompt generation, one of these categories is **randomly selected** to ensure semantic diversity and relevance across generated queries.

Relevant Schema (Tables and Columns)

- This section explicitly lists the tables and columns available in the schema.
- By doing so, it **prevents schema hallucination**, ensuring that the model only generates SQL queries using valid Osquery tables and attributes.

Sample Values

- Provides realistic examples of values found in the database, such as process names, usernames, or port numbers.
- This helps the model formulate **plausible and executable filters**, minimizing unrealistic query logic.

SQL Complexity Level

- Controls the structural difficulty of the generated SQL query.
- Four levels of complexity are defined and **randomly assigned** during generation:
    1. **Easy:** Simple SELECT statements on a single table.
    2. **Medium:** Queries with basic filters, conditions and simple joins.
    3. **Complex:** Multi-table joins or grouped aggregations.
    4. **Highly Complex:** Nested queries, multiple joins, or advanced conditions.
- This ensures a balanced dataset covering a wide range of SQL difficulties.

Output Format (JSON)

- The final SQL query is generated in a **structured JSON format**.
- This enables **machine-readable post-processing and validation**, ensuring the generated SQL can be programmatically verified for correctness.

2.2. Post-Processing

After generation, we apply a strict filtering and validation pipeline:

- Remove non-SELECT queries using rule-based filters.
- Execute SQL queries on the osquery schema to eliminate syntax errors or invalid table references.
- Remove duplicates or semantically similar queries.
- Ensure one-to-one mapping between natural language and SQL intent.

NL Question Synthesis

After creating SQL queries, the next step is to transform them into natural language (NL) questions that convey the same meaning. Based on the method described in a recent research paper, this step aims to generate a variety of clear and accurate NL questions corresponding to each SQL query.

To make the generated questions more diverse and realistic, we define nine common ways in which people typically ask questions: formal, colloquial, imperative, interrogative, descriptive, concise, vague, and metaphorical.

The first six styles (formal, colloquial, imperative, interrogative, descriptive, and concise) demonstrate how users can clearly express their intent using different tones or sentence structures. In contrast, the vague and metaphorical styles mimic situations in which users employ unclear or figurative language, requiring additional interpretation by the model.

The prompt for generating natural language questions is designed to balance accuracy and stylistic variation. It consists of the following key components:

- **Task Instruction:** Specifies that the large language model (LLM) should first explain the SQL query in plain text and then convert it into a corresponding natural language question.
- **SQL Query:** Provides the SQL statement to be transformed, ensuring the generated question accurately reflects the logic of that specific query.
- **SQL-related Column Information:** Lists relevant column names and their contextual meanings. This helps the LLM clarify abbreviations and maintain semantic accuracy.
- **Desired Language Style:** Indicates one of the nine language styles, selected randomly for each query. Each style is accompanied by a short description and an example question to help the LLM produce stylistically varied outputs.

For the first six styles, the model generates questions directly based on the SQL query. For the vague and metaphorical styles, it produces questions that are intentionally less direct, reflecting the ambiguity often present in natural human communication.

For each SQL query, multiple natural language questions are generated. To ensure their correctness and relevance, a **semantic consistency selector module**—adapted from the same research paper—is employed.

We use Sentence Transformers to obtain high-dimensional semantic representations of each generated question. For every question, the average semantic similarity with all other questions is calculated. The question with the highest average similarity is selected, as it best represents the central meaning of the SQL query. This approach filters out outlier questions and preserves the one that captures the core intent.

By combining stylistic diversity with semantic consistency selection, the generated natural language questions remain faithful to the SQL logic while exhibiting natural variation in phrasing. This ensures that the NL2SQL model is trained on a rich set of natural language expressions, improving its ability to handle diverse real-world user inputs.

2.3. Pre-Processing

We use embedding retrieval for schema linking in order to bridge natural language queries with structured database elements. To achieve this, we generate embeddings for database tables by incorporating not just the table names, but also rich metadata such as table descriptions, column names, data types, and column-level descriptions. This additional context helps create semantically meaningful representations of the schema components. We use a transformer-based model to encode this structured information into dense vectors. For each table, we format the metadata in a consistent, natural-language-friendly structure before embedding. During query time, a user's natural language input is also converted into an embedding, and we perform similarity search to retrieve the most relevant schema elements.

