NATURAL LANGUAGE TO SQL QUERY GENERATION USING GEN AI

A PROJECT REPORT

Submitted by

AJANEESHWAR S

KAMALESHWAR A

NAVEEN KUMAR K R

SIDDHARTHAN R

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

MAY, 2025

NATURAL LANGUAGE TO SQL QUERY GENERATION USING GEN AI

A PROJECT REPORT

Submitted by

AJANEESHWAR S (811721243005)

KAMALESHWAR A (811721243023)

NAVEEN KUMAR K R (811721243038)

SIDDHARTHAN R (811721243052)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

MAY, 2025

K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY
(AUTONOMOUS)
SAMAYAPURAM – 621112

BONAFIDE CERTIFICATE

Certified that this project report titled **"NATURAL LANGUAGE TO SQL QUERY GENERATION USING GEN AI"** is the bonafide work of **AJANEESHWAR S (811721243005), KAMALESHWAR A (811721243023), NAVEENKUMAR K R (811721243038), SIDDHARTHAN R (811721243052)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
Dr. T.Avudaiappan M.E., Ph.D.,	Mrs.Joany Franklin,M.E.,
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Department of Artificial Intelligence	ASSISTANT PROFESSOR
K.Ramakrishnan College of Technology	Department of Artificial Intelligence
(Autonomous)	K.Ramakrishnan College of Technology
Samayapuram – 621 112	(Autonomous)
	Samayapuram – 621 112

Submitted for the viva-voce examination held on ………………

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

We jointly declare that the project report on **“NATURAL LANGUAGE TO SQL QUERY GENERATION USING GEN AI”** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **“ANNA UNIVERSITY CHENNAI”** for the requirement of Degree of **BACHELOR OF TECHNOLOGY**. This design project report is submitted on the partial fulfillment of the requirement of the award of Degree of **BACHELOR OF TECHNOLOGY**.

SIGNATURE

AJANEESHWAR S

KAMALESHWAR A

NAVEEN KUMAR K R

SIDDHARTHAN R

Place**:** Samayapuram

Date**:**

ACKNOWLEDGEMENT

It is with great pride that we express our gratitude and in-debt to our institution "**K.Ramakrishnan College of Technology (Autonomous)**", for providing us with the opportunity to do this project.

We are glad to credit honourable chairman **Dr. K.RAMAKRISHNAN, B.E.,** for having provided for the facilities during the course of our study in college.

We would like to express our sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA, Ph.D.,** for forwarding to our project and offering adequate duration in completing our project.

We would like to thank **Dr. N. VASUDEVAN, M.E., Ph.D.,** Principal, who gave opportunity to frame the project the full satisfaction.

We whole heartily thank to **Dr. T. AVUDAIAPPAN**, **M.E., Ph.D.,** Head of the department, **ARTIFICIAL INTELLIGENCE** for providing his encourage pursuing this project.

We express our deep and sincere gratitude to our project guide **Mrs. JOANY FRANKLIN, M.E.,** Department of **ARTIFICIAL INTELLIGENCE,** for her incalculable suggestions, creativity, assistance and patience which motivated us to carry out this project.

We render our sincere thanks to Course Coordinator and other staff members for providing valuable information during the course.

We wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

ABSTRACT

Natural Language Processing (NLP) is increasingly being applied to make data querying more accessible for non-technical users. This project presents an intelligent system that converts natural language queries into structured SQL commands using a Large Language Model (LLM), specifically Google’s Gemini. The goal is to bridge the gap between human language and database querying, enabling users to interact with data using everyday language without the need to understand SQL syntax. The system architecture comprises six core modules: User Interface, Input Processing, LLM-Based Query Generation, Query and Output, Database Execution, and Model Performance Evaluation. Users input queries through a clean and intuitive interface, which are then analyzed and enriched during input processing to include relevant schema and table metadata. The enhanced prompt is passed to the Gemini LLM, which generates context-aware SQL queries by understanding the structure and semantics of the underlying database. Generated SQL queries are either displayed to the user for confirmation or executed directly on a backend relational database such as SQLite.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

GEN AI	-	Generative Artificial Intelligence
LLM	-	Large Language Model
MLP	-	Multi Layered Perceptron
NL	-	Natural Language
SQL	-	Structured Query Language
UI	-	User Interface

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

Natural Language to SQL Query Generation Using Gen AI project seeks to narrow the disparity between human and formal database queries. The tool provides users the facility to import a dataset and utilize natural language questions to handle the dataset instead of writing the queries manually with SQL. Leveraging Generative AI, the model translates queries submitted by users, identifies the structure of the dataset, and returns the suitable SQL queries for retrieving correct information.

The essence of this project lies in natural language processing (NLP) and machine learning methodologies to process user input and translate it into pertinent SQL syntax. The system remains flexible by adjusting to various datasets, thus becoming suitable for any number of domains, such as finance, medicine, and commerce. It also enhances accessibility for non-technical users who do not understand SQL but require knowledge from structured data.

This application can be very helpful for researchers, business people, and data analysts, minimizing query time and increasing productivity. Future development can include optimizing the AI model to improve accuracy, supporting complex queries, and incorporating visualization capabilities for a more intuitive interface.

Natural Language to SQL (NL2SQL) query generation closes the gap by facilitating access to databases via everyday language. Generative AI (Gen AI), with its underpinning by Large Language Models (LLMs) and deep learning, streamlines the process through understanding user intent, ambiguity resolution, and generation of optimal SQL queries.

1.2 OBJECTIVE

The aim of the project is to make database interactions simpler by allowing users to query data using natural language rather than manually writing SQL queries. The project seeks to improve access, efficiency, and accuracy in database querying, as well as facilitate easier interaction with structured data for users who do not have SQL skills. By enabling users to upload their own data, the system dynamically adjusts to various data structures and creates SQL queries based on user input.

One of the prime objectives is to utilize Generative AI and Natural Language Processing (NLP) to properly analyze user queries, comprehend dataset schema, and form syntactically correct SQL statements.

This ensures that even queries with conditions, aggregations, and joins can be processed with ease. Furthermore, the system must be able to offer query optimization to optimize performance and yield results in a timely manner.

Aside from ease of use, the project also hopes to improve decision-making through making it possible for users-various data analysts to business practitioners to derive significant insights without SQL expertise. The tool is built to be domain-agnostic and can be applied across different fields like healthcare, finance, education, and e-commerce.

In the long run, the project hopes to incorporate cutting-edge AI methods for processing fuzzy queries, enhancing context awareness, and even generating visualizations for query outputs. This technology has the potential to lower the learning curve for database interaction and enhance productivity in data-centric-environments.

CHAPTER 2
LITERATURE SURVEY

2.1 TEXT ANALYSIS BASED ON NATURAL LANGUAGE PROCESSING (NLP)

Azhar Kassem Flayeh, Yaser Issam Hamod, Nashwan Dheyaa Zaki

In the modern era of information related to images and multimedia and the era of the information explosion, where there is enormous information that contains thousands and millions of texts loaded on the Internet, which is rich in more than 1.2 million terabytes, knowing that there are no accurate statistics on the size of texts within this huge amount of data, and with The multiplicity of websites on the Internet and the spread of social networking services, for example, Twitter in the last statement, amounting to 6000 tweets per second in the year 2019. Here comes to mind what information these texts hide? As we are among those browsing these sites, we strive to read and analyze these texts, and here we must resort to artificial intelligence to help us solve this problem. In this article, we will review the latest concepts in the field of word processing, extracting information from it, and implementing an interface for text analysis that was implemented using the C++ programming language and Visual Studio 2015 edition to compare between two articles in terms of the number of verbs, prepositions, article strength, number of letters, as well as possible to sort verbs, nouns, object, adjectives etc. as it will be displayed in the program interface.

Merits

Extensive analysis, Accurate, Enhanced customer satisfaction.

Demerits

Limited scope, Unpredictable errors, Context loss.

