CAPSTONE THESIS REPORT

DUAL-MODEL FRAMEWORK FOR NATURAL LANGUAGE TO SQL TRANSLATION IN MIMIC-IV HEALTHCARE DATABASES

Aaryan Nagpal

Supervisor: Dr. Lipika Dey

Student ID: 1020211786

Department of Computer Science

Ashoka University

Spring 2025

ABSTRACT

This study explores natural language to SQL conversion for the MIMIC-IV healthcare database, addressing the gap between clinical expertise and database querying capabilities. A novel two-stage pipeline architecture for converting natural language questions into executable SQL queries was implemented, specifically optimized for medical database structures.

The research methodology involved a systematic evaluation of nine language models across three categories: general-purpose LLMs, medical-specialized models, and SQL-focused models. The experimental methodology followed a systematic progression through four distinct development phases: zero-shot inferencing, few-shot learning, schema-aware prompting, and parameter fine-tuning. Quantitative analysis of the EHRSQL 2024 dataset revealed superior performance from heterogeneous model combinations, with the final implementation utilizing a fine-tuned Qwen 2.5 generator and Phi 4 validator in a conditional architecture achieving 30% result accuracy and 60% structural accuracy.

A notable finding was the inverse performance pattern between components, where generation models experienced regression during schema-aware phases while validation models improved, indicating distinct information processing requirements for these complementary tasks. The pipeline incorporates selective validation mechanisms to optimize computational efficiency while maintaining query accuracy.

This study aims to establish a foundation for domain-specific text-to-SQL conversion in healthcare, demonstrating effective techniques for democratizing access to clinical data through natural language interfaces towards future advancements in medical database accessibility.

Keywords MIMIC-IV, Natural Language to SQL, Healthcare Data Accessibility, Query Validation Pipeline

Acknowledgement

I would like to express my deepest gratitude to my advisor and mentor, Dr. Lipika Dey from the Computer Science Department at Ashoka University, for her invaluable guidance, insightful contributions, and active involvement throughout the course of this project. Her thoughtful feedback played a crucial role in shaping the direction and outcomes of my research, without which this project would not have been possible.

I am grateful to Dr. Mayank Garg from the Koita Center for Digital Health (KCDH-A) for his biological insights and support. My thanks also extend to the Computer Science Department at Ashoka University, the Koita Center for Digital Health (KCDH-A), and Mphasis Lab for providing the computational resources necessary for the execution of this project, which greatly contributed to its successful completion.

The successful implementation of this project would not have been possible without the use of essential technologies and libraries. I extend my appreciation to the developers of pandas, NumPy, and SQLite for data manipulation; Hugging Face Transformers and llama-cpp for large language model implementations; tqdm and concurrent.futures for efficient batch processing; and matplotlib, and plotly for visualization and analysis of results. These tools formed the technological foundation that enabled the development and evaluation of my text-to-SQL conversion pipeline.

I am especially grateful to my peers, Maanas Kejriwal and Tarush Chaudhary, whose thoughtful discussions, constructive feedback, and unwavering camaraderie provided both intellectual stimulation and moral support throughout this research journey.

Lastly, I would like to express my heartfelt appreciation to my family and friends for their unwavering support, encouragement, and understanding throughout the course of this research. Their belief in me kept me motivated and focused, and I am deeply grateful for their constant and unconditional presence.

Certificate

This is to certify that the capstone thesis titled **"Dual-Model Framework for Natural Language to SQL Translation in MIMIC-IV Healthcare Databases"** submitted by **Aaryan Nagpal (1020211786)** has been carried out under my supervision and has been found satisfactory.

Dr. Lipika Dey
Advisor

Contents

1 Introduction **1**

2 Background and Motivation **2**

2.1 Background 2

2.1.1 Clinical Databases and Accessibility Barriers 2

2.1.2 Text-to-SQL Systems in Medical Domains 2

2.2 Motivation 3

2.2.1 Democratizing Healthcare Data Access 3

2.2.2 Addressing Domain-Specific Complexities 3

3 Literature Survey **4**

3.1 Previous Literature 4

3.1.1 From Natural Language to SQL: Review of LLM-based Text-to-SQL Systems Mohammadjafari et al. (2025) 4

3.1.2 Towards Understanding the Generalization of Medical Text-to-SQL Models and Datasets Tarbell et al. (2023) 5

3.1.3 Retrieval Augmented Text-to-SQL Generation for Epidemiological Question Answering Using Electronic Health Records Ziletti and D'Ambrosi (2024) 5

3.1.4 Deep Learning Driven Natural Languages Text-to-SQL Query Conversion: A Survey Kumar et al. (2022) 5

3.1.5 Large Language Model Enhanced Text-to-SQL Generation: A Survey Zhu et al. (2024) 6

3.1.6 MIMIC-IV, a Freely Accessible Electronic Health Record Dataset Johnson et al. (2023) 6

3.2 MIMIC-IV Database 7

3.3 Gap Analysis 7

3.3.1 Insufficient Validation Mechanisms 7

3.3.2 Limited Evaluation in Medical Contexts 8

3.4 Research Questions 8
3.4.1 Development and Evaluation of a Two-Stage Pipeline for Medical Text-to-SQL Conversion 8
3.4.2 Comparative Evaluation of Language Models for Medical Text-to-SQL on MIMIC-IV 8

4 Problem Statement and Objectives **9**
4.1 Problem Statement 9
4.2 Objectives 9
4.2.1 Comparative Model Analysis for MIMIC Query Tasks 9
4.2.2 Optimized Two-Stage Pipeline Development 10

5 Scope **11**
5.1 Data and Resource Constraints 11
5.2 Methodological Boundaries 11
5.3 Research Compliance 11

6 Methodology, Work Done and Challenges **12**
6.1 PhysioNet Training 13
6.2 Model Selection 13
6.3 Dataset Analysis and Preparation 14
6.3.1 Dataset Characteristics 14
6.3.2 MIMIC-IV Database Adaptation 15
6.3.3 Dataset Analysis 16
6.4 Query Comparison Methodology 22
6.4.1 Execution-Based Metrics 22
6.4.2 Structural Similarity Analysis 22
6.4.3 Component-Level Analysis 23
6.4.4 Execution Plan Analysis 23
6.4.5 Error Categorization 24

6.4.6 Composite Evaluation Score 24
6.5 Generation Model (M1) Development 25
6.5.1 Zero-Shot Implementation 25
6.5.2 Few-Shot Refinement 29
6.5.3 Schema-Aware Enhancement 33
6.5.4 Fine-Tuning Implementation 37
6.6 Validation Model (M2) Development 41
6.6.1 Zero-Shot Implementation 42
6.6.2 Few-Shot Refinement 45
6.6.3 Schema-Aware Enhancement 49
6.6.4 Fine-Tuning Implementation 53

7 Results and Discussions 58
7.1 Pipeline Configuration Evaluation 58
7.2 Model Performance Progression 62
7.2.1 Composite Score Evolution 62
7.2.2 Overall Performance Trends 65

8 Conclusions 69

9 Future Work and Extensions 70
9.1 Architectural and Computational Enhancements 70
9.2 Cross-Database Adaptability 70
9.3 User Interface Enhancements 70
9.4 Evaluation and Integration 71
9.5 Dataset Enhancements 71

10 Appendix 72
10.1 Figures, Code and Tables from 6 72
10.1.1 Modified Schema from 6.3.2 72

10.1.2 Sample Execution Plan from 6.4.4 73
10.2 Listings from 7 73
10.2.1 Examples of NLQ-to-SQL Conversion from 7.1 73

References **76**

1 Introduction

Electronic Health Records (EHRs) contain valuable clinical data that remains underutilized due to a critical skills gap: clinicians lack database query expertise, while data engineers lack medical knowledge. This challenge is compounded by the scarcity of medical databases, with the few that exist primarily designed for operational purposes, billing, and insurance rather than research or clinical decision support.

This disconnect hinders healthcare innovation and clinical decision-making by creating a critical barrier to timely data access. For complex databases like MIMIC-IV, the stakes are particularly high as query errors can lead to incorrect clinical conclusions with potentially serious consequences.

MIMIC-IV's complex schema, with its interrelated tables, specialized terminology, coding systems, and temporal relationships, presents unique challenges. While recent advances in Large Language Models (LLMs) have opened new avenues for natural language processing, facilitating effective communication between these models and databases remains difficult, particularly for obtaining correct values in objective clinical queries.

This thesis develops a two-stage natural language to SQL conversion pipeline for MIMIC-IV, progressing from zero-shot to fine-tuning approaches, with a validation stage that verifies generated queries. The research aims to explore a practical framework for medical database querying and questioning while providing insights into effective prompting strategies and model architectures for domain-specific text-to-SQL conversion.

Through comprehensive evaluation metrics combining execution accuracy and structural similarity analysis, this thesis demonstrates how progressive refinement of prompting strategies and targeted fine-tuning substantially improve performance in specialized domains while maintaining computational efficiency.

This thesis aims to bridge the gap between natural language and SQL in healthcare settings to democratize access to clinical data and potentially accelerate advances in medical research and patient care.

2 Background and Motivation

2.1 Background

2.1.1 Clinical Databases and Accessibility Barriers

Electronic Health Records (EHRs) have revolutionized healthcare data management by establishing structured repositories of clinical information. These database systems capture critical patient information, including demographics, diagnoses, procedures, medications, and laboratory measurements, thereby supporting both research and clinical decision-making. Such databases primarily rely on Structured Query Language (SQL) to fetch data, whose utilization requires precise syntax and a thorough understanding of database schema relationships.

However, this technical requirement has created a fundamental disconnect in healthcare informatics: clinicians and medical researchers with the most domain expertise often lack the technical skills to directly extract and analyze relevant data. Meanwhile, data engineers and informaticians proficient in SQL may lack the medical context necessary to formulate clinically meaningful queries.

This barrier has significantly impeded data-driven healthcare improvement, particularly in time-sensitive clinical environments where rapid information retrieval could directly impact patient outcomes.

2.1.2 Text-to-SQL Systems in Medical Domains

Advancements in Natural Language Processing (NLP) have facilitated significant developments in translating human language to formal query languages. Text-to-SQL conversion in healthcare contexts presents domain-specific challenges, including complex medical terminology interpretation, temporal reasoning requirements, and stringent accuracy demands essential for clinical applications.

To address these hurdles, specialized Text-to-SQL frameworks incorporate clinical ontologies, entity-linking strategies, and schema-aware encoders to accurately map medical concepts and temporal expressions into structured database queries.

Recent research further explores fine-tuning large pre-trained language models on electronic health record (EHR) schemas, and enforcing strict protocols to meet regulatory standards and ensure reliable performance in high-stakes clinical decision support systems.

2.2 Motivation

2.2.1 Democratizing Healthcare Data Access

The proliferation of EHRs has generated vast repositories of structured clinical data with significant potential for advancing patient care and medical research. However, while healthcare professionals possess the clinical knowledge to formulate relevant questions, they often lack the technical expertise required to navigate complex database systems like MIMIC.

In this context, by developing a Text-to-SQL framework for the clinical databases, clinicians and researchers can interact with patient data using natural language.

By translating plain-language queries into accurate and schema-aware SQL, such frameworks can eliminate technical barriers, reduce query formulation time, and enable more inclusive, data-driven decision-making in clinical and research settings, representing a critical step toward fully utilizing the wealth of information contained in modern healthcare systems.

2.2.2 Addressing Domain-Specific Complexities

Healthcare databases present unique challenges beyond standard data access problems, including intricate temporal relationships, specialized terminology, and complex relational schemas, coupled with the high-stakes nature of medical data analysis demanding exceptional query accuracy as errors may lead to incorrect clinical conclusions.

Current text-to-SQL systems demonstrate insufficient performance when handling these domain-specific complexities, particularly with nested queries, bundled constraints, and specialized medical vocabularies, underscoring the necessity for systems specifically engineered to accommodate healthcare data structures and terminology.

The design of an effective text-to-SQL system to handle the domain-specific complexities of healthcare databases can lead to the development of a framework specifically tailored to the clinical domain.

Such a framework, capable of accurately interpreting user intent and generating precise queries, would enable reliable data retrieval, critical for research and decision-making in high-stakes medical environments ultimately bridging a significant gap between Natural Language Processing and medicine.

3 Literature Survey

Natural language to SQL conversion has emerged as a critical area of research at the intersection of natural language processing and database systems, aiming to democratize data access by enabling users without specialized SQL knowledge to query databases using everyday language. Recent advancements in large language models (LLMs) have significantly improved the capability and accuracy of text-to-SQL systems, pushing the boundaries of what is possible in this field of research.

However, applying this research in medical domain presents unique challenges for text-to-SQL systems due to complex domain-specific terminology, intricate database schemas, and high stakes for query accuracy.

This literature survey examines the evolution of text-to-SQL approaches with a particular focus on medical applications. The review analyzes recent methodological advancements, evaluation frameworks, and persistent challenges in the field.

By systematically examining existing research, this section establishes the foundation for identifying significant research gaps that this thesis addresses, particularly in developing robust natural language interfaces for complex medical databases.

3.1 Previous Literature

3.1.1 From Natural Language to SQL: Review of LLM-based Text-to-SQL Systems Mohammadjafari et al. (2025)

Objective

This comprehensive review examines the architectural innovations, training methodologies, and performance characteristics of LLM-based text-to-SQL systems, comparing traditional neural approaches with emerging large language model frameworks.

Relevance

The review provided a critical analysis of how prompt engineering, context-aware processing, and few-shot learning influence SQL generation quality. Their framework for comparing model architectures informed our approach to evaluating different LLMs on the MIMIC dataset. However, the review primarily focuses on general-domain datasets like Spider and WikiSQL, highlighting the need for **domain-specific analysis** in healthcare contexts.

3.1.2 Towards Understanding the Generalization of Medical Text-to-SQL Models and Datasets Tarbell et al. (2023)

Objective

The research specifically investigates the generalization capabilities of text-to-SQL models in medical contexts, analyzing how model performance varies across different medical database schemas and question types.

Relevance

This research most closely aligns with our research objectives, providing a detailed analysis of the challenges in medical text-to-SQL conversion. Their evaluation of model performance across diverse medical schemas highlighted the difficulty of generalization in this domain. However, their research did not explore validation mechanisms for ensuring query correctness—an important shortcoming given the high-stakes nature of medical applications—which our proposed methodology aims to address.

3.1.3 Retrieval Augmented Text-to-SQL Generation for Epidemiological Question Answering Using Electronic Health Records Ziletti and D'Ambrosi (2024)

Objective

This research introduces a retrieval-augmented approach for generating SQL queries from epidemiological questions using electronic health records, demonstrating how domain-specific retrieval enhances query accuracy.

Relevance

The retrieval augmentation methodology mentioned directly addresses the challenge of domain knowledge incorporation in medical text-to-SQL systems. The research showed how integrating retrieval-augmented generation (RAG) into text-to-SQL systems significantly enhances performance in translating epidemiological questions into SQL queries over EHR data, particularly within small, domain-specific biomedical datasets.

3.1.4 Deep Learning Driven Natural Languages Text-to-SQL Query Conversion: A Survey Kumar et al. (2022)

Objective

This survey comprehensively examined the transition from traditional rule-based approaches to deep learning methodologies for text-to-SQL conversion, analyzing architectural innovations and performance benchmarks across various application domains.

Relevance

The research contained a systematic categorization of text-to-SQL approaches based on their underlying architectures and learning paradigms. The analysis of encoder-decoder frameworks and attention mechanisms informs our model selection process, even though it does not specifically address the challenges of medical data querying.

3.1.5 Large Language Model Enhanced Text-to-SQL Generation: A Survey Zhu et al. (2024)

Objective

This survey examined how recent advances in large language models have transformed text-to-SQL generation, focusing on prompt engineering strategies, few-shot learning capabilities, and the integration of external knowledge sources.

Relevance

The survey provided crucial insights into leveraging the advanced reasoning capabilities of LLMs for SQL generation. The analysis of prompt engineering strategies directly informs our approach to designing zero-shot and few-shot prompts for medical query generation. The survey acknowledges limitations in domain-specific applications but does not propose concrete solutions for medical contexts, which our methodology builds upon by incorporating schema compliance checks.

3.1.6 MIMIC-IV, a Freely Accessible Electronic Health Record Dataset Johnson et al. (2023)

Objective

This paper detailed the transition from MIMIC-III to MIMIC-IV, discussing how the structure of the data has changed. It introduces the table structure of MIMIC-IV and gives a high-level overview of the table connections. The paper further discusses patient trajectory graphs, contributing to an understanding of longitudinal data in MIMIC-IV

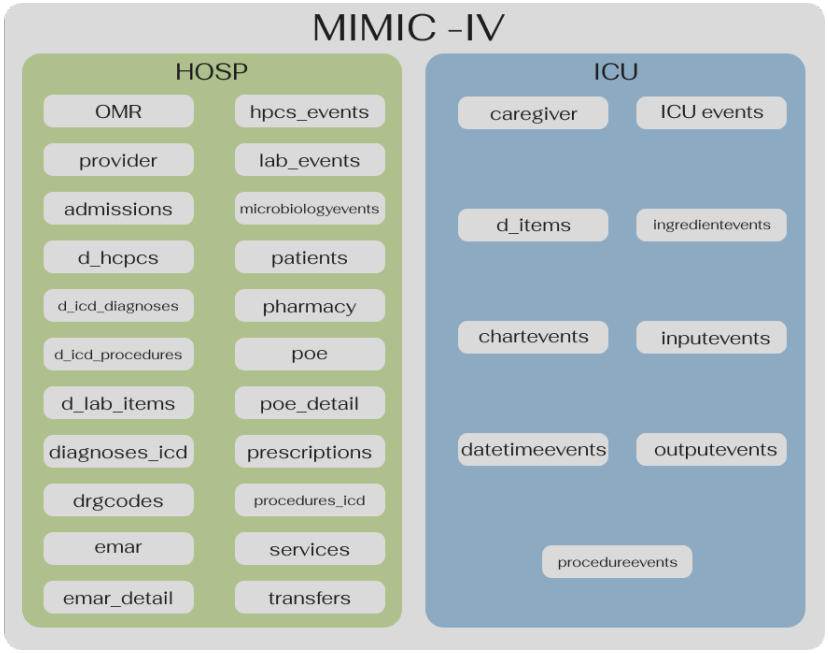
Relevance

The description of MIMIC-IV and its structure was helpful for a database of this size. Additionally, most prior literature focuses on MIMIC-III, making this paper valuable for contextualizing research in the newer dataset. Finally, the paper covered challenges concerning data sparsity and missing data, which were introduced before accessing the data.

3.2 MIMIC-IV Database

MIMIC-IV is an electronic health records database containing 383,220 patient admissions (2008–2019) from Beth Israel Deaconess Medical Center. Its modular relational structure comprises Core, Hosp, ICU, ED, and CXR components with complex schema relationships.

With 31 tables and hundreds of attributes connected by dozens of foreign keys, MIMIC-IV far exceeds usual benchmarks, being more clinically and globally relevant than MIMIC-III, with updated coding standards (ICD-10, LOINC) that align with international healthcare systems and broader inpatient coverage beyond ICU data.



Its intricate temporal dependencies, specialized medical terminology, and cross-module join requirements further compound challenges for text-to-SQL systems.

3.3 Gap Analysis

The comprehensive examination of existing literature reveals two critical research gaps in medical text-to-SQL conversion that this thesis directly addresses.

3.3.1 Insufficient Validation Mechanisms

Current text-to-SQL systems typically generate queries in a single pass, without post-generation checks for schema compliance or semantic integrity. In the MIMIC-IV context, unchecked JOINs or WHERE clauses risk producing misleading cohorts or timelines. As Tarbell et al. (2023) note, "current models lack mechanisms to verify generated queries

against complex medical schemas." Introducing a lightweight validation that enforces syntactical and logical consistency could help catch errors before they impact downstream analyses.

3.3.2 Limited Evaluation in Medical Contexts

Most surveys and benchmarks (e.g., Zhu et al. (2024); Mohammadjafari et al. (2025)) assess text-to-SQL on general-domain databases with modest schemas. These evaluations overlook the nested JOINs, specialized code systems (ICD, HCPCS), and strict temporal relationships characteristic of EHR data. A focused comparison of general and domain-tuned language models on representative MIMIC-IV queries would shed light on how well different approaches handle these real-world challenges and point toward targeted enhancements such as schema-aware validation or temporal reasoning.

3.4 Research Questions

3.4.1 Development and Evaluation of a Two-Stage Pipeline for Medical Text-to-SQL Conversion

The project evaluates the effect of introducing a secondary LLM–based validation pass on SQL generated for MIMIC-IV. Execution success rate, semantic accuracy, and result similarity are measured on a fixed set of clinical queries before and after validation, and inference latency is recorded to determine whether this two-stage design improves reliability without prohibitive overhead.

Error analysis highlights issues such as missing JOIN conditions or misplaced filters, illustrating the validator's contribution to reducing syntactic and logical failures. These metrics aim to quantify practical gains in correctness and efficiency when interacting with the complex MIMIC-IV schema.

3.4.2 Comparative Evaluation of Language Models for Medical Text-to-SQL on MIMIC-IV

The project benchmarks both general-purpose and medical-tuned LLMs on a curated suite of MIMIC-IV queries spanning cohort selection, temporal aggregations, and code-system filters. For each model, syntactic validity, execution accuracy against ground-truth results, and end-to-end latency are captured to identify architectures that optimally balance precision and performance in clinical SQL generation. Detailed comparison of error types and inference speed will inform best practices for deploying text-to-SQL systems in electronic health record environments.

4 Problem Statement and Objectives

4.1 Problem Statement

Hospitals generate massive amounts of Electronic Health Record (EHR) data, and systems like MIMIC-IV hold tremendous potential for improving patient care and advancing medical research. However, effectively querying such complex medical databases can be difficult due to several key challenges:

1. **Technical and Clinical Knowledge Gap**
   Clinicians often lack the SQL expertise needed to query structured data, while database professionals may not fully grasp the medical context. This disconnect prevents effective and timely use of healthcare data, slowing down decision-making and research.
2. **Limitations of Existing Methods**
   Most current text-to-SQL systems generate queries in a single step without validating the schema or semantics, leading to potential errors. In healthcare, even small mistakes can lead to incorrect patient cohorts or misleading clinical insights, which poses significant risks for decision-making.
3. **Specific Challenges with MIMIC-IV**
   MIMIC-IV's complex schema, multi-table joins, and specialized medical codes require precise querying. The intricate relationships between tables and strict temporal constraints make it difficult for existing systems to generate accurate queries, highlighting the need for tailored solutions in the healthcare domain.