2.4. Embedding Model

An **embedding model** projects heterogeneous inputs such as text or tables into a shared high-dimensional vector space, enabling semantic similarity comparisons. Similar inputs are mapped to nearby points, capturing their underlying meaning or structure. In our setup, both natural language queries and structured tables are encoded into a common embedding space, facilitating retrieval through vector similarity:

$$\text{Retrieve}(q) = \underset{ti}{\text{argmax}}\ \text{sim}\left(E_q(q), E_t(t_i)\right),$$

2.5. NL2SQL Translation

As discussed in Section 2, two prominent modeling approaches are commonly used for this task: the encoder–decoder architecture and the decoder-only architecture. In this section, we do a detailed discussion of the training methodology, including optimization strategies and model fine-tuning, and finally, an overview of the retrieval-based models incorporated to enhance query understanding and schema grounding. All experiments were conducted on a remote server equipped with four NVIDIA RTX A6000 GPUs.

Encoder–Decoder Architecture

We employed a popular encoder–decoder architecture based on the CodeT5 model, a variant of the T5 language model specifically designed for code understanding and generation. Both CodeT5-base and CodeT5-large variants were explored using a supervised training setup. The model received the natural language query along with the corresponding database schema as input and was trained to generate the target SQL query as output.

Decoder-Only Architecture

In the decoder-only language model setup, we employed the Qwen-7B model. The model was first pretrained on a large, standard SQL dataset to develop a general understanding of SQL syntax and structure. After this pretraining phase, it was further fine-tuned on our domain-specific dataset to adapt to the nuances of our task.

2.6. Post-Processing

Initially, the model employs a **beam search decoding strategy** to generate multiple candidate SQL queries for each natural language input. Specifically, the model generates the top-k hypotheses (with $k = 5$ in our implementation), representing the five most probable SQL sequences according to the model's output distribution. This approach increases the likelihood of producing at least one valid and semantically accurate query, especially in cases where the top-1 prediction may be syntactically correct but semantically invalid.

Following generation, each candidate query undergoes an **execution validation step**. The queries are executed within the target schema environment (in this case, the OSQuery schema) to verify their syntactic and operational correctness. If the top-ranked query fails to execute—due to missing tables, incorrect column references, or logical inconsistencies—the system triggers a **feedback loop**. In this loop, error information (e.g., syntax errors, invalid references) is programmatically analyzed and fed back to the model or decoding module to guide the selection of an alternative candidate from the remaining beam outputs.

Through this post-processing mechanism, the system mitigates execution failures but also enhances the practical usability of the model in real-world database query scenarios, where robustness and accuracy are critical.

