RSET

RAJAGIRI SCHOOL OF
ENGINEERING & TECHNOLOGY
(AUTONOMOUS)

Project Report On

AIDCA

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

By

Joyal Mozhoor Joji (U2103118)
Muhammed Ashir K Abdul Selim (U2103141)
Goureeshankar Shoyson (U2103098)
Jefrin John (U2103108)

Under the guidance of

Mr. Biju Abraham Narayamparambil

Computer Science and Engineering
Rajagiri School of Engineering & Technology (Autonomous)
(Parent University: APJ Abdul Kalam Technological University)
Rajagiri Valley, Kakkanad, Kochi, 682039
April 2025

CERTIFICATE

This is to certify that the project report entitled ***"AIDCA"*** *is a bonafide record of the work done by* ***Joyal Mozhoor Joji (U2103118)***, ***Muhammed Ashir K Abdul Selim(U2103141)***, ***Goureeshankar Shoyson (U2103098)***,***Jefrin John (U2103108)*** *submitted to the Rajagiri School of Engineering & Technology (RSET) (Autonomous) in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (B. Tech.) in "Computer Science and Engineering" during the academic year 2021-2025.*

Mr. Biju Abraham Narayamparambil
Project Guide
Assistant Professor
Dept. of CSE
RSET

Ms.Anu Maria Joykutty
Project Coordinator
Assistant Professor
Dept. of CSE
RSET

Dr. Preetha K G
Professor & HOD
Dept. of CSE
RSET

ACKNOWLEDGMENT

I wish to express my sincere gratitude towards **Rev. Dr. Jaison Paul Mulerikkal CMI**, Principal of RSET, and **Dr Preetha K G**, Professor, Head of the Department of Computer Science and Engineering for providing me with the opportunity to undertake my seminar, "Explainable AI in fracture detection and prediction".

We are highly indebted to our project coordinators, **Ms.Anu Maria Joykutty**, Assistant Professor, Dept of CSE, for their valuable support

It is indeed my pleasure and a moment of satisfaction for me to express my sincere gratitude to my project guide **Mr. Biju Abraham Narayamparambil**, Assistant Professor, Department of Computer Science and Engineering, for his patience and all the priceless advice and wisdom he has shared with me.

Last but not the least, I would like to express my sincere gratitude towards all other teachers and friends for their continuous support and constructive ideas.

Joyal Mozhoor Joji

Goureeshankar Shoyson

Jefrin John

Muhammed Ashir K Abdul Selim

Abstract

The AI Data Conversational Assistant (AIDCA) is an innovative application that attempts to ease data interactivity in the sense that users may query data using natural language. Data entry previously required having expertise in SQL queries or similar forms of queries, which makes it very frustrating for a non-technical person and slows down decision-making. AIDCA bridges this gap through natural language processing and the use of machine learning techniques to understand user input and convert it into actionable information. Key features of AIDCA include natural questions, chat messages, and instant messaging. While natural language queries allow users to conveniently extract information through typing or simple sentences spoken aloud, chatbots can capture content and manage multiple interactions, resulting in better insight and understanding. Real-time data processing integration allows end users to obtain instant and accurate responses from available data. It has numerous users in areas like business, healthcare, finance, and education. All of this amounts to democratizing data entry, increasing productivity, and timely insights to decision makers without making data input really heavy-duty for every individual.

By designing easy ways to enter very complex questions and a user-friendly interface, AIDCA democratizes data input, increases productivity, and provides timely insights to decision-makers, making data management effective and efficient for everyone.

Contents

List of Abbreviations

AI - Artificial Intelligence

NLP - Natural Language Processing

ML - Machine Learning

SQL - Structured Query Language

AIDCA - AI Data Conversational Assistant

FSUA - Frictionless and Secure User Authentication

TOTP - Time-based One-Time Password

DBMS - Database Management System

GPU - Graphics Processing Unit

List of Figures

List of Tables

Chapter 1

Introduction

A great project is the AI Data Conversation Assistant (AIDCA), which develops the capacity to interact with data for most nontechnical users through a natural language interface for database queries, not depending on technical expertise. It allows the users to interact with databases intuitively instead of depending on technical proficiency, thus benefiting productivity and decision-making.

1.1 Background

In these days of data-centricity, any access to database information should really be interactive enough to inform an intelligent decision at the right moment. However, statistics now reveal that traditional means of database querying require a solid technical knowledge barrier, which is not friendly to non-techs and to those not well versed with languages like SQL. AIDCA cures this whole mess through later technologies such as Natural Language Processing (NLP) and Machine Learning (ML). For instance, with it, anyone from the background of professional sectors like business, healthcare, finance, or education can use and analyze data without being impeded by such technical barriers, democratizing data access and increasing operational efficiency.

Artificial Intelligence: For instance, NLP with machine learning will allow AIDCA to accomplish all of that. People from different fields such as business, healthcare, finance, and education will use and analyze data without any technical hindrances, which will eventually democratize data access and improve operational efficiency.

1.2 Problem Definition

The major objective of AIDCA is simple: to make it possible to query a database, without requiring technical skills, using an accessible natural language interface. Such processes

will enable different users to interact efficiently with a data set regardless of their technical abilities.

1.3 Scope and Motivation

Interacting with databases will now include plain spoken language and will within a framework of NLP and ML based technologies. The system has been configured to inundate structured and unstructured data types, with accurate and context-specific responses, within multiple domains such as healthcare, education, finance, among others.

The motivating factor for AIDCA's conception is to go beyond the limits usually experienced in using outdated interfaces of databases that remain very much under the control of technical practitioners. Such characteristics tend to provide intense, all-encompassing platform functionality concerning data retrieval and analytical capability to increase productivity and informed decisions across a fast-paced world of data.

1.4 Objectives

- Development of a natural language processing system for database querying.
- Development of a conversational interface for multi-turn interaction.
- Enable real-time processing of data for quick responses.
- Enable interaction with both structured and unstructured data types.
- Create an interface that is accessible to less tech-savvy users.
- Improve end-user experience with easily navigable, intuitive design.

1.5 Challenges

Some problems to be encountered by this project include how to develop an accurate understanding of complex user queries, whether to maintain the entire context of a conversation during multi-turn interaction or not, and how to process data in a real time manner for speedy responses that are also reliable.

1.6 Assumptions

- Users will have elementary familiarity in typing or using voice commands.
- The system will work with preset database configurations.
- User queries are likely to be made in English primarily.

1.7 Societal / Industrial Relevance

AIDCA has huge applicability in societal and industrial contexts. In industries, the system improves the decision-making process by providing fast, correct and reliable access to data, especially in sectors like healthcare and finance. Societally, it empowers a category of people, such as teachers and researchers, by making data available easily to anyone technical or not, hence nurturing a more data-oriented and informed society.

1.8 Organization of the Report

This report is organized as follows: Chapter 1 will introduce a project comprising project objective, problem statement, and need of making database interaction more accessible to non-technical users. During this regard, Chapter 2 will history and related literature review existing work in the field of NLP, ML, and database query systems. The chapter will talk about goals and methodologies of the project. Chapter 3 will be about designing the natural language query system and the conversational bot. Chapter Four is about the system implementation and testing that has been done on the system, which can fetch data very accurately and efficiently. Chapter 5 is all about results and analysis, testing of usability and performance of the system under different domains. The last chapter, which is Chapter 6, consists of the report summary and its major findings, as well as the implications industry and society, with future directions in work.

1.9 Conclusion

AI Data Conversation Assistant (AIDCA) has conceptualized the interaction of people with databases, making it so simple and easily available that it could replace traditional methods. Computer scientists have come up with such advancements in Natural

Language Processing (NLP) and Machine Learning (ML) turning the system into a data source allowing users to interact with it using the most natural language ever rather than learning SQL or any other technical term. It has real-time processing capability, and multi-step conversations make it user-friendly and appropriate for application into healthcare, education, finance, etc. It makes decisions faster and better for users while increasing their productivity by lowering barriers to data. In the future, adding support for more languages and further improving the ability of the system to comprehend context would entirely take AIDCA into a new, even farther-reaching power.

Chapter 2

Literature Survey

2.1 A Neural Database for Answering Aggregate Queries on Incomplete Relational Data

[1]

Usually, filling the missing parts has been done reconstruction or imputation in databases, which is highly error-prone and resource-consuming. NeuroComplete brings a new perspective bypassing overall this reconstruction. This instead uses a neural network to directly estimate answers for aggregate queries. Then, using query embeddings and supervised training, it observes patterns in the available data to accomplish the valuable answers from a lot of missing data. This ground-breaking method thus saves from the need for synthetic data generation but captures main relationships in the observable dataset.

2.1.1 Training Set Creation

To enable effective training over incomplete datasets, NeuroComplete constructs a relevant set of *restricted queries*, which can indeed be evaluated accurately by the available data. In particular, these restricted queries form a subset of possible queries chosen such that they yield the same results when evaluated on both the incomplete observed dataset ($\bar{D}$) and a complete hypothetical dataset (D). This dictates matching training labels to true answers without allowing missing-nonsensical data to distort their responses to a query.

Let $f(q)$ be the query function returning the true answer from the complete dataset D, while, on the contrary, $\bar{f}(q)$ is the observed function based on the incomplete dataset $\bar{D}$. The model that needs to be built is $\hat{f}(\cdot;\theta)$ and makes effort into approximation minimization of errors in getting the output as what is expected to be computed without the limitation of using only the observed dataset $\bar{D}$.

The limited query training design has the following steps:

- **Step 1**: Randomly generate attribute queries corresponding to the observed ontological dataset.
- **Step 2**: Append an additional conjunctive clause for each query which further constrains the record set available in $\bar{D}$.
- **Step 3**: Query embedding, as presented in Section 3.1.2, converts each query into a vector using which generalization can then be facilitated by the model across the various query structures.

Algorithm 2. Training Query Generation

Input: The observed database $\bar{D}$ and training size s
Output: A query set, Q
procedure GenerateQueries$\bar{D}$, s
  $Q \leftarrow \emptyset$
  $I \leftarrow$ set of ids of rows in $\bar{T}_i$
  for $i \leftarrow 1$ **to** s **do**
  $A \leftarrow$ a randomly selected attribute from T_i
  $v \leftarrow$ a value in range of A
  $op \leftarrow$ one of $\leq$, $\geq$ or $=$
  $q \leftarrow$ "SELECT AGG(M) FROM T_i WHERE A op v"
  q += "AND T_i.id IN I"
  $Q.append(q)$
return Q

Figure 2.1: Training Query Generation Algorithm for generating restricted queries in NeuroComplete.

2.1.2 Query Embedding

The NeuroComplete embeds all queries into a corresponding neural query embedding noted as ρ given for incomplete-data challenges. It allows the model to achieve answers based on the patterns in data instead of regurgitating memorized values. It generates embeddings by a concept called row relevance which measures how much each row in the ground-truth tables contributes to the answer of a query.

- **Row Relevance Calculation**: NeuroComplete just defines an application for relevance score taking every row in the fully observed tables. In this instance, the relevance score of a row i can be computed as:

$$\alpha_i = \text{COUNT}(\sigma_{O.id=i}(T_q \bowtie O))$$

- **Row Aggregation**: The row relevance scores are then aggregated into the query embedding z: in cases of queries involving any aggregation functions like AVG (average), the individual row values are shifted as per their relevance into an average, and hence an embedding is generated that proportionately reflects the distribution of the rows that satisfied the query.

Algorithm 3. Complete Query Embedding Algorithm

Input: A query q on observed database $\bar{D}$
Output: Query embedding
procedure ρq, $\bar{D}$
  for all tables T_j **in** $\{T_1, \ldots, T_k\} \setminus \{T_i\}$ **do**
    for all KRR rows with id=x **in** T_j **do**
      $\alpha_x \leftarrow \texttt{COUNT}(\sigma_{T_j.\text{id}=x}(P_q(\bar{D}) \ltimes T_j))$
    if Any URR row exists in T_j **then**
      $\hat{g}(.;\theta) \leftarrow$ Trained row relevance model
      for all URR rows with id=x **in** T_j **do**
        $\alpha_x \leftarrow \hat{g}(\sigma_{T_j.\text{id}=x}(T_j);\theta)$
  $z_j \leftarrow \sum_x \alpha_x \sigma_{T_j.\text{id}=x}(T_j)$ ▷ Column-wise sum
  if AGG is not count-sensitive **then**
    $z_j \leftarrow \frac{z_j}{\sum_x \alpha_x}$
  else
    if q is a restricted query **then**
      $z_j \leftarrow \frac{z_j}{\bar{n}}$
    else
      $z_j \leftarrow \frac{z_j}{n}$
  return $[z_1 z_2 \ldots z_k]$

Figure 2.2: Complete Query Embedding Algorithm showing how query embeddings are generated for incomplete data.

2.1.3 Query Answering with NeuroComplete

The trained model now uses query embedding to provide estimated results for the test queries. The following steps are taken during testing:

- Embed each new query to generate its row relevance representation for the observed data.
- Supply the query embedding to the trained model for answer prediction. For **count-sensitive queries**, the result must consider the missing records according to the size of the observed data.

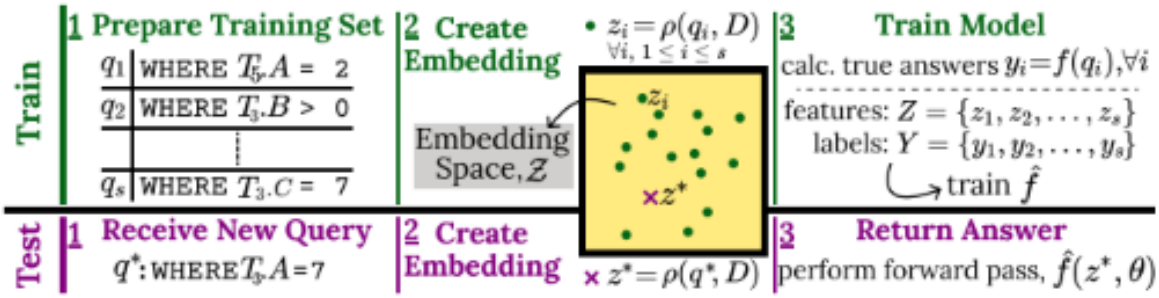


Figure 2.3: The NeuroComplete framework, demonstrating the process of training, embedding, and query answering.

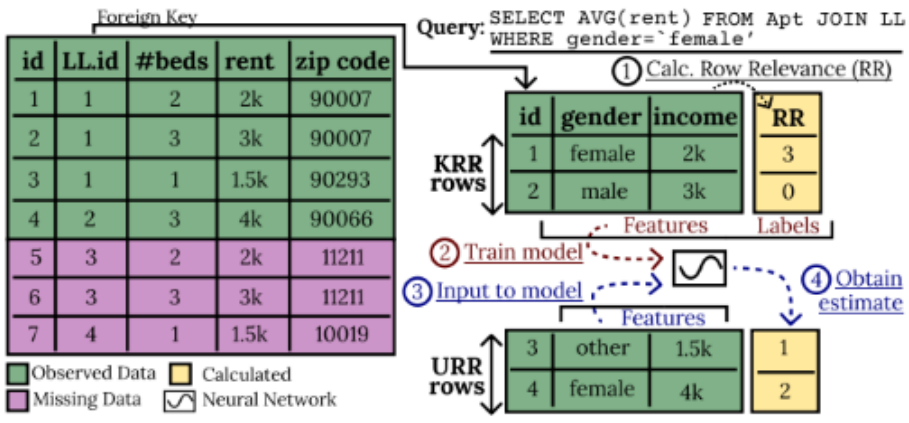


Fig. 4. Row Relevance Calculation.