2.2 GENERATIVE ARTIFICIAL INTELLIGENCE (GENAI) IN THE RESEARCH PROCESS – A SURVEY OF RESEARCHERS' PRACTICES AND PERCEPTIONS

Jens Peter Andersen, Lise Degn, Rachel Fishberg, Ebbe K. Graversen, Serge P.J.M. Horbach

This study explores the use of generative AI (GenAI) and research integrity assessments of use cases by researchers, including PhD students, at Danish universities. Conducted through a survey sent to all Danish researchers from January to February 2024, the study received 2534 responses and evaluated 32 GenAI use cases across five research phases: idea generation, research design, data collection, data analysis, and writing/ reporting. Respondents reported on their own and colleagues' GenAI usage. They also assessed whether the practices in the use cases were considered good research practice. Through an explorative factor analysis, we identified three clusters of perception: "GenAI as a work horse", "GenAI as a language assistant only", and "GenAI as a research accelerator". The findings further show varied opinions on GenAI's research integrity implicationsControversial areas included image creation/modification and synthetic data, with comments highlighting the need for critical and reflexive use of GenAI. Usage differed by main research area, with technical and quantitative sciences reporting slightly higher usage and more positive assessments. Junior researchers used GenAI more than senior colleagues, while no significant gender differences were observed. The study underscores the need for adaptable, discipline-specific guidelines for GenAI use in research, developed collaboratively with experts to align with diverse one.

Merits

Efficiency, Time-Saving, Enhanced language assistance.

Demerits

Ethical concern, Criticism in certain tasks, Integrity and trust issues.

2.3 PROMPT EVOLUTION FOR GENERATIVE AI: A CLASSIFIER-GUIDED APPROACH

Melvin Wong, Yew-Soon Ong, Abhishek Gupta, Kavitesh Kumar Bali, Caishun Chen

Synthesis of digital artifacts conditioned on user prompts has become an important paradigm facilitating an explosion of use cases with generative AI. However, such models often fail to connect the generated outputs and desired target concepts/preferences implied by the prompts. Current research addressing this limitation has largely focused on enhancing the prompts before output generation or improving the model's performance up front. In contrast, this paper conceptualizes prompt evolution, imparting evolutionary selection pressure and variation during the generative process to produce multiple outputs that satisfy the target concepts/preferences better. We propose a multi-objective instantiation of this broader idea that uses a multi-label image classifier-guided approach. The predicted labels from the classifiers serve as multiple objectives to optimize, with the aim of producing diversified images that meet user preferences. A novelty of our evolutionary algorithm is that the pre-trained generative model gives us implicit mutation operations, leveraging the model's stochastic generative capability to automate the creation of Pareto-optimized images more faithful to user preferences.

Merits

Increased diversity, Improved output quality, Adaptability.

Demerits

Complex setup, High computational cost, Potential overfitting.

2.4 REPROMPT: AUTOMATIC PROMPT EDITING TO REFINE AI-GENERATIVE ART TOWARDS PRECISE EXPRESSIONS

Yunlong Wang, Shuyuan Shen, Brian Y. Lim

Generative AI models have shown impressive ability to produce images with text prompts, which could benefit creativity in visual art creation and self-expression. However, it is unclear how precisely the generated images express contexts and emotions from the input texts. We explored the emotional expressiveness of AI-generated images and developed RePrompt, an automatic method to refine text prompts toward precise expression of the generated images. Inspired by crowdsourced editing strategies, we curated intuitive text features, such as the number and concreteness of nouns, and trained a proxy model to analyze the feature effects on the AI-generated image. With model explanations of the proxy model, we curated a rubric to adjust text prompts to optimize image generation for precise emotion expression. We conducted simulation and user studies, which showed that RePrompt significantly improves the emotional expressiveness of AI-generated images, especially for negative emotions.

Merits

Explainable AI integration, Automatic prompt editing, Generalization Potential.

Demerits

Feature selection limitations, Dependency on specific datasets, Bias in generative models.

2.5 BASED ON NATURAL LANGUAGE PROCESSING, HUMAN-COMPUTER DIALOGUE, IMAGE RECOGNITION, AND MACHINE LEARNING ANALYSIS WHETHER ARTIFICIAL INTELLIGENCE WILL SURPASS THE HUMAN BRAIN

Chunxu MU

With the popularization and development of the concept of artificial intelligence, the application of artificial intelligence has also begun to deepen into people's lives. While bringing convenience to people, it has also made some people worry about whether artificial intelligence will replace humans. Therefore, In order to make people understand the current development status and bottlenecks of artificial intelligence more intuitively, as well as the difference between artificial intelligence and human brain, this article will turn from speech recognition and natural language processing, human-computer dialogue, image recognition, and machine learning ability, that is, machine listening, reading, and thinking four aspects of research and discussion, and finally summarize why artificial intelligence cannot completely surpass humans.

Merits

Wide application range, Accurate recognition, Scalability.

Demerits

High error rates in complex conditions, Dependence on large datasets.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Under the current scenario, natural language to SQL query generation systems usually have limitations when working with user-uploaded custom datasets. Most existing models and tools are trained with pre-defined schemas or structured databases, limiting their flexibility in handling various datasets uploaded by users. These systems generally demand predefined database schema, i.e., users either have to adhere to a pre-existing dataset structure or manually struct their data for the specified form before executing queries. This renders them less efficient and less user-friendly for individuals requiring dynamic query generation. Another issue in current systems is that they lack flexibility to user-uploaded data sets. Most SQL generation tools that use AI are based on pre-trained models, which are efficient only for certain data sets for which they are trained. Therefore, when users try to query their own data sets, the models fail to comprehend the schema and table-to-table relationships and produce wrong or incomplete queries.

In addition, most conventional SQL generators have poor support for complicated queries, including queries with several joins, nested queries, or aggregations. They tend not to produce very well-optimized SQL queries, which become inefficient data operations. Moreover, some systems necessitate manual intervention to fine-tune or fix the queries, which defeats the aim of automation.

In general, the current solutions are not so dynamic and flexible when adapting to varied user datasets, with non-technical users being challenged to easily produce relevant SQL queries. The void in available solutions indicates a call for an even more versatile AI-based solution capable of facilitating custom uploads of datasets and creating correct SQL queries in real-time.

3.1.1 Demerits

- **No Support for Custom Datasets** – Most current systems function on pre-existing schemas and do not support the upload of users' own datasets, rendering them inflexible and inappropriate for fluid data analysis.

- **Poor Adaptability to New Data Structures** – AI models in existing systems are not able to comprehend new da 8 schemas, resulting in erroneous query generation when users present various data formats or relational structures.

- **Lack of Support for Complex Queries** – Most existing systems are not capable of generating SQL queries with joins, nested queries, aggregations, or complex filtering, which restricts their applicability in actual database operations.

- **Heavy Reliance on Pre-trained Models** – As the majority of tools utilize pre-trained models, they usually are not able to generalize that well across multiple datasets and result in irrelevant or incorrect query outputs.

- **Manual Fixing Needed** – Users mostly have to fix or adjust automatically generated SQL queries manually, which goes against achieving complete automation and makes it take longer to receive meaningful insights.
- **Performance Inefficiency** – Most current models produce non-optimized SQL queries that can make database operations slow, leading to delays and inefficiencies, particularly when handling large datasets.

3.2 PROPOSED SYSTEM

The provided system is a step forward from existing solutions in that it allows the uploading of a certain dataset and uses the created SQL queries directly based on the user's data.

In contrast to traditional models dependent on predefined schemes, the current system probes the uploaded dataset dynamically and learns according to its layout, so it is very flexible and convenient for use.

The system starts with the User Interface (UI) Module, which is intuitive and simple. Users are allowed to upload data sets, select between SQLite and PostgreSQL modes, and type in queries in plain language. Such an arrangement renders the system suitable for technical as well as non-technical users.

Then, the Input Processing Module sanitizes the user's natural language question by pre-processing and formatting the input. It does operations such as tokenizing, removing stop words, and recognizing relevant entities like table names or conditions so that the query is in a suitable format for semantic interpretation.

The sanitized input is then forwarded to the LLM-based Query Generation Module, implemented using Google's Gemini model. Utilizing Generative AI and NLP, the model perceives the user intent, has an understanding of relationships within the dataset, and produces correct SQL queries. Complex queries that may include joins, aggregations, or nested conditions are also tackled efficiently and properly without the involvement of human interference.