3. Result and Discussion

3.1. Query Statistics

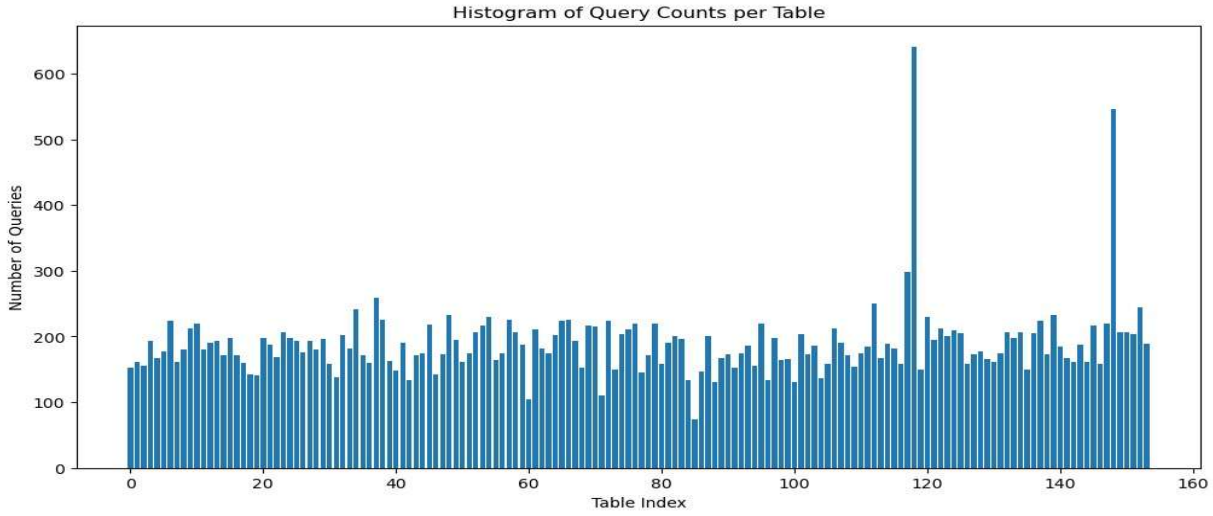
For this study, we focused on a subset of **154** tables from the original **284** tables present in the Osquery schema Table 3.1 summarises the dataset statistics. The constructed synthetic dataset consists of **47,306** samples, with a designated test set comprising **472** queries. The training dataset exhibits a diverse difficulty-level distribution, including **20,365** medium, **15,897** easy, **6,890** extra-hard, and **4,154** hard queries, ensuring balanced representation across varying levels of query complexity. On average, each natural language query is associated with **1.59** tables, indicating moderate query-table interactions.

Furthermore, approximately **53.7%** of the queries involve aggregation operations, while **23.3%** include subqueries, and **14.4%** contain Common Table Expressions (CTEs). In total, **25,393** queries include aggregation, **11,006** contain subqueries, and **6,833** incorporate CTEs. The dataset comprises **1,954,042** tokens, with individual queries ranging from a minimum of **5** tokens to a maximum of **323** tokens, and an average length of **41.31** tokens per query.

Additionally, **6,833** unique SQL skeletons were identified, alongside **93** distinct SQL functions, highlighting the syntactic and functional diversity of the generated corpus.

Table 3.1: Aggregate SQL Query Statistics

Statistic	Value
Total Queries	47,306
Average JOINs per Query	0.48
Average CTEs per Query	0.15
Average Subqueries per Query	0.10
Average Tokens per Query	23.19
Average Aggregations per Query	0.45
Queries with HAVING	13,301
Queries with GROUP BY	22,043
Queries with ORDER BY	11,208



3.2. Retrieval Evaluation

We conducted a series of experiments to evaluate the most effective methods for **schema linking** within the **NL2SQL framework**, focusing on both the **embedding-based retriever** and the **cross-encoder reranker**. The baseline retrieval model achieved a **Recall@5** of **0.4270**. After fine-tuning, the retriever's performance improved significantly, reaching a **Recall@5** of **0.4910** and a **Mean Reciprocal Rank (MRR)** of **0.8388**.

We evaluated several pre-trained cross-encoder models using three key criteria: Full Recall, Mean Reciprocal Rank (MRR), and inference latency, with the goal of identifying the most effective reranker for the table retrieval task. As presented in Table 3.2, the ms-marco-MiniLM-L-6-v2 model demonstrated the best trade-off between effectiveness and efficiency, achieving a **Full Recall** of **0.484**, an **MRR** of **0.8278**, and an average inference time of **45.12 ms per query**. In the retrieval pipeline, the initial retriever selects the top-20 candidate tables, which are subsequently reranked by the cross-encoder to produce the final top-5 results.

Table 3.2: Comparison of Cross-Encoder Models for Reranking

Model	Full Recall	MRR Avg.	Time (ms)
cross-encoder/ms-marco-MiniLM-L-6-v2	**0.484**	**0.8278**	**45.12**
cross-encoder/ms-marco-MiniLM-L-12-v2	0.481	0.8259	69.17
cross-encoder/ms-marco-electra-base	0.455	0.7828	167.54

3.3. NL2SQL Translation Model

To evaluate the accuracy of our NL2SQL model, we employ **Exact Match Accuracy** and **Component Matching Accuracy** as our primary metrics. Although *Execution Accuracy* is a widely adopted standard metric for assessing NL2SQL model performance, it was not feasible to use in our case. This limitation arises because our model operates on the OSQuery schema, where real table values are unavailable for every table instance. Consequently, it is not possible to execute the generated SQL queries to verify whether their results align with the ground truth answers.