4.2 Objectives

4.2.1 Comparative Model Analysis for MIMIC Query Tasks

The first objective is to conduct a comprehensive analysis of different language models on MIMIC-IV natural language queries to assess their performance across various implementation techniques:

1. Evaluate both general-purpose and medical-domain LLMs (including Phi-4, Qwen 1.5/2.5, MedQwen, Meditron, Medalpaca, and specialized SQL models)
2. Compare performance under zero-shot and few-shot prompting regimes
3. Assess model capabilities across two distinct tasks:
   (a) Query generation: Converting natural language questions to executable SQL

   (b) Query validation: Verifying and correcting generated SQL for schema compliance and logical consistency

4. Develop robust evaluation metrics focused on execution success, result accuracy, and clinical relevance

4.2.2 Optimized Two-Stage Pipeline Development

Building on the model analysis findings, the second, and primary, objective is to develop and implement an efficient two-stage natural language to SQL pipeline specifically optimized for the MIMIC-IV database:

1. Select the best-performing models for both generation and validation stages based on comparative analysis
2. Fine-tune selected models to enhance performance on medical text-to-SQL tasks
3. Design an integrated pipeline architecture that balances accuracy with practical efficiency
4. Implement schema-aware validation mechanisms to ensure query correctness
5. Evaluate the end-to-end pipeline against baseline approaches using clinically relevant metrics

5 Scope

5.1 Data and Resource Constraints

1. **Dataset Scope:**
   The project exclusively utilizes the EHRSQL 2024 competition dataset by Lee et al. (2024), containing paired natural language queries and SQL for a modified version of the MIMIC-IV database[1]. The specific database modifications, necessary for query compatibility and execution, are detailed in Section 6.
2. **Hardware Limitations:**
   All experimentation is constrained to a single GPU with 24GB memory, which limits model selection to those that can operate within this boundary. This constraint is particularly significant for the two-stage pipeline where both generation and validation models must simultaneously fit within available memory.
3. **Model Selection Parameters:**
   The research is restricted to non-proprietary language models including domain-specific medical models and SQL-specialized variants. Proprietary models and those exceeding hardware constraints are excluded from the analysis.

5.2 Methodological Boundaries

1. **Evaluation Framework:**
   Evaluation is primarily limited to result matching between generated and ground truth queries, with secondary assessment of structural similarity when results don't match. This includes table access, column selection, and query component analysis using custom evaluation metrics.
2. **Implementation Focus:**
   The project focuses solely on backend implementation without user interface components. Queries exceeding predefined execution thresholds are terminated to prevent system freezing, potentially limiting complete analysis of highly complex queries.

5.3 Research Compliance

Research operates within permitted use guidelines for MIMIC-IV data, with results and analyses conducted in private environments per data usage agreements. While methodology is shareable, specific data structures remain protected.

[1] This dataset was chosen as the only available resource specifically targeting MIMIC-IV with complex clinical queries.

6 Methodology, Work Done and Challenges

This research implemented a sequential approach to develop a **two-stage natural language to SQL conversion pipeline** specifically for MIMIC-IV, the architecture for which can be seen in Figure 1.

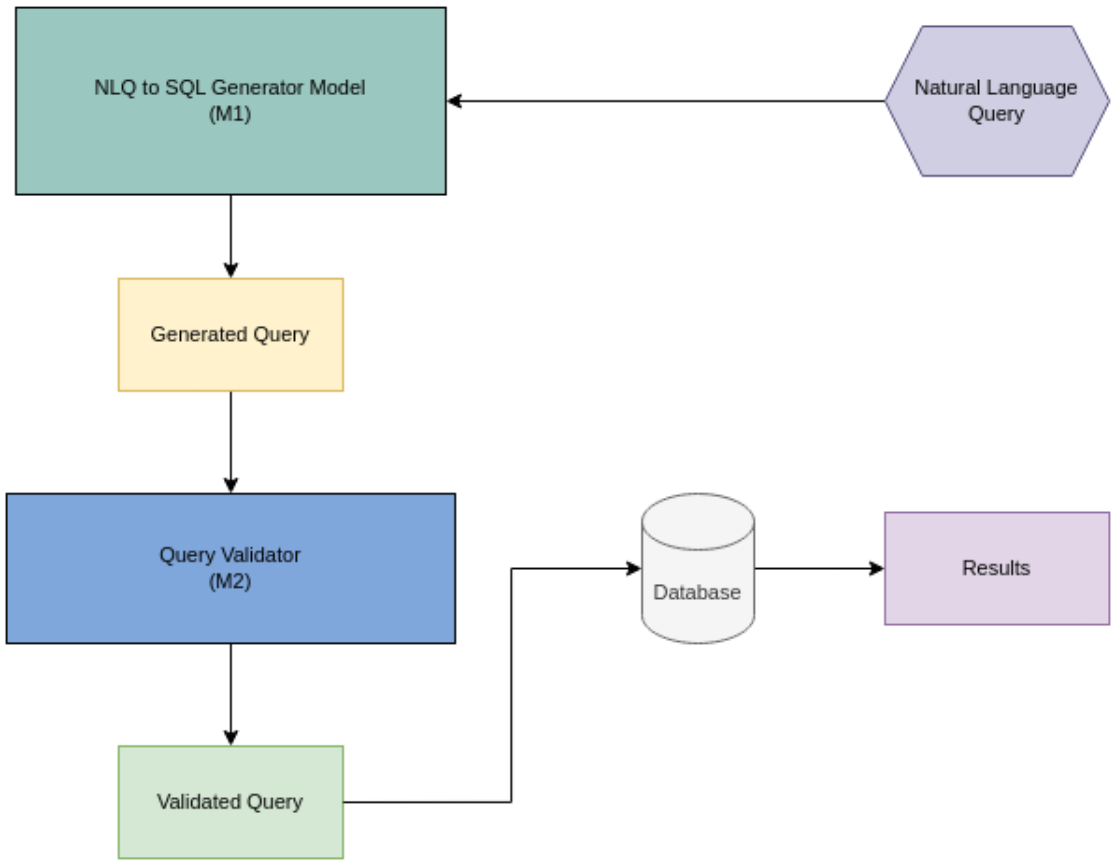


Figure 1: Dual Framework Architecture

The methodology progressed through three distinct phases:

1. **Resource Identification and Preparation.**
   Initially, attempts were made to create a custom dataset through rule-based natural language generation from randomly generated, medically valid queries. This approach was abandoned due to verification challenges and scope limitations. Subsequently, the EHRSQL 2024 competition dataset was identified as a suitable resource containing MIMIC-IV valid queries. Concurrently, candidate models were identified

through literature review and repository exploration, focusing on non-proprietary models compatible with 24GB GPU memory constraints.

2. **Generation Model (M1) Development.**
   Model development progressed systematically from **zero-shot inferencing** to **few-shot learning**, and to **schema-aware prompting**. Each stage involved executing generated queries against MIMIC-IV, analyzing result accuracy and query structure, and selecting top-performing models for advancement. The most promising model was then fine-tuned for enhanced performance.
3. **Validation Model (M2) Development and Integration.**
   A parallel experimental progression focused on developing the validation component, specifically optimizing its ability to identify and correct errors in M1-generated queries. Both components were ultimately integrated into a cohesive pipeline that invoked validation and correction on generated queries to ensure correct results.

6.1 PhysioNet Training

Since the MIMIC-IV dataset is only given access to for people with certain credentials, we were required to pass the CITI Data or Specimens only Research course with a minimum score of 80%. This course contained 13 modules, relating to data ethics and usage agreements. Taking the course ensured compliance with strict ethical standards, which is a prerequisite for working with such sensitive healthcare data.

6.2 Model Selection

Model selection followed a systematic approach balancing technical capabilities with practical implementation constraints. As highlighted by Mohammadjafari et al. (2025), model architecture significantly impacts text-to-SQL performance, particularly for domain-specific applications. Following this insight, candidates were categorized into three strategic classes:

1. **General-purpose LLMs** with strong reasoning capabilities
2. **Medical-domain specialized models** with healthcare knowledge
3. **SQL-specialized models** with query generation expertise

The selection process integrated quantitative metrics from Hugging Face leaderboards with qualitative assessments from the literature surveyed. For each category, three representative models were chosen based on their performance profiles and memory efficiency, as shown in Table 1.

Category	Model	Parameters	GGUF Quantization
General	Phi-4	-	Q5_K_M
	Qwen 1.5	14B	Q5_K_M
	Qwen 2.5	14B	Q5_K_M
Medical	MedQwen Reasoning i1	-	Q5_K_M
	Meditron	7B	Q8
	Medalpaca	13B	Q6_K
SQL-Specialized	Qwen 2 Math	7B	Q8
	DuckDB NSQL	7B	Q8
	SQLCoder	7B	Q5_K_M

Table 1: Selected Models by Category

The inclusion of medical-specialized models aligns with findings by Tarbell et al. (2023) that domain knowledge significantly impacts text-to-SQL performance in healthcare contexts. This strategic diversification allowed potential comparative analysis across model types to determine whether general reasoning capabilities, domain knowledge, or SQL specialization contributed more effectively to medical query generation.

For implementation, **GGUF** (GPT-Generated Unified Format) quantized versions were selected for all models to optimize memory usage while preserving inference capabilities. This approach facilitated model loading within the 24GB GPU constraint, with particular attention to quantization levels for a balanced memory efficiency and performance as noted by Zhu et al. (2024).

6.3 Dataset Analysis and Preparation

For the thesis, the primary dataset that was used was the EHRSQL 2024 competition dataset, released by Lee et al. (2024).

This dataset served as a unique resource for evaluating natural language to SQL conversion systems specifically targeting the MIMIC-IV database, unlike other datasets which focused on the MIMIC-III version, an outdated variant.

6.3.1 Dataset Characteristics

The dataset consisted of paired natural language questions and corresponding SQL queries organized into training, validation, and test sets. The number of records per split are mentioned below in Table 2.

Split	Number of Rows
train	5124
test	1167
valid	1163

Table 2: Caption

Each record followed a three-field structure: a unique identifier, a natural language clinical question, and its corresponding SQL query for the MIMIC-IV database. One such example is showed below in Table 3

id	**question**	**true_query**
7615134b4fa7ff6ddf73b525	Tell me the total number of patients who were discharged from the hospital in 2100	`SELECT COUNT(DISTINCT admissions.subject_id) FROM admissions WHERE admissions.dischtime IS NOT NULL AND strftime('%Y', admissions.dischtime) = '2100'`

Table 3: A sample from the EHRSQL 2024 dataset

The questions represented realistic clinical information needs, ranging from simple patient demographic queries to complex temporal analyses involving multiple tables and nested conditions.

6.3.2 MIMIC-IV Database Adaptation

Detailed comparison between the dataset's expected schema and the standard MIMIC-IV database revealed significant structural differences requiring adaptation. The dataset assumed modifications to the standard MIMIC-IV schema, including:

1. **Column Simplification:** Many tables in the competition schema contained fewer columns than the original MIMIC-IV database. For example, the `d_icd_procedures` table was reduced from containing both `icd_code` and `icd_version` to only including `icd_code`.

2. **Temporal Information Enhancement:** Additional timestamp columns were added to several clinical event tables. Most notably, the `diagnoses_icd` and `procedures_icd` tables incorporated a `charttime` column not present in the standard schema.
3. **Demographic Data Restructuring:** The `patients` table was significantly modified, replacing the `anchor_age` and `anchor_year` system with direct `dob` (date of birth) fields, allowing for more straightforward age calculations.
4. **Additional Tables:** A new `cost` table was introduced to support financial analysis queries, with no direct equivalent in the standard MIMIC-IV schema.
5. **Schema Reduction:** A significant simplification of the overall database structure, retaining only 17 of the original 31 tables in the MIMIC-IV schema. This focused the dataset on the most clinically relevant entities while reducing schema complexity.

To address these modifications, a modified version of the MIMIC-IV database was created by transforming the standard schema to match the expected structure. The schema structure for the modified MIMIC database can be found in Appendix 6.3.2.

6.3.3 Dataset Analysis

Prior to model development, comprehensive exploratory analysis was performed on the training split of the dataset to understand its characteristics and challenges, leading to the subsequent methodological decisions taken, even highlighting key areas requiring attention.

We first analyzed the `question` and `true_query` columns, revealing **8.78%** of entries in the dataset contained questions without any corresponding SQL queries, as seen in Figure 2.

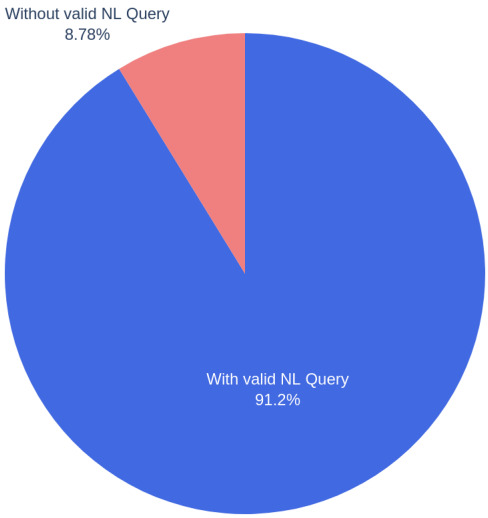


Figure 2: Distribution of Valid and Invalid Questions

The subset with questions having no corresponding queries was part of the design of the dataset, intending to allow learning models to know what questions are possible to query and what aren't[2].

One such analysis performed was a **Word Frequency analysis**, which revealed distinct vocabulary patterns between the two groups.

Questions with SQL predominantly contained terms related to patient data, temporal references, and specific measurable attributes (e.g., "prescribed", "since", "test", "hospital", "first"), while questions without SQL frequently included subjective or interpretive terminology (e.g., "tell", "rooms"), they also often contained references to private or unhosted data (e.g., "history", "dr") that may not be directly represented in the database schema and are thus less amenable to precise query generation, as seen in Figure 3.

[2]This subset was analyzed separately to identify patterns that might explain the absence of SQL formulations.

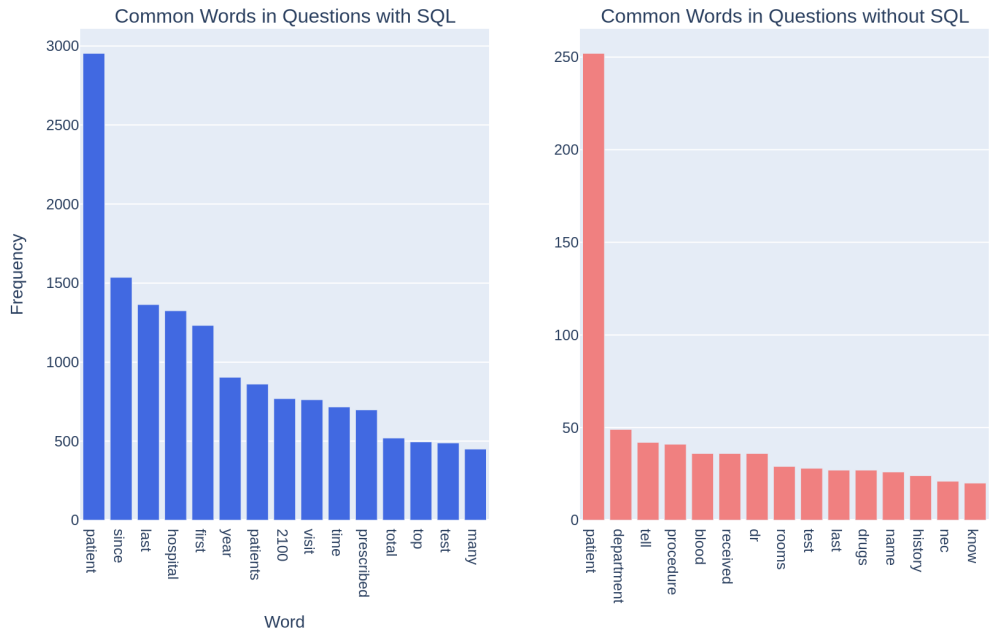


Figure 3: Most Common Words in Questions With and Without SQL

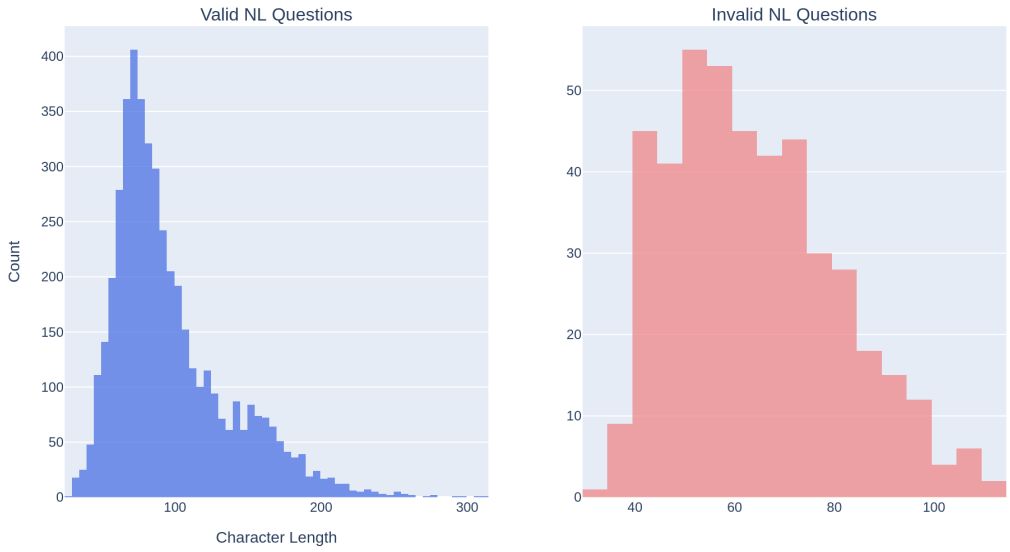


Figure 4: Distribution of Length of Questions

As seen in Figure 4, questions with corresponding SQL queries had a distinctly different length distribution compared to those without, with SQL-paired questions generally containing more words on average. Despite their greater length, SQL-paired questions demonstrated more focused information needs that could be readily translated to database queries.

A **semantic analysis** of questions revealed a distribution of query intents across the dataset.

As seen in Figure 5, **count-related** queries were the most common type, focusing on enumerating patients or events matching specific criteria. **Existence-related** queries formed the second most frequent category, determining the presence of specific conditions or events in patient records. The third major category comprised **measurement-related** queries, which sought to retrieve specific clinical measurements or values from patient records.

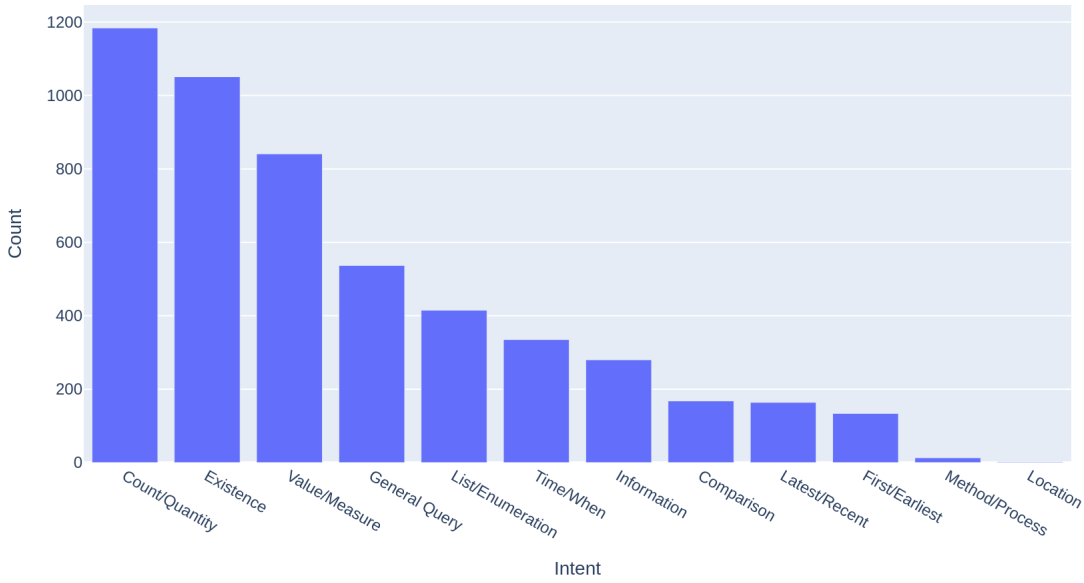


Figure 5: Distribution of Query Intents

This distribution allowed insight into the design of prompting strategies, particularly for few-shot examples needed to ensure model generalization across the full spectrum of the dataset.

After analyzing the `questions` field, we proceeded to examine the `true_query` field, which contains the SQL queries corresponding to the natural language questions[3].

[3]This analysis focuses specifically on questions that have an associated SQL query, excluding those without a query pair.

The query analysis started with a **complexity analysis** of the queries. To quantify query complexity, a custom scoring system was developed that weighted different SQL components according to their cognitive and computational complexity, assigning weights to each SQL component and calculating a final score.