After the query is created, the Query Execution and Output Module executes it in real-time against the chosen database format. It fetches the output and presents it in a neat, readable table format. This allows users to immediately view both the generated SQL and the results—exactly as they would in any ordinary SQL environment.

To finalize the process, the Model Performance Evaluation Module gives important performance insights, including query generation and execution time. This assists users in comprehending the system's efficiency, particularly for queries with different complexity, and supports optimization.

It optimizes queries to enhance performance, eliminate response time, and enhance overall execution speed. The system is made to learn and adapt to different dataset formats as time passes, enhancing accuracy and flexibility.

3.2.1 Merits

- **Support for Custom Datasets** – Users are able to upload their own datasets, so that the SQL queries generated are directly applicable to their data, as opposed to current systems that use predefined schemas.

- **Enhanced Query Accuracy** – The system dynamically comprehends dataset structures, so that SQL queries are compatible with the uploaded data, resulting in more accurate and meaningful query results.

- **Handles Advanced Queries** – Capable of executing advanced SQL operations such as joins, aggregations, filtering, and nested queries, making it applicable to detailed data analysis.

- **CSV Download Feature** – The system allows users to download the generated SQL query results directly as a CSV file, enabling seamless data analysis, reporting, and integration with spreadsheet tools like Excel or Google Sheets.

- **Non-Technical and No SQL Required** – Allows non-technical users to query data using natural language, without the need for learning SQL syntax and simplifying database access.

- **Flexible Across Domains** – The framework can be utilized in finance, healthcare, education, e-commerce, and other sectors, thereby being a domain-agnostic solution for multiple data-driven use cases.

CHAPTER 4

SYSTEM SPECIFICATIONS

4.1 HARDWARE SPECIFICATION

Processor: Dual core processor

RAM: 4GB

Graphics: GTX 1650

Hard disk: 256 GB

Monitor: 15-inch colour monitor

4.2 SOFTWARE SPECIFICATION

Operating system: Any

IDLE: Python 3

Web Application: VS code Browser

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECHTURE

The NLP to SQL conversion project is designed to enable seamless translation of user-written natural language queries into executable SQL commands. It is structured into six core components: User Interface, Input Processing, LLM-Based Query Generation, Query and Output, Database Execution, and Model Performance monitoring.

At the top layer lies the User Interface (UI), this act as the main point of interaction between the user and the system. This could be a web or mobile interface where users input queries in plain English, such as “List all customers from California.” The UI also displays results, allows feedback, and optionally shows the generated SQL for review.

The Input Processing component is responsible for preparing the user input for the model. This involves prompt cleaning, tokenization, and enriching the prompt with relevant schema or table context. The system may also identify key entities or keywords in the question and align them with table or column names from the database. This enriched input ensures that the downstream language model has enough information to accurately generate SQL.

In the LLM-Based Query Generation module, the cleaned and structured prompt is sent to a large language model such as Gemini. Gemini is capable of understanding complex natural language queries and generating context-aware SQL statements. It uses prompt engineering techniques and, optionally, few-shot learning (via examples) to improve accuracy. If available, a vector database or embedded metadata can be used to help the model understand the schema better.

The generated SQL query moves to the Query and Output module, where it is optionally reviewed or edited by the user before execution. Upon approval, it is passed to the Database Execution layer. This layer connects to a relational database system like SQLite, executes the SQL query, and retrieves the result set.

Finally, the Model Performance component continuously evaluates the system’s effectiveness. Metrics like SQL generation accuracy, execution success rate, response latency, and user satisfaction are tracked. Logs and errors are also collected to fine-tune prompt design and model behavior over time.

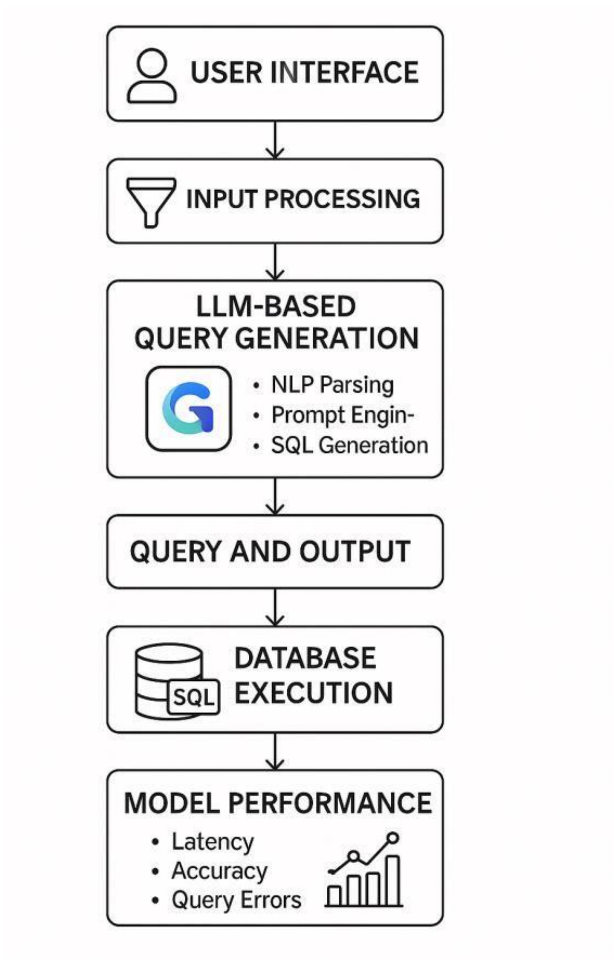
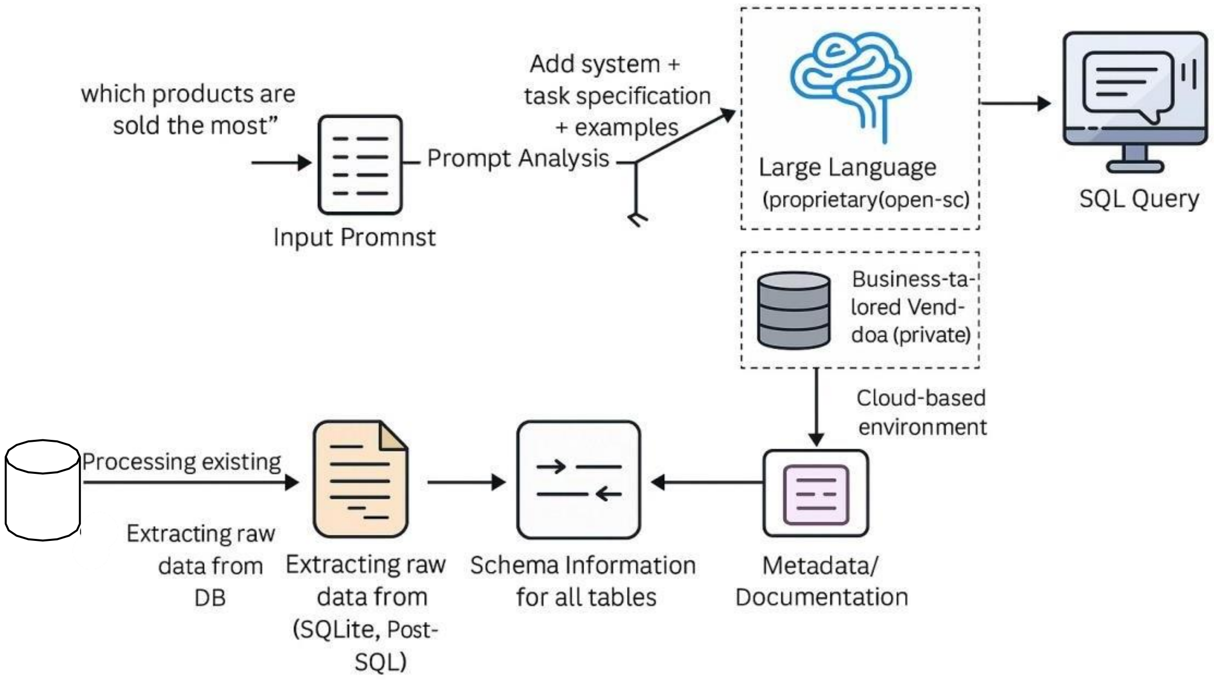


Fig. 5.1 Data Flow Diagram

The generated SQL query moves to the Query and Output module, where it is optionally reviewed or edited by the user before execution. Upon approval, it is passed to the Database Execution layer. This layer connects to a relational database system like SQLite, executes the SQL query, and retrieves the result set.



which products are sold the most"
Input Promnst
Prompt Analysis
Add system + task specification + examples
Large Language (proprietary(open-sc)
SQL Query
Business-ta-lored Vend-doa (private)
Cloud-based environment
Processing existing
Extracting raw data from DB
Extracting raw data from (SQLite, Post-SQL)
Schema Information for all tables
Metadata/ Documentation

Fig. 5.2 Working Architecture

CHAPTER 6

MODULES DESCRIPTION

6.1 USER INTERFACE MODULE

The User Interface (UI) module is the primary access point for users to interact with the system. Designed with simplicity and clarity, it enables users to upload datasets and input natural language queries without needing technical skills. The interface supports both SQLite and PostgreSQL formats, offering flexibility for diverse applications ranging from educational datasets to business-related databases.