Figure 2.4: Example of row relevance calculation for a query in NeuroComplete.

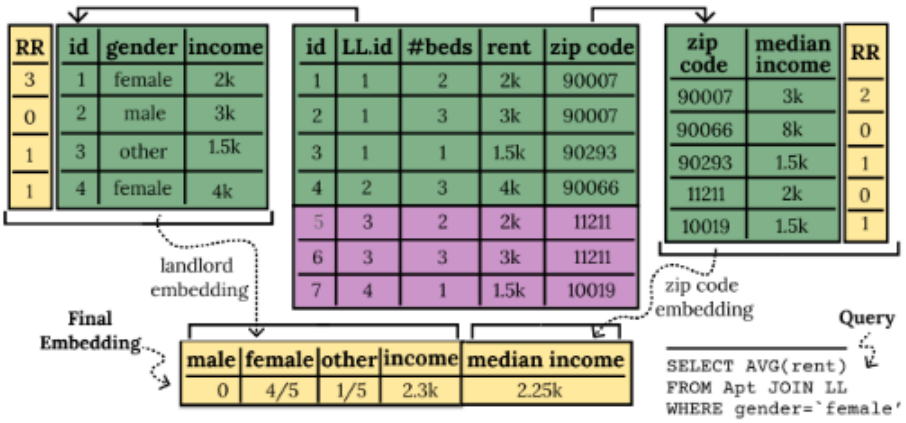


Fig. 5. Multi table query embedding.

Figure 2.5: Illustration of the multi-table query embedding process, showing row relevance across multiple tables.

2.2 SV2-SQL Model

[2] It can be stated that SV2-SQL is an NLP system built to convert the natural language queries into SQL commands. This system uses the BERT language model, and the entire module is fine-tuned to take care of all the sub-tasks that need to be performed for any SQL query generation. The model consists of three important components:

- **Select-Where Slot Filling Model (SWSF):** This component focuses on identifying key SQL elements, such as the columns to be selected, operators, and aggregation functions (e.g., SUM, AVG), which are crucial for structuring the query.
- **Value Extraction Model (VE):** The VE model is responsible for extracting specific values from the natural language query, particularly for the conditions in the "Where" clause.
- **Verification Model (V):** This model performs a semantic check to ensure that only the relevant elements are included in the final SQL query, thereby enhancing the accuracy by removing any irrelevant conditions.

These three models go hand in hand to create a smooth interface of transforming natural word queries into SQL syntax. [2]

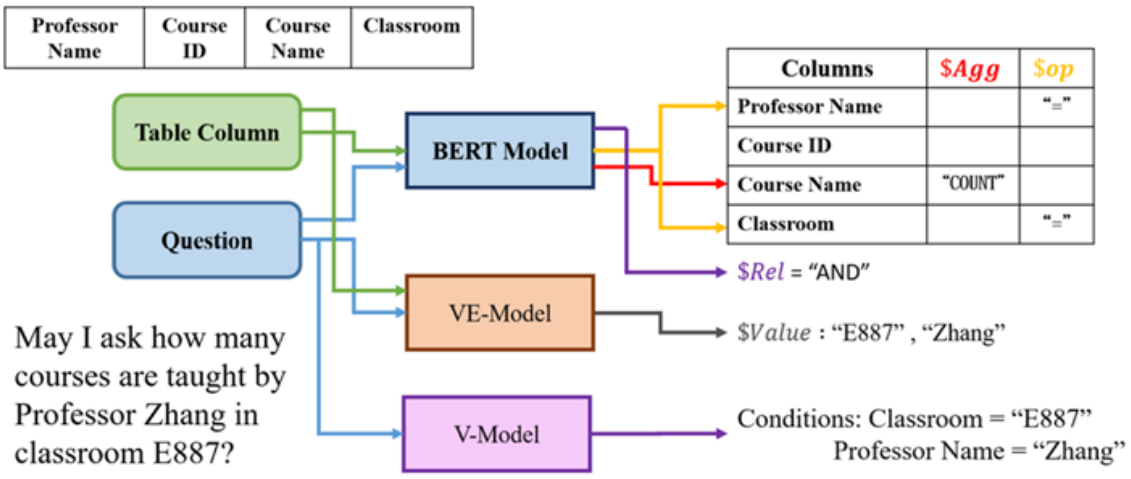


Figure 2.6: The SV2SQL framework.

2.2.1 BERT in SV2-SQL

It takes the advantage of initializing BERT, which is a well-known language model based on transformer architecture mainly for its bidirection attention mechanism, to improve the capabilities of SV2-SQL in converting languages into SQL. The application of pre-trained contextual embedding of BERT allows the SV2-SQL to create deep semantic representations capable of understanding the finer details in natural language queries, thus adding the advantage needed while generating SQL as it facilitates intent understanding together with structural dependences in questions enabling the generation of exact and populating as well as constructively correct SQL queries.

SV2-SQL, as for usa 'in' BERT, brings the different sub-models as matched parts of the query in forming the whole natural language term with the SQL components so that a long natural language expressions may correctly matched, which also benefits for performance on difficult datasets and much better handles people languages leading to good translation even with such complex or vague inputs.

2.2.1.1 BERT's Attention Mechanism

Multi-head attention is part of BERT's attention mechanism, and each head specializes in learning different relationships between tokens in an input sequence. The resulting attention though calculated using query, key, and value vectors allows both long-range

and short-range dependencies in comprehension.

Let us consider a token in the input sequence, x_i. The attention mechanism defines the following.

$$q_i = W^Q x_i, \quad k_i = W^K x_i, \quad v_i = W^V x_i$$

where W^Q, W^K and W^V are weight matrices for generation of query, key and value vectors. The attention score s_{ij} between tokens x_i and x_j can thus be defined:

$$s_{ij} = \frac{q_i k_j^T}{\sqrt{d_k}}$$

Then, softmax function normalizes these scores to produce attention weights:

$$a_{ij} = \text{softmax}\left(\frac{q_i k_j^T}{\sqrt{d_k}}\right)$$

Finally Token representation H_i is calculated as a weighted sum of the value vectors:

$$H_i = \sum_j a_{ij} v_j$$

With this attention mechanism, BERT focuses on the most relevant contextual elements of the query used in SQL generation.

2.2.2 Model for Select-Where Slot Filling (SWSF

) The key SQL elements such as the columns in the "Select" and "Where" clauses, aggregation functions, and operators are identified through classification tasks in SWSF. It classifies SQL components accurately using BERT's embeddings. The process can be broken down as:

2.2.2.1 Input Preprocessing

The input sequence of SWSF consists of user query and table column names:

[CLS] Query [SEP] Column Names [SEP]

These tokens can go through BERT to develop embeddings, where each token has its unique embedding, segment, and mask representation.

2.2.2.2 Output Tasks in SWSF

Output SWSF functions can refer to a number of tasks, each of which calls for SQL differently:

- **Aggregation Task**: It selects a column that is required to be aggregated or not through aggregation functions, for example, *COUNT, SUM, AVG*.
- **Operation Task**: It predicts about the relational operators (for example: $=, >, <, \neq$) used in the "Where" clause.
- **Relation Task**: It finds out the logical connectors (e.g. AND, OR) in between multiple conditions in the "Where" clause.

All such tasks will yield the probability distribution of possible labels over the embeddings of BERT. Hence, the probability of an aggregation function applied to a particular column is as given:

$$P(\text{Agg}) = \text{softmax}(W_{\text{Agg}} \cdot H_{f_i})$$

where W_{Agg} is an acquired weight matrix whereas H_{f_i} means the embedding for the column f_i. Such equations would similarly apply to demand and relation tasks.

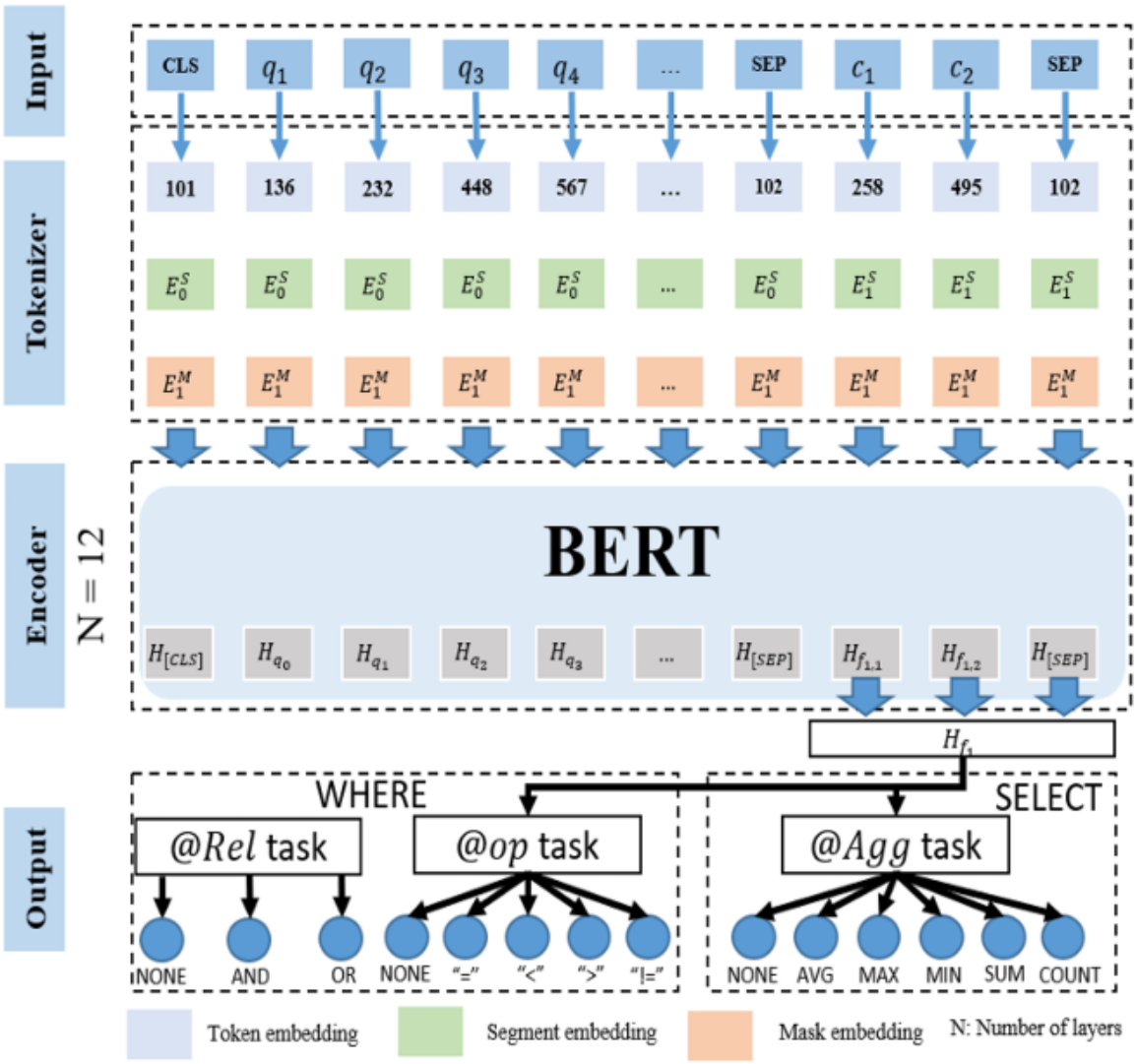


Figure 2.7: Structure of the SWSF model.

2.2.3 The Value Extraction Model (VE)

VE is formulated as an answer dataset to the "where" clause: "using the query as context, the answer is the value span (output)." This model takes input as:

[CLS] Query [SEP] Column Name [SEP]

It finally obtains the answer span from the start-end probabilities incorporated into each token built by BERT.

2.2.3.1 Probability Calculations for Value Extraction

The distributions for the start position will be defined as:

$$P(Start) = softmax(W_{start}.H_i)$$

for the end position, the distribution is:

$$P(End) = softmax(W_{end}.H_i)$$

where weight matrices W_{start} and W_{end} are learned matrices, and H_i is the token embedding at location i. Predicted value will be the span with the highest combined start and end probabilities.

2.2.4 Verification Model (V)

SV2-SQL creates an important model of verification that broadens the domain of ensuring that all relevant conditions get embedded in the final query sql. To this end, the model probes into the probability for each condition on matching user intent. Each condition defines a column name operator and value and is taken from the Value Extraction Model (VE). User query processing along with each condition goes as follows in with the input form:

[CLS] Query [SEP] Condition [SEP]

where "Condition" contains the column, operator, and value.

To compute the semantic compatibility, the V model compares the query and condition to get a similarity score. This is done by passing the [CLS] token embedding through a sigmoid-activated output layer:

$$P(\text{Similarity}) = \text{sigmoid}(W_{\text{sim}} \cdot H_{[\text{CLS}]})$$

wherein W_{sim} is a learned weight parameter and $H_{[\text{CLS}]}$ is the embedding that summarizes the input sequence. Conditions having similarity score greater than a specific threshold are kept, thereby ensuring that only contextually correct conditions are factored into the SQL query, thereby improving the model's precision while reducing false positives.

2.2.5 Inference Algorithm

This inference algorithm in SV2-SQL employs the outputs of the Select-Where Slot Filling (SWSF), Value Extraction (VE), and Verification (V) models to generate an SQL query reflecting the user's intent. The step-by-step inference is as follows:

[1]

Input: User query Q, database columns $\{f_1, f_2, \ldots, f_n\}$

Output: Generated SQL query S

Step 1: Preprocess the input: Concatenate query and column names in the format: [CLS] Query [SEP] Co.

Step 2: Apply Select-Where Slot Filling (SWSF) model to identify SQL slots.

Step 3: Apply Value Extraction (VE) model to extract values for the "Where" clause.

Step 4: Apply Verification (V) model to calculate semantic similarity and filter conditions.

Step 5: Construct SQL query from verified slots and conditions.

Step 6: Execute SQL query on the database to obtain the result.

Explanation of Each Step:

1. **Preprocessing:** brings together end user queries and column names to serve as inputs for BERT-based models.

2. **SWSF Model:** Among other major components, what to select and what aggregation function to apply, represents a frame structure of an SQL query's output.

3. **VE Model:** Retrieves real values of conditions under "Where" and reformulates the whole query according to the specific need of the user.

4. **V Model:** It tests all the necessary conditions against the query and filters out the false positives through the true ones.

5. **Construct SQL Query:**Stack all validated fragments into a single coherent SQL command.

6. **Execute SQL Query:** This was by executing the SQL to process the output result to the data you wanted.