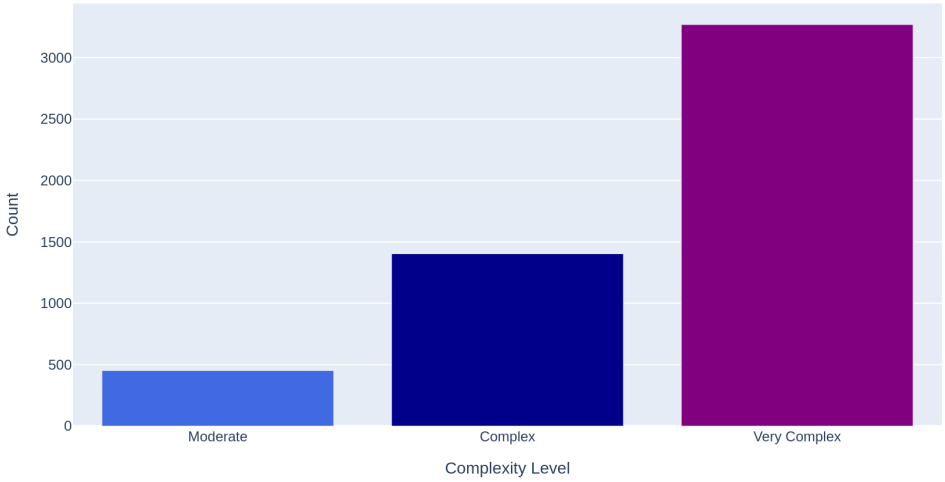


Figure 6: Distribution of SQL Query Complexity Levels

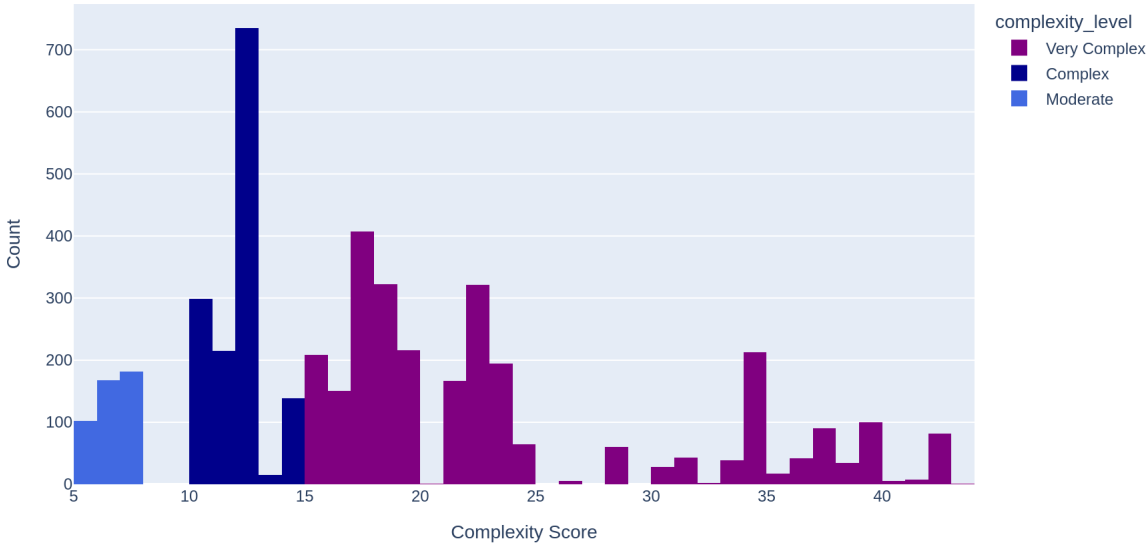


Figure 7: Distribution of SQL Query Complexity Scores

Based on the resulting scores, queries were categorized into four complexity levels: **Simple** (score < 5), **Moderate** (score 5-9), **Complex** (score 10-14), and **Very Complex** (score $\geq$ 15).

This granular classification and distribution in Figure 6 and 7 revealed the dataset's significant structural complexity, with a substantial portion of queries falling into the Complex and Very Complex categories, highlighting the existence of multi-part queries with nested structures and multiple table joins, as well as the need for models capable of handling them.

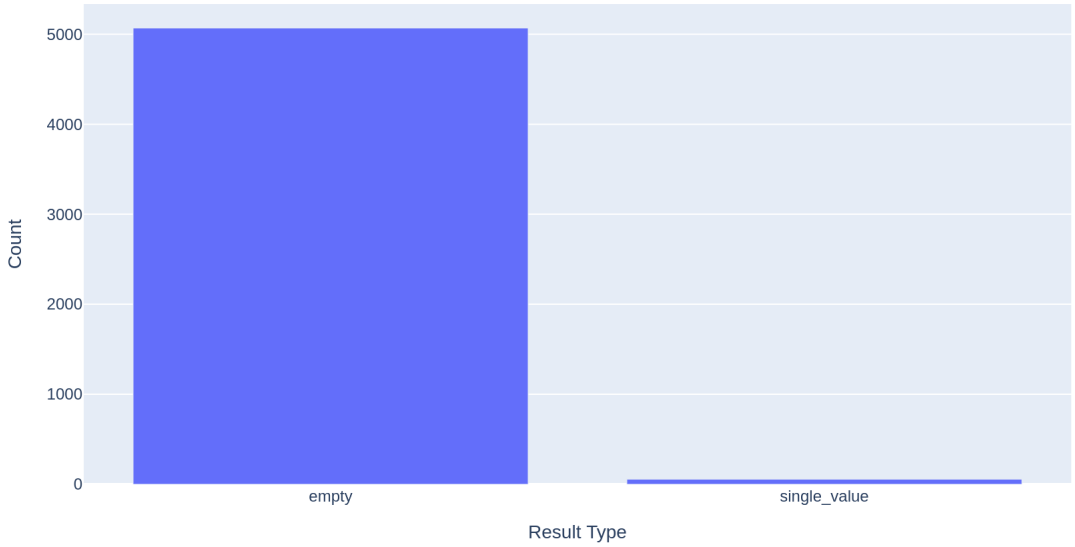


Figure 8: Distribution of Query Execution Outcomes

A critical finding emerged when executing the dataset's queries against the adapted MIMIC-IV database: approximately 98% of queries returned empty result sets.

This unexpected outcome prompted further investigation through value substitution experiments. To determine whether empty results were due to specific value constraints, 1,000 randomly selected queries were modified by **substituting random values** sampled from the relevant database columns.

This approach failed to significantly increase the result yield, suggesting structural rather than value-specific issues with the queries. **Despite returning empty results, the vast majority of queries executed without errors, indicating syntactic and schema compatibility but possible semantic misalignment with the available data.**

The prevalence of empty result sets presented a methodological challenge: models needed to be evaluated not just on their ability to generate syntactically correct queries, but on structural and semantic similarity to ground truth queries, since execution results alone would often be insufficient for meaningful comparison.

6.4 Query Comparison Methodology

A particularly challenging aspect of this research was establishing robust evaluation metrics for comparing the ground truth `true_query` field and our generated queries.

The overwhelming prevalence of queries returning empty result sets necessitated looking beyond simple execution outcomes to assess model performance, leading to the development of a comprehensive multi-dimensional evaluation framework that combined execution-based metrics with structural and semantic analysis.

6.4.1 Execution-Based Metrics

The primary execution metrics focused on query executability and result correctness:

- **Execution Success Rate:** The percentage of generated queries that executed without syntax or schema errors, providing a basic measure of grammatical correctness and schema compatibility.
- **Result Match Rate:** For successfully executed queries, this metric determined whether the generated query produced identical results to the ground truth query[4].

6.4.2 Structural Similarity Analysis

To provide deeper insights beyond execution results, the following structural analysis metrics were implemented:

- **Table Access Accuracy:** Measured using the Jaccard similarity coefficient between tables referenced in the generated and ground truth queries:

$$\text{Table Access Accuracy} = \frac{|\text{Tables}_{\text{true}} \cap \text{Tables}_{\text{gen}}|}{|\text{Tables}_{\text{true}} \cup \text{Tables}_{\text{gen}}|} \tag{1}$$

  This metric assessed whether the model correctly identified the relevant database tables regardless of query syntax variations.

[4]The comparison algorithm accounted for row order variations while ensuring identical column composition and data values.

- **Column Access Accuracy:** Similarly calculated for columns referenced across both queries:

$$\text{Column Access Accuracy} = \frac{|\text{Columns}_{\text{true}} \cap \text{Columns}_{\text{gen}}|}{|\text{Columns}_{\text{true}} \cup \text{Columns}_{\text{gen}}|} \tag{2}$$

  This metric evaluated the model's ability to identify the correct data attributes within the medical database schema.

- **Query Text Similarity:** Calculated using sequence matching algorithms to determine overall textual similarity between ground truth and generated queries.

6.4.3 Component-Level Analysis

To achieve fine-grained analysis of query generation quality, the following component-level decomposition and evaluation were implemented:

- **Component Similarity:** Each query was parsed into its constituent clauses (e.g. 'SELECT', 'FROM', 'WHERE' etc.) with special handling for nested subqueries for independent evaluations using sequence matching algorithms.
- **Subquery Analysis:** For complex queries, we recursively analyzed nested subqueries, preserving their hierarchical relationships and ensuring accurate component extraction despite high nesting levels.

This analysis proved particularly valuable for identifying patterns in model errors[5].

6.4.4 Execution Plan Analysis

A rather novel approach of this evaluation framework was the incorporation of execution plan comparison. This involved utilizing the database engine's **execution plan**, which detailed the optimization strategies and execution steps. One type of execution plan, accessed using the 'EXPLAIN' clause, can be seen in Figure 50 in the Appendix.

- **Plan Extraction:** For each successfully executed query (both generated and ground truth), we captured the database engine's execution plan, which detailed the optimization strategies and execution steps.
- **Operation Matching:** The proportion of operations present in both execution plans was calculated:

[5]For instance, a model might correctly identify tables and conditions (high 'FROM' and 'WHERE' clause similarity) but struggle with complex result selection logic (low 'SELECT' clause similarity).

$$\text{Operation Match} = \frac{|\text{Operations}_{\text{true}} \cap \text{Operations}_{\text{gen}}|}{|\text{Operations}_{\text{true}} \cup \text{Operations}_{\text{gen}}|} \tag{3}$$

- **Sequence Similarity:** Beyond operation overlap, we measured the sequential similarity of execution steps, capturing the structural equivalence of query execution strategies.

This execution plan analysis offered insights beyond surface syntax, revealing whether the database would process and optimize the queries similarly.

6.4.5 Error Categorization

For queries that failed to execute, a systematic error categorization framework was implemented :

- **Syntax Errors:** Fundamental grammatical issues in SQL formation
- **Schema Errors:** References to non-existent tables or columns
- **Type Errors:** Data type mismatches or incompatible operations
- **Function Errors:** Incorrect use of SQL functions or aggregates
- **Constraint Errors:** Violations of database constraints or integrity rules
- **Timeout Errors:** Queries exceeding execution time thresholds

As mentioned by Ning et al. (2024), this categorization is to allow targeted analysis of failure modes across different models and prompted strategies.

6.4.6 Composite Evaluation Score

A weighted composite score[6], combining the normalized evaluation dimensions, was formulated to have some holistic comparison across models.

[6]From a computational theory perspective, determining whether two arbitrary SQL queries are semantically equivalent is **generally undecidable**. Our composite score does not attempt to solve this theoretical challenge, but instead functions as **a domain-specific heuristic** for **model assessment and improvement** under **constrained conditions** (fixed schema, defined question set). The score establishes a practical partial ordering of models while acknowledging theoretical limitations, focusing on pragmatic clinical utility rather than theoretical query equivalence determination.

Metric	Weight
Result Match Rate	0.3
Table Access Accuracy	0.2
Column Access Accuracy	0.2
Query Text Similarity	0.1
Execution Plan Similarity	0.2
Total	**1.0**

Table 4: Weight distribution for the composite evaluation score

This score prioritizes functional correctness (result matching) while acknowledging the importance of structural accuracy (table/column access accuracy) and execution details, as done in Yu et al. (2018), particularly given the proportion of queries with empty results.

Robustness tests were conducted using Monte Carlo perturbations and tornado analysis on a sample test set; the resulting Spearman'ρ values (min = 0.983, median = 1.0 for 20% variation over 1000 samples) and invariant model rankings indicated the score is stable under changes in weight configuration.

6.5 Generation Model (M1) Development

The development of the generation model component followed a systematic progression through increasingly sophisticated prompting strategies. For model development and fine-tuning procedures, we utilized the training split of the dataset. Subsequent evaluation of the fine-tuned models was conducted exclusively on the testing split to ensure unbiased performance assessment.

Each stage will built upon findings from the previous iteration, enabling incremental improvements in model performance while maintaining a controlled experimental environment, within the 24GB GPU memory constraint.

6.5.1 Zero-Shot Implementation

The initial phase employed zero-shot prompting to establish a performance baseline across all candidate models. Zero-shot evaluation provides insight into models' inherent capabilities without task-specific examples, revealing their fundamental understanding of SQL syntax and database querying concepts.

Listing 1: M1 Zeroshot Prompt

```
Convert this question to a SQL query in [brackets]:
```

```
Question: {question}
SQL:
```

A minimalist prompt design, as seen above in Listing 1, that provided clear instructions without additional context and required the model to rely solely on its pre-trained knowledge of SQL and medical terminology, was implemented[7]. All nine candidate models were evaluated using the same prompt across the entire evaluation dataset.

Query generation was performed with temperature set to **0.0** to ensure reproducibility[8], and a maximum token limit of **256** was applied to accommodate the longest anticipated queries.

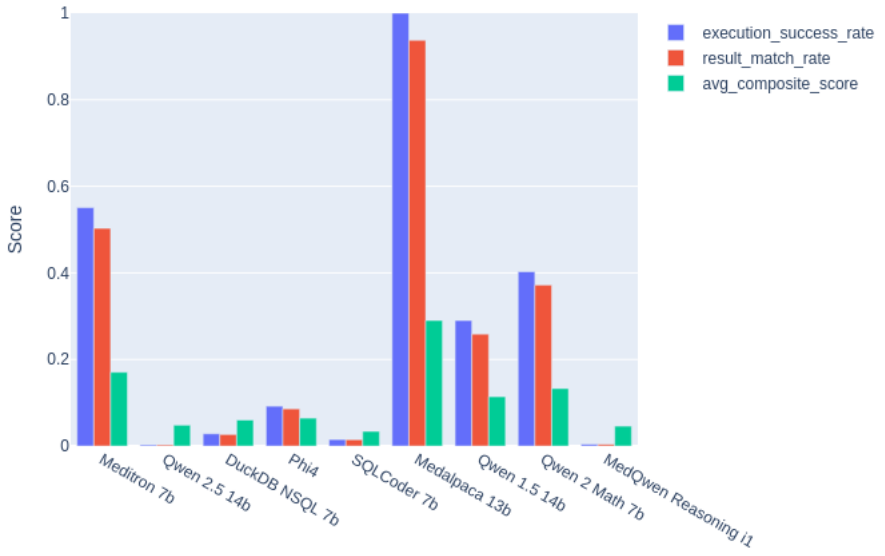


Figure 9: Comparative Zero-Shot Performance Across Models

The results look promising in Figure 9, but upon seeing the other component performance, significant limitations were revealed, particularly in handling complex joins, nested queries, and medical terminology as seen in Figure 10, 11 and 12.

[7]The instruction to enclose SQL within brackets served two purposes: establishing a clear boundary for extraction and signaling that a structured, executable query was required rather than explanatory text.

[8]To ensure a deterministic output.

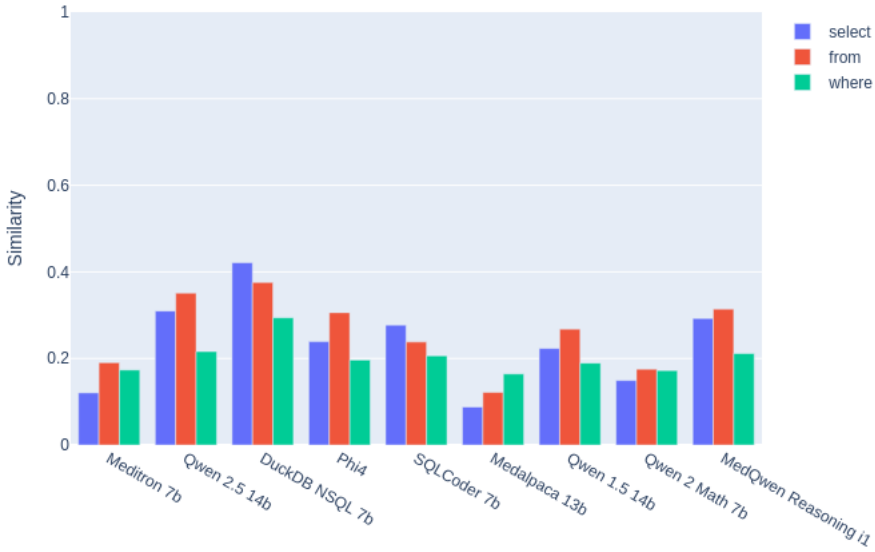


Figure 10: Comparative Zero-Shot Component Similarity

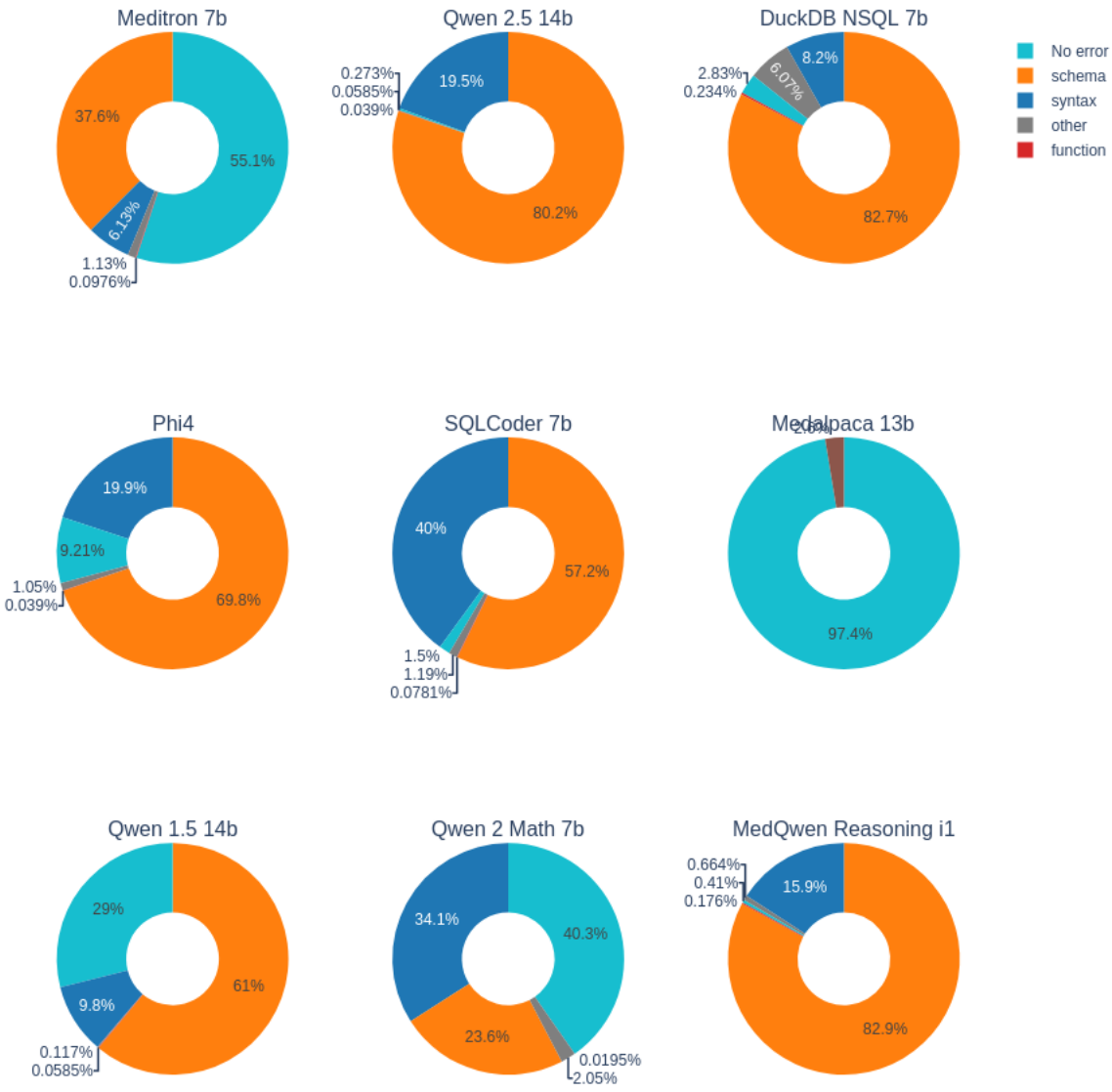


Figure 11: Combined Zero-Shot Error Categories

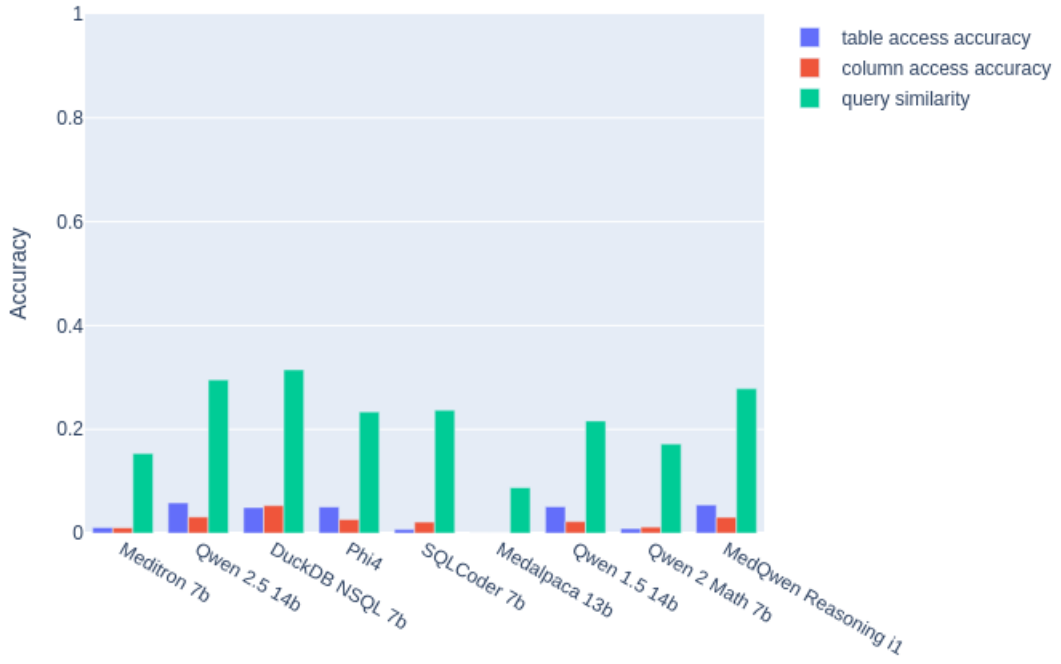


Figure 12: Comparative Zero-Shot Structural Accuracy

No models were eliminated at this stage, as the goal was to establish baseline performance across architectural approaches and identify inherent strengths in specific query types.

6.5.2 Few-Shot Refinement

To address limitations observed in zero-shot performance, a few-shot learning approach, that provided models with illustrative examples of the text-to-SQL task, was implemented.