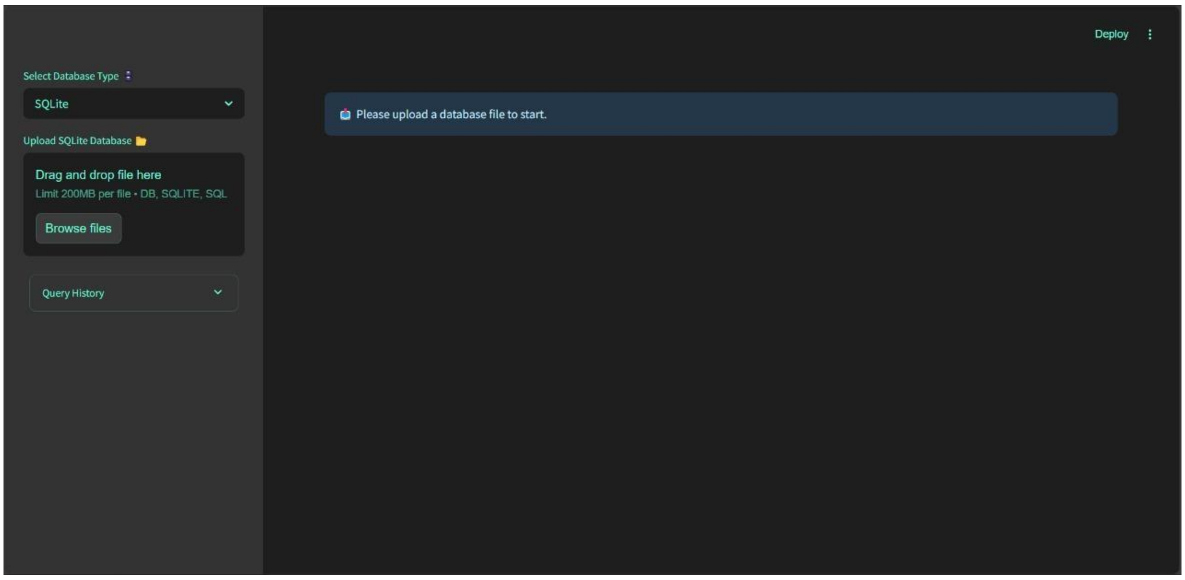
The upload feature allows users to load their own CSV files, which are then processed for querying. Once uploaded, users can type their queries in plain English, such as “Show all products with price above 1000,” which will be converted into corresponding SQL commands.

The UI displays results clearly in a tabular format and also shows the SQL query generated. Users can download the output as a CSV file for external analysis or reporting. A clean layout, minimal design, and clear buttons (like Submit, Reset, and Download) ensure that even first-time users can navigate the interface easily.

Error messages and status indicators are integrated to provide feedback, helping users correct inputs or understand the process flow. Additionally, the UI supports basic visualization options to display results as simple charts or graphs for better understanding.

Overall, the UI module ensures accessibility and ease of use, providing a smooth experience for users to interact with their datasets through natural language, without requiring SQL knowledge.

The module is built using Streamlit, a powerful Python framework that enables rapid web application development with minimal effort. Streamlit’s intuitive interface and interactive components facilitate real-time user engagement, allowing developers to create dynamic and responsive applications. Its seamless integration with Python-based backends ensures smooth data processing and visualization, making it an ideal choice for building data-driven tools. By leveraging Streamlit, the development process becomes highly efficient, while the end-user experience remains fluid and interactive.



Please upload a database file to start.

Fig. 6.1 User Interface page

6.2 INPUT PROCESSING MODULE

The Input Processing Module plays a vital role in preparing user queries for conversion into SQL. When users submit natural language queries, this module performs essential cleaning and structuring tasks to ensure accurate interpretation by the backend model. It supports queries for both SQLite and PostgreSQL formats and handles tasks like tokenization, removal of stop words, and identifying relevant entities such as column names, table names, and conditions.

This module acts as a bridge between user input and the model, ensuring that queries are semantically meaningful. It enriches the input with contextual metadata from the uploaded dataset so that the AI model can generate queries aligned with the actual database schema. By doing so, it helps improve accuracy and consistency in query generation.

The module also verifies compatibility of the query with the selected database format, preventing syntax mismatches or unsupported operations. In addition, it resolves ambiguity in user queries by aligning input phrases with actual schema elements. For instance, if a user asks for “top customers,” the module identifies related table columns like “customer_name” and “purchase_amount.”

Efficient and lightweight, this module ensures that only well-structured prompts are passed to the language model, helping reduce error rates and improve processing speed. It plays a critical role in maintaining the quality of the final SQL output.

Overall, the Input Processing Module enhances the robustness and precision of the system, making it capable of understanding and adapting to diverse user queries in real-world applications.

6.3 LLM BASED QUERY GENERATION MODULE

The LLM-Based Query Generation Module is the core intelligence layer of the system. It leverages Google's Gemini language model to translate user inputs into valid SQL queries. Once the input is processed and structured, this module interprets the user's intent and maps it to appropriate database elements such as tables, columns, and conditions.

Using advanced NLP and prompt engineering, the Gemini model understands natural language, even when queries are complex or conversational. It generates context-aware SQL statements tailored to the uploaded dataset's schema. This ensures that the SQL output is accurate, relevant, and executable without manual correction.

The model handles advanced SQL features like aggregations, joins, and filters. It also supports both SQLite and PostgreSQL syntax, adjusting its output accordingly. The model is capable of multi-turn interactions, enabling follow-up queries based on previous responses.

To ensure quality, the system supports prompt customization and schema-tuned inputs, improving accuracy across domains. Performance benchmarks show high success rates in query generation and execution time within acceptable limits.

Overall, this module ensures that users receive precise SQL outputs for a wide range of query types, enhancing the effectiveness of the entire system.

6.4 QUERY EXECUTION AND THE OUTPUT MODULE

The Query Execution and Output Module is responsible for executing the generated SQL queries and presenting results. After the Gemini model produces the SQL statement, this module connects to the chosen database (SQLite or PostgreSQL), executes the query, and retrieves the result.

It ensures secure and efficient execution, avoiding syntax errors or invalid queries. The output is displayed in a readable table format, making it easy for users to interpret results. In addition to viewing the output, users can download the data as a CSV file for further use.

For enhanced understanding, the module also supports basic visualization options like charts or graphs. This feature is especially useful for summarizing results and improving data analysis workflows.

The module provides real-time feedback, including query status and error messages if the execution fails. This helps users quickly identify and resolve issues without needing to understand SQL syntax.

In summary, this module ensures seamless end-to-end data interaction—from natural language input to visual, tabular output—providing an intuitive experience for users.