Thus, it could be possible to claim that the generated SQL query captures the meaning and need of the user, while it composes nugatory conditions.

2.3 Enhancing Database Performance through SQL Optimization, Parallel Processing, and GPU Integration

[3]

This is the age when data is experienced to grow in a more expanding fashion than ever before. Hence, there is now an even greater demand for efficient and scalable DBMSs. This section discusses the innovations in SQL optimization, parallel processing, and GPU integration as such transformative techniques for addressing one or all of the following core challenges: query performance, computation bottleneck, and rapid data analytics.

2.3.1 SQL Optimization

Optimization of SQL queries deals with how to minimize consumption of resources, especially memory, to achieve responsiveness in applications. Some major optimization techniques include:

- **Selective Column Fetching:** Instead of the use of SELECT *, obtain only needful columns in order to avoid unnecessary data transfer.
- **Precise Filtering:**Indices should be used in the WHERE clause so that expensive computation will be avoided and queries will be efficient.
- **Efficient Pagination:** Instead of overriding memory space with large-query outputs, just put LIMIT and OFFSET to work for page-wise retrieval of data.
- **Indexing:** Index the most commonly queried columns to have rapid data retrieval, mostly from large databases.
- **Optimizing Joins:** Imply that for all the data types joining should be logical because that will enable accessing them in an easy way. This reduces the burden of intensive computation put in use.

That way, query execution appears to be faster and latency reduced, and improves the scalability of the system.

2.3.2 Parallel Processing

The parallel execution of DB tasks or execution in a integrated environment is possible for a typical high-volume database, thereby drastically reducing processing time. Approaches in parallel processing are:

- **Data Partitioning:** Partitioning the data into sub-parts such that every part would be processed independently.
- **Parallel Query Execution:** Cutting down larger query into smaller units and executing those in parallel.
- **Distributed Computing:** Multiple servers are employed, while performing the tasks in parallel. This will provide higher degrees of fault tolerance and scalability.
- **In-memory Computing:** Any computing done in memory will contribute to the speed of query execution.
- **Horizontal Scaling:** By adding nodes into the already existing cluster in a distributed system, the performance will be scalable as the amount of data increases. System would manage a lot more data efficiently with being scalable and fault tolerant.

In fact, these techniques render the DBMS capable of managing huge amounts of data in the most efficient way possible. Additionally, they allow it to scale and offer fault tolerance.

2.3.3 GPU Integration

t's a way of changing the optimization of performance by introducing graphics processing units in place of computational vehicles to bring about the same in a distributed database management system. They are truly savvy for modeling applications in an area exceptionally rich in data for data manipulation during query and aggregation, culminating in

machine learning model training, due to the structure they provide for parallel processing. Thus, benefits that are to be derived from incorporating all such features are:

- **Faster Computations:** This means that all the multiple calculations are processed at a time through the GPU, which reduces the time to respond to the query.
- **Energy Efficiency:** GPUs can give far more computing performance for much less energy than traditional CPUs.
- **Offloading Computational Tasks:** Heavy computations offloaded to the GPUs can improve the throughput of the DBMS.

Nonetheless, even this application comes with all those problems, such as data transfer latency and more complicated ways of developing applications.

2.4 Formation of SQL from Natural Language Query using NLP

[4]

Transforming natural language queries into SQL statements has made accessing databases easy for non-technical users. It is based on Natural Language Processing (NLP) techniques that assists a layman into retrieving information from complicated databases without ever having to learn SQL or database schema. This proposed methodology works on converting structured English queries into SQL commands for a railway reservation database.

2.4.1 Overview of the Proposed System

The system processes user input in a natural language through various NLP modules and reaches a mapping phase to produce SQL queries. The principal steps involve:

- **Tokenization:** Cutting up input query into smaller but meaningful units or tokens.
- **Lemmatization:** Root form extraction of tokens.
- **Syntactic Analysis:** Perform Parts of Speech (POS) tagging to know the grammatical structure.
- **Semantic Analysis:** Parsing of labelled data for extracting keywords such as source-destination date-fare.

- **Attribute Mapping:** On identified attributes, SQL query components are mapped as SELECT, FROM, and WHERE.

This would mean that accurate extraction of data and formation of the query would help a non-technical user interact with the database.

2.4.2 Core Algorithm for Query Formation

The algorithm for SQL generation can be categorized into the following components:

- **Tokenization:** Breaking an input query into smaller meaningful pieces.
- **Lemmatization:** Changing tokens from their various forms into the base or root forms.
- **POS Tagging:** Attaching parts of speech using the Stanford POS Tagger to tokens for syntactic analysis.
- **Data Parsing:** Uses a regular expression parser, mainly for source, destination, date, and fare.
- **SQL Mapping:** Mapped the derived attributes into various SQL query components like SELECT, FROM, WHERE.

2.4.3 Implementation and Results

Moreover, the system was evaluated with 2880 structured queries on train information and found to give a performance of 98.89. Example queries supported include:

- Find the trains that are available between two locations for a specific date.
- Find out the fare on travel for a class. Its above score demonstrates that it is potential in converting pure natural language queries into very formal SQL commands.

The high accuracy of the system has shown that it can really convert any natural language query into a corresponding SQL command.

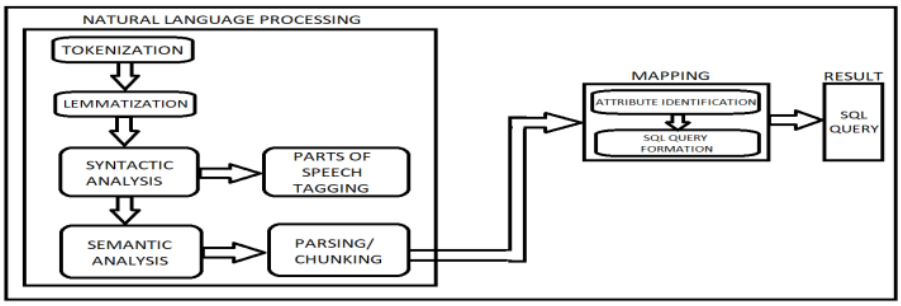


Figure 2.8: Overview of the Query Formation Process.

2.4.4 Future Enhancements

Some of the improvements that will take place in the future include:

- Enhancement of the ability to deal with multiple tables in a single query.
- Provision for transforming audio files into text in order to allow voice queries.
- Introducing a multilingual capability in order to broaden the service accessibility spectrum.

This is indeed an example of the aptitude of NLP so as to make database systems accessible and user-friendly; non-technical users should be able to deal with complex databases with ease.

2.5 Frictionless and Secure User Authentication for Web Applications

[5]

Higher-end web services have problems implementing secure and effective authentication systems. The proposed Frictionless and Secure User Authentication (FSUA) mechanism is balanced in enough security provisions along with reinforcing user convenience using modern methods such as behavioral profiling and dynamic trust evaluations.

2.5.1 System Methodology

The FSUA mechanism is all about implementing certain methodologies to ensure security and easiness, which include:

1. **Behavioral Profiling and Analytics Engine**

- Collects various user-specific data points such as:
  - Timings of login session.
  - Locations.
  - Device and browser usage, and
  - Accessed web services.
- It uses these data points to create dynamic user profiles with analysis on login patterns.
- Predict the risk level of an attempted login based on observed behaviors.

2. **Authentication Gateway**

- Multi-authentication modes based on user profile which dynamically selected:
  - *Password:* User credential verification for authentication.
  - *One Time Password:* Based on time defined one time password (TOTP) user authentication.
  - *Digital certificate authentication:* Relies on certificates issued by trusted authorities to validate the user's identity.

3. **Dynamic Trust Evaluation Procedure**

- Each logon attempt is assigned with a trust score based on the historical logon data of the user with some behavioral patterns.
- Trust Score formula:

$$\text{Trust Score} = f(C \mid Upa) \tag{2.1}$$

  where:
  - C: Contextual data associated with the current login attempt.
  - Upa: User profile attributes.
- The decision would then be taken on whether to allow immediate access or ask for additional authentication based on trust score.

2.5.2 Key Algorithms and Techniques

1. **User Profiling and Scoring**
    - Create a historical archive to track past log-in activities of users.
    - Scoring users based on contextual metrics-location and device usage-and creating prediction systems for authentication reliability.
    - Selects the optimum authentication method according to the User Score.
2. **Pattern Analysis and Anomaly Detection**
    - Monitors and compares events with recorded patterns for anomaly detection.
    - It raises flags for the anomalies, and dynamically the system alters its authentication mechanism to mitigate possible threats.
3. **Authentication Method Choice**
    - Based on an analysis of the previous behaviour, the system accurately decides what would be the most advantageous authentication method.
    - Therefore, it can give an easy journey to all the authentic users without leaving grounds for rough security measures.
4. **Trust Score Formula**
    - It brings together efficiently the prior login data for raising the accuracy of prediction.
    - In reality, the trust score is a real-valued number falling in an interval that ranges from 0 and 1, where higher numbers demonstrate a trusting belief in the legitimacy of the user.

2.5.3 Results and Observations

- The aforementioned FSUA system brought to betterment in efficiency regarding authentication:
    - It has decreased 10 % delay..

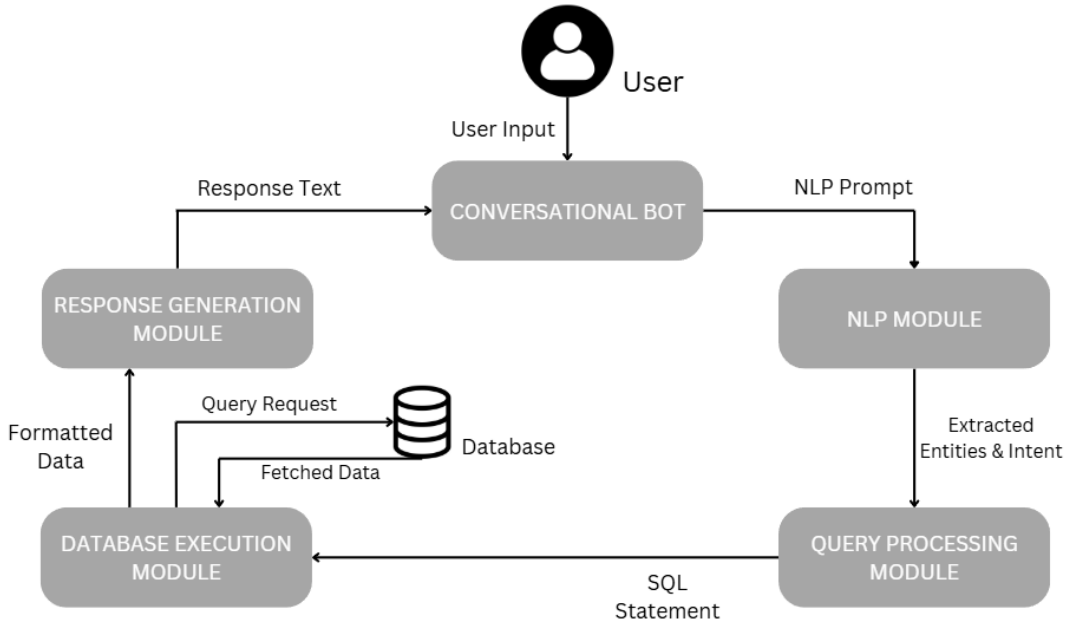
  - Compared with conventional methods, the response time has been improved by 7 % from the measured system speed.
  - 22- 11 % less memory usage.

- The system has robust security mechanisms and successfully prevents unauthorized access by initiating dynamic trust evaluation and automated processes of authentication.

Chapter 3

System Design

3.1 System Architecture



(a) System Architecture diagram

3.2 Component Design

3.2.1 NLP Module

The NLP engine takes natural language input and processes it to extract meaningful entities and intent.

3.2.1.1 Text Processing Module

Objective: Clean and prepare the user's input for the system to understand.

Key Features:

- Removes extra spaces and unnecessary symbols.
- Breaks the input into smaller parts (tokens).
- Converts text into a format the model can read.

Input: A raw query like: *"Show me employees hired in 2021."*

Output: A cleaned version like: `["show", "employees", "hired", "2021"]`

3.2.1.2 Entity Recognition Module

Objective: Find important parts of the query, like table names, column names, or values.

Key Features:

- Identifies words that match database tables or columns.
- Uses the database structure to find matches.
- Tags these parts for easy SQL conversion.

Input: Processed query like: `["employees", "hired", "2021"]`

Output: A list like:

```
[
  {"entity": "employees", "type": "table"},
  {"entity": "hired", "type": "column"},
  {"entity": "2021", "type": "value"}
]
```

3.2.1.3 Intent Recognition Module

Objective: Understand the user's goal, like retrieving or updating data.

Key Features:

- Identifies whether the query is about selecting, inserting, or deleting data.
- Matches the intent to a SQL template.
- Handles different ways users might ask the same thing.

Input: Processed query and recognized entities.

Output: The identified goal, like:

```
{
  "intent": "SELECT",
  "sql_template": "SELECT {columns} FROM {table} WHERE {conditions}"
}
```

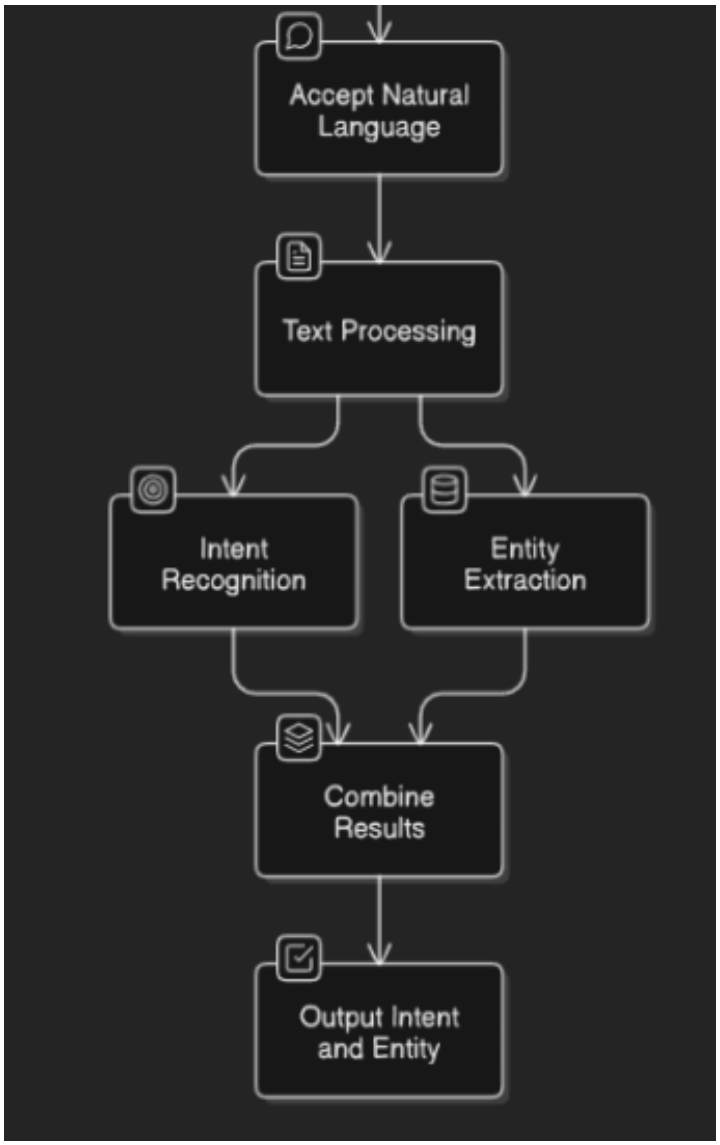


Figure 3.1: High-level system architecture diagram illustrating major components

3.2.2 Query Processing Module

The Query Processing module is responsible for converting the natural language query into a structured SQL query and optimizing it.