Four carefully curated examples were selected to represent diverse query patterns and complexity levels:

- A basic patient-specific query involving procedure verification
- A medical advice question that cannot be answered with SQL (to establish appropriate boundaries)
- A complex temporal query with aggregation and ordering
- A multi-table join with time-based filtering

The same examples were used across all models to maintain evaluation consistency.

The few-shot prompt, as seen in Listing 2, maintained the same core instruction but included exemplars before the target question.

Listing 2: M1 Fewshot Prompt

```
Convert these questions to SQL queries in [brackets]. If the
   question cannot be
answered with a SQL query, respond with [No SQL query can answer
   this]:

Question: Has patient 10014078 undergone a central venous catheter
   placement
with guidance procedure?
SQL: [SELECT COUNT(*)>0 FROM procedures_icd WHERE procedures_icd.
   icd_code =
( SELECT d_icd_procedures.icd_code FROM d_icd_procedures WHERE
d_icd_procedures.long_title = 'central venous catheter placement
   with guidance' )
AND procedures_icd.hadm_id IN ( SELECT admissions.hadm_id FROM
   admissions
WHERE admissions.subject_id = 10014078 )]

Question: What precautions should i take after a closed uterine
   biopsy procedure?
SQL: [No SQL query can answer this. This requires medical advice,
   not database querying.]

[Additional examples...]

Question: {question}
SQL:
```

Few-shot prompting substantially improved performance across all models, with the most notable gains in **reducing schema errors** by demonstrating proper table selection and join patterns, **avoiding answering non-SQL questions** through improved identification of questions that required medical interpretation rather than direct data retrieval as seen in Figure 13, 14, 15 and 16.

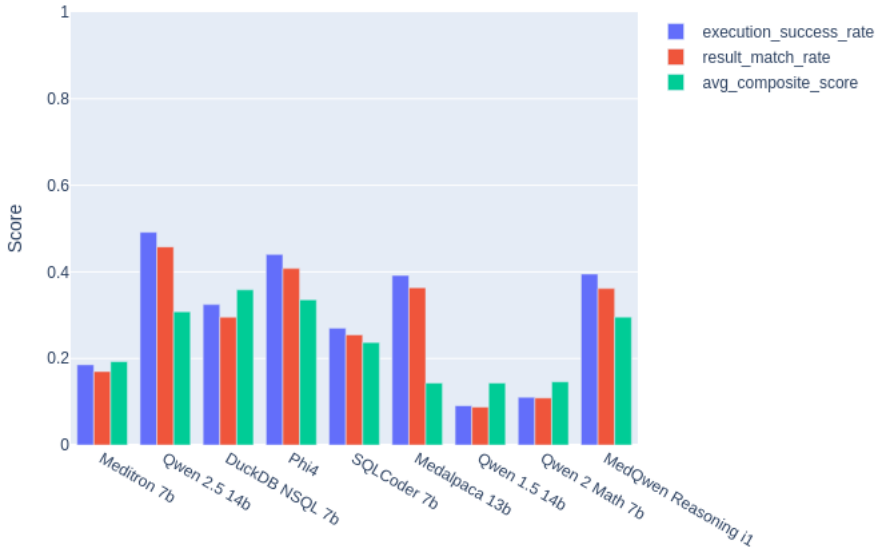


Figure 13: Few-Shot Performance Improvement Over Zero-Shot Baseline

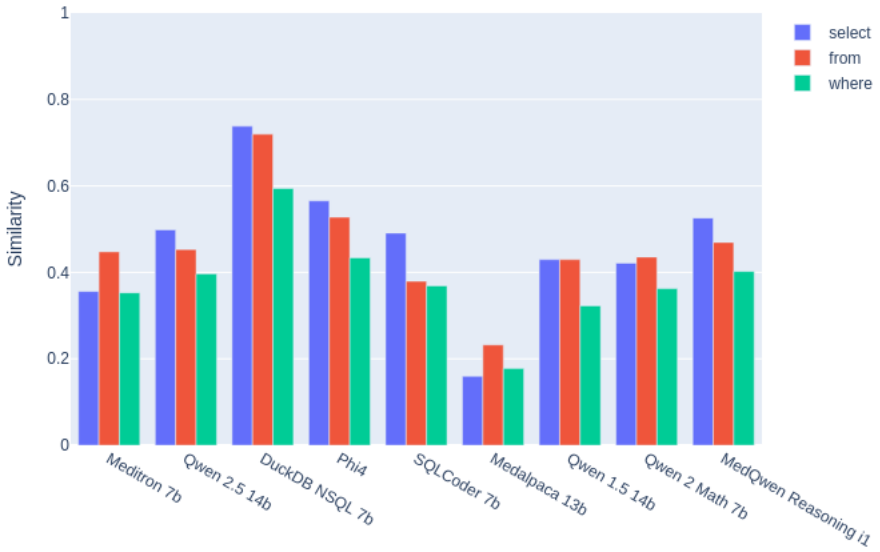


Figure 14: Comparative Few-Shot Component Similarity

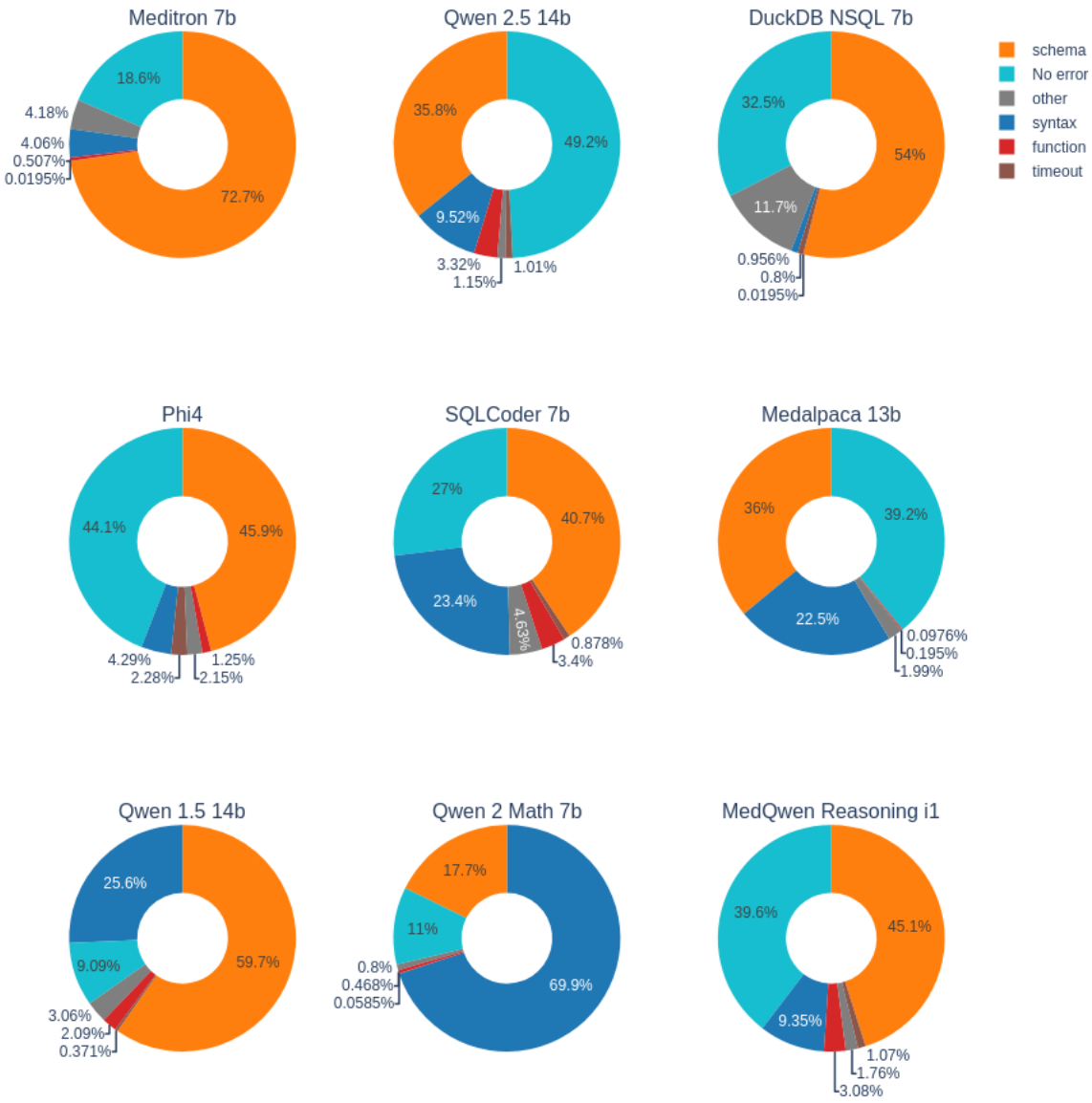


Figure 15: Combined Few-Shot Error Categories

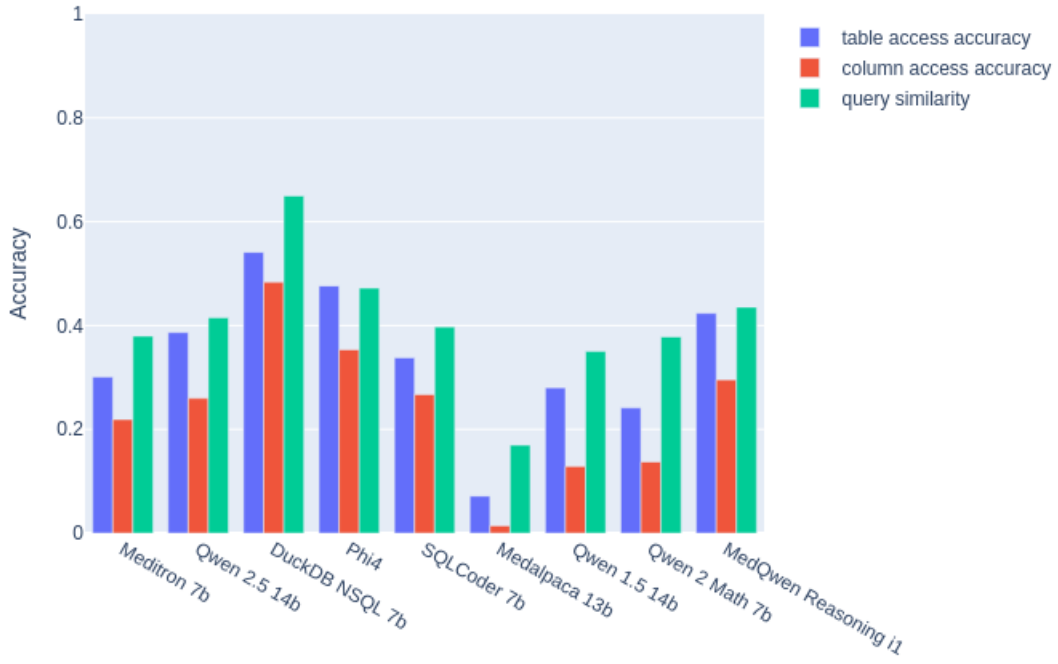


Figure 16: Comparative Few-Shot Structural Accuracy

Based on composite evaluation scores, the **top five** performing models were selected for advancement to the schema-aware phase.

6.5.3 Schema-Aware Enhancement

This phase represented a significant advancement in prompting strategy as explicit database schema information was incorporated into the prompt.

This approach directly addressed the structural complexity of the MIMIC-IV database by providing models with table definitions, key columns, and common join patterns.

A concise yet comprehensive schema-aware prompt that balanced informativeness with prompt length constraints was constructed as seen in Listing 3.

Listing 3: M1 Schema-Aware Prompt

```
Convert these questions to SQL queries in [brackets]. If the
    question cannot be
answered with a SQL query, respond with [No SQL query can answer
    this].
Verify joins and filters accordingly.
```

```
Schema:
Database Schema:
- patients table: Contains patient demographic information (key
    columns: subject_id)
- admissions table: Contains hospital admission records (key columns
    : subject_id, hadm_id)
- icustays table: Contains ICU stay information (key columns:
    subject_id, hadm_id, stay_id)
...

Common table joins:
- Join patients to admissions using subject_id
- Join admissions to icustays using hadm_id and subject_id
...

IMPORTANT: Not all questions can be answered with SQL queries. If a
    question asks for
medical advice, interpretation of results, or contains information
    not present in the
database, respond with 'No SQL query can answer this question' and
    briefly explain why.

Question: {question}
SQL:
```

Brief functional description of each table's purpose (e.g., "patients table: Contains patient demographic information"), identification of primary identifiers (e.g. subject_id, hadm_id, stay_id, etc.) and explicit documentation of standard table relationships (e.g., "Join patients to admissions using subject_id") were added to the prompt as well.

Since the representation of the modified MIMIC-IV was considerably smaller than the complete database schema, focusing on the most essential information for query construction while minimizing token consumption became much simpler.

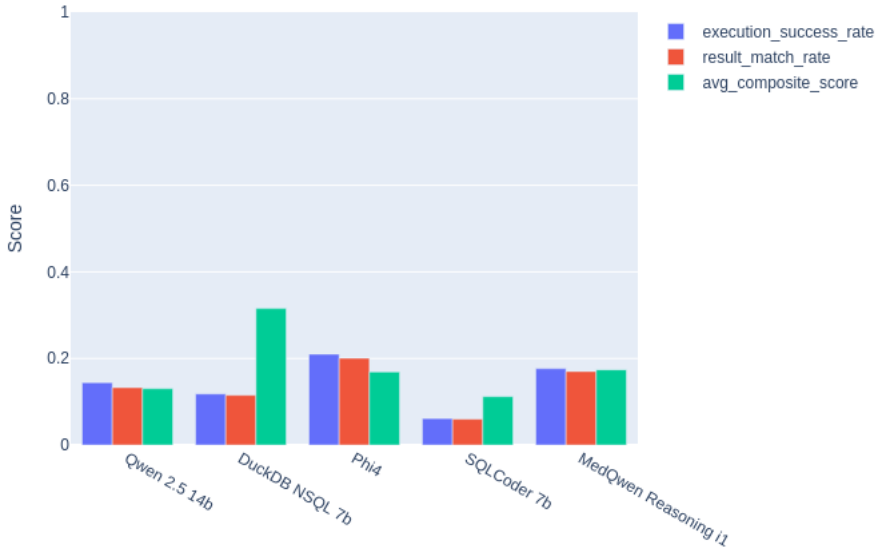


Figure 17: Schema-Aware Performance Improvement Over Zero-Shot Baseline

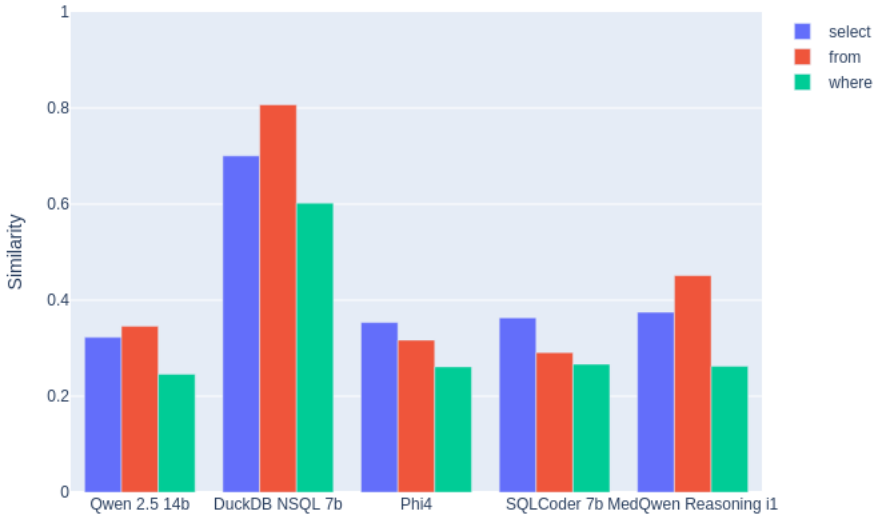


Figure 18: Comparative Schema-Aware Component Similarity

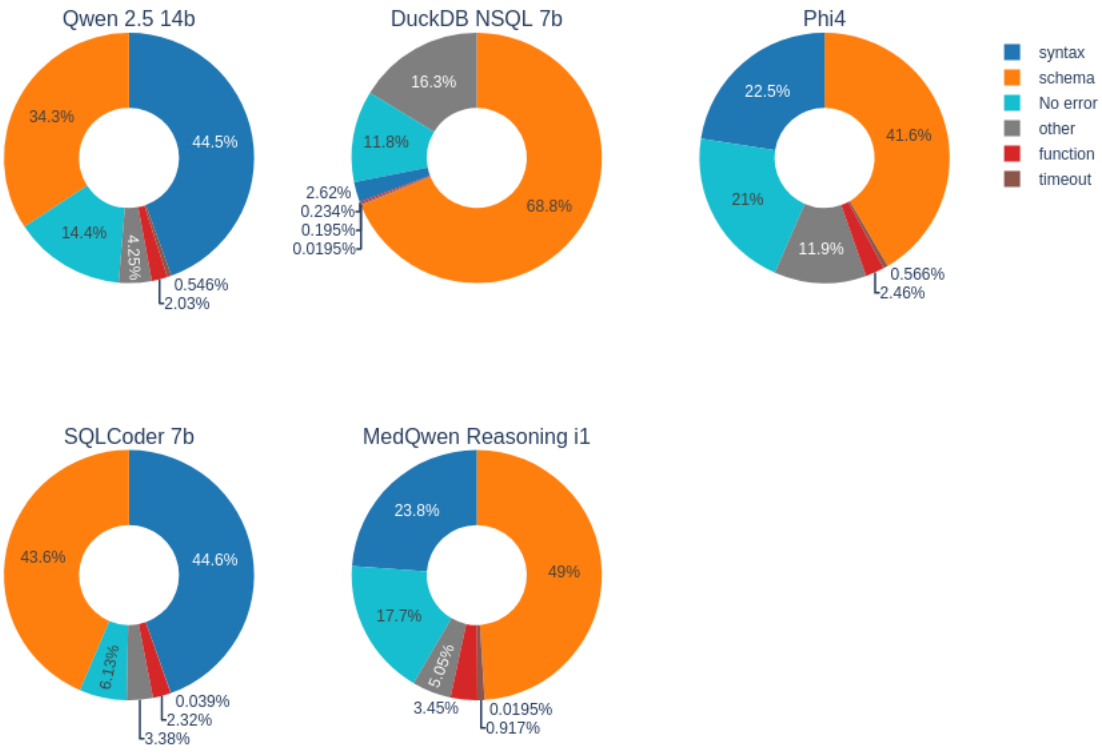


Figure 19: Combined Schema-Aware Error Categories

As seen in Figure 17, 18, 19 and 20, **lower performance across most metrics was observed** with schema-aware prompting compared to few-shot prompting.

This performance regression may be explained by the context window getting overcrowded and information overload being created for the models by the addition of extensive schema information, which is also addressed by Zhu et al. (2024).

According to Tarbell et al. (2023), attention may have been forced to be divided between processing schema details and generating the actual query, potentially disrupting any efficient reasoning patterns established during few-shot learning.

A potential solution for this limitation can be found in fine-tuning, through which schema knowledge may be stored **within model parameters** rather than through valuable context window space, embedding the knowledge of the database structure within the model through parameter updating.

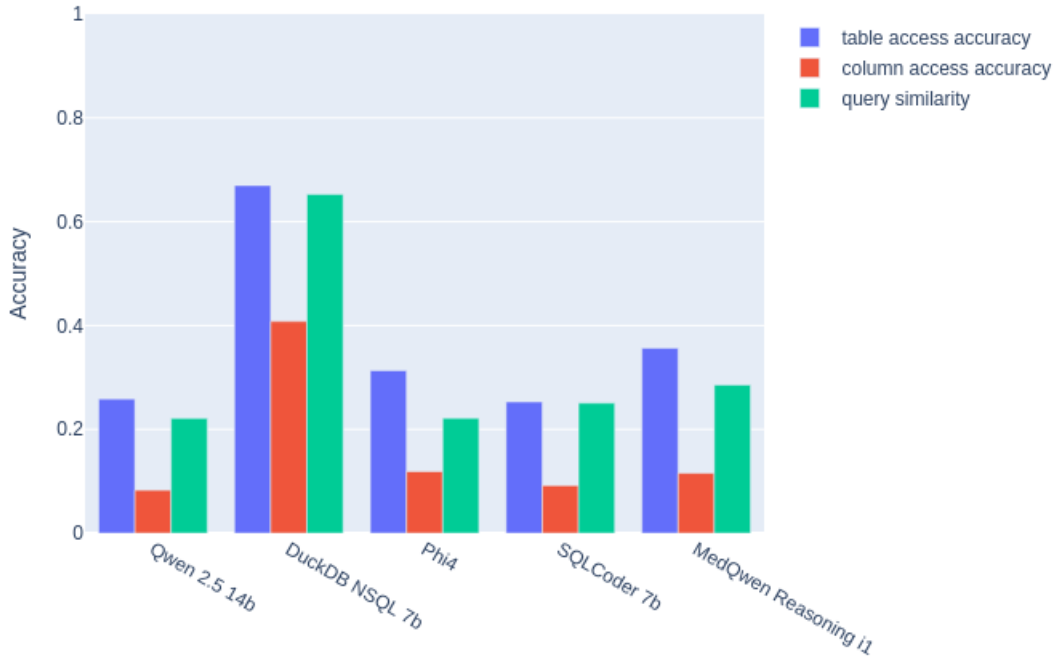


Figure 20: Comparative Schema-Aware Structural Accuracy

If this approach is successful, schema information would no longer need to be processed as part of the model's working memory during inference, whereby cognitive resources could be freed for the primary task of accurate SQL generation.

Therefore, fine-tuning the **Top 3** performing models from the schema-aware evaluation would be done for the final phase of the generation model development.

6.5.4 Fine-Tuning Implementation

Based on the performance analysis of schema-aware prompting, the top three performing models (**Qwen 2.5**[9], **Phi4**, and **DuckDB NSQL**) were selected for fine-tuning.

A parameter-efficient approach was implemented to overcome the 24GB GPU memory constraint while model adaptation was optimized for the medical SQL generation task.

[9]Although MedQwen Reasoning i1 outperformed Qwen 2.5, it did not have native LoRA support.