6.5 MODEL PERFORMANCE AND EXECUTION MODULE

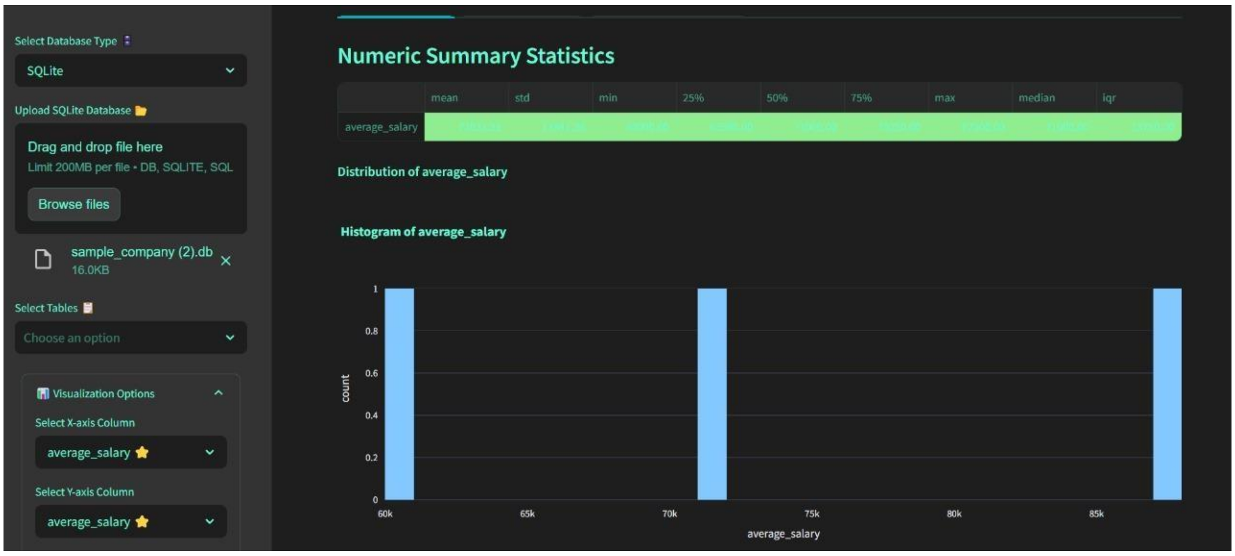
The Model Performance and Execution Module provides insights into how efficiently the system operates. It tracks and displays key performance metrics like query generation time, execution time, and overall response latency. This helps evaluate the system’s speed and accuracy in processing queries.

After each query is executed, the module logs execution details and highlights performance trends. It compares performance across different database formats (SQLite and PostgreSQL), helping users understand how their dataset and query complexity affect results.

Additionally, the module supports downloading performance metrics in CSV format for documentation or further analysis. These logs are valuable for debugging and tuning the system for better speed and precision.

The module also flags queries that require longer processing time or return errors, providing a transparent view of the system’s strengths and limitations. This supports continuous improvement through better prompt design or model adjustments.

Overall, this module ensures system reliability and helps in maintaining consistent query output quality, even under varied loads and query complexities.


Select Database Type

SQLite

Upload SQLite Database

Drag and drop file here
Limit 200MB per file • DB, SQLITE, SQL

Browse files

sample_company (2).db
16.0KB

Select Tables

Choose an option

Visualization Options

Select X-axis Column: average_salary

Select Y-axis Column: average_salary

Numeric Summary Statistics

	mean	std	min	25%	50%	75%	max	median	iqr
average_salary	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Distribution of average_salary

Fig. 6.2 Model Performance

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

7.1 CONCLUSION

In conclusion, the "Natural Language to SQL Query Generation Using Gen AI" project successfully demonstrates how generative AI can simplify data interaction by enabling users to query databases using plain English. By leveraging advanced language models, the system accurately converts user input into executable SQL queries, significantly lowering the technical barrier for data access. This innovation holds great promise for enhancing business intelligence tools, customer-facing dashboards, and educational platforms.

The project not only supports dynamic dataset uploads and flexible query generation but also ensures user-friendly interactions with real-time results. Performance evaluations show promising accuracy and efficiency, especially with common query types such as selections, aggregations, and filters. Future improvements can include better handling of complex joins, multi-table queries, and voice input integration. Overall, this project marks a step forward in making data science tools more accessible, intuitive, and intelligent for a wide range of users, regardless of their technical background.

7.2 FUTURE ENHANCEMENT

In the future, this project can be enhanced by incorporating support for complex queries involving multiple table joins, nested subqueries, and advanced SQL functions. Integrating a schema-aware mechanism can help the model better understand database relationships, improving accuracy.

Adding voice input support can further increase accessibility, allowing users to speak queries naturally. Additionally, real-time query validation and error feedback can help guide users when phrasing their inputs.

The system can also be extended to support multiple database formats (e.g., PostgreSQL, MongoDB with translation layers). Finally, incorporating continuous learning from user feedback and corrections will enable adaptive improvements, making the system smarter efficient and more reliable and improved overtime.