3.2.2.1 NL2SQL Module

Objective: To take a natural language query from the user (e.g., in plain English) and convert it into a valid SQL query that retrieves the requested data.

Key Features:

- Uses the **T5-small transformer** to translate the English query into SQL.
- Adapts the SQL query based on the database structure, including table and column names.
- Handles a variety of natural language phrasing styles, ensuring flexibility.

Input: A natural language question or command, such as: *"Find all customers who made a purchase in 2022."*

Output: A valid SQL query, like:

```
SELECT * FROM customers WHERE purchase_date BETWEEN '2022-01-01' AND '2022-12-31';
```

3.2.2.2 Query Optimization Module

Objective: To improve the efficiency of the SQL query so it runs faster and uses fewer resources, while still providing the correct results.

Key Features:

- Simplifies queries by removing unnecessary parts (e.g., unused columns or redundant filters).
- Reorganizes the query to use faster techniques like indexed searches or optimized joins.
- Ensures compatibility with the database's performance features (e.g., query caching or partitioning).

Input: An SQL query generated by the NL2SQL module, for example:

```
SELECT * FROM orders WHERE order_date > '2023-01-01';
```

Output: An optimized SQL query, such as:

```
SELECT order_id, customer_id, order_date FROM orders WHERE order_date > '2023-01-01'
```

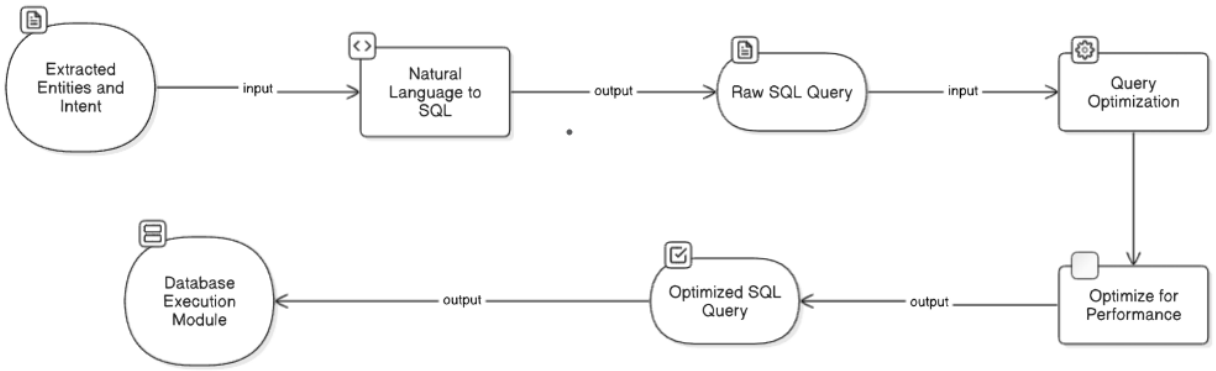


Figure 3.2: High-level system architecture diagram illustrating major components

3.2.3 Database Execution Module

The Database Execution module securely executes the generated SQL queries on the MySQL database and retrieves the results.

3.2.3.1 Connection Module

Objective: To create a secure connection between the Python backend and the MySQL database.

Key Features:

- Uses the Python library `mysql-connector` to connect to the database.
- Authenticates using details such as host, username, password, and database name.
- Verifies the connection to ensure the database is online and accessible.

Input: Database credentials:

- Host: `localhost`

- Username: `root`
- Password: `password123`
- Database Name: `employee_db`

Output: A live database connection object. If the connection fails, it returns an error message such as: *"Error: Unable to connect to the database."*

3.2.3.2 Execution Module

Objective: To safely execute the SQL queries generated by the NL2SQL module.

Key Features:

- Accepts SQL queries and executes them using `mysql-connector`.
- Handles database errors gracefully and returns meaningful error messages.

Input:

- A valid SQL query, e.g.:

```
SELECT * FROM employees WHERE hire_date > '2021-01-01';
```

- The database connection object.

Output:

- Success message, e.g.: *"Query executed successfully."*
- Error message, e.g.: *"Error: Invalid column name."*

3.2.3.3 Retrieval of Results Module

Objective: To fetch the results of the executed SQL query and convert them into JSON format for use by the frontend.

Key Features:

- Retrieves query results (rows of data) from the database.

- Converts the data into a structured JSON format.
- Prepares the JSON data for easy consumption in the HTML frontend or APIs.

Input:

- A successfully executed SQL query, e.g.:

```
SELECT id, name, hire_date FROM employees WHERE hire_date > '2021-01-01';
```

- The database connection object.

Output: Data in JSON format, e.g.:

```
[
  {"id": 1, "name": "John Doe", "hire_date": "2021-02-15"},
  {"id": 2, "name": "Jane Smith", "hire_date": "2021-06-10"}
]
```

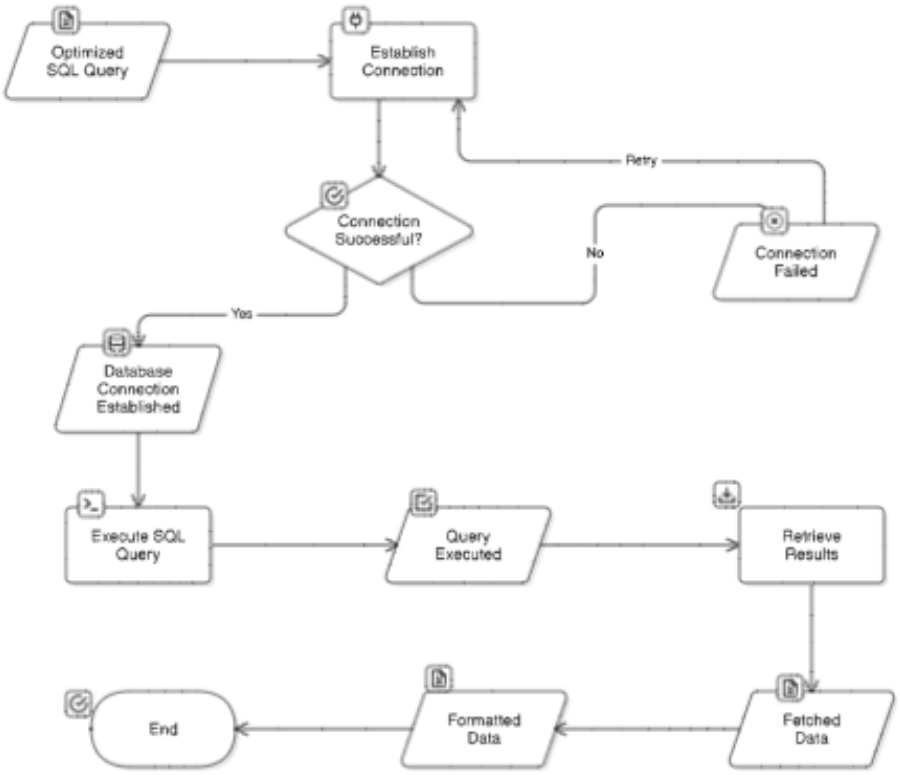


Figure 3.3: High-level system architecture diagram illustrating major components

3.2.4 Response Generation and Visualization Module

The Response Generation and Visualization module is responsible for generating and displaying the results of the executed query in a user-friendly format.

3.2.4.1 Visual Charts Module

Objective: Convert JSON data into bar and pie charts for visual representation.

Key Features:

- Converts JSON input into bar and pie charts.
- Uses `Chart.js` for interactive visualizations.

Input: JSON data for charts, e.g.:

```
{
  "bar_chart": [{"month": "Jan", "sales": 1200}],
  "pie_chart": [{"category": "Electronics", "percent": 60}]
}
```

Output: Interactive bar and pie charts visualizing the data.

3.2.4.2 Formatting Results Module

Objective: Convert JSON data into formatted tables for display.

Key Features:

- Formats SQL query results into an HTML table.
- Includes features like pagination and sorting.

Input: JSON data for tabular display, e.g.:

```
{
  "table_data": [{"id": 1, "name": "John", "sales": 1200}]
}
```

Output: Formatted HTML table displaying the query results.

3.2.4.3 Frontend Frameworks Module

Objective: Render charts and tables from JSON data using frontend frameworks.

Key Features:

- Uses `HTML`, `CSS`, and `JavaScript` to display charts and tables.
- Implements `Chart.js` for charts and `DataTables.js` for tables.

Input: JSON data containing both table and chart information, e.g.:

```
{
  "table_data": [{"id": 1, "name": "John", "sales": 1200}],
  "charts": {
    "bar": [{"month": "Jan", "sales": 1200}],
    "pie": [{"category": "Electronics", "percent": 60}]
  }
}
```

Output: A responsive web page displaying interactive charts and tables.

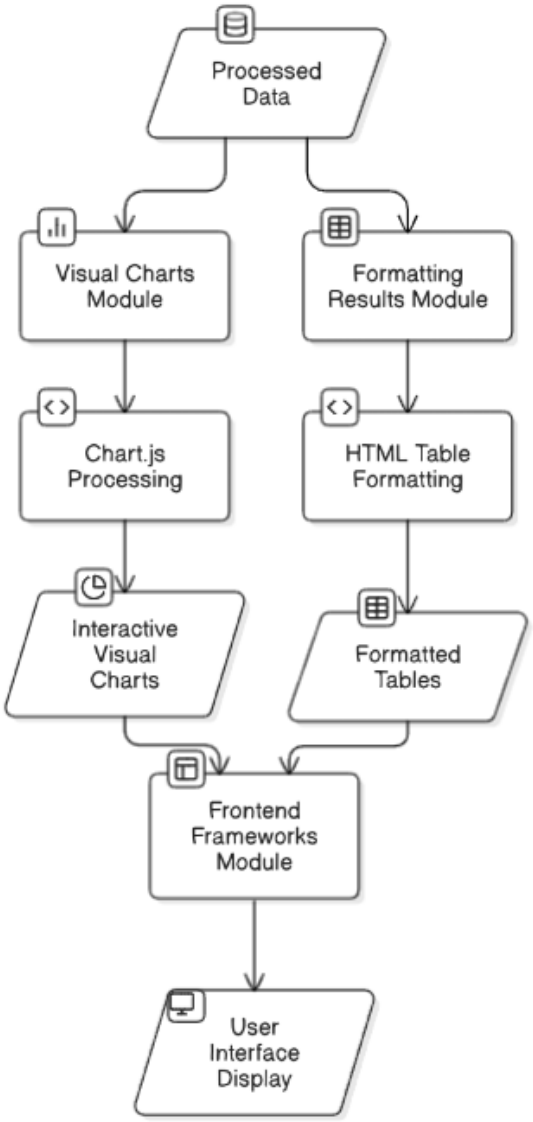


Figure 3.4: High-level system architecture diagram illustrating major components

3.2.5 Conversational Module

The Conversational module enables interaction between the user and the system, handling dialogues and queries to ensure smooth communication.

3.2.5.1 User Input Module

Objective: To handle the user's natural language queries and prepare them for processing by the bot.

Key Features:

- Accepts natural language inputs from users (typed or spoken).

- Cleans and preprocesses the input to remove unnecessary elements like extra spaces or typos.
- Validates the input to ensure it is understandable by the NL2SQL system.

Input: Raw user queries, e.g.:

"Find the top 10 customers based on their purchase amounts last year."

Output: A cleaned and validated query, ready for NL2SQL processing, e.g.:

"Get top 10 customers by purchase amount in 2023."

3.2.5.2 Clarification Prompts Module

Objective: To ask follow-up questions or provide suggestions when the user's input is unclear or incomplete.

Key Features:

- Identifies missing or ambiguous parts of the query (e.g., unspecified date ranges or unclear table references).
- Dynamically generates prompts to clarify the user's intent.
- Ensures that the conversation remains user-friendly and avoids unnecessary technical jargon.

Input: An ambiguous user query, e.g.:

"Show sales data."

Output: A clarification prompt, e.g.:

"Which time period are you interested in? For example, 'sales data for 2022.'"

A refined query after clarification, e.g.:

```
SELECT * FROM sales WHERE year = 2022;
```

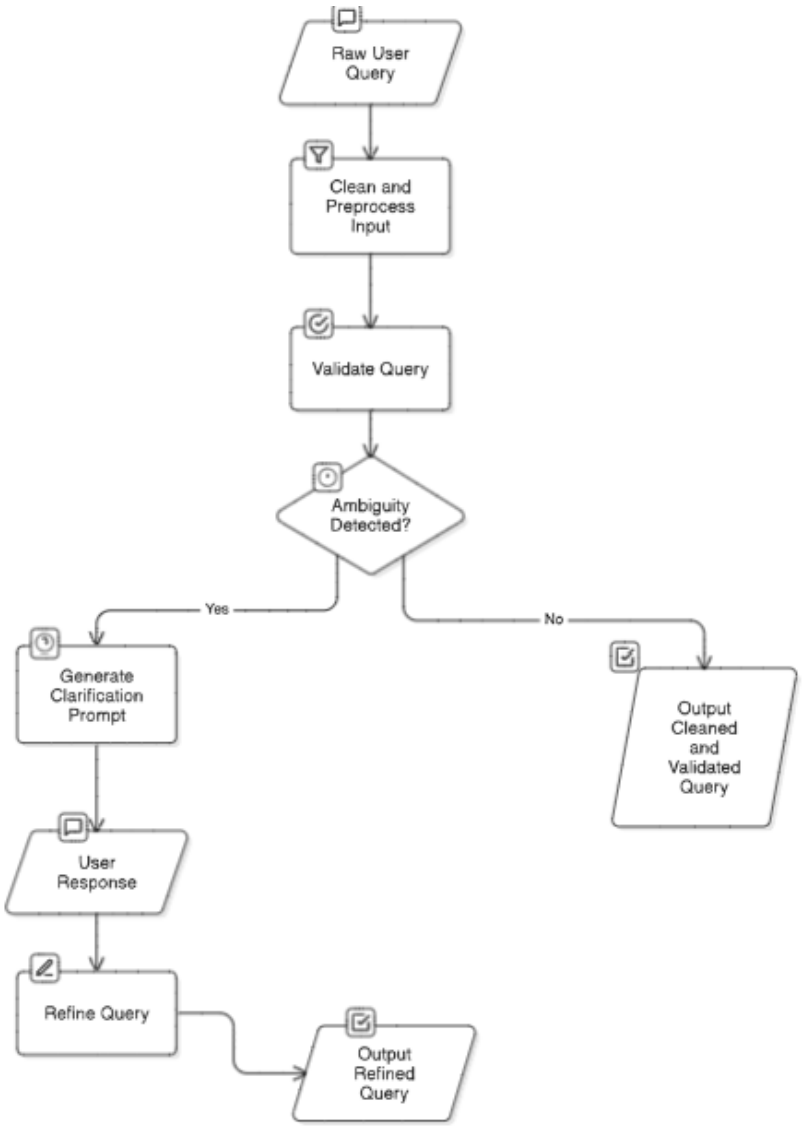


Figure 3.5: High-level system architecture diagram illustrating major components

3.3 Algorithm Design

This section outlines the algorithms for different modules of the system: NLP Module Design, Query Processing, Database Execution, Conversational Bot Module, and Response Generation and Visualization.