Low-Rank Adaptation (LoRA) was employed as the primary fine-tuning methodology and was implemented using the **Unsloth FastLanguageModel library** by Daniel Han and team (2023), which accelerated training by 2-3 times and optimized memory[10].

Additional optimizations, such as quantization and gradient checkpointing[11], were applied to minimize memory usage during backpropagation.

The LoRA configuration, along with **fp16** precision, was tailored specifically for balancing parameter efficiency to learn complex query structures for each model.

By this configuration, only approximately 2% of the model parameters were made trainable, whereby memory requirements were dramatically reduced while adaptation capability was maintained. For the models being fine-tuned, this translated to tens of millions of trainable parameters instead of billions.

The training dataset was prepared with a specialized format through which schema information was embedded directly within the instruction context:

Listing 4: M1 Fine-Tune Data Format

```
<|im_start|>system
You are a SQL assistant specialized in medical database queries.
Your task is to convert natural language questions into SQL queries
for the MIMIC medical database.

[Schema information]
<|im_end|>
<|im_start|>user
[question]
<|im_end|>
<|im_start|>assistant
[sql_query]
<|im_end|>
```

As seen in Listing 4, associations between natural language questions, schema context, and SQL responses attempt to get established during training, whereby schema information no longer needed repetition during inference.

As seen in Table 5, the training process was configured with key hyperparameters selected for each model based on their architecture to balance optimization quality with the hardware

[10]This optimization framework was essential given the 24GB memory constraint, as it enabled the fine-tuning of 14B parameter models that would otherwise be infeasible on a single GPU.

[11]Allowed recomputation of intermediate activations rather than storing them.

constraint, maintaining training efficiency while enabling effective adaptation within the 24GB GPU limitation.

Parameter	**Qwen 2.5**	**Phi 4**	**DuckDB NSQL**
Epochs	3	5	3
Gradient Accumulation	8	12	16
Learning Rate	2e-4	5e-5	1e-4
Target Modules	`q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj`	`q_proj, k_proj, v_proj, o_proj, up_proj, down_proj`	`q_proj, k_proj, v_proj, o_proj, up_proj, down_proj`
Scheduler Type	Cosine	Linear	Cosine
Alpha	32	16	48
Rank	16	8	24

Table 5: Key Training Hyperparameters

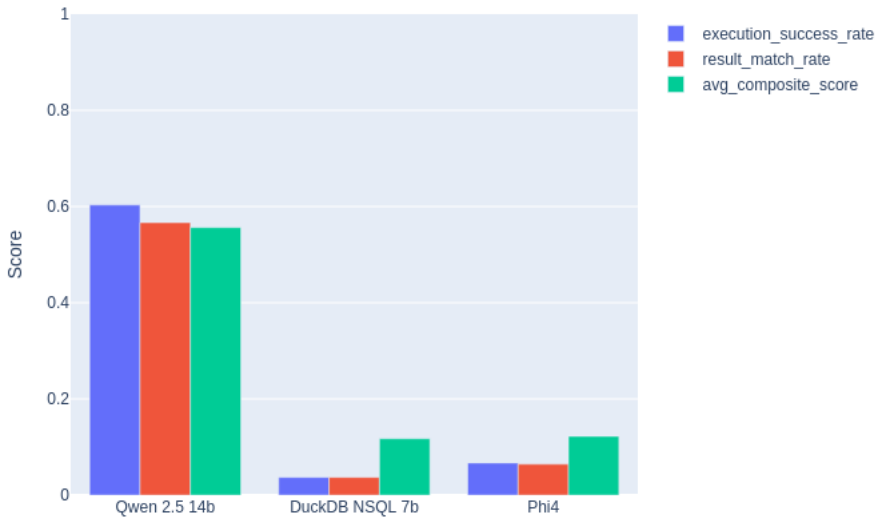


Figure 21: Fine-Tuning Performance Improvement Over Schema-Aware Baseline

As seen in Figure 37, 38, 39 and 40, **significant performance improvements across most metrics were observed** with **Qwen 2.5**, which demonstrated exceptional capability compared to the alternatives.

As shown in the performance metrics, the fine-tuned Qwen 2.5 achieved approximately **80%** success across column/table access accuracy as well as query component similarity, far exceeding Phi4 and DuckDB NSQL models and being the only model to successfully internalize the database structure knowledge during fine-tuning.

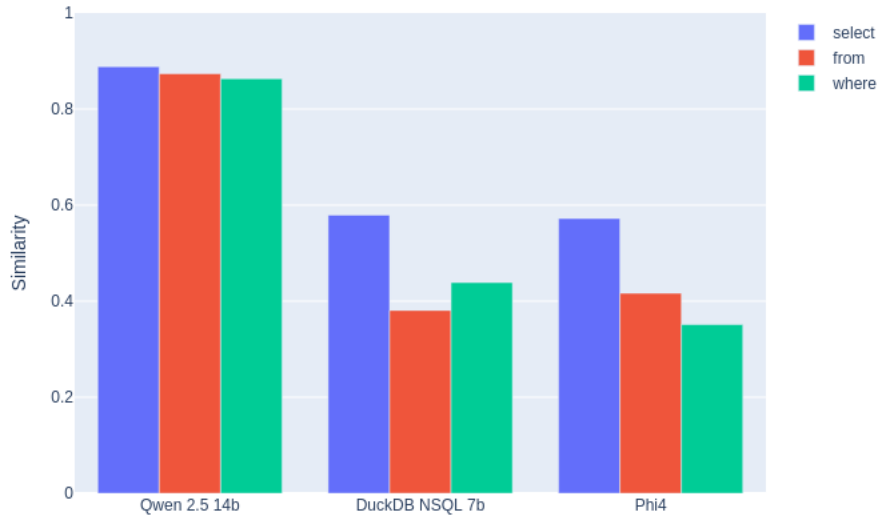


Figure 22: Comparative Fine-Tuned Component Similarity

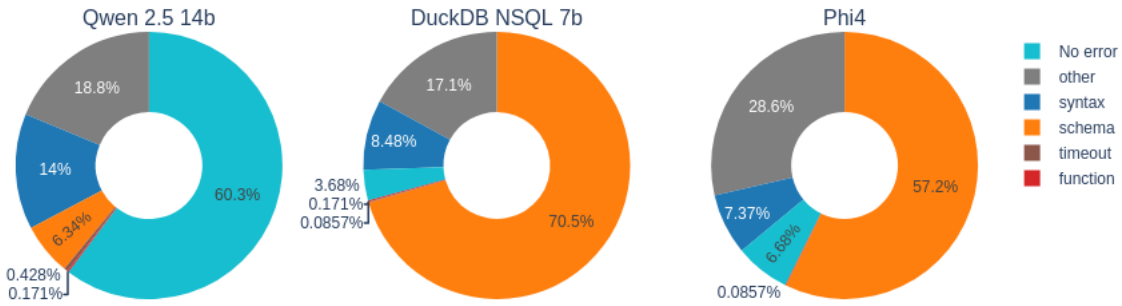


Figure 23: Combined Fine-Tuned Error Categories

The most telling contrast appears in the error analysis, where Qwen 2.5 resolved **60.3%** of queries **without errors**, while DuckDB NSQL and Phi4 exhibited persistent schema errors (70.5% and 57.2% respectively).

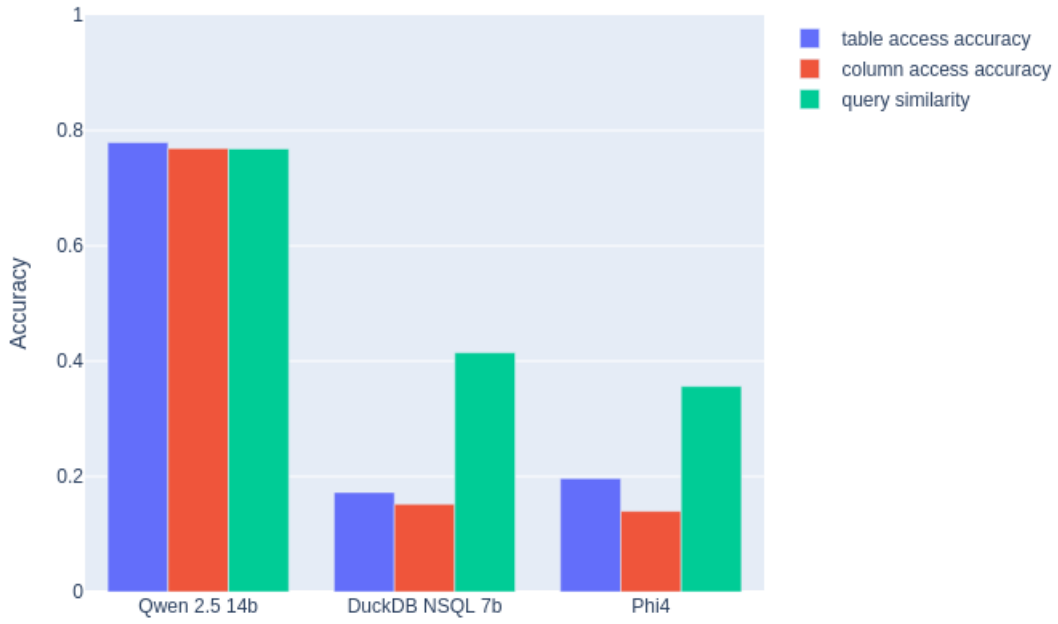


Figure 24: Comparative Fine-Tuned Structural Accuracy

The component analysis further highlights this divide, with Qwen achieving approximately 85-90% similarity across SELECT, FROM, and WHERE clauses, compared to significantly lower values for the alternatives.

This extraordinary performance gap led to the definitive selection of **fine-tuned Qwen 2.5 14b as the primary generation model (M1)** for our pipeline.

6.6 Validation Model (M2) Development

The development of the validation model component followed the same systematic progression as the generation model, through increasingly sophisticated prompting strategies. For the development and fine-tuning of our validation component, we constructed a specialized dataset comprising failed queries generated by the top-performing M1 model when applied to the training split. The evaluation of these fine-tuned validation models was subsequently

conducted using an analogous collection of failed queries identified within the testing split, ensuring rigorous assessment under realistic conditions.

Each stage built upon findings from the previous iteration, enabling incremental improvements in validation performance while maintaining a controlled experimental environment, within the 24GB GPU memory constraint.

6.6.1 Zero-Shot Implementation

The initial phase employed zero-shot prompting to establish a performance baseline across all candidate models. Similar to M1, zero-shot evaluation provides insight into models' inherent capabilities to identify and fix SQL errors without task-specific examples.

Listing 5: M2 Zeroshot Prompt

```
You are an expert at fixing SQL queries specilized for MIMIC-IV
   Database.
Please fix this SQL query that has errors.

Question: {question}

Original SQL query: {original_query}

Error message: {error_message}

Fixed SQL query:
```

The focused prompt design seen above in Listing 5, similar to Listing 1, provided clear instructions for query correction without additional context. The prompt required the model to rely solely on its pre-trained knowledge of SQL syntax, error patterns, and medical database structures to diagnose and repair faulty queries.

All nine candidate models were evaluated using the same prompt across the entire set of failed queries from the top-performing M1 model (**DuckDB NSQL 7B**) obtained after the schema-aware stage.

Query validation was performed with temperature set to **0.1** to balance determinism with some exploration of correction options, and a maximum token limit of **512** was applied to accommodate potentially complex query repairs.

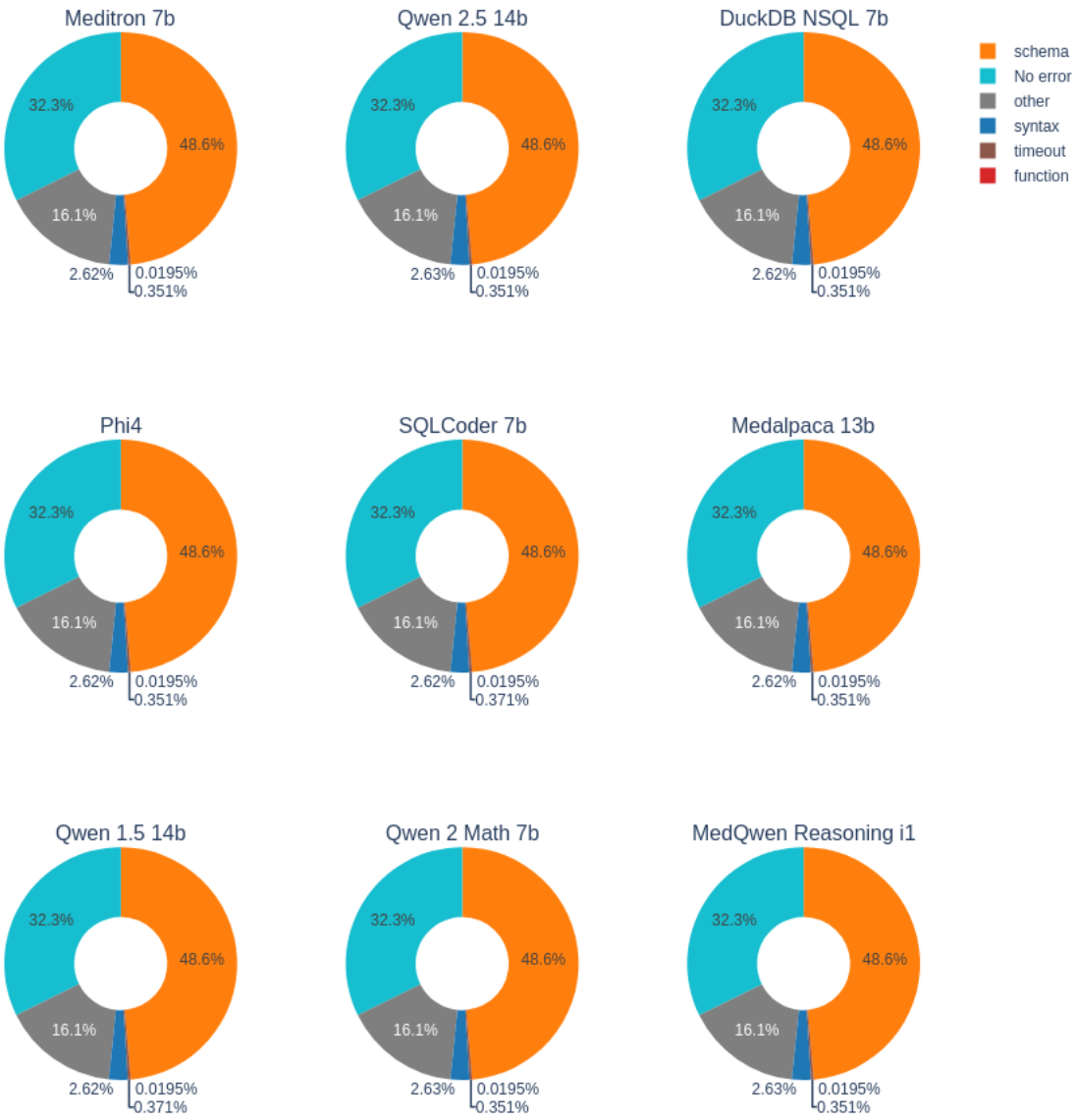


Figure 25: Combined Zero-Shot Error Categories

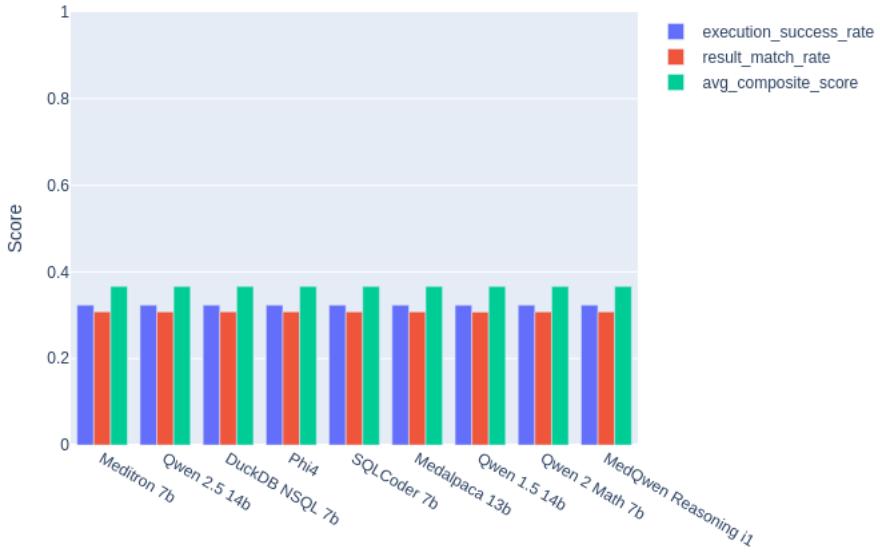


Figure 26: Comparative Zero-Shot Performance Across Models

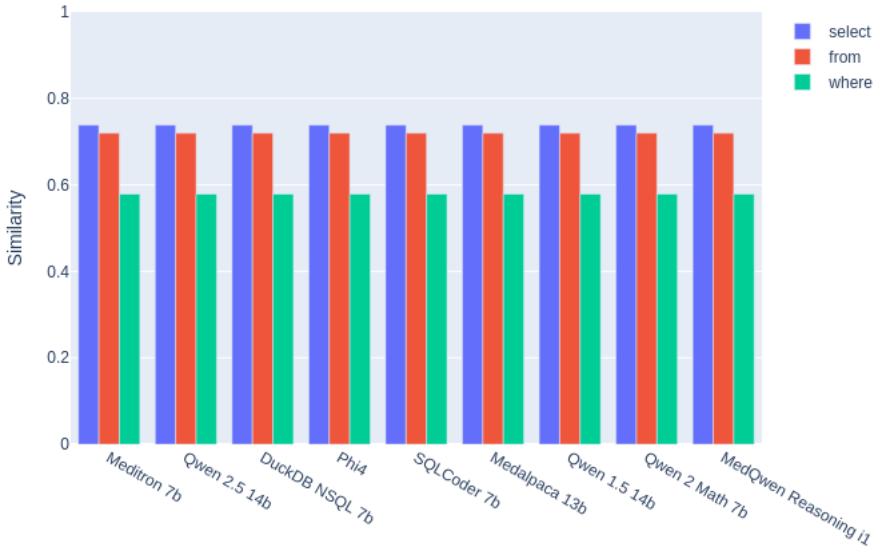


Figure 27: Comparative Zero-Shot Component Similarity

Despite initial expectations, the results were disappointing, with no observable improvements derived from the input dataset. A closer analysis of component performance revealed

substantial shortcomings in handling complex error correction, nested queries, and table relationship resolution, as shown in Figures 25, 26, 27, and 28.

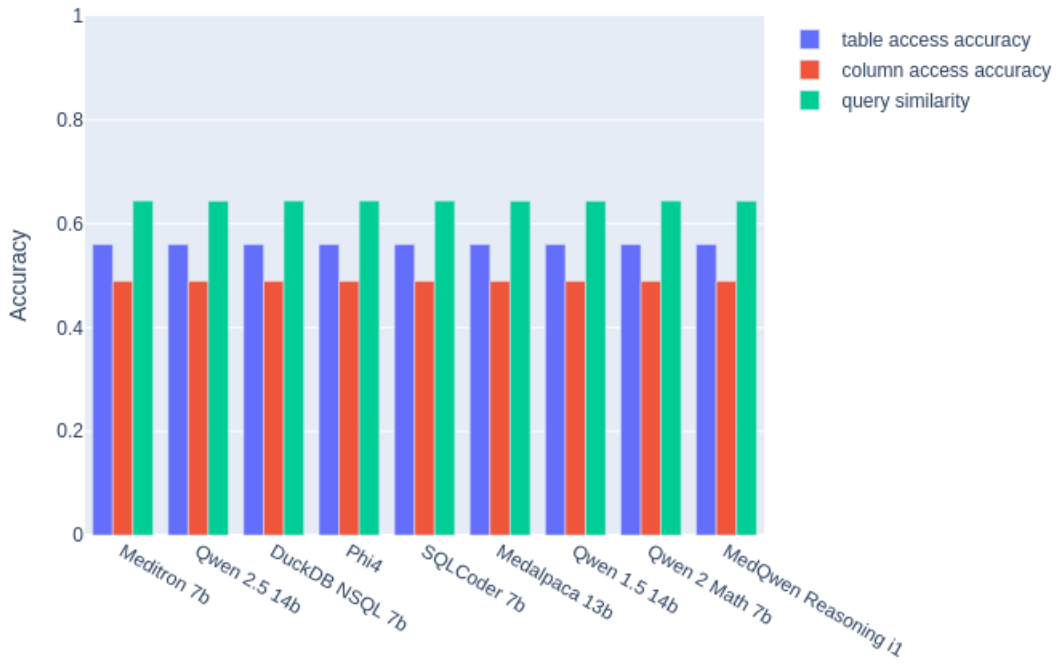


Figure 28: Comparative Zero-Shot Structural Accuracy

No models were eliminated at this stage, as the goal was to establish baseline performance across architectural approaches and identify inherent strengths in specific error correction patterns.

6.6.2 Few-Shot Refinement

To address limitations observed in zero-shot performance, a few-shot learning approach that provided models with illustrative examples of SQL error correction was implemented.

Four carefully curated examples were selected to represent diverse error patterns and complexity levels:

- A column reference error where the query incorrectly referred to a non-existent column
- An incomplete input error with malformed query structure
- A timeout error requiring query optimization

- A complex join error with incorrect relationship handling

The same examples were used across all models to maintain evaluation consistency.

The few-shot prompt, as seen in Listing 6, maintained the same core instruction but included exemplars of error correction before the target query.