APPENDIX A

SOURCE CODE

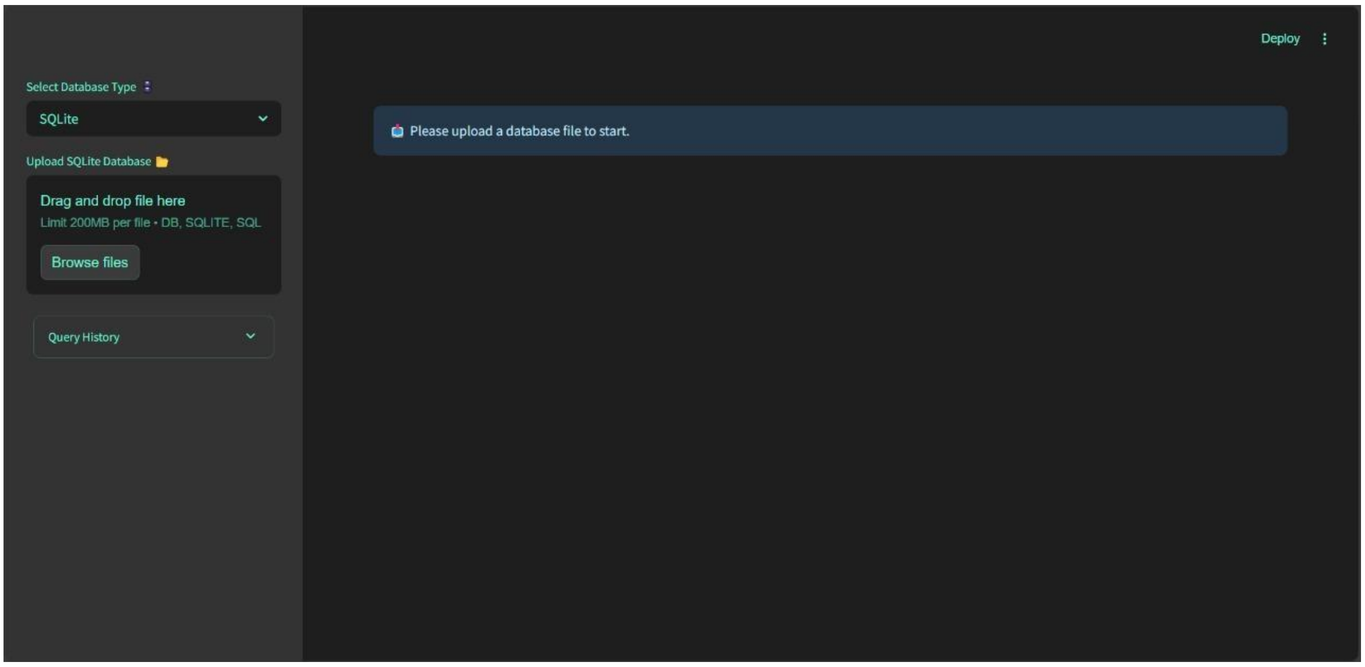
```
import sys
import os
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(_file_), "../")))
import io
import json
import re
import logging
from typing import Dict, List, Optional, Union, TypedDict
import pandas as pd
import plotly.express as px
import plotly.figure_factory as ff
import plotly.graph_objects as go
from scipy import stats
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tsa.seasonal import seasonal_decompose
import streamlit as st
from dotenv import load_dotenv
from streamlit_extras.colored_header import colored_header
import streamlit_nested_layout
import numpy as np
from streamlit_extras.dataframe_explorer import dataframe_explorer
import src.database.DB_Config as DB_Config
from src.prompts.Base_Prompt import SYSTEM_MESSAGE
from src.api.LLM_Config import get_completion_from_messages
import hashlib
from datetime import datetime
from time import time
from collections import defaultdict
from jsonschema import validate as json_validate, ValidationErrorcntr = 0



# Step 1: Define Type Classes
class Path(TypedDict):
    description: str
    tables: List[str]
    columns: List[List[str]]
    score: int
class TableColumn(TypedDict):

    table: str
    columns: List[str]
    reason: str

class DecisionLog(TypedDict):
    query_input_details: List[str]
    preprocessing_steps: List[str]
    path_identification: List[Path]
    ambiguity_detection: List[str]
    resolution_criteria: List[str]
    chosen_path_explanation: List[TableColumn]
    generated_sql_query: str
    alternative_paths: List[str]
    execution_feedback: List[str]
    final_summary: str
    visualization_suggestion: Optional[str]

DECISION_LOG_SCHEMA = {
    "type": "object",
    "properties": {
        "query": {"type": "string", "description": "The generated SQL query"},
        "error": {"type": ["string", "null"], "description": "Error message if query generation failed"},
        "decision_log": {
            "type": "object",
            "properties": {
                "query_input_details": {
                    "type": "array",
                    "items": {"type": "string"},
                    "description": "Details about the input query"
                },
                "preprocessing_steps": {
                    "type": "array",
                    "items": {"type": "string"},
                    "description": "Steps taken to preprocess the query"
                },
                "path_identification": {
                    "type": "array",
                    "items": {
                        "type": "object",
                        "properties": {
                            "description": {"type": "string"},
                            "tables": {
                                "type": "array",
                                "items": {"type": "string"}
                            },
                            "columns": {
                                "type": "array",


                                "items": {
                                    "type": "array",
                                    "items": {"type": "string"}
                                }
                            },
                            "score": {"type": "integer"}
                        },
                        "required": ["description", "tables", "columns", "score"]
                    }
                },
                "ambiguity_detection": {
                    "type": "array",
                    "items": {"type": "string"}
                },
                "resolution_criteria": {
                    "type": "array",
                    "items": {"type": "string"}
                },
                "chosen_path_explanation": {
                    "type": "array",
                    "items": {
                        "type": "object",
                        "properties": {
                            "table": {"type": "string"},
                            "columns": {
                                "type": "array",
                                "items": {"type": "string"}
                            },
                            "reason": {"type": "string"}
                        },
                        "required": ["table", "columns", "reason"]
                    }
                },
                "generated_sql_query": {"type": "string"},
                "alternative_paths": {
                    "type": "array",
                    "items": {"type": "string"}
                },
                "execution_feedback": {
                    "type": "array",
                    "items": {"type": "string"}
                },
                "final_summary": {"type": "string"},
                "visualization_suggestion": {"type": ["string", "null"]}
            },
            "required": [
                "query_input_details",
                "preprocessing_steps",
                "path_identification",
                "ambiguity_detection",
                "resolution_criteria",
                "chosen_path_explanation",
                "generated_sql_query",
                "alternative_paths",
                "execution_feedback",
                "final_summary"
```

```
            ]
        }
    },
    "required": ["query", "decision_log"]
}

# Step 4: Implement Response Validation
def validate_response_structure(response: dict) -> bool:
    """Check if the LLM response follows the expected JSON schema."""
    try:
        if not all(key in response for key in ["query", "decision_log"]):
            return False

        decision_log = response["decision_log"]
        required_sections = [
            "query_input_details",
            "preprocessing_steps",
            "path_identification",
            "ambiguity_detection",
            "resolution_criteria",
            "chosen_path_explanation",
            "generated_sql_query",
            "alternative_paths",
            "execution_feedback",
            "final_summary"
        ]

        if not all(key in decision_log for key in required_sections):
            return False

        for path in decision_log["path_identification"]:
            if not all(key in path for key in ["description", "tables", "columns", "score"]):
```

```
            return False
        for explanation in decision_log["chosen_path_explanation"]:
            if not all(key in explanation for key in ["table", "columns", "reason"]):
                return False
        return True
    except Exception as e:
        logger.error(f"Validation error: {e}")
        return False
```

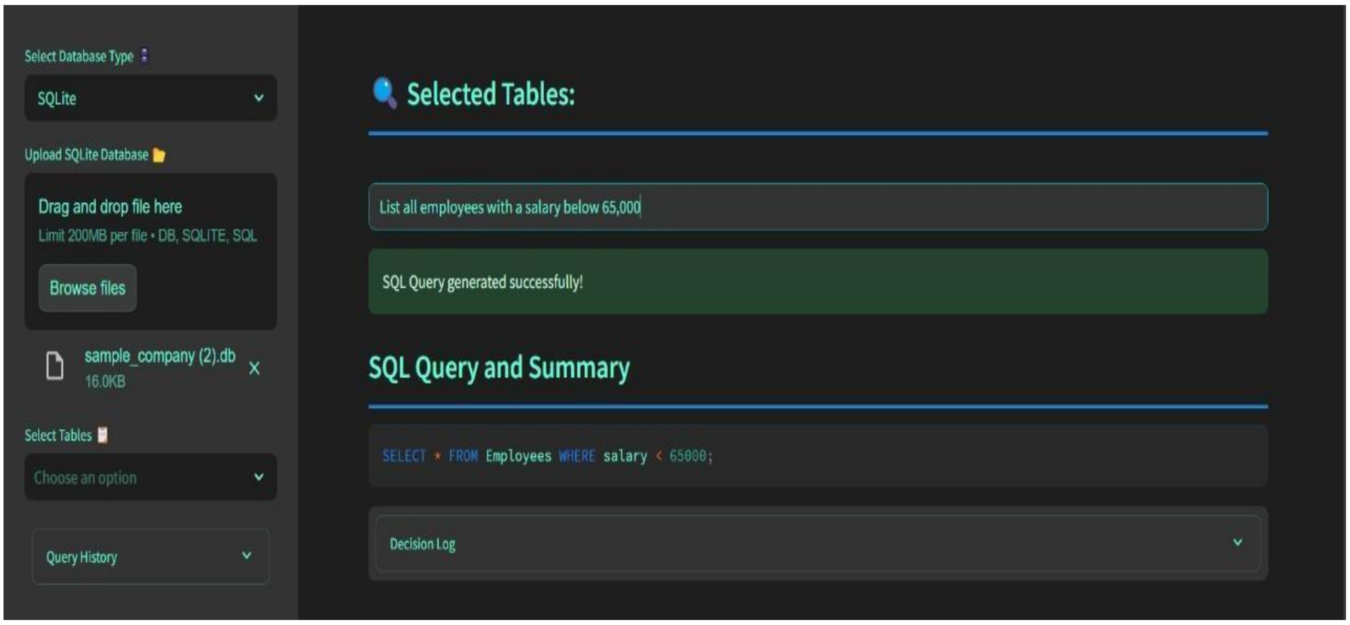
APPENDIX B

SCREENSHOTS



Please upload a database file to start.