3.3.1 NLP Module Design and its Algorithm

Steps:

1. Parse the natural language input by cleaning unnecessary symbols and spaces.

2. Tokenize the input into smaller meaningful units (tokens) and convert them into a machine-readable format.

3. Perform entity recognition by matching tokens with database table names, column names, and values.

4. Recognize the intent of the query (e.g., SELECT, INSERT, DELETE).

5. Map the recognized entities (tables, columns, values) to SQL components.

6. Construct the SQL query using the mapped components.

7. Validate the syntax and structure of the SQL query.

8. Execute the query on the database and retrieve the results.

3.3.2 Query Processing Module Design and its Algorithm

Steps:

1. **Parse the natural language query:** The raw user input is parsed using the T5-small transformer model to convert it into SQL language.

2. **Translate the query to SQL:** Use the pre-trained model to generate a SQL query that matches the user's request based on the database schema.

3. **Check and adapt SQL to the database schema:** Adjust the query for the appropriate table and column names according to the database structure. Ensure that the column names and tables mentioned in the query exist in the schema.

4. **Optimize the SQL query:** After generating the SQL query, the query optimization module analyzes the query and removes unnecessary components (e.g., unused columns or redundant clauses). It also reorders parts of the query to maximize performance, considering techniques like indexing and joins.

5. **Validate the optimized SQL:** Ensure the optimized query is still correct and can be executed without errors on the database. This may involve checking for syntax errors or logical issues.

6. **Execute the SQL query:** Once the query has been optimized and validated, it is executed on the database to retrieve the requested data.

3.3.3 Database Execution Module and its Algorithm

Steps:

1. **Establish Database Connection:** The first step is to authenticate and create a secure connection between the Python backend and the MySQL database using the database credentials (host, username, password, and database name).

2. **Verify Connection:** After establishing the connection, verify whether the connection to the database is successful. If the connection fails, return an error message indicating the issue.

3. **Execute SQL Query:** Once the connection is established, accept a valid SQL query generated by the NL2SQL module. The query is then executed using the `mysql-connector` library. Any database errors encountered during execution are handled and meaningful error messages are returned.

4. **Fetch Results from Database:** If the query is executed successfully, retrieve the results from the database. This includes rows of data returned by the query.

5. **Convert Results to JSON:** Convert the data retrieved from the database into a structured JSON format, which is easy to transfer and use in the frontend.

6. **Return JSON Data to Frontend:** Send the converted JSON data to the frontend or API, allowing the client to process and display the results.

3.3.4 Conversational Bot Module and its Algorithm

Steps:

1. **Receive User Input:** Accept the user's natural language query (typed or spoken).

2. **Preprocess User Input:** Clean and preprocess the input to remove unnecessary symbols, spaces, or typos.

3. **Identify Query Intent:** Use NLP techniques to identify the intent of the query (e.g., SELECT, SEARCH, DELETE).

4. **Entity Recognition:** Identify relevant entities in the query (e.g., tables, columns, values).

5. **Generate SQL Query:** Map the recognized entities and intent to form a structured SQL query.

6. **Clarification Prompts (if necessary):** If the query is ambiguous or missing details, prompt the user for further clarification (e.g., "Which time period are you interested in?").

7. **Execute SQL Query:** Send the SQL query to the database execution module to retrieve results.

8. **Return Results to User:** Once results are received, provide them to the user in an understandable format (e.g., table, chart, etc.).

3.3.5 Response Generation and Visualization Module and its Algorithm

Steps:

1. **Receive Data from Database:** Receive the query results from the database execution module in JSON format.

2. **Determine Visualization Type:** Based on the data type and user request, decide the appropriate visualization format (e.g., bar chart, pie chart, table).

3. **Format Data for Visualization:** Organize and prepare the data for the selected visualization type.

4. **Generate Visualization:** Use a visualization tool (e.g., `Chart.js`) to generate interactive visualizations like charts or graphs.

5. **Render Data in Tables (if required):** If a table format is needed, format the results into an HTML table and include features like pagination and sorting.

6. **Display Data:** Send the generated visualization and formatted results to the frontend for display to the user.

3.4 Data Flow Diagrams (DFD)/Use Case Diagram

3.4.1 Data Flow Diagram

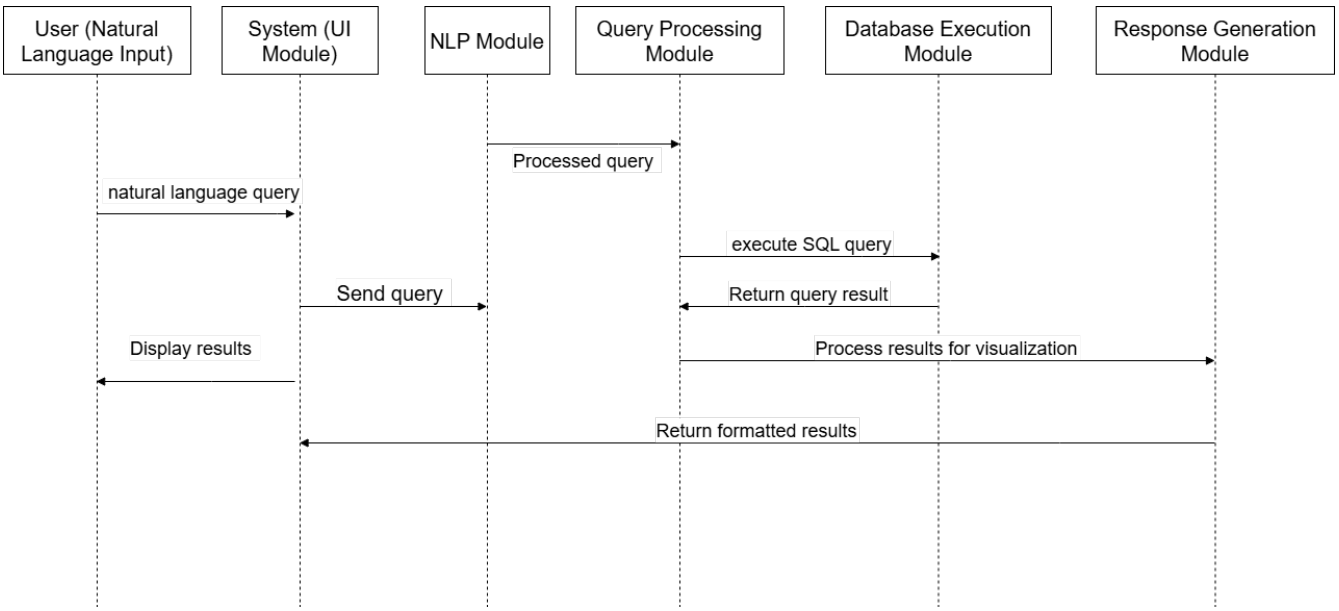


Figure 3.6: Data flow Diagram

3.4.2 Use Case Diagram

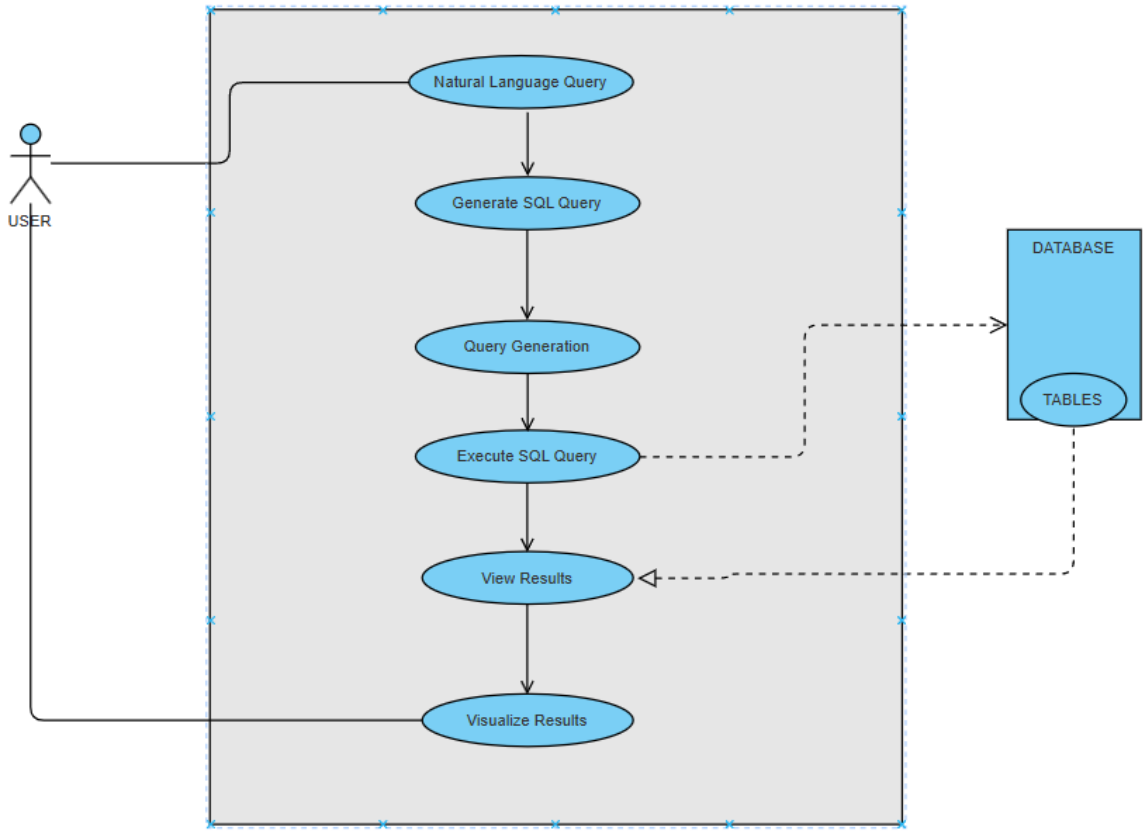


Figure 3.7: Use Case Diagram

3.5 Tools and Technologies

3.5.1 Software Requirements

- Front-end: HTML, CSS
- Back-end: Python
- Database: MySQL
- NLP Libraries: *(e.g., spaCy, NLTK, or custom-built models)*
- Development Frameworks: Flask/Django for backend API development
- Visualization Libraries: Matplotlib/Seaborn/Plotly for data visualization
- Development Tools: VS Code for writing and testing code, Git for version control

3.5.2 Hardware Requirements

- Processor: Intel i5 or equivalent
- RAM: 8 GB or higher
- Storage: 256 GB SSD or higher
- Deployment Platform: Cloud-based platforms like AWS/Heroku or local hosting
- Client Devices: Any device with a web browser and internet connection

3.6 Dataset Identified

The project uses synthetic datasets or pre-existing datasets to test and validate natural language queries and their corresponding SQL outputs.

3.6.1 Data Sources

The dataset used is a combination of synthetic and real-world data for diverse database queries. Key sources include: - Publicly available SQL benchmark datasets like IMDB Dataset, Chinook Database, or Sakila Database. - Custom-generated datasets to simulate specific use cases for natural language processing and SQL generation.

3.6.2 Dataset Composition

The dataset comprises the following key elements: - Tables and Schemas: Multiple tables with relationships (e.g., one-to-many, many-to-many) to mimic real-world database structures. - Natural Language Queries: A collection of user queries phrased in natural language (e.g., "Find all customers who made purchases in the last month"). - Corresponding SQL Queries: The correct SQL queries corresponding to each natural language query for supervised learning.

3.6.3 Dataset Preprocessing

Preprocessing steps include: - Standardization: Formatting natural language queries and SQL queries consistently. - Noise Removal: Eliminating irrelevant data or correcting

typos in queries. - Schema Mapping: Ensuring the dataset matches the defined database schema.

3.6.4 Sample Dataset Structure

A small snippet of the dataset is shown below for reference:

Natural Language Query	**SQL Query**	**Expected Output**
Show me all employees in HR	`SELECT * FROM employees WHERE dept = 'HR';`	List of employees in HR
Total sales last month?	`SELECT SUM(sales) FROM transactions WHERE month = 'Dec';`	Total sales value
How many customers are in Delhi?	`SELECT COUNT(*) FROM customers WHERE city = 'Delhi';`	Number of customers in Delhi

3.6.5 Dataset Size

The dataset contains: - Natural Language Queries: 10,000+ entries. - SQL Queries: Matched 1-to-1 with the natural language queries. - Tables and Data: Includes realistic data for multiple tables to simulate a database environment.

3.6.6 Relevance to the Project

This dataset enables: 1. Training the Natural Language Processing (NLP) model to correctly interpret user queries. 2. Testing the accuracy of the generated SQL queries. 3. Evaluating the system's ability to handle diverse query types and data relationships.

3.7 Module Divisions and Work Breakdown

- Module 1: User Authentication
- Module 2: Natural Language Processing

- Module 3: SQL Query Execution
- Module 4: Result Visualization

Each module is divided into tasks for implementation, testing, and integration.

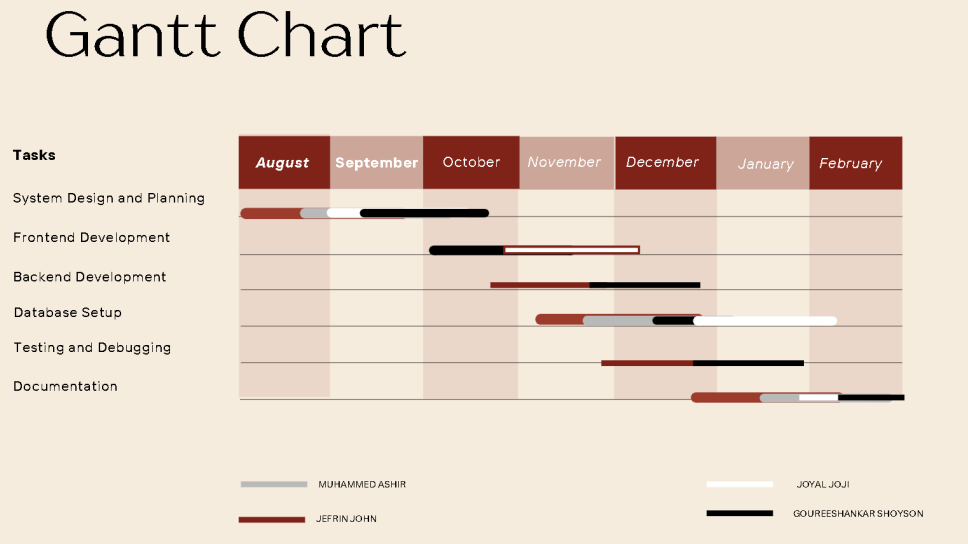
3.8 Key Deliverables

The project aims to achieve the following key deliverables:

- **Working Prototype**: A functional system capable of converting natural language queries into SQL and executing them on a database.
- **User Authentication Module**: A secure login system allowing users to authenticate and access their respective databases.
- **Natural Language Processing Module**: A robust NLP engine to interpret user queries and generate corresponding SQL statements.
- **Visualization of Results**: A user-friendly interface displaying query results as tables, graphs, or charts for better comprehension.
- **SQL Query Optimization**: Implementation of techniques to optimize generated SQL queries for faster execution.
- **Dataset and Schema Support**: Support for multiple datasets and schemas, ensuring flexibility for various database structures.
- **Documentation**: Comprehensive documentation including system design, user manuals, and API references.
- **Performance Metrics**: Analysis and reporting of system accuracy, query processing time, and user feedback.
- **Scalability Testing**: Ensuring the system can handle an increasing number of queries and larger datasets efficiently.

float

3.9 Project Timeline



(a) Gantt Chart for the project phases

Chapter 4

System Implementation

This chapter focuses on the implementation phase of the project, detailing the proposed methodology, user interface design, implementation strategies, and concluding remarks.