Listing 6: M2 Fewshot Prompt

```
You are an expert at fixing SQL queries specialized for MIMIC-IV
   Database.
Fix the SQL query based on the following examples and the error
   message.

Example 1:
Question: Show me the top four diagnoses with the highest 6-month
   mortality rate.
Original SQL query: SELECT d_icd_diagnoses.short_title FROM
   d_icd_diagnoses WHERE d_icd_diagnoses.icd_code IN (SELECT t2.
   icd_code FROM (SELECT t1.icd_code, DENSE_RANK() OVER (ORDER BY t1
   .c1 DESC) AS c2 FROM...
Error message: no such column: d_icd_diagnoses.short_title
Fixed SQL query: SELECT d_icd_diagnoses.long_title FROM
   d_icd_diagnoses WHERE d_icd_diagnoses.icd_code IN ( SELECT T4.
   icd_code FROM ( SELECT T3.icd_code, DENSE_RANK() OVER ( ORDER BY
   T3.C2 ASC ) AS C3 FROM...

[Additional examples...]

Now fix this query:
Question: {question}
Original SQL query: {original_query}
Error message: {error_message}
Fixed SQL query:
```

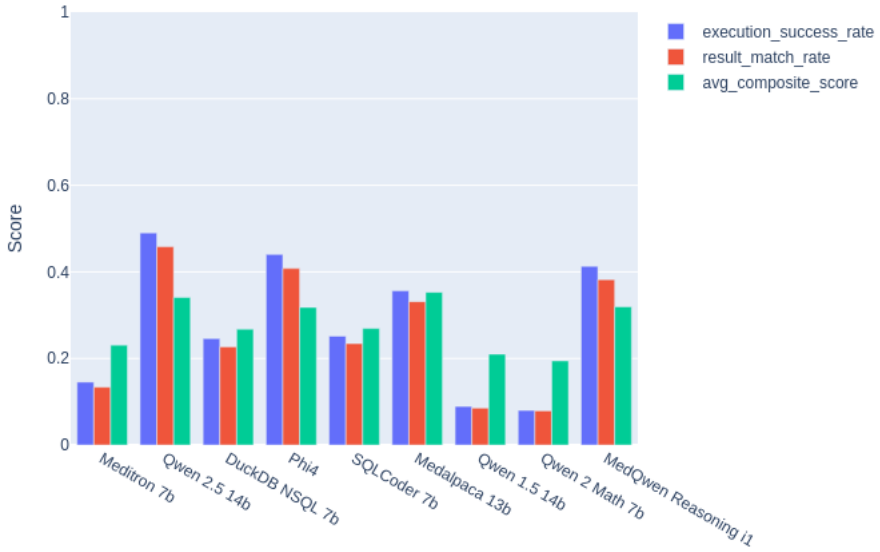


Figure 29: Few-Shot Performance Improvement Over Zero-Shot Baseline

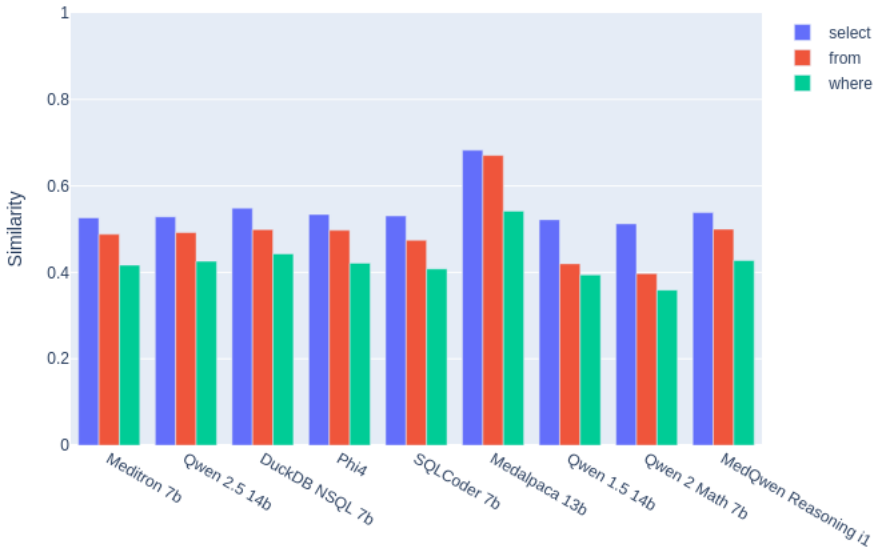


Figure 30: Comparative Few-Shot Component Similarity

Few-shot prompting substantially improved performance across all models, with the most notable gains in **correcting schema errors** by demonstrating proper column references and table relationships, **fixing incomplete queries** through structural completion patterns,

and **optimizing timeout-prone queries** through more efficient query formulation as seen in Figure 29, 30, 31 and 32.

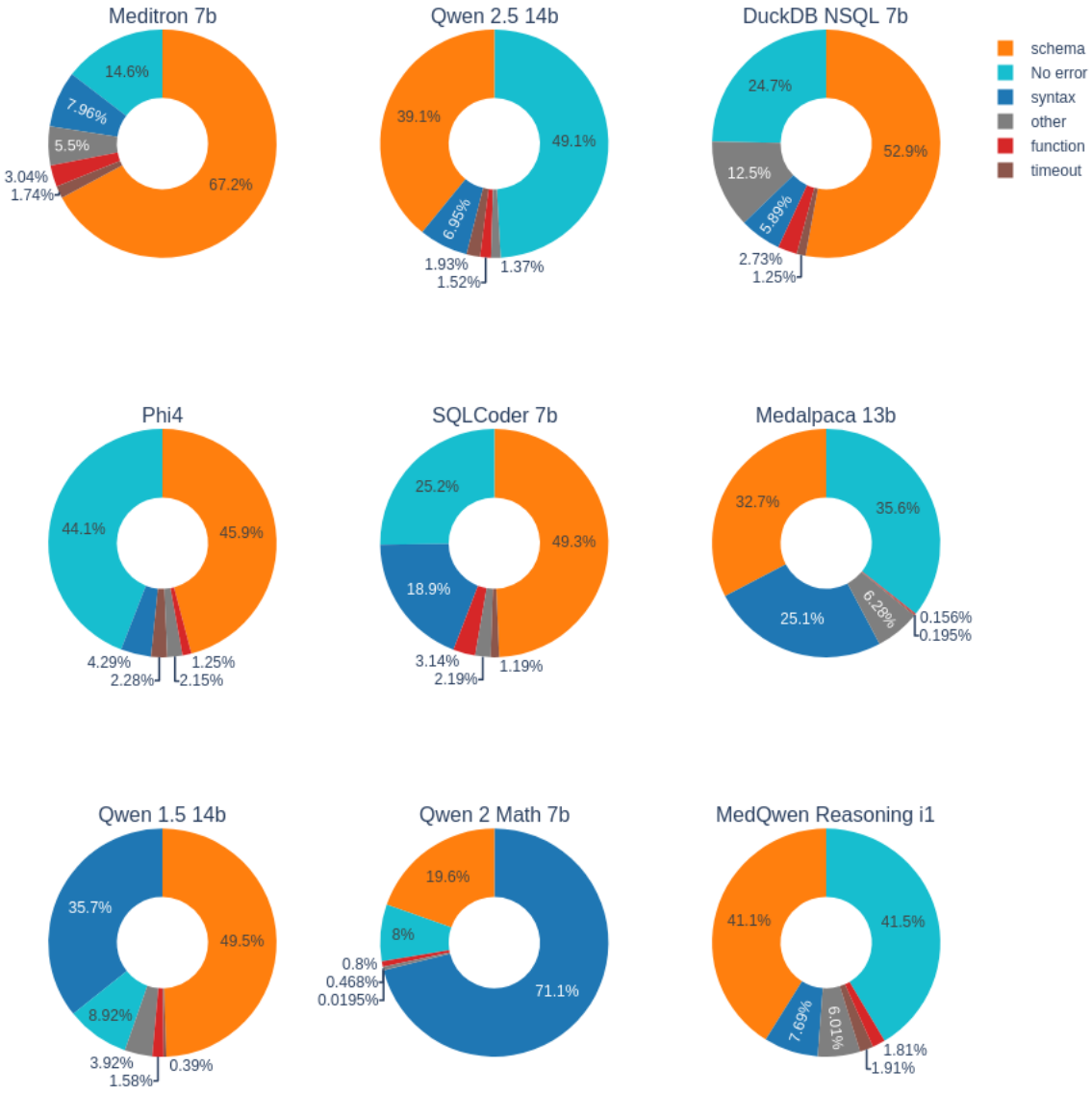


Figure 31: Combined Few-Shot Error Categories

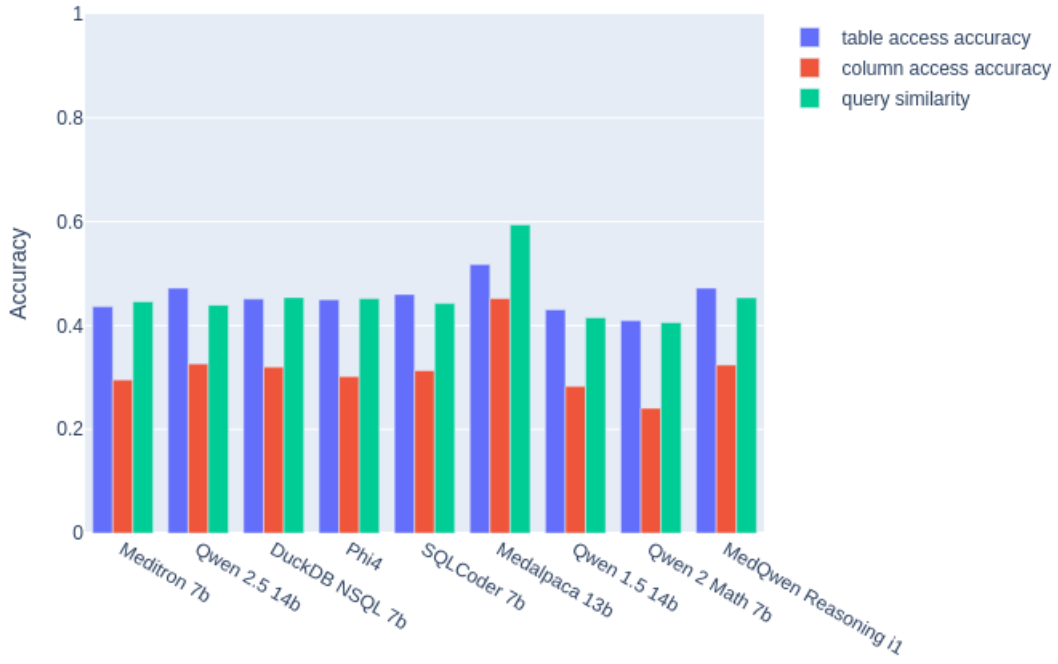


Figure 32: Comparative Few-Shot Structural Accuracy

Based on composite evaluation scores, the **top five** performing models were selected for advancement to the schema-aware phase.

6.6.3 Schema-Aware Enhancement

This phase represented a significant advancement in prompting strategy as explicit database schema information was incorporated into the prompt.

This approach directly followed from M1 and addressed the structural complexity of the MIMIC-IV database by providing models with table definitions, key columns, and common join patterns to inform error correction.

A concise yet comprehensive schema representation, that balanced informativeness with prompt length constraints, was constructed similar to Listing 7, same as seen in Listing 3.

Listing 7: M2 Schema-Aware Prompt

```
You are an expert at fixing SQL queries specialized for MIMIC-IV
    Database.
Fix the SQL query based on the error message and database schema
    information.
```

```
Verify joins and filters accordingly.

Schema:
Database Schema:
- patients table: Contains patient demographic information (key
    columns: subject_id)
...

Common table joins:
- Join patients to admissions using subject_id
...

IMPORTANT: Not all questions can be answered with SQL queries. If a
    question asks for
medical advice, interpretation of results, or contains information
    not present in the
database, respond with 'No SQL query can answer this question' and
    briefly explain why.

Question: {question}

Original SQL query: {original_query}

Error message: {error_message}

Fixed SQL query:
```

Similarly, a brief functional description of each table's purpose, identification of primary identifiers, and explicit documentation of standard table relationships were added to the prompt as well.

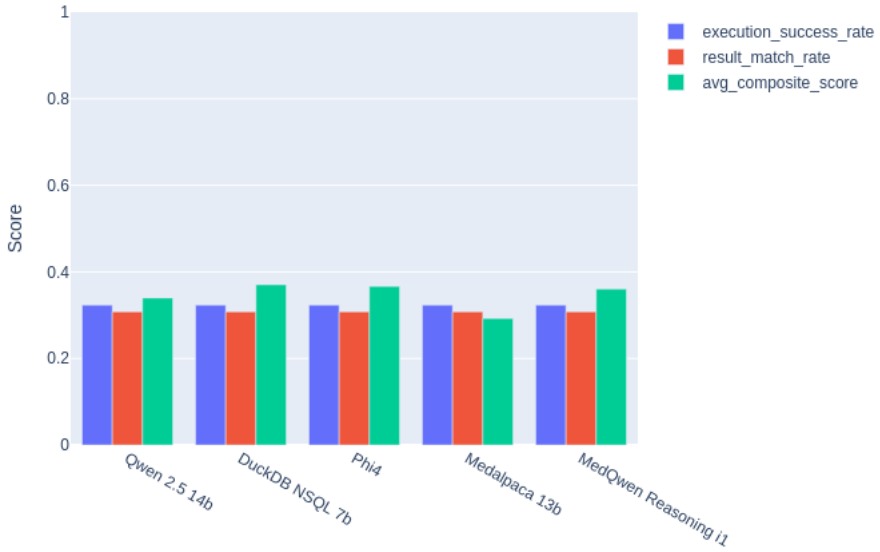


Figure 33: Schema-Aware Performance Improvement Over Zero-Shot Baseline

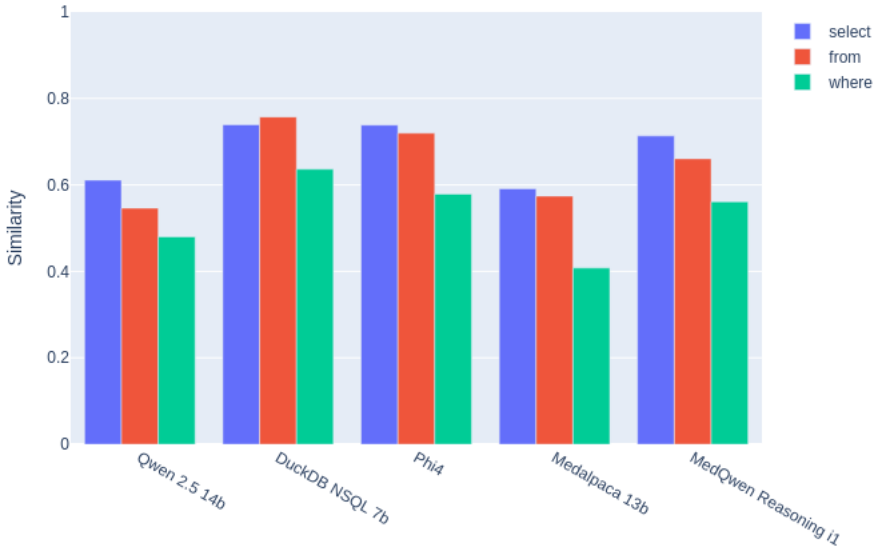


Figure 34: Comparative Schema-Aware Component Similarity

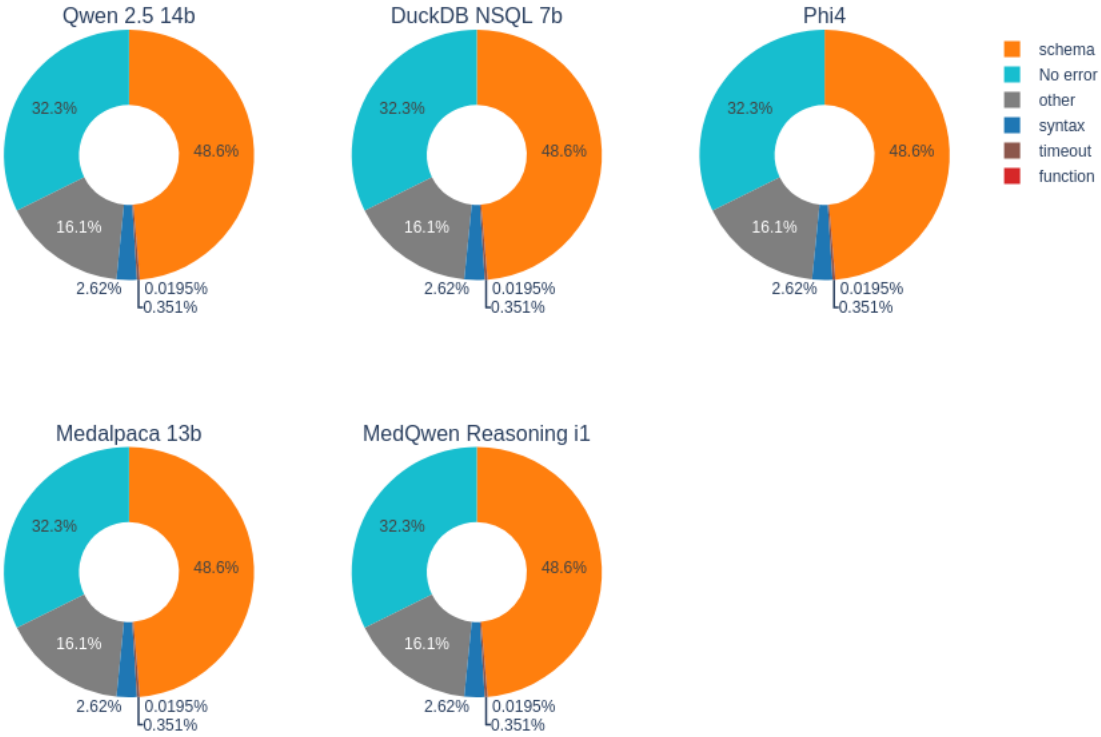


Figure 35: Combined Schema-Aware Error Categories

As seen in Figure 33, 34, 35 and 36, **better performance across most metrics was observed** with schema-aware prompting compared to few-shot prompting, **except error-related and results-related metrics, where no significant improvement from the input dataset was observed**.

This pattern differs from M1 development, suggesting that schema information is particularly valuable for explicit structural guidance, helping models identify and fix structural issues in queries. While Zhu et al. (2024) noted potential challenges with context window limitations when including schema details, our validation results indicate that for specialized correction tasks, the benefits outweigh these constraints.

Building on these positive schema-aware results, fine-tuning was identified as a way to further enhance performance. By preserving schema knowledge within the context window rather than embedding it in model parameters, cognitive resources could be more effectively directed toward the specialized task of error diagnosis and correction.

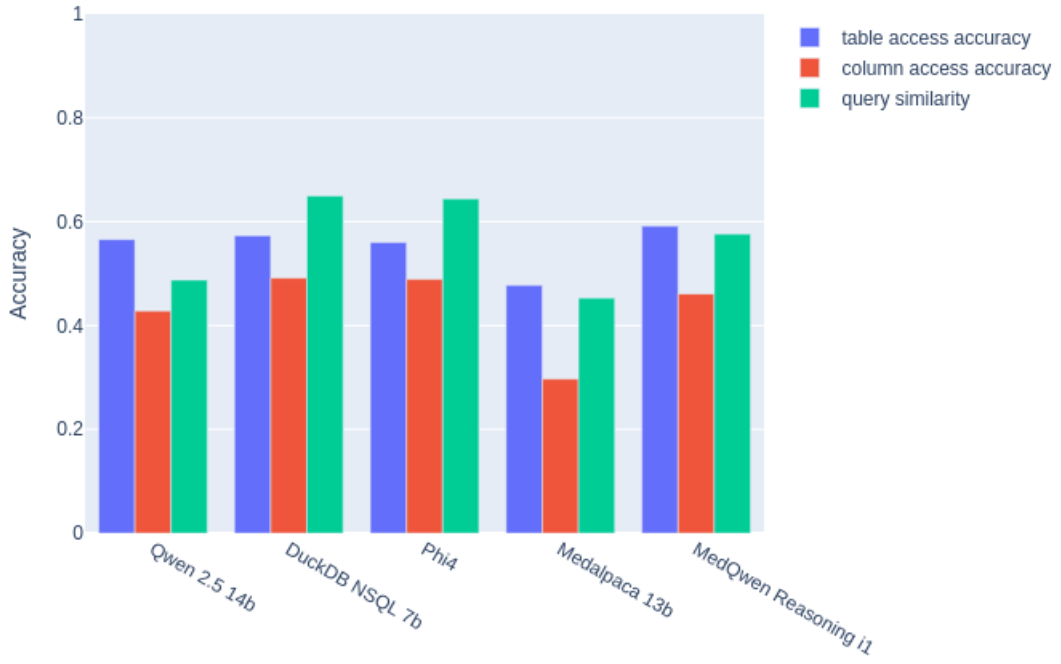


Figure 36: Comparative Schema-Aware Structural Accuracy

Therefore, fine-tuning the **Top 3** performing models from the schema-aware evaluation was undertaken for the final phase of the validation model development.

6.6.4 Fine-Tuning Implementation

Similar to M1, the fine-tuning of the M2 model was focused on adapting top three base models to the medical SQL query **fixing** rather than generation from scratch.

Based on the performance analysis of schema-aware prompting, the top three performing models (**Qwen 2.5**[12], **Phi4**, and **DuckDB NSQL**) were selected for fine-tuning.

Similar to M1, a memory-efficient fine-tuning strategy was used within the 24GB GPU limit, employing **LoRA** via the Unsloth FastLanguageModel library along with 4-bit quantization and gradient checkpointing. LoRA parameters (rank, alpha, target modules) were tuned for transformer compatibility, and `fp16` precision reduced memory use.

As in M1, only approximately 1% of parameters were trainable, keeping active weights in the tens of millions.

[12]Similar to M1, MedQwen Reasoning i1 was not selected due to a lack of native LoRA support despite outperforming Qwen 2.5.

Listing 8: M2 Fine-Tune Data Format

```
<|im_start|>system
You are an expert at fixing SQL queries specialized for medical
   databases.
<|im_end|>
<|im_start|>user
You are an expert at fixing SQL queries specialized for MIMIC-IV
   Database. Fix the SQL query based on the error message and
   database schema information.

Verify joins and filters accordingly.

Schema: [schema_string]

Question: [question]

Original SQL query: [original_sql]
Error message: [error_text]

Fixed SQL query:
<|im_end|>
<|im_start|>assistant
[corrected_sql]
<|im_end|>
```

As seen in Listings 8, this schema-aware instruction format allowed M2 to learn correction strategies for real-world execution errors by leveraging database context, logical alignment, and error diagnostics—all embedded at training time, unlike M1, which focused on initial SQL generation.