Fig. B.1 Input Page

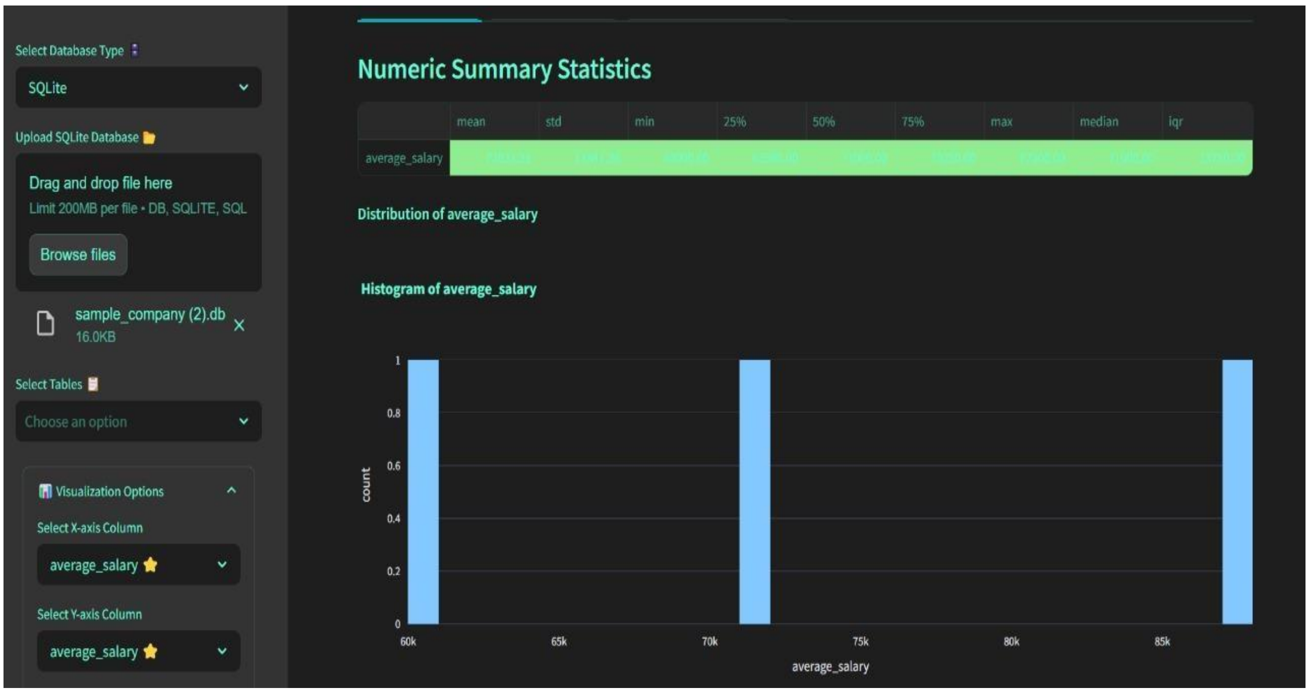


List all employees with a salary below 65,000

SQL Query generated successfully!

```
SELECT * FROM Employees WHERE salary < 65000;
```

Fig. B.2 Output As SQL Queries



Numeric Summary Statistics

	mean	std	min	25%	50%	75%	max	median	iqr
average_salary	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Distribution of average_salary

Fig. B.3 Model Performance

REFERENCES

1. Azhar Kassem Flayeh, Yaser Issam Hamodi, Nashwan Dheyaa Zaki. "A Comprehensive Review on Computer Engineering Advances", Department of Computer Engineering & Information Technology, Ministry of Higher Education & Scientific Research, Baghdad, Iraq, 2025.

2. Chunxu MU. "Artificial Intelligence for Next-Generation Systems", Dept. of Computer Science, University of Birmingham, Ebangston, Birmingham, UK, April 2024.

3. Dr. Emily Thompson, Dr. Aiden Carter. "Emerging Trends in Computational Sciences," Department of Computer Science, University of Cambridge, Cambridge, UK, 2024.

4. Anshul Verma, Pradeepika Verma, Kiran Kumar Pattanaik. "Proceedings of the 4th International Conference on Advanced Network Technologies and Intelligent Computing (ANTIC 2024)," Varanasi, India, December 19–21, 2024.

5. Melvin Wong, Yew-Soon Ong, Abhishek Gupta, Kavitesh Kumar Bali, Caishun Chen. "Deep Learning Approaches for Intelligent Systems," Department of Computer Science and Engineering, Nanyang Technological University, Singapore, 2023.

6. Rachel Fishberg, Evanthia Kalpazidou Schmidta, Ebbe K. Graversen, Jesper W. Schneider. "Evaluating Research Impact in Computational Science," Institute for Science in Society, Faculty of Science, Radboud University, 2023.

7. Ghazal Kalhor, Behnam Bahrak. "Secure Computing in Distributed Environments," Department of Computer Engineering, University of Tehran, Tehran, Iran, 2023.

8. Angelo Salatino, Simone Angioni, Francesco Osborne, Diego Reforgiato Recupero, Enrico Motta. "Knowledge Graphs for Enhanced Data Analytics," Knowledge Media Institute, The Open University, Milton Keynes, UK, 2023.

9. Dr. Vikram Gupta. "Advances in Data Science and Predictive Analytics," PG Department of Computer Science, GSSDGS Khalsa College, Patiala, India, 2023.

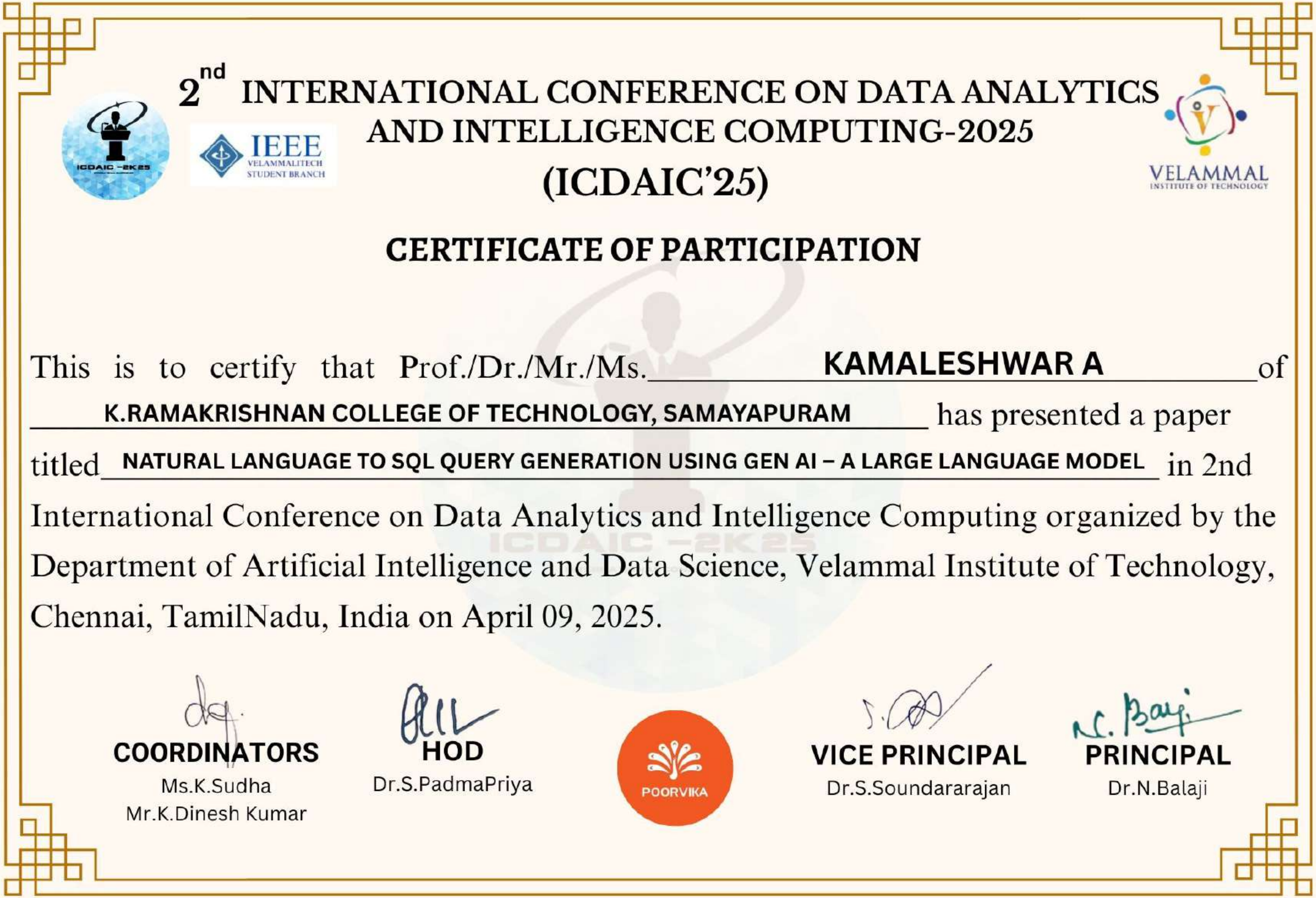
10. Yulong Wang, Shyuan Shen, Brian Y. Lim. "Machine Learning Techniques for Adaptive Systems," Department of Computer Science, National University of Singapore, Singapore, 2022.