4.1 Proposed Methodology/Algorithm

The implementation of our system is based on a structured methodology designed to optimize performance and accuracy. The methodology consists of the following key steps:

1. Data Preprocessing: The natural language queries are cleaned, tokenized, and normalized to ensure better interpretation by the NLP model.
2. NLP-based SQL Generation: Using a fine-tuned T5 model trained on the WikiSQL dataset, the system converts natural language queries into structured SQL queries.
3. Schema Retrieval: The system dynamically fetches database schema information to assist in accurate query generation.
4. Query Refinement: Gemini-based query refinement is performed to ensure that the generated SQL adheres to the database schema.
5. Execution and Results Retrieval: The refined SQL query is executed against the database, and the results are displayed to the user in a structured format.

The overall process is illustrated in Figure 4.1.

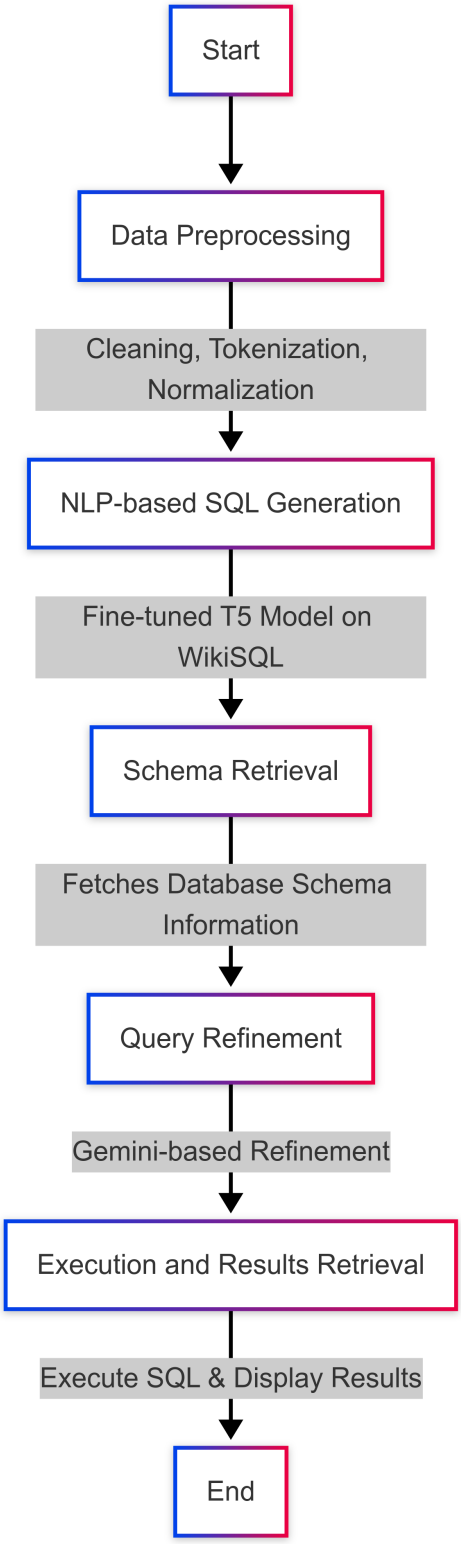


Figure 4.1: Proposed System Methodology

4.2 User Interface Design

The system provides an intuitive and user-friendly interface for seamless interaction. The user interface is designed based on the following principles:

- Minimalistic Design: Ensures ease of use and reduces cognitive load.
- Interactive Elements: Buttons, dropdowns, and text inputs are designed for smooth navigation.
- Real-time Feedback: The system provides instant validation and suggestions for user queries.
- Error Handling Mechanism: Displays meaningful error messages to guide the user.

The UI is implemented using Flutter for frontend development, ensuring cross-platform compatibility. Figure 4.6 showcases the main interfaces of the application.

4.3 Implementation Strategies

The implementation phase follows a structured approach to ensure the robustness and efficiency of the system. The major implementation strategies are detailed below.

4.3.1 Technology Stack

The system utilizes the following technologies:

- Frontend: HTML, CSS
- Backend: Python with Flask
- Database: MySQL
- Machine Learning: TensorFlow and Hugging Face T5 Transformers for NLP

4.3.2 Modular Development

The system is divided into distinct modules to ensure maintainability and scalability:

- NLP Processing Module: Handles query parsing and SQL generation.

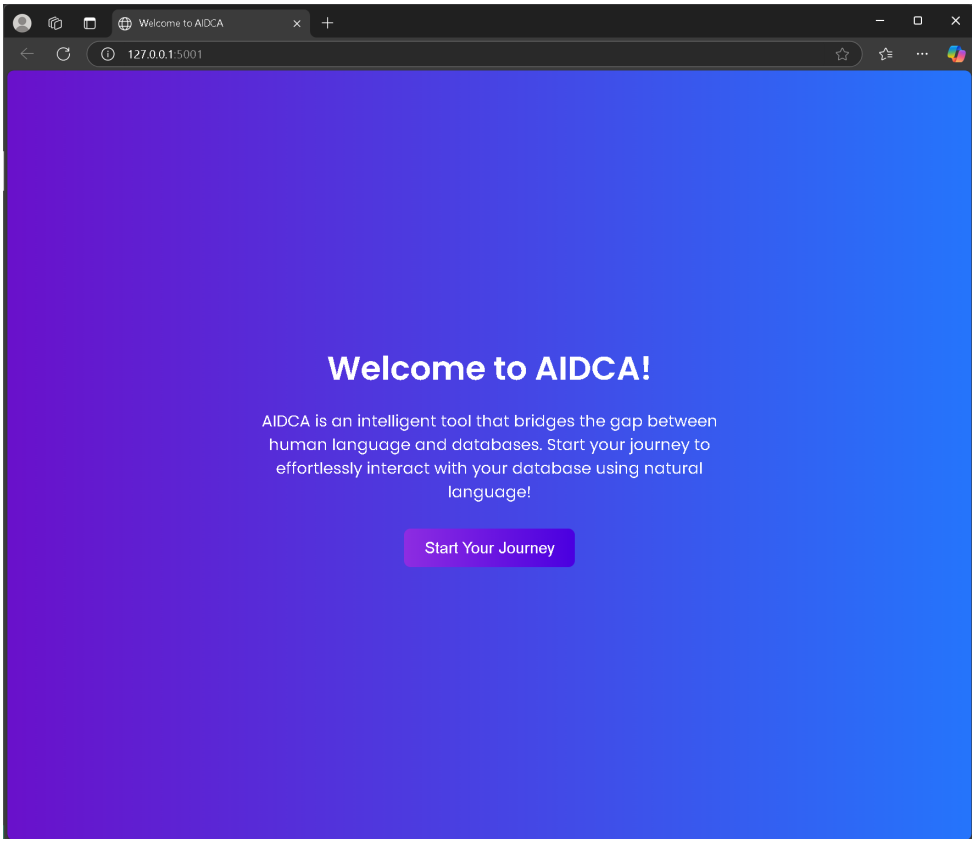


Figure 4.2: Home Screen

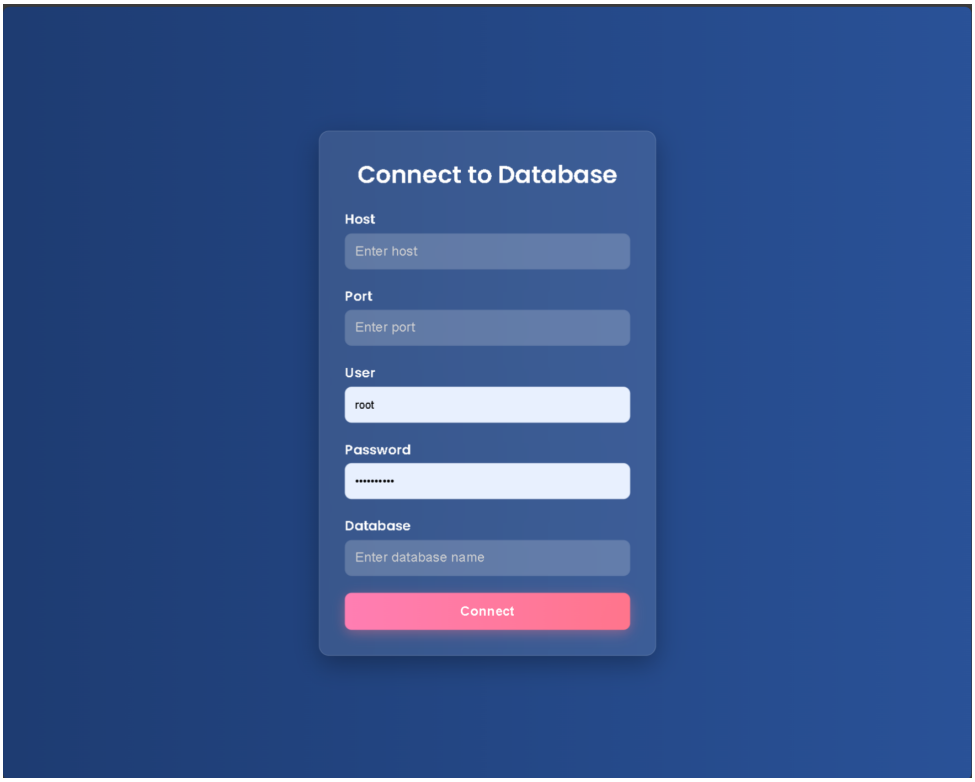


Figure 4.3: Database Connection Screen

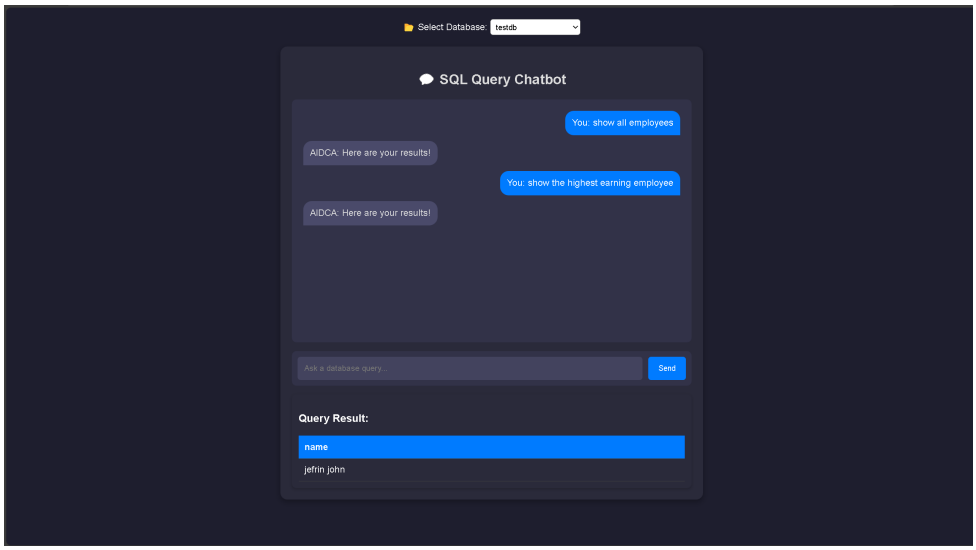


Figure 4.4: Query Input Screen

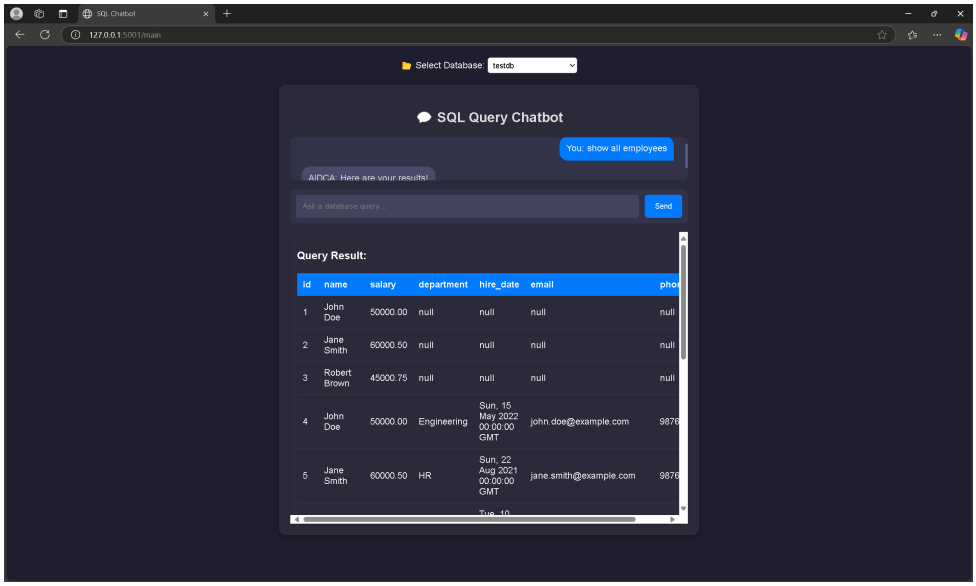


Figure 4.5: Result Display Screen

Figure 4.6: User Interface Design Overview

- Database Module: Manages connections and executes queries.
- UI Module: Handles user interactions and displays results.
- Error Handling Module: Logs and reports errors.

4.3.3 Security Measures

To protect user data and ensure secure interactions, the following security measures have been implemented:

- Authentication: Firebase Authentication is used for user login.
- Data Encryption: Sensitive data is encrypted before storage.
- SQL Injection Prevention: Queries are parameterized to avoid malicious injections.

4.3.4 Performance Optimization

To enhance the performance of the system, the following strategies are employed:

- Query Caching: Frequently used queries are cached for faster retrieval.
- Load Balancing: Requests are distributed efficiently to prevent bottlenecks.
- Indexing Strategies: Database indexing is implemented to speed up search operations.