The **Exact Match** metric measures the proportion of predicted SQL queries that perfectly match the gold (reference) SQL queries in both structure and content. However, since small variations in SQL syntax can lead to functionally equivalent queries, Exact Match alone may not fully reflect the semantic correctness of the generated SQL.

To provide a more nuanced assessment, we also use **Component Matching**, which evaluates the correctness of individual SQL components—such as the SELECT, WHERE, GROUP BY, and ORDER BY clauses—independently. This component-level evaluation helps capture partial correctness and provides deeper insight into the model's ability to generate accurate substructures of SQL queries.

Table 4.3: Performance comparison of NL2SQL models trained on the OSQuery schema using Exact Match and Component Matching metrics.

Model	Exact Match (%)	Component Matching (%)	Parameters
CodeT5-base	32.4	55.8	220M
CodeT5-large	38.7	61.2	770M
Qwen1.5B	44.3	67.8	1.5B
Qwen7B	**47.9**	**70.6**	7B

4. Conclusion and Future Work

Phase 1 of our project focused on establishing the correctness of our approach to generating SQL queries. This initial phase demonstrated that our method is capable of producing syntactically and semantically valid queries. However, we identified several limitations that we plan to address in future work.

Improving Value Grounding

One of the key challenges observed was in **value grounding**. In the current implementation, we use placeholders for values (e.g., constants in WHERE clauses), which limits the practical applicability of the generated queries. To improve this, we plan to incorporate a Named Entity Recognition (NER) model for more accurate value extraction and grounding. Alternatively, we may provide the model with explicit value annotations during training to better learn value associations.

Enhancing Conditional Logic with Reinforcement Learning

We aim to enhance the model's performance in generating conditional logic in WHERE clauses. For this, we are exploring the use of **reinforcement learning (RL)** techniques. RL could help optimize query generation by providing feedback on the accuracy and relevance of generated conditions, thereby improving the model's ability to construct meaningful and executable SQL queries.

Dataset Augmentation

Another future direction involves **dataset augmentation**. While the current dataset provides a solid foundation for training and evaluation, we plan to increase its size and diversity through automated augmentation techniques. This expansion will expose the model to a broader range of query structures and linguistic variations, ultimately improving its generalization and robustness.

Integrated Evaluation of Retrieval and Generation Models

In this phase, the **retrieval model** and the **generation model** were evaluated separately to establish their independent effectiveness. In future iterations, we plan to perform a joint evaluation and fine-tuning process, enabling the models to operate in an end-to-end fashion. This integrated assessment will provide a more realistic measure of overall system performance and facilitate smoother interaction between retrieval and generation components.

Knowledge Distillation for Model Efficiency

Finally, we plan to apply **knowledge distillation** using our current 7B model as a teacher. This approach will enable us to train a smaller, more efficient student model that retains much of the performance of the larger model while being more suitable for deployment in resource-constrained environments.

5. Reference

[1] Gao, C., Li, B., Zhang, W., Lam, W., Li, B., Huang, F., Si, L., and Li, Y. (2025). Towards generalizable and robust text-to-sql parsing.

[2] Guo, J., Zhan, Z., Gao, Y., Xiao, Y., Lou, J.-G., Liu, T., and Zhang, D. (2025). Towards complex text-to-sql in cross-domain databases with intermediate representation.

[3] Li, B., Luo, Y., Chai, C., et al. (2025a). The dawn of natural language to sql: Are we fully ready?

[4] Li, B., Luo, Y., Chai, C., Zhao, H., Chen, Q., and Yu, T. (2024a). The dawn of natural language to sql: Are we fully ready? *arXiv preprint arXiv:2406.01265*.

[5] Li, H., Wu, S., Zhang, X., Huang, X., Zhang, J., Jiang, F., Wang, S., Zhang, T., Chen, J., Shi, R., Chen, H., and Li, C. (2024b). Omnisql: Synthesizing high-quality text-to-sql data at scale.