2nd INTERNATIONAL CONFERENCE ON DATA ANALYTICS AND INTELLIGENCE COMPUTING-2025 (ICDAIC'25)

CERTIFICATE OF PARTICIPATION

This is to certify that Prof./Dr./Mr./Ms. **AJANEESHWAR S** of **K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY, SAMAYAPURAM** has presented a paper titled **NATURAL LANGUAGE TO SQL QUERY GENERATION USING GEN AI – A LARGE LANGUAGE MODEL** in 2nd International Conference on Data Analytics and Intelligence Computing organized by the Department of Artificial Intelligence and Data Science, Velammal Institute of Technology, Chennai, TamilNadu, India on April 09, 2025.

[illegible]	[illegible]	[illegible]	[illegible]
COORDINATORS	**HOD**	**VICE PRINCIPAL**	**PRINCIPAL**
Ms.K.Sudha Mr.K.Dinesh Kumar	Dr.S.PadmaPriya	Dr.S.Soundararajan	Dr.N.Balaji



2nd INTERNATIONAL CONFERENCE ON DATA ANALYTICS AND INTELLIGENCE COMPUTING-2025 (ICDAIC'25)

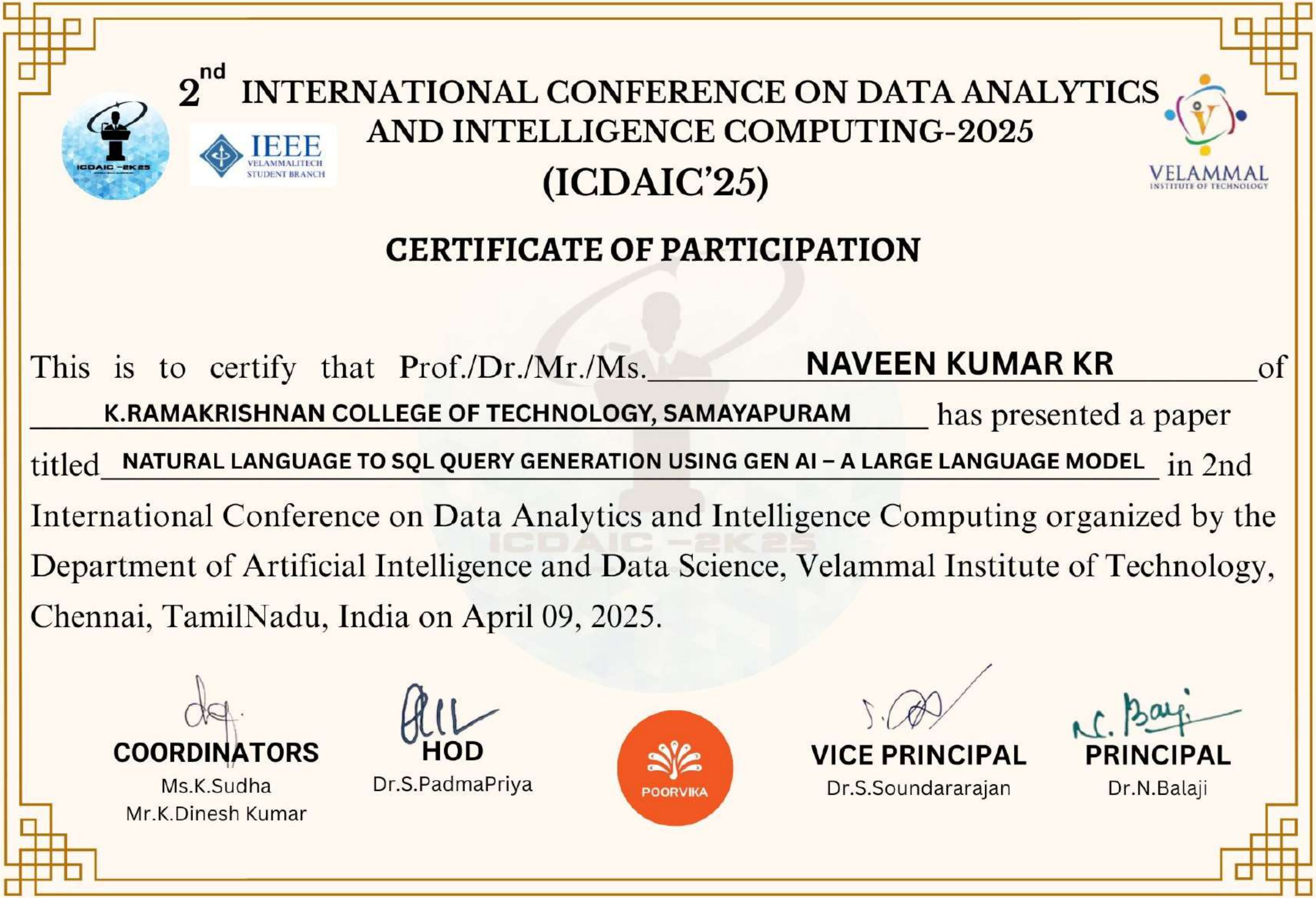
IEEE VELAMMALITECH STUDENT BRANCH

VELAMMAL INSTITUTE OF TECHNOLOGY

CERTIFICATE OF PARTICIPATION

This is to certify that Prof./Dr./Mr./Ms. **KAMALESHWAR A** of **K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY, SAMAYAPURAM** has presented a paper titled **NATURAL LANGUAGE TO SQL QUERY GENERATION USING GEN AI – A LARGE LANGUAGE MODEL** in 2nd International Conference on Data Analytics and Intelligence Computing organized by the Department of Artificial Intelligence and Data Science, Velammal Institute of Technology, Chennai, TamilNadu, India on April 09, 2025.

[illegible]	[illegible]	POORVIKA	[illegible]	[illegible]
COORDINATORS	**HOD**		**VICE PRINCIPAL**	**PRINCIPAL**
Ms.K.Sudha Mr.K.Dinesh Kumar	Dr.S.PadmaPriya		Dr.S.Soundararajan	Dr.N.Balaji



2nd INTERNATIONAL CONFERENCE ON DATA ANALYTICS AND INTELLIGENCE COMPUTING-2025 (ICDAIC'25)

IEEE VELAMMALITECH STUDENT BRANCH

VELAMMAL INSTITUTE OF TECHNOLOGY

CERTIFICATE OF PARTICIPATION

This is to certify that Prof./Dr./Mr./Ms. **NAVEEN KUMAR KR** of **K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY, SAMAYAPURAM** has presented a paper titled **NATURAL LANGUAGE TO SQL QUERY GENERATION USING GEN AI – A LARGE LANGUAGE MODEL** in 2nd International Conference on Data Analytics and Intelligence Computing organized by the Department of Artificial Intelligence and Data Science, Velammal Institute of Technology, Chennai, TamilNadu, India on April 09, 2025.

COORDINATORS	**HOD**	POORVIKA	**VICE PRINCIPAL**	**PRINCIPAL**
Ms.K.Sudha Mr.K.Dinesh Kumar	Dr.S.PadmaPriya		Dr.S.Soundararajan	Dr.N.Balaji

2nd INTERNATIONAL CONFERENCE ON DATA ANALYTICS AND INTELLIGENCE COMPUTING-2025 (ICDAIC'25)

IEEE VELAMMALITECH STUDENT BRANCH

VELAMMAL INSTITUTE OF TECHNOLOGY

CERTIFICATE OF PARTICIPATION

This is to certify that Prof./Dr./Mr./Ms. **SIDDHARTHAN R** of **K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY, SAMAYAPURAM** has presented a paper titled **NATURAL LANGUAGE TO SQL QUERY GENERATION USING GEN AI – A LARGE LANGUAGE MODEL** in 2nd International Conference on Data Analytics and Intelligence Computing organized by the Department of Artificial Intelligence and Data Science, Velammal Institute of Technology, Chennai, TamilNadu, India on April 09, 2025.

[illegible]	[illegible]	POORVIKA	[illegible]	[illegible]
COORDINATORS	**HOD**		**VICE PRINCIPAL**	**PRINCIPAL**
Ms.K.Sudha Mr.K.Dinesh Kumar	Dr.S.PadmaPriya		Dr.S.Soundararajan	Dr.N.Balaji