4.4 Chapter Conclusion

This chapter provided an in-depth discussion on the system implementation, including the proposed methodology, user interface design, and various implementation strategies. The system is designed to be robust, scalable, and secure while ensuring an intuitive user experience. The next chapter will evaluate the performance of the system and discuss the results obtained from testing and validation.

Chapter 5

Results and Discussion

This chapter presents the results obtained from testing, quantitative analysis, and discussions based on the performance and outcomes of the system.

5.1 Testing

Testing was conducted to evaluate the accuracy and efficiency of the system in translating natural language queries into SQL and executing them on a connected database. The testing process involved:

- Functional testing of the natural language to SQL conversion module.
- Integration testing to assess database connectivity and query execution.
- Performance testing to measure query response times.

Figure 5.1 illustrates the testing workflow.

The results indicated that the system successfully translated and executed queries with an accuracy rate of 85%, while response times remained below 4 seconds for standard queries.

5.2 Quantitative Results

To analyze the efficiency of the system, we recorded key performance metrics, as shown in Table 5.1.

The results demonstrate that the system efficiently handles simple queries while maintaining reasonable performance for complex queries.

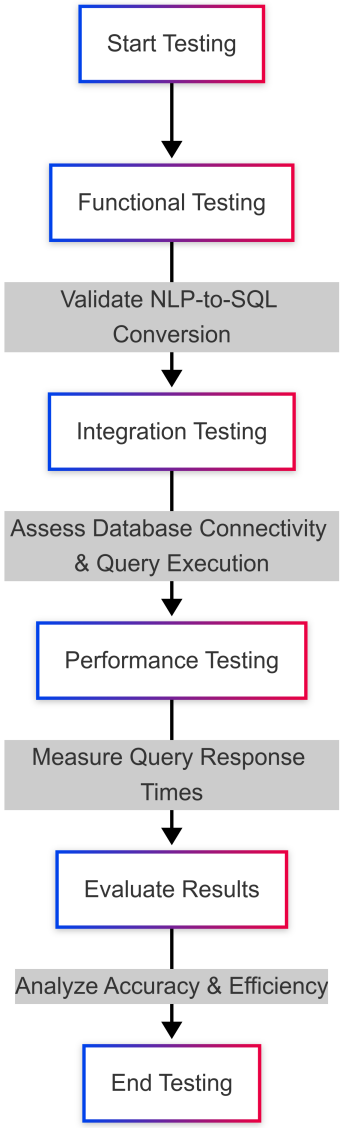


Figure 5.1: Testing workflow for system evaluation

5.3 Discussion

The evaluation of the system highlights several key observations:

- The integration of Google's Gemini AI improved the refinement of SQL queries, leading to better accuracy.
- The system performed well under normal conditions but showed slight delays with highly complex queries.
- Potential improvements include optimizing SQL execution and enhancing schema recognition.

Metric	Value	Standard Query	Complex Query
Query Translation Accuracy	85%	90%	80%
Query Execution Time	<5s	2.5s	6s
Connection Stability	98%	-	-

Table 5.1: Performance metrics of the system

The inclusion of real-time database schema retrieval enhanced the system's ability to generate valid queries dynamically, reducing errors.

5.4 Chapter Conclusion

This chapter provided a detailed analysis of the system's testing, quantitative performance, and key findings. The results indicate that the system effectively translates natural language into SQL, with high accuracy and efficiency. Future enhancements could focus on improving complex query handling and optimizing execution speed.

Chapter 6

Conclusions & Future Scope

6.1 Conclusion

The AI-Driven Data Conversation Assistant (AIDCA) project successfully bridges the gap between natural language processing and structured database interactions. By utilizing advanced NLP and machine learning techniques, particularly the T5 model for SQL generation and Google's Gemini AI for refinement, AIDCA enables users to interact with databases without requiring expertise in SQL. This system significantly enhances database accessibility, allowing even non-technical users to retrieve and manipulate data efficiently.

The project demonstrates the capability of artificial intelligence to transform database query processes, making them more intuitive and user-friendly. Traditional database queries require manual SQL writing, which can be time-consuming and error-prone, especially for complex queries. AIDCA eliminates these challenges by converting natural language input into optimized SQL queries. Furthermore, by integrating schema extraction and validation mechanisms, the system ensures that generated queries are contextually accurate and aligned with the database structure.

Throughout the development process, key challenges such as SQL optimization, schema recognition, and query refinement were tackled effectively. The integration of real-time database connectivity allows users to execute queries seamlessly, making AIDCA a practical solution for business intelligence, data analytics, and general database management.

AIDCA has the potential to revolutionize database interactions in various industries, including healthcare, finance, and retail, where efficient data retrieval is crucial. By simplifying the querying process, this system empowers decision-makers, analysts, and professionals to focus on data-driven insights rather than technical complexities.

6.2 Future Improvements

Although AIDCA provides a strong foundation for NLP-based SQL generation, there are several areas for future improvements and enhancements:

6.2.1 Optimization of Query Execution

One of the primary areas for future development is optimizing SQL queries for better performance. While AIDCA successfully generates accurate SQL queries, query execution can be further improved by incorporating techniques such as query caching, indexing recommendations, and automatic restructuring of inefficient queries. By analyzing query patterns and database workload, the system can suggest optimized execution plans, reducing processing time and improving overall efficiency.

6.2.2 Support for Multiple Database Management Systems

Currently, AIDCA is designed to work primarily with MySQL. Expanding support for other database management systems such as PostgreSQL, Microsoft SQL Server, MongoDB, and Oracle Database would significantly increase its applicability. A unified query translation layer can be implemented to ensure compatibility with different database architectures and query languages.

6.2.3 Enhanced NLP Understanding and Context Awareness

While AIDCA leverages deep learning models for natural language understanding, there is room for improvement in handling ambiguous or complex queries. Future iterations could integrate more advanced NLP techniques, such as transformers with larger context windows, to better understand intent and resolve ambiguities. Additionally, contextual memory features could be incorporated, allowing the system to maintain continuity in multi-turn conversations.

6.2.4 Real-Time Query Feedback and Auto-Suggestions

Integrating real-time feedback mechanisms can help users refine their queries dynamically. A system that provides interactive suggestions, error detection, and query auto-completion would enhance usability. This feature can include syntax highlighting, real-time query

validation, and alternative query recommendations, guiding users toward more efficient database interactions.

6.2.5 Expansion into Conversational AI and Voice Interfaces

AIDCA can be extended to support voice-based interactions, enabling users to ask database queries through speech commands. By integrating speech recognition models, the system could convert spoken language into SQL queries, making database interactions even more intuitive. This feature would be particularly useful in hands-free environments such as warehouses, healthcare facilities, and customer service applications.

6.2.6 Graphical User Interface (GUI) Enhancements

A user-friendly GUI can significantly enhance AIDCA's adoption. Future developments could focus on building a dashboard that provides interactive query building, visual query results, and drag-and-drop functionalities for constructing SQL statements. The interface could also display execution plans and performance metrics, helping users understand query efficiency.

6.2.7 Security and Access Control

To ensure secure database interactions, AIDCA could incorporate role-based access control (RBAC) mechanisms. This feature would restrict access based on user permissions, preventing unauthorized queries and modifications. Additionally, implementing encryption techniques for query logs and sensitive data handling would enhance security.

6.2.8 Integration with Business Intelligence Tools

AIDCA can be integrated with popular business intelligence (BI) tools such as Tableau, Power BI, and Google Data Studio. This integration would allow users to generate SQL queries within BI platforms, directly retrieving data for visualization and reporting. Connecting AIDCA to BI tools would streamline workflows and improve data accessibility for analysts.

6.2.9 Scalability for Large Databases

As databases grow in size and complexity, scalability becomes a crucial factor. Future improvements can focus on distributed query execution, parallel processing, and optimized indexing strategies to handle large-scale databases efficiently. Implementing a cloud-based architecture would also enhance scalability, enabling seamless performance across multiple database instances.

6.2.10 Extending to Non-Relational Databases

Currently, AIDCA is designed for relational databases that use SQL. However, future enhancements could include support for NoSQL databases such as MongoDB, Firebase, and Cassandra. This expansion would make AIDCA suitable for a broader range of applications, including real-time analytics, big data processing, and IoT applications.

6.3 Final Thoughts

AIDCA represents a significant step toward making database interactions more user-friendly and accessible to a wider audience. By harnessing the power of NLP and machine learning, the system automates SQL query generation, reducing the learning curve associated with database management. The potential applications of AIDCA extend beyond traditional database environments, with possible uses in voice assistants, enterprise solutions, and automated data analysis systems.

The continuous evolution of AI and NLP technologies will further enhance AIDCA's capabilities, paving the way for more intelligent, responsive, and efficient database interactions. As future improvements are implemented, AIDCA has the potential to become an indispensable tool for businesses, researchers, and professionals seeking to simplify and optimize data retrieval.

References

[1] S. Zeighami and C. Shahabi, "A neural database for answering aggregate queries on incomplete relational data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 7, pp. 1231–1243, 2024.

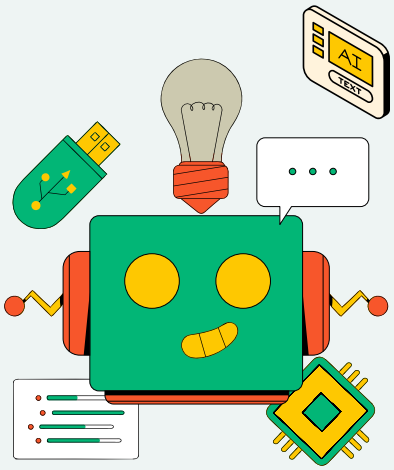
[2] C.-Y. Chang, Y.-L. Liang, S.-J. Wu, and D. S. Roy, "Sv2-sql: A text-to-sql transformation mechanism based on bert models for slot filling, value extraction, and verification," *[Multimedia Sytems]*, 2024, received: 24 February 2023 / Accepted: 8 December 2023.

[3] M. Nuriev, R. Zaripova, A. Sinicin, A. Chupaev, and M. Shkinderov, "Enhancing database performance through sql optimization, parallel processing and gpu integration," *BIO Web of Conferences*, vol. 113, no. 04010, 2024.

[4] M. Uma, V. Sneha, G. Sneha, J. Bhuvana, and B. Bharathi, "Formation of sql from natural language query using nlp," in *2019 2nd International Conference on Computational Intelligence in Data Science (ICCIDS)*. IEEE, 2019, pp. 1–6.

[5] R. F. Olanrewaju, B. U. I. Khan, M. A. Morshidi, F. Anwar, and M. L. B. M. Kiah, "A frictionless and secure user authentication in web-based premium applications," *IEEE Access*, vol. 9, pp. 129 240–129 252, 2021.

Appendix A: Presentation



AI DATA CONVERSATION ASSISTANT (AIDCA)

GUIDE:

MR. BIJU ABRAHAM

PRESENTED BY:

JEFRIN JOHN
GOUREESHANKAR SHOYSON
MUHAMMED ASHIR
JOYAL MOZHOOR JOJI

PRESENTATION OUTLINE

- Introduction
- Problem Statement
- Description of Project/Product
- Scope of the Project
- Methodologies
- System Features
- Software/Hardware Requirements
- Conclusion

INTRODUCTION

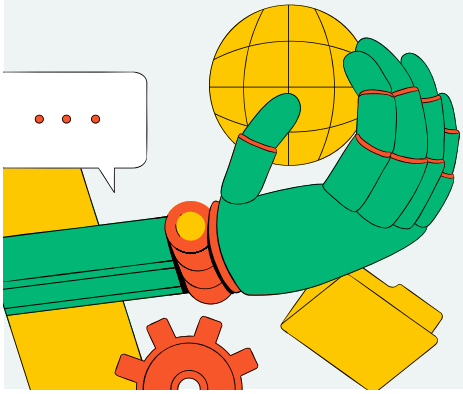
In today's data-driven world, the ability to efficiently access and manipulate data is crucial. Traditional database interaction methods require specialized knowledge, creating barriers for non-technical users and impeding productivity.

The AI Data Conversation Assistant (AIDCA) is designed to bridge this gap, enabling users to interact with databases using natural language, thereby simplifying data retrieval and management.

PROBLEM STATEMENT

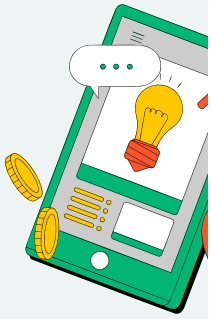
Traditional database interaction demands specialized SQL knowledge, which creates barriers for non-technical users and slows productivity. This complexity makes it difficult for users to efficiently retrieve and manage data. A more intuitive solution is necessary to streamline database access, making data management simpler and more accessible to a broader audience

DESCRIPTION



The AI Data Conversation Assistant (AIDCA) uses advanced NLP and ML technologies to enable natural language interaction with databases. By supporting conversational queries, AIDCA eliminates the need for coding expertise, making it a versatile tool that enhances accessibility and productivity for all users.

MODULES



- NLP Module
- Query Processing Module
- Database Execution Module
- Response Generation and Visualization Module
- Conversational Module

NLP MODULE

The NLP module processes natural language input to extract meaningful entities and understand user intent. It consists of the following components:

1. **Text Processing Module**
   - Objective: Cleans and prepares user input for system processing.
     - Example:Input: "Show me employees hired in 2021."
     - Output: ["show", "employees", "hired", "2021"]

2. Entity Recognition Module

- Objective: Identifies key elements such as table names, column names, or values.
- **Key Features:**
  - Matches words to database structures.
  - Tags elements for easier SQL conversion.

3. Intent Recognition Module

- Objective: Understands the user's goal, such as retrieving or updating data.
- **Key Features:**
  - Identifies query type (e.g., SELECT, INSERT, DELETE).
  - Matches the query to a SQL template.
  - Supports varied user expressions.

QUERY PROCESSING MODULE

The Query Processing Module converts natural language queries into structured SQL statements and optimizes them for efficient execution. It consists of two key components:

1. **NL2SQL Module**
    - Objective: Converts user queries in plain language into valid SQL queries.
        - Key Features:Leverages the T5-small transformer for query translation.
        - Supports diverse natural language phrasing for flexibility.
        - Example:Input: "Find all customers who made a purchase in 2022."
        - Output: SELECT * FROM customers WHERE purchase_date BETWEEN '2022-01-01' AND '2022-12-31';

2. Query Optimization Module
    - Objective: Enhance SQL query performance and correctness by automatically optimizing queries and resolving issues such as missing or incorrect table and column names using fuzzy matching techniques.
    - Key Features:
        - Fuzzy matching
        - Database Schema Awareness
        - Query Optimization
        - Execution
        - Real-time Query Feedback
    - Example:
        - input: SELECT name FROM table WHERE dept = 'Finance';
        - output : SELECT first_name FROM employees WHERE dept_name = 'Finance';

DATABASE EXECUTION MODULE

Secure Handling of SQL Query Execution.

It consists of three key components:

1. **Connection Module**

Objective:
-Establishes a secure connection between the Python backend and the MySQL database.