[6] Li, H., Zhang, J., Li, C., and Chen, H. (2025b). Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql.

[7] Li, H., Zhang, J., Liu, H., Fan, J., Zhang, X., Zhu, J., Wei, R., Pan, H., Li, C., and Chen, H. (2025c). Codes: Towards building open-source language models for text-to-sql.

[8] Li, J., Hui, B., Cheng, R., Qin, B., Ma, C., Huo, N., Huang, F., Du, W., Si, L., and Li, Y. (2025d). Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing.

[9] Li, J., Hui, B., Qu, G., Zhang, Z., Xu, R., He, B., Chen, Q., and Yu, T. (2024c). Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

[10] Lin, X. V., Socher, R., and Xiong, C. (2025). Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[11] Liu, X., Shen, S., Li, B., Ma, P., Jiang, R., Zhang, Y., Fan, J., Li, G., Tang, N., and Luo, Y. (2025). A survey of nl2sql with large language models: Where are we, and where are we going?

[12] Pourreza, M. and Rafiei, D. (2025). Din-sql: Decomposed in-context learning of textto-sql with self-correction.

[13] Scholak, T., Schucher, N., and Bahdanau, D. (2025). Picard: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[14] Talaei, S., Pourreza, M., Chang, Y., Mirhoseini, A., and Saberi, A. (2025). Chess: Contextual harnessing for efficient sql synthesis.

[15] Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Zhang, Q., Yan, Z., and Li, Z. (2025a). Mac-sql: A multi-agent collaborative framework for text-to-sql.

Annexure B

अनुप्रयुक्त कृत्रिम बुद्धिमत्ता हेतु टेक्नोक्राफ्ट केंद्र
एच.एस.एस. परिशिष्ट
भारतीय प्रौद्योगिकी संस्थान मुंबई
पवई, मुंबई-400 076, भारत

Technocraft Centre for Applied Artificial Intelligence
HSS- Annexe
Indian Institute of Technology Bombay
Powai, Mumbai-400 076, India

Phone: (+91-22) 2576 6662 (O)
Fax : (+91-22) 2572 6875
Email : office.tcaai@iitb.ac.in
Web : www.tcaai.iitb.ac.in

IIT Bombay

18th June 2025

Nitish Kumar,

I am happy to offer you internship position at Technocraft Center for Applied Artificial Intelligence at IIT Bombay. The topic of internship is cybersecurity. During the Internship, you will work on data collection, analysis and training of AI algorithms to detect cyber threat in IoT networks and end points. The terms of the internship are as follows:

1. The internship duration is from 1st July 2025 to 1st March 2026.
2. The work done during the internship can be part of the your MTech thesis. You will be encouraged to publish the work as per the guidelines of your parent institute.
3. No accommodation is guaranteed on IIT Bombay campus during the entire duration of the internship. You have to make your own arrangement.
4. Monthly attendance details will be shared with DIAT to aid in AICTE M.Tech scholarship release.

We hope this internship opportunity will help you to learn and address various issues in cybersecurity. As a response to this letter, please confirm your acceptance of the offer with an endorsement from your advisor.

Looking forward to your joining as an intern at TCA2I, IIT Bombay.

Sincerely,

Manjesh Kumar Hanawal
Professor-In-Charge, TCA2I
IIT Bombay

टेक्नोक्राफ्ट अनुप्रयुक्त कृत्रिम बुद्धिमता केंद्र
Technocraft Centre for Applied Artificial Intelligence
भारतीय प्रौद्योगिकी संस्थान मुंबई
Indian Institute of Technology Bombay
पवई, मुंबई-४०००७६/Powai, Mumbai-400076

Annexure C

None

Annexure D

COURSE COMPLETION CERTIFICATE

The certificate is awarded to

Nitish Kumar

for successfully completing the course

Fine Tuning Large Language Models

on November 14, 2025

Infosys | Springboard

Congratulations! You make us proud!

Satheesha B.N.
Satheesha B. Nanjappa
Senior Vice President and Head
Education, Training and Assessment
Infosys Limited

Issued on: Friday, November 14, 2025
To verify, scan the QR code at https://verify.onwingspan.com