Parameter	Qwen 2.5	Phi 4	DuckDB NSQL
Epochs	3	5	3
Gradient Accumulation	8	12	16
Learning Rate	2e-4	5e-5	1e-4
Target Modules	`q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj`	`q_proj, k_proj, v_proj, o_proj, up_proj, down_proj`	`q_proj, k_proj, v_proj, o_proj, up_proj, down_proj`
Scheduler Type	Cosine	Linear	Cosine
Alpha	32	16	48
Rank	16	8	24

Table 6: Key Training Hyperparameters

As seen in Table 6, similar to M1, the training process was configured with key hyperparameters to balance optimize quality and maintain training efficiency for each model.

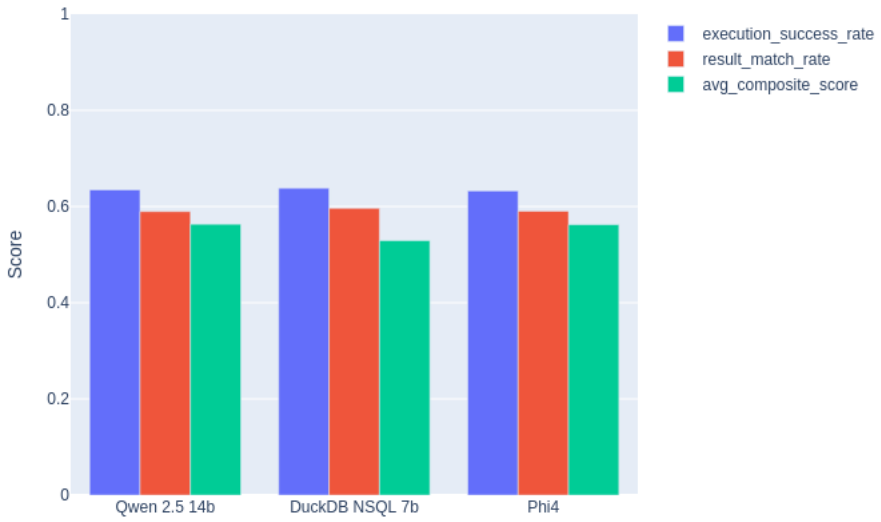


Figure 37: Fine-Tuning Performance Improvement Over Schema-Aware Baseline

As seen in Figure 37, 38, 39 and 40, **significant performance improvements across most metrics were observed** with **Qwen 2.5**.

While DuckDB NSQL showed marginally higher execution success rates (63.8% versus 63.5%), Qwen 2.5's superior structural understanding led to significantly better composite performance scores (**56.4%** versus 52.9%).

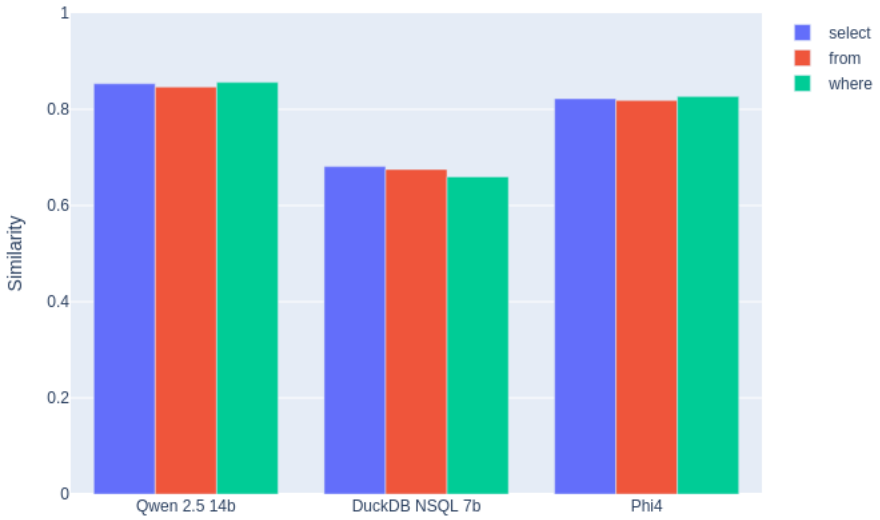


Figure 38: Comparative Fine-Tuned Component Similarity

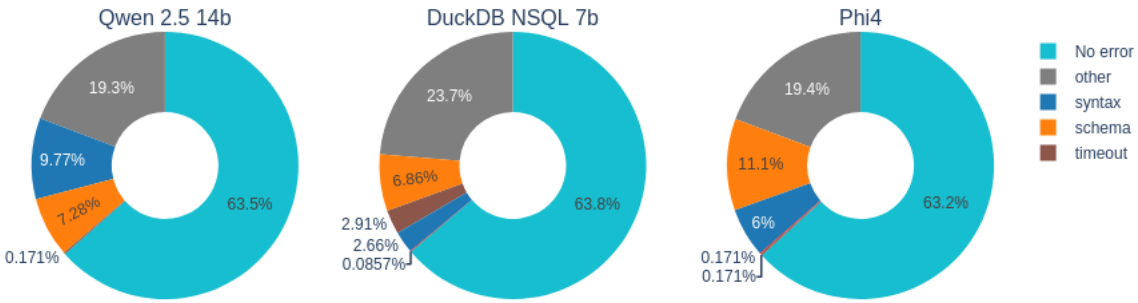


Figure 39: Combined Fine-Tuned Error Categories

As shown in the performance metrics, the fine-tuned Qwen 2.5 achieved approximately **78%** success across column/table access accuracy, with near **85%** similarity in query components (SELECT, FROM, and WHERE clauses).

Surprisingly, **fine-tuned Phi 4** demonstrated nearly identical performance with approximately **77%** success across column/table access accuracy and **82%** query component similarity. While fine-tuned DuckDB NSQL also performed adequately with **70%** structural accuracy and **67%** component similarity, it was notably outperformed by both Qwen 2.5 and Phi 4 across most structural metrics.

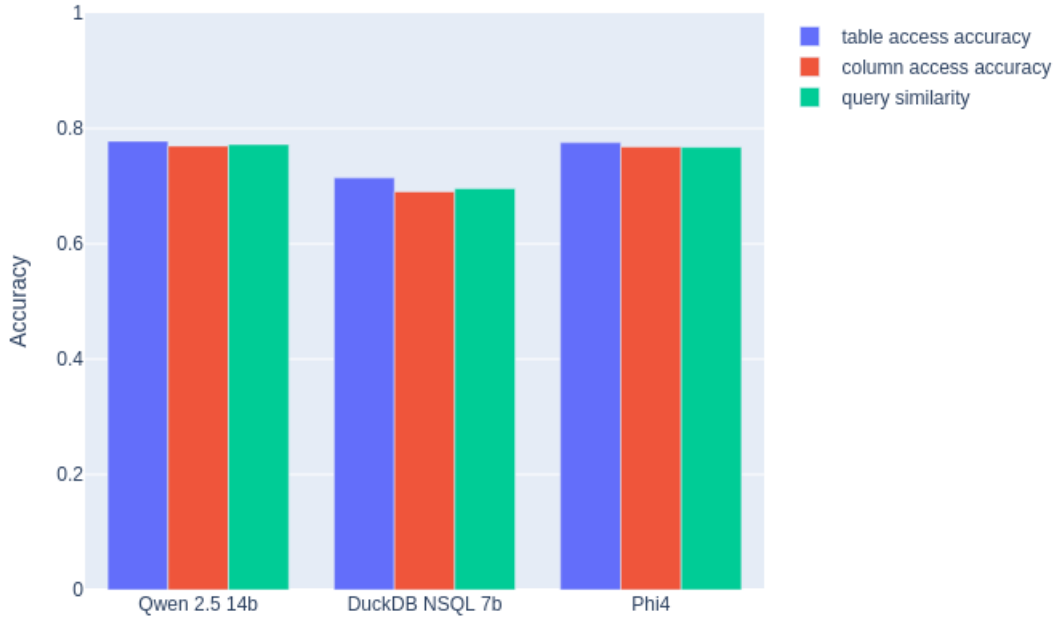


Figure 40: Comparative Fine-Tuned Structural Accuracy

It is worth noting that Phi4 performed **remarkably close** to Qwen 2.5, with nearly identical table access accuracy (77.5%), column access accuracy (76.8%), and composite score (56.2%).

Given this close performance between Qwen 2.5 and Phi4, **two versions** of the pipeline, with each model taking turns as the validator (M2) to experimentally determine the optimal configuration in the complete system, will be developed.

7 Results and Discussions

7.1 Pipeline Configuration Evaluation

Following the successful development of individual components, a complete end-to-end pipeline combining the fine-tuned generation and validation models was implemented and evaluated.

The pipeline architecture was designed to maximize query execution success while minimizing computational overhead through conditional validation[13].

The implementation follows a **two-stage processing flow**:

1. The natural language question is processed by the generator model (M1) to produce an initial SQL query
2. The generated query is executed against the MIMIC-IV database
3. If execution succeeds, results are returned without further processing
4. If execution fails, the validator model (M2) is triggered with the error message and original query
5. The validated query is executed, and final results are returned

Two pipeline configurations were evaluated to determine the optimal combination of models:

- **Configuration A**: Qwen 2.5 (Generator) + Qwen 2.5 (Validator)
- **Configuration B**: Qwen 2.5 (Generator) + Phi 4 (Validator)

Both configurations utilized the same **fine-tuned Qwen 2.5 14B model** as the primary generator (M1) but differed in the validation component (M2).

The evaluation was conducted on **100 samples of the validation split** of the dataset, analyzing both the individual component success rates and the cumulative pipeline performance.

[13]This conditional validation strategy optimizes computational efficiency, though the pipeline architecture remains flexible enough to verify all generated queries regardless of initial correctness assessment when needed.

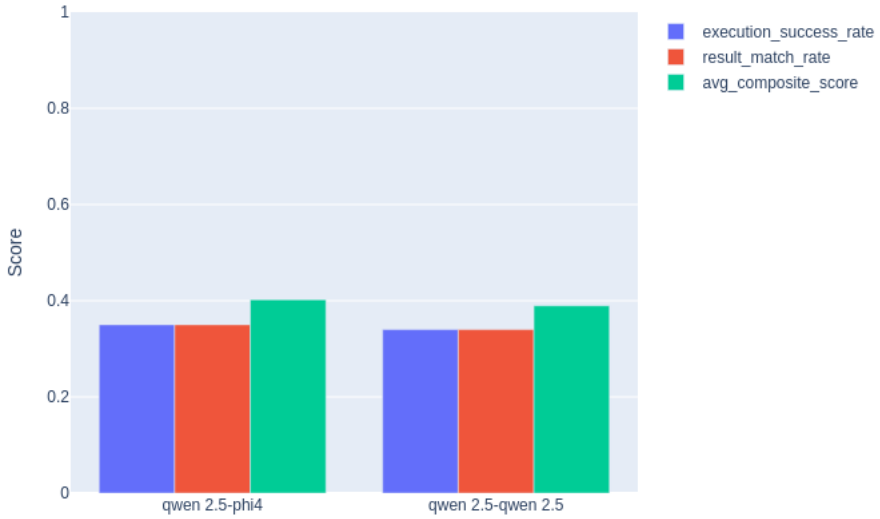


Figure 41: Overall Performance of Pipeline Configurations

As evident from Figure 41, both pipeline configurations showed overall results that were lower than those of individual components when evaluated in isolation.

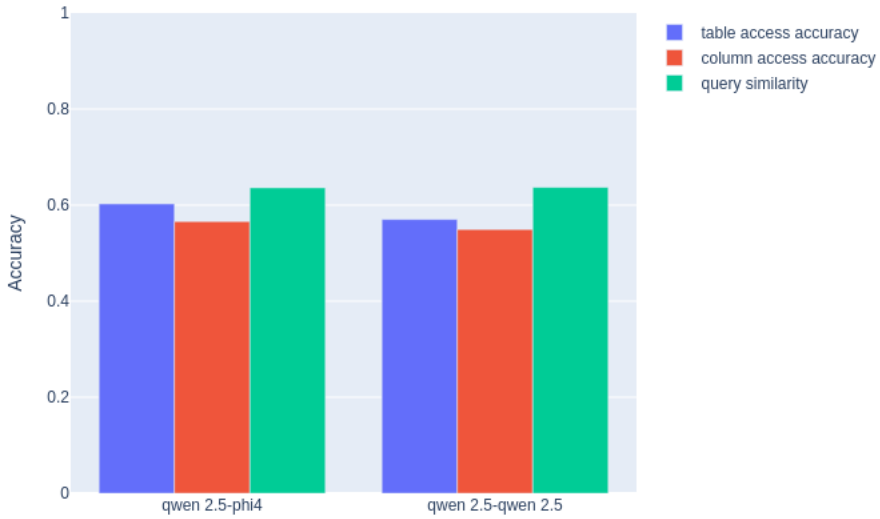


Figure 42: Structural Performance of Pipeline Configurations

Configuration B (Qwen 2.5 + Phi 4) showed slightly better execution success and composite scores than Configuration A. This suggests diverse model architectures might offer complementary strengths in our specific case, though the relative advantages of heterogeneous versus homogeneous model combinations remain largely unexplored and require further investigation.

Nonetheless, seeing Figure 42, the structural and component accuracy metrics remained surprisingly robust at approximately 60%, indicating that the foundational query architecture was well-preserved through the pipeline process.

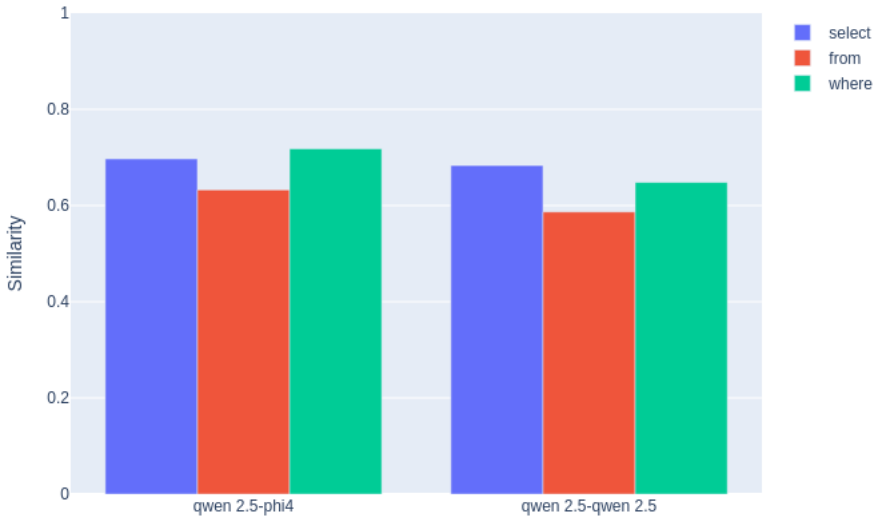


Figure 43: Component Performance of Pipeline Configurations

Despite the performance gap between individual components and the integrated pipeline, the system showed particular strength in handling questions with well-defined query structures.

The conditional validation architecture proved efficient by only activating the validator for queries that failed execution, thereby reducing unnecessary computation while effectively rehabilitating a portion of the failed queries.

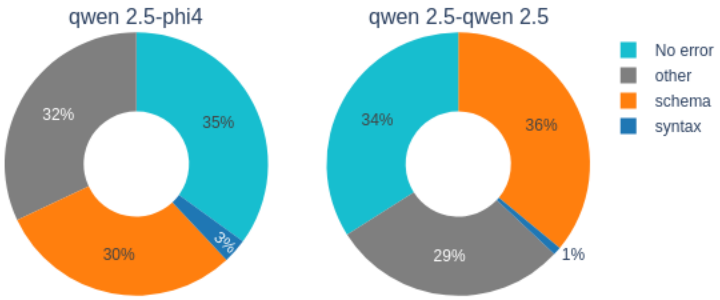


Figure 44: Error Analysis of Pipeline Configurations

Looking at Figure 43 and 44, the preservation of component similarity metrics suggests that while the system maintains basic structural coherence, schema-related errors remain a significant challenge.

These issues could potentially be addressed through targeted schema-specific fine-tuning or by incorporating more detailed database structure knowledge during the validation phase to improve the pipeline's understanding of complex table relationships and column properties.

To address schema-related challenges, two key enhancements are proposed: **schema-aware parameter adaptation** through targeted fine-tuning protocols focused on database structural comprehension, and enhanced structural representation **integrating more comprehensive database architecture specifications** to improve understanding of complex table relationships and column properties.

Pipeline optimization could be achieved through developing a **bidirectional feedback loop** between components for iterative query refinement, and incorporating **adaptive complexity management algorithms** to simplify execution-intensive queries while preserving semantic intent.

Based on the comprehensive evaluation, **Configuration B** (Qwen 2.5 Generator + Phi 4 Validator) is selected as the optimal pipeline configuration for the final system, balancing high execution success rates with strong schema comprehension and component accuracy.

Listing 9: Example Pipeline Execution

```
QUESTION: What was the last procedure icd code for patient 10014078
GENERATED SQL: SELECT procedures_icd.icd_code FROM procedures_icd
   WHERE procedures_icd.hadm_id IN ( SELECT admissions.hadm_id FROM
   admissions WHERE admissions.subject_id = 10014078 ) ORDER BY
   procedures_icd.charttime DESC LIMIT 1
```

```
EXECUTION SUCCESS: True
EXECUTION TIME: 0.0389 seconds
RESULTS:
icd_code
icd9|9671
```

As demonstrated in Listing 9, the pipeline successfully processed a natural language question about patient procedures, generating syntactically correct SQL that executes efficiently against the MIMIC-IV database.

This example shows direct generation without requiring validation, as the initial query executed successfully. Additional examples showcasing more complex queries and the validation process can be found in Listings 10 and 11 in the Appendix.

7.2 Model Performance Progression

7.2.1 Composite Score Evolution

Figure 45 and 46 illustrate the composite performance scores for individual models across each developmental stage.

As evident from Figure 45 and Table 7, individual generator models (M1) demonstrated distinct performance trajectories through the development pipeline, with Medalpaca 13b establishing the strongest zero-shot baseline with a composite score of **28.9%**[14], followed by Meditron 7b at **17%** and Qwen 2 Math 7b at **13.2%**.

[14]This model performed worst, returning empty queries for all inputs. Its apparent accuracy was due to these empty queries matching the expected null results in many cases, which was later identified from their lack of structure.

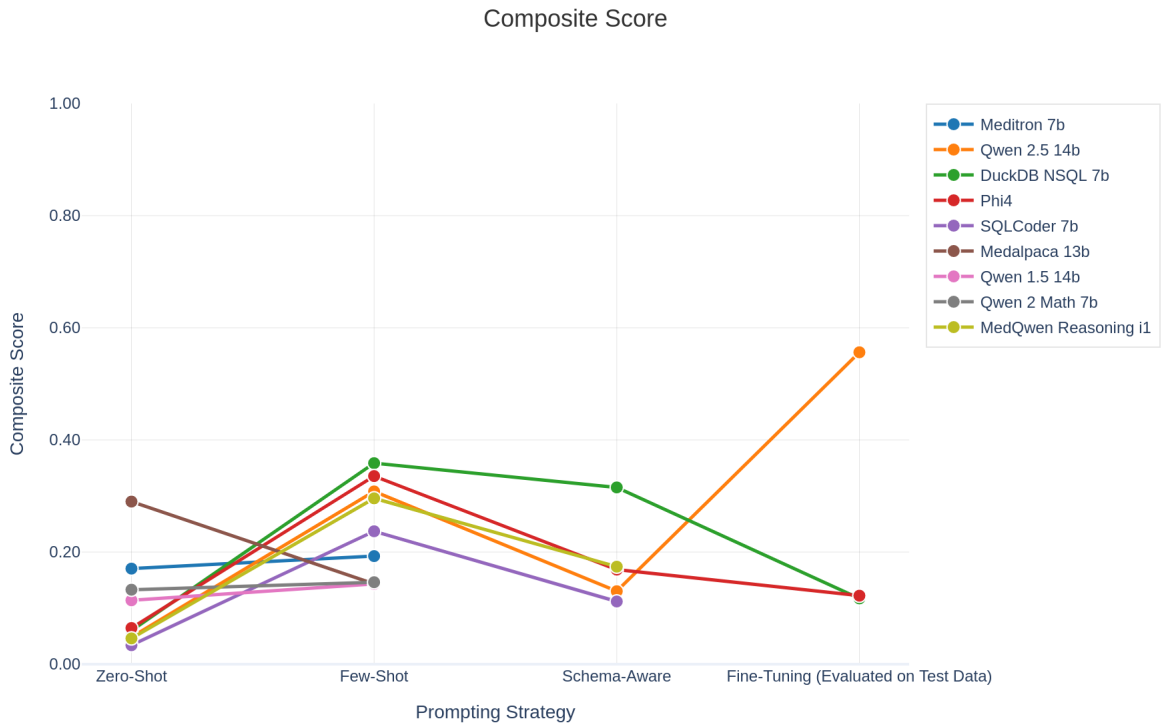


Figure 45: M1 Composite Scores Across Development Stages

In the few-shot stage, DuckDB NSQL 7b established the strongest baseline with a composite score of **35.8%**, followed by Phi4 at **33.5%** and Qwen 2.5 14b at **30.8%**.

The schema-aware phase revealed interesting model-specific behaviors. All selected M1 models experienced severe performance regression, with DuckDB NSQL 7b dropping to **31.5%**. This consistent pattern suggests inherent limitations in context window management when processing extensive schema information.

The fine-tuning phase produced the most significant differentiation between models. Qwen 2.5 demonstrated exceptional adaptability, achieving a composite score of **55.6%**.

Model	Zero-shot (%)	Few-shot (%)	Schema-aware (%)	Finetuned (%)
DuckDB NSQL 7b	6.00	**35.86**	**31.53**	11.74
MedQwen Reasoning i1	4.60	29.58	**17.39**	–
Medalpaca 13b	**29.00**	14.32	–	–
Meditron 7b	**17.05**	19.28	–	–
Phi4	6.45	**33.55**	16.87	12.22
Qwen 1.5 14b	11.40	14.31	–	–
Qwen 2 Math 7b	**13.28**	14.62	–	–
Qwen 2.5 14b	4.83	**30.81**	13.02	**55.63**
SQLCoder 7b	3.39	23.71	11.19	–

Table 7: M1 Model performance (%) across different stages

As evident from Figure 46 and Table 8, zero-shot validator models (M2) showed no meaningful improvement over their input queries from the best Schema Aware M1 model, falling short of expectations for error correction capabilities in this initial configuration.

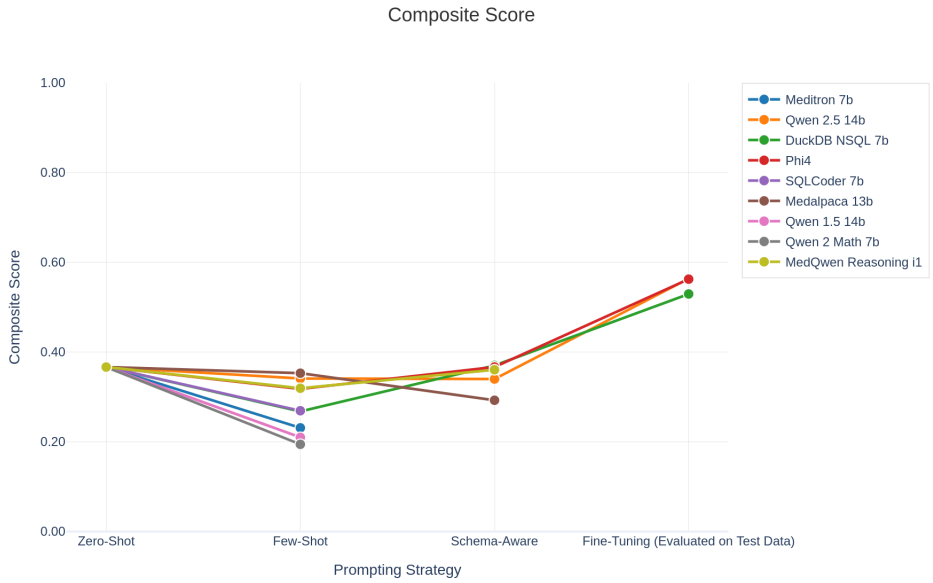


Figure 46: M2 Composite Scores Across Development Stages

In the few-shot stage, Medalpaca 13b established the strongest baseline with a composite score of **35.2%**, followed by Qwen 2.5 14b at **34.1%** and MedQwen Reasoning i1 at **31.9%**.

In the schema-aware phase, most models performed similar to few-shot stage, with DuckDB NSQL 7b leading with a composite score of **37%**, followed by Phi4 at **36.6%** and MedQwen Reasoning i1 at **36%**.

The fine-tuning phase produced the most significant improvements within models, with Qwen 2.5 and Phi4 demonstrating exceptional adaptability, achieving a composite score of **56.3%** and **56.2%** respectively.

Model	**Zero-shot (%)**	**Few-shot (%)**	**Schema-aware (%)**	**Finetuned (%)**
DuckDB NSQL 7b	36.67	26.76	**37.02**	52.94
MedQwen Reasoning i1	36.66	**31.93**	**36.06**	–
Medalpaca 13b	36.66	**35.29**	29.26	–
Meditron 7b	36.67	23.11	–	–
Phi4	36.67	31.80	**36.67**	**56.25**
Qwen 1.5 14b	36.65	20.99	–	–
Qwen 2 Math 7b	36.66	19.44	–	–
Qwen 2.5 14b	36.66	**34.10**	33.98	**56.35**
SQLCoder 7b	36.66	26.92	–	–

Table 8: M2 Model performance (%) across different stages

7.2.2 Overall Performance Trends

Figure 47 and 48 present the averaged performance metrics across all models for each development stage, highlighting collective improvement patterns.

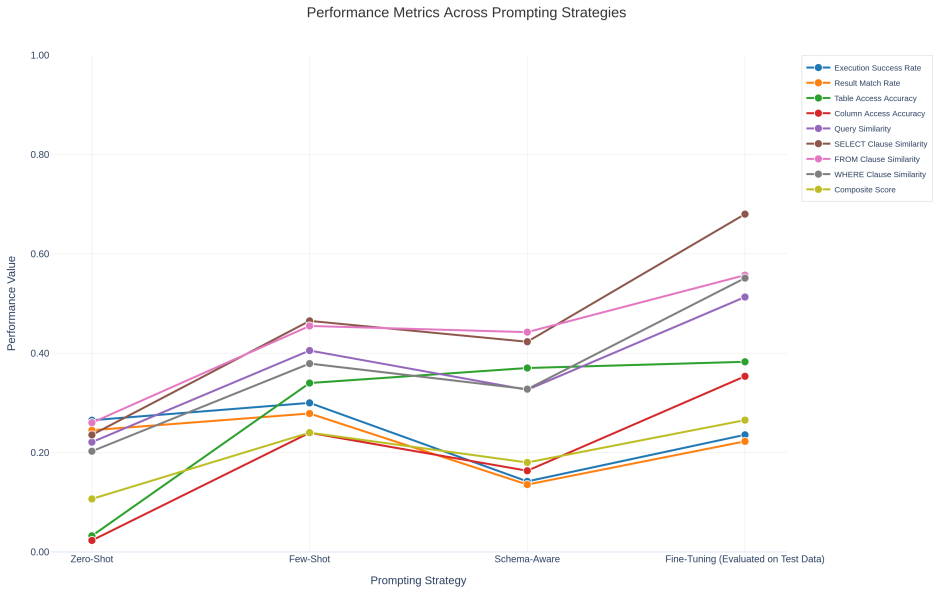


Figure 47: Average Performance Metrics for M1

Metric	**Zero-shot (%)**	**Few-shot (%)**	**Schema-aware (%)**	**Finetuned (%)**
Execution Success Rate	26.51	30.00	14.19	23.56
Result Match Rate	24.49	27.86	13.55	22.25
Avg. Table Access Accuracy	3.24	33.99	37.02	38.27
Avg. Column Access Accuracy	2.31	23.98	16.34	35.35
Avg. Query Similarity	22.10	40.53	32.63	51.30
Avg. SELECT Similarity	23.55	46.50	42.30	67.99
Avg. FROM Similarity	26.00	45.49	44.24	55.71
Avg. WHERE Similarity	20.26	37.91	32.76	55.12
Avg. Composite Score	10.67	24.01	18.00	26.53

Table 9: M1 Evaluation metrics (%) across different experiment settings

As seen in Figure 47 and Table 9, on average, generator models (M1) showed a clear progression pattern through the development stages.

From initial poor performance in zero-shot learning, models improved substantially with few-shot learning. The schema-aware phase demonstrated a slight regression in all metrics,

while fine-tuning produced dramatic improvements across table access accuracy, column access, and query similarity.

Figure 48: Average Performance Metrics for M2

As seen in Figure 48 and Table 10, validator models (M2) exhibited a consistent improvement pattern across all development stages.

Metric	**Zero-shot (%)**	**Few-shot (%)**	**Schema-aware (%)**	**Finetuned (%)**
Execution Success Rate	32.32	27.91	32.34	63.52
Result Match Rate	30.76	25.98	30.78	59.21
Avg. Table Access Accuracy	56.03	45.56	55.37	75.59
Avg. Column Access Accuracy	48.93	31.74	43.34	74.25
Avg. Query Similarity	64.39	45.59	56.21	74.50
Avg. SELECT Similarity	73.85	54.70	67.88	78.55
Avg. FROM Similarity	71.98	49.32	65.16	78.01
Avg. WHERE Similarity	57.89	42.64	53.29	78.10
Avg. Composite Score	36.66	27.81	34.60	55.18

Table 10: M2 Evaluation metrics (%) across different experiment settings

Starting with modest performance in the zero-shot implementation, these models showed steady gains when transitioning to few-shot learning. Unlike generator models, the schema-aware phase produced slight improvements rather than regression, suggesting that schema information provides particular value for error correction tasks.

After fine-tuning, validator models demonstrated substantial enhancements in table access accuracy, column access accuracy, and execution success rates, confirming the effectiveness of parameter-efficient adaptation for SQL validation and correction.

8 Conclusions

In this research, we developed a **two-stage pipeline** for converting natural language questions to SQL queries targeting the MIMIC healthcare database. Through our iterative development process, we combined a fine-tuned Qwen 2.5 generator with a Phi 4 validator in a **conditional architecture**, achieving approximately 30% result accuracy and 60% structural accuracy while optimizing computational resources through selective validation.

Our comprehensive evaluation across different model configurations revealed valuable insights into the dynamics of text-to-SQL conversion in the medical domain. **Heterogeneous model combinations** appeared to slightly outperform homogeneous configurations in our limited testing, though the difference was marginal, with Qwen 2.5 showing exceptional adaptability through fine-tuning (55.6% composite score for generation).

We observed an intriguing pattern where generator models experienced **performance regression** during the schema-aware phase, while validator models showed improvements in the same phase, suggesting fundamentally different information processing requirements between these complementary tasks.

Despite promising advances in structural accuracy, **schema-related errors** remain a significant challenge in our implementation. The complex relationships between medical database tables continue to present difficulties for modern language models.

In conclusion, this work lays a foundation for future research in medical text-to-SQL systems, highlighting the importance of adaptability and domain-specific optimization in medical NLP systems. We envision several promising directions, including implementing **bidirectional feedback mechanisms** between components, developing more sophisticated schema representation techniques, and exploring adaptive complexity management algorithms for handling increasingly complex queries.

While our current results demonstrate meaningful progress, we recognize that our contribution is just one step in the broader effort to bridge the gap between natural language and structured medical data access.

9 Future Work and Extensions

9.1 Architectural and Computational Enhancements

The current implementation faces limitations due to hardware constraints, with both models sharing a single 24GB GPU. Future work could explore several architectural improvements:

- Incorporating larger state-of-the-art models such as GPT-4, DeepSeek, and Qwen 3, could significantly enhance performance but would require expanded computational resources.
- A bidirectional iterative approach where generation and validation models communicate until an optimal query is achieved could improve accuracy beyond the current two-stage pipeline. This could be implemented as a conversational agent framework where the generator proposes a query, the validator identifies issues, and the generator refines based on this feedback, progressively narrowing uncertainty through multiple iterations, potentially handling edge cases that the current pipeline misses.

  This approach would also align with recent advances in agentic AI systems, which have demonstrated improved problem-solving through recursive self-improvement.
- Addressing bottlenecks for queries that return extensive results would improve system responsiveness and scalability for complex clinical analyses.

9.2 Cross-Database Adaptability

- Adapting the approach to other electronic health record systems beyond MIMIC-IV by linking alternative databases to the Query Execution module and fine-tuning models on their respective schemas.
- Exploring whether models trained on one medical database schema can effectively transfer to others, potentially reducing training requirements for new implementations.

9.3 User Interface Enhancements

- Creating a user interface that supports natural language input, potentially including voice-to-text functionality and guided query construction for more focused searches.
- Developing advanced guided query interfaces that allow users to blend natural language with structured field selection, enabling specific parameters (like patient

demographics, date ranges, or specific medical codes) directly into the query while using natural language for complex relationships and conditions.

9.4 Evaluation and Integration

- Implementing result confidence scores to provide users with transparency regarding the system's certainty in its query formulation and results.
- Investigating how the system could integrate into existing healthcare workflows to maximize utility while maintaining appropriate privacy safeguards.
- Conducting user studies with healthcare stakeholders, including clinicians, researchers, and administrators of varying technical expertise, to evaluate system usability and assess how effectively non-technical users can formulate complex queries, and whether outputs facilitate clinical decision-making.

9.5 Dataset Enhancements

- Developing methods to create larger training datasets through rule-based synthetic query generation with language model refinement. This pipeline would generate diverse SQL queries covering MIMIC-IV tables, create template-based natural language questions, and apply style transfer techniques to produce more naturalistic phrasing.
- Implementing verification mechanisms to ensure synthetic question-query pairs maintain semantic equivalence while introducing linguistic diversity. These would validate that generated questions accurately represent the intended SQL queries and retrieve expected database results.

10 Appendix

10.1 Figures, Code and Tables from 6

10.1.1 Modified Schema from 6.3.2

```
{
  "admissions": ["subject_id", "hadm_id", "admittime", "dischtime",
  ↪  "admission_type", "admission_location", "discharge_location", "insurance",
  ↪  "language", "marital_status", "age"],
  "chartevents": ["subject_id", "hadm_id", "stay_id", "itemid", "charttime",
  ↪  "valuenum", "valueuom"],
  "cost": ["subject_id", "hadm_id", "event_type", "event_id", "chargetime",
  ↪  "cost"],
  "d_icd_diagnoses": ["icd_code", "long_title"],
  "d_icd_procedures": ["icd_code", "long_title"],
  "d_items": ["itemid", "label"],
  "d_labitems": ["itemid", "label"],
  "diagnoses_icd": ["subject_id", "hadm_id", "icd_code", "charttime"],
  "icustays": ["subject_id", "hadm_id", "stay_id", "first_careunit",
  ↪  "last_careunit", "intime", "outtime"],
  "inputevents": ["subject_id", "hadm_id", "stay_id", "starttime", "itemid",
  ↪  "totalamount", "totalamountuom"],
  "labevents": ["subject_id", "hadm_id", "charttime", "valuenum", "valueuom"],
  "microbiologyevents": ["subject_id", "hadm_id", "charttime", "spec_type_desc",
  ↪  "test_name", "org_name"],
  "outputevents": ["subject_id", "hadm_id", "stay_id", "charttime", "itemid",
  ↪  "value", "valueuom"],
  "patients": ["subject_id", "gender", "dob", "dod"],
  "prescriptions": ["subject_id", "hadm_id", "starttime", "stoptime", "drug",
  ↪  "dose_val_rx", "dose_unit_rx", "route"],
  "procedures_icd": ["subject_id", "hadm_id", "icd_code", "charttime"],
  "transfers": ["subject_id", "hadm_id", "transfer_id", "eventtype", "careunit",
  ↪  "intime", "outtime"]
}
```

Figure 49: Tables and their columns from the modified MIMIC schema

10.1.2 Sample Execution Plan from 6.4.4

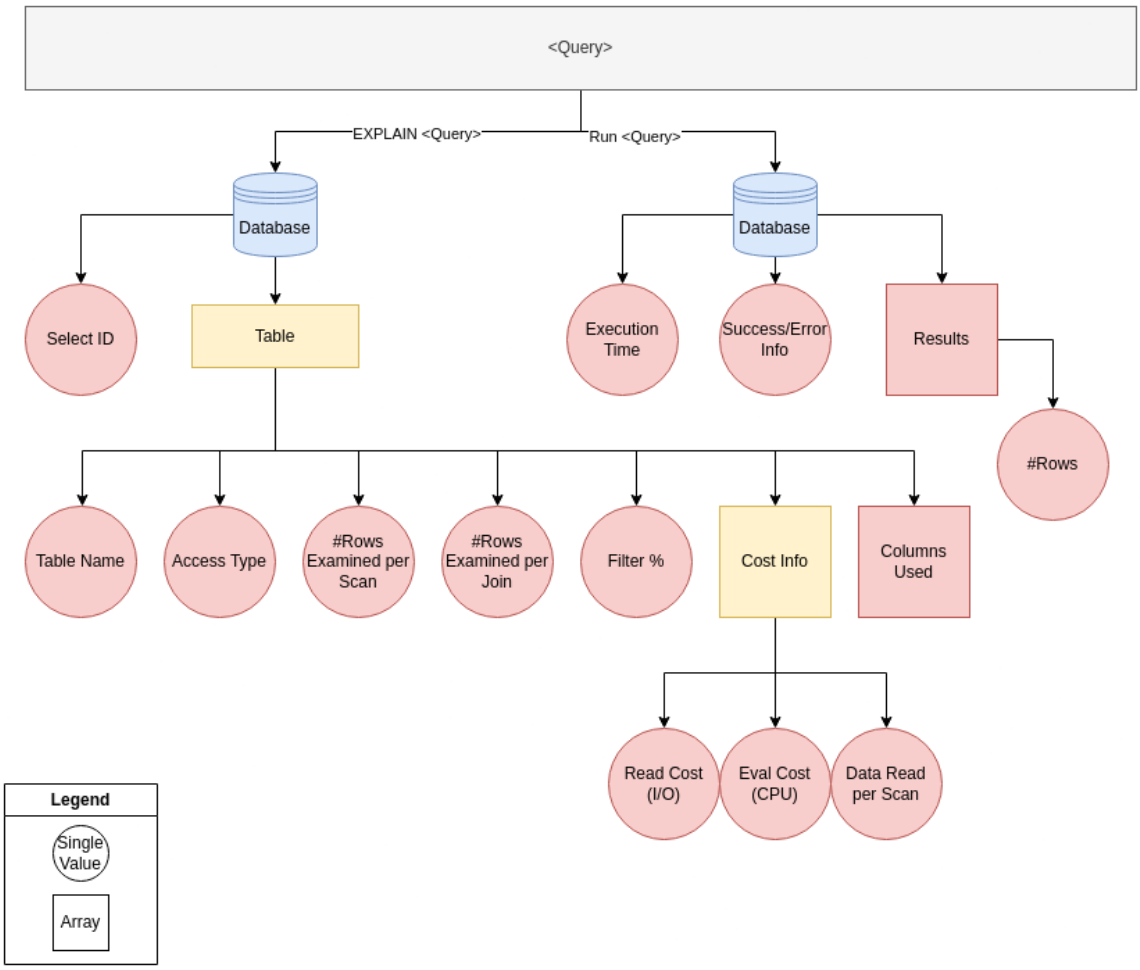


Figure 50: Sample Result Structure of EXPLAIN clause

10.2 Listings from 7

10.2.1 Examples of NLQ-to-SQL Conversion from 7.1

Listing 10: Pipeline Validation Example 2

```
QUESTION: Has any midazolam been prescribed to patient 10019172?
GENERATED SQL: SELECT COUNT(*)>0 FROM prescriptions WHERE
   prescriptions.drug = 'midazolam' AND prescriptions.patient_id =
   10019172
VALIDATION TRIGGERED: Yes
ERROR: None
VALIDATED SQL: SELECT COUNT(*)>0 FROM prescriptions WHERE
   prescriptions.drug = 'midazolam' AND prescriptions.hadm_id IN (
```

```
   SELECT admissions.hadm_id FROM admissions WHERE admissions.
   subject_id = 10019172 );
EXECUTION SUCCESS: True
EXECUTION TIME: 0.3882 seconds
RESULTS:
COUNT(*)>0
1
```

In Listing 10, we observe the validator successfully addressing a schema-related error in the generator's output. The initial query incorrectly referenced a non-existent `patient_id` column in the `prescriptions` table. The validator recognized this schema inconsistency and reformulated the query using a subquery that properly joins `prescriptions` with the `admissions` table using `hadm_id` as the appropriate foreign key relationship.

Listing 11: Pipeline Validation Example 3

```
QUESTION: What drug was first prescribed to patient 10010471 via neb
    route since 12/2100?
GENERATED SQL: SELECT drugs.drug FROM ( SELECT admissions.subject_id
   , prescriptions.drug, prescriptions.starttime FROM prescriptions
   JOIN admissions ON prescriptions.hadm_id = admissions.hadm_id
   WHERE admissions.subject_id = 10010471 AND prescriptions.route =
   'neb' AND strftime('%Y-%m',prescriptions.starttime) >= '2100-12'
   ORDER BY prescriptions.starttime ASC LIMIT 1 ) AS
   first_neb_prescription
VALIDATION TRIGGERED: Yes
ERROR: None
VALIDATED SQL: SELECT prescriptions.drug FROM prescriptions WHERE
   prescriptions.hadm_id IN ( SELECT admissions.hadm_id FROM
   admissions WHERE admissions.subject_id = 10010471 ) AND
   prescriptions.route = 'neb' AND strftime('%Y-%m',prescriptions.
   starttime) >= '2100-12' ORDER BY prescriptions.starttime ASC
   LIMIT 1;
EXECUTION SUCCESS: True
EXECUTION TIME: 0.4174 seconds
RESULTS:
drug
ipratropium-albuterol neb
```

Listing 11 demonstrates the validator's ability to simplify unnecessarily complex queries while maintaining correct logical structure. The initial query contained an overly elaborate nested subquery with an erroneous reference to a non-existent `drugs` table and improper query structure. The validator successfully restructured the query into a more direct form using a cleaner subquery pattern, while preserving the essential filtering criteria and sort order to identify the first medication administered via nebulizer route.

References

Michael Han Daniel Han and Unsloth team. Unsloth, 2023. URL http://github.com/unslothai/unsloth.

Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-Wei H Lehman, Mengling Feng, Marzyeh Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iv: A freely accessible critical care database. *Scientific Data*, 10(1):1–14, 2023. doi:10.1038/s41597-023-02055-7.

Ayush Kumar, Parth Nagarkar, Prabhav Nalhe, and Sanjeev Vijayakumar. Deep learning driven natural languages text to sql query conversion: A survey. *arXiv preprint arXiv:2208.04415*, 2022. URL https://arxiv.org/abs/2208.04415.

Gyubok Lee, Sunjun Kweon, Seongsu Bae, and Edward Choi. Overview of the EHRSQL 2024 shared task on reliable text-to-SQL modeling on electronic health records. In *Proceedings of the 6th Clinical Natural Language Processing Workshop*, pages 644–654, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi:10.18653/v1/2024.clinicalnlp-1.62. URL https://aclanthology.org/2024.clinicalnlp-1.62/.

Ali Mohammadjafari, Anthony S. Maida, and Raju Gottumukkala. From natural language to sql: Review of llm-based text-to-sql systems, 2025. URL https://arxiv.org/abs/2410.01066.

Zheng Ning, Yuan Tian, Zheng Zhang, Tianyi Zhang, and Toby Jia-Jun Li. Insights into natural language database query errors: From attention misalignment to user handling strategies. *arXiv preprint arXiv:2402.07304*, 2024. URL https://arxiv.org/abs/2402.07304.

Richard Tarbell, Kim-Kwang Raymond Choo, Glenn Dietrich, and Anthony Rios. Towards understanding the generalization of medical text-to-sql models and datasets. *arXiv preprint arXiv:2303.12898*, 2023. URL https://arxiv.org/abs/2303.12898.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task, 2018. URL https://aclanthology.org/D18-1425/.

Xiaohu Zhu, Qian Li, Lizhen Cui, and Yongkang Liu. Large language model enhanced text-to-sql generation: A survey. *arXiv preprint arXiv:2410.06011*, 2024. URL https://arxiv.org/abs/2410.06011.

Angelo Ziletti and Leonardo D'Ambrosi. Retrieval augmented text-to-sql generation for epidemiological question answering using electronic health records. In *Proceedings of the 6th Clinical Natural Language Processing Workshop*, pages 47–53. Association for Computational Linguistics, 2024. doi:10.18653/v1/2024.clinicalnlp-1.4. URL http://dx.doi.org/10.18653/v1/2024.clinicalnlp-1.4.