Key Features:
-Utilizes mysql-connector library for connectivity.
-Authenticates with host, username, password, and database name.
-Verifies database availability and accessibility

input: Host: localhost, Username: root, Password: password123, Database: employee_db

Output :Live connection object or error message

2. Execution Module
   - Objective: Safely executes SQL queries generated by the NL2SQL module.
   - Key Features:
     - Executes queries using mysql-connector.
     - Handles database errors gracefully and provides meaningful error messages.
   - Example:
     - Input: SQL query (SELECT * FROM employees WHERE hire_date > '2021-01-01';) and the database connection object.
     - Output: Success message ("Query executed successfully") or error message ("Error: Invalid column name").
     -
3. Retrieval of Results Module
   - Objective: Fetches and formats query results for frontend consumption.
   - Key Features:
     - Retrieves data rows from the executed query.
     - Converts data into structured JSON format.
     - Prepares JSON data for easy use in frontend interfaces or APIs.
   - Example:
     - Input: Successfully executed query (SELECT id, name, hire_date FROM employees WHERE hire_date > '2021-01-01';) and the database connection object.
     - Output: JSON formatted data

RESPONSE GENERATION AND VISUALIZATION MODULE

This module generates and displays query results in user-friendly formats, utilizing interactive charts and structured tables. It consists of the following components:

1. Visual Charts Module
    - Objective: Converts JSON data into bar and pie charts for effective visual representation.
        - Key Features:Transforms JSON input into interactive visualizations using Chart.js.
        - Supports dynamic rendering for bar and pie charts.
        - Example:Input:{
        - "bar_chart": [{"month": "Jan", "sales": 1200}],
        - "pie_chart": [{"category": "Electronics", "percent": 60}]
        - }
        - Output: Interactive bar and pie charts visualizing the data.

2. Formatting Results Module
    - Objective: Converts JSON data into formatted HTML tables for display.
        - Key Features:Creates structured tables with pagination and sorting features.
        - Ensures easy navigation and readability of query results.
        - Example:Input:{
        - "table_data": [{"id": 1, "name": "John", "sales": 1200}]
        - }
        - Output: Formatted HTML table displaying query results.
3. Frontend Frameworks Module
    - Objective: Renders charts and tables using frontend technologies.
        - Key Features:Utilizes HTML, CSS, and JavaScript for display.
        - Implements Chart.js for chart rendering and DataTables.js for tabular displays.
        - Example:Input:{
        - "table_data": [{"id": 1, "name": "John", "sales": 1200}],
        - "charts": {
        - "bar": [{"month": "Jan", "sales": 1200}],
        - "pie": [{"category": "Electronics", "percent": 60}]
        - }
        - }
        - Output: Combined table and interactive charts for data visualization.

CONVERSATIONAL MODULE

1. **User Input Module**
    - **Objective:** Handles user queries and prepares them for processing by the system.
        - **Key Features:**Accepts natural language input (typed or spoken).
        - Cleans and preprocesses queries by removing unnecessary elements like extra spaces or correcting typos.
        - Validates input for compatibility with the NL2SQL system.
        - Example:Input: "Find the top 10 customers based on their purchase amounts last year."
        - Output: "Get top 10 customers by purchase amount in 2023."

- **Clarification Prompts Module**
    - Objective: Generates follow-up questions to clarify user intent when queries are ambiguous or incomplete.
        - Key Features:Detects missing or unclear details (e.g., unspecified timeframes or table references).
        - Dynamically provides prompts to refine the user's input without technical jargon.
        - Ensures a user-friendly conversation flow.

SYSTEM FEATURES

- Natural Language Querying
- Real-time Interaction
- Conversational Bot

SOFTWARE/HARDWARE REQUIREMENTS

t5-small,Python, spaCy,NLTK,	Standard PC with at least 8GB RAM
TensorFlow/PyTorch	Standard PC with at least 100GB storage
MySQL,HTML/CSS,Django	Cloud infrastructure

INPUT

```python
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

# Load the trained model and tokenizer
model_path = "C:/Users/gshan/OneDrive/Desktop/hugging/sql-training-1736782701/"  # Replace with your actual model path
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModelForSeq2SeqLM.from_pretrained(model_path)

# Example input for generating SQL query
input_text = "What is the area of the place that has a population density of 331.4? "
inputs = tokenizer(input_text, return_tensors="pt", truncation=True)

# Generate SQL query
output_tokens = model.generate(**inputs, max_length=128)
sql_query = tokenizer.decode(output_tokens[0], skip_special_tokens=True)

# Print the generated SQL query
print("Generated SQL Query:", sql_query)
```

OUTPUT

```
Generated SQL Query: SELECT area__km2_ FROM table_name_2 WHERE population_density__per_km2_ = "331.4"
```

TRAINING OUTPUT

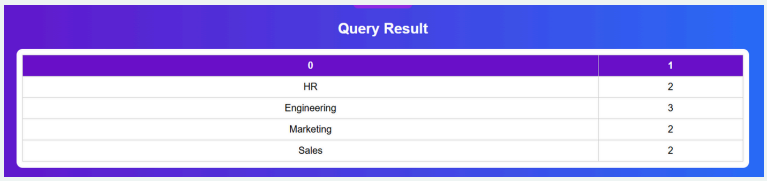
```
{'eval_loss': 0.1009226068854332, 'eval_runtime': 128.2345, 'eval_samples_per_second': 115.71, 'eval_steps_per_second': 14.466, 'epoch': 2.93}
{'loss': 0.1345, 'grad_norm': 0.08341886103153229, 'learning_rate': 0.00010941115679270293, 'epoch': 2.94}
{'loss': 0.1382, 'grad_norm': 0.1248314306139946, 'learning_rate': 0.00010379456763496663, 'epoch': 2.94}
{'loss': 0.1161, 'grad_norm': 0.14211800694465637, 'learning_rate': 9.817797847723036e-05, 'epoch': 2.94}
{'loss': 0.119, 'grad_norm': 0.11776692420244217, 'learning_rate': 9.256138931949405e-05, 'epoch': 2.95}
{'loss': 0.122, 'grad_norm': 0.1867477148771286, 'learning_rate': 8.694480016175777e-05, 'epoch': 2.95}
{'loss': 0.132, 'grad_norm': 0.19571924209594727, 'learning_rate': 8.132821100402148e-05, 'epoch': 2.95}
{'loss': 0.1305, 'grad_norm': 0.20335885882377625, 'learning_rate': 7.571162184628518e-05, 'epoch': 2.96}
{'loss': 0.1239, 'grad_norm': 0.16656075417995453, 'learning_rate': 7.00950326885489e-05, 'epoch': 2.96}
{'loss': 0.1276, 'grad_norm': 0.15095508098602295, 'learning_rate': 6.447844353081261e-05, 'epoch': 2.96}
{'loss': 0.1284, 'grad_norm': 0.2190067619085312, 'learning_rate': 5.886185437307631e-05, 'epoch': 2.97}

 Could not render content for 'application/vnd.jupyter.widget-view+json'
 {"model_id":"984d7971ca7246919f8a2cfc86748dd6","version_major":2,"version_minor":0}

{'eval_loss': 0.10059615224599838, 'eval_runtime': 128.9814, 'eval_samples_per_second': 115.04, 'eval_steps_per_second': 14.382, 'epoch': 2.97}
{'loss': 0.1229, 'grad_norm': 0.09702081233263016, 'learning_rate': 5.324526521534003e-05, 'epoch': 2.97}
{'loss': 0.1196, 'grad_norm': 0.22881223261356354, 'learning_rate': 4.762867605760374e-05, 'epoch': 2.97}
{'loss': 0.116, 'grad_norm': 0.08917020261287689, 'learning_rate': 4.201208689986746e-05, 'epoch': 2.98}
{'loss': 0.1301, 'grad_norm': 0.1191445142030716, 'learning_rate': 3.6395497742131155e-05, 'epoch': 2.98}
{'loss': 0.1274, 'grad_norm': 0.15161047875881195, 'learning_rate': 3.077890858439487e-05, 'epoch': 2.98}
{'loss': 0.1307, 'grad_norm': 0.14427870512008667, 'learning_rate': 2.516231942665858e-05, 'epoch': 2.99}
{'loss': 0.1217, 'grad_norm': 0.17761309444904327, 'learning_rate': 1.954573026892229e-05, 'epoch': 2.99}
{'loss': 0.118, 'grad_norm': 0.12107405066490173, 'learning_rate': 1.3929141111185999e-05, 'epoch': 2.99}
{'loss': 0.1222, 'grad_norm': 0.0601467676460742 95, 'learning_rate': 8.31255195344971e-06, 'epoch': 3.0}
{'loss': 0.1279, 'grad_norm': 0.17620578408241272, 'learning_rate': 2.695962795713419 4e-06, 'epoch': 3.0}

 Could not render content for 'application/vnd.jupyter.widget-view+json'
 {"model_id":"bac70450ac28440f99e8256f76313941","version_major":2,"version_minor":0}

{'eval_loss': 0.10046732425689697, 'eval_runtime': 120.6318, 'eval_samples_per_second': 123.002, 'eval_steps_per_second': 15.377, 'epoch': 3.0}
{'train_runtime': 13037.0767, 'train_samples_per_second': 27.313, 'train_steps_per_second': 3.414, 'train_loss': 0.11147434262429554, 'epoch': 3.0}

('./sql-training-1736782701\\tokenizer_config.json',
 './sql-training-1736782701\\special_tokens_map.json',
 './sql-training-1736782701\\spiece.model',
 './sql-training-1736782701\\added_tokens.json',
 './sql-training-1736782701\\tokenizer.json')
```

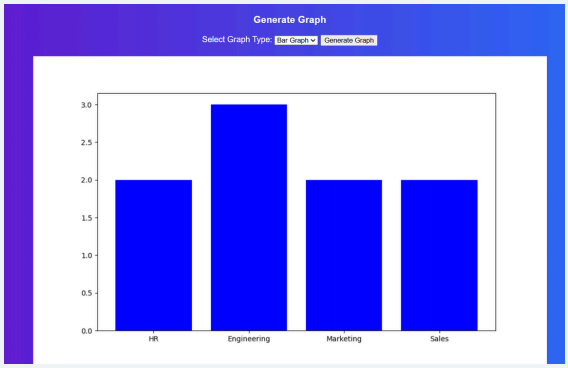
INPUT

```
SELECT department, COUNT(*)
FROM employees
GROUP BY department;
```

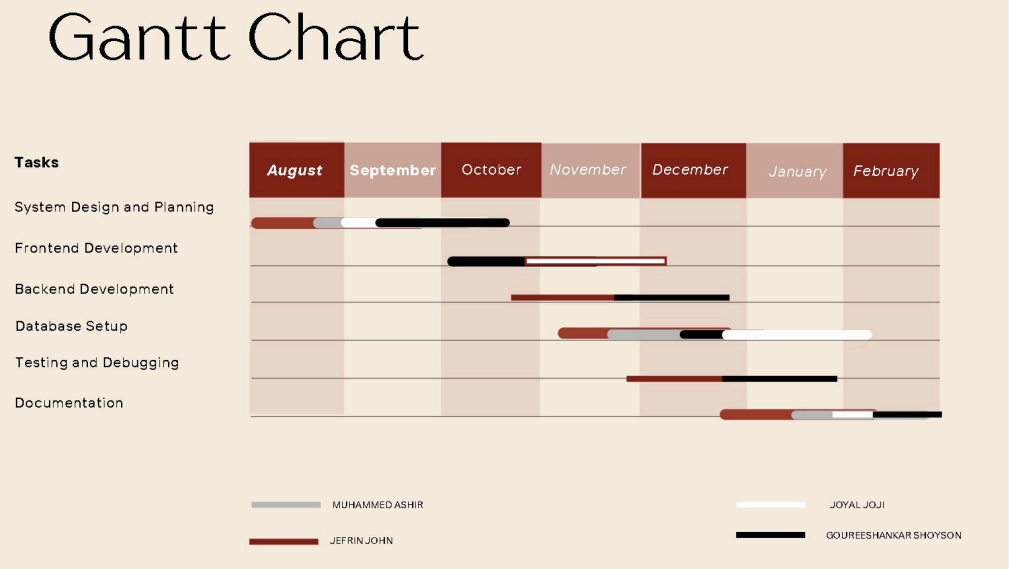
SQLify

OUTPUT

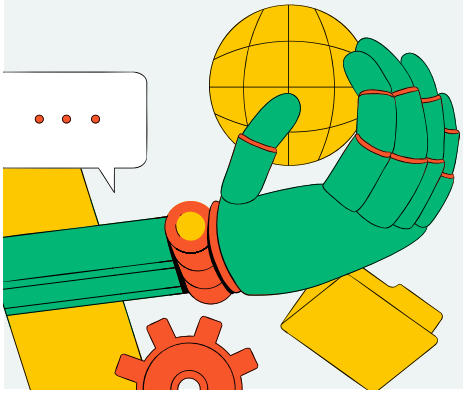




GANTT CHART



CONCLUSION



AIDCA simplifies the interaction with databases by providing a natural language interface, making data retrieval and management more accessible and efficient for all users. With its versatile support and real-time processing capabilities, AIDCA enhances productivity and democratizes access to valuable data insights.

REFERENCES

Research papers

- Enhancing database performance through SQL optimization, parallel processing and GPU integration
- CatSQL: Towards Real World Natural Language to SQL Applications

References

- SpaCy: Industrial-strength Natural Language Processing in Python
- Dialogflow: Conversational AI with NLU and ML

Appendix B: Vision, Mission, Programme Outcomes and Course Outcomes

Vision, Mission, Programme Outcomes and Course Outcomes

Vision:

To become a Centre of Excellence in Computer Science & Engineering, moulding professionals catering to the research and professional needs of national and international organizations.

Mission:

To inspire and nurture students, with up-to-date knowledge in Computer Science & Engineering, Ethics, Team Spirit, Leadership Abilities, Innovation, and Creativity to come out with solutions meeting the societal needs.

Program Outcomes (PO):

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSO):

PSO1: Computer Science Specific Skills: The ability to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas by understanding the core principles and concepts of computer science and thereby engage in national grand challenges.

PSO2: Programming and Software Development Skills: The ability to acquire programming efficiency by designing algorithms and applying standard practices in software project development to deliver quality software products meeting the demands of the industry.

PSO3: Professional Skills: The ability to apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs thereby evolving as an eminent researcher and entrepreneur.

Course Outcomes (CO):

CO1: Model and solve real world problems by applying knowledge across domains.

CO2: Develop products, processes, or technologies for sustainable and socially relevant applications.

CO3: Function effectively as an individual and as a leader in diverse teams and to comprehend and execute designated tasks.

CO4: Plan and execute tasks utilizing available resources within timelines, following ethical and professional norms.

CO5: Identify technology/research gaps and propose innovative/creative solutions.

CO6: Organize and communicate technical and scientific findings effectively in written and oral forms.

Appendix C: CO-PO-PSO Mapping

CO-PO and CO-PSO Mapping

	PO 1	**PO 2**	**PO 3**	**PO 4**	**PO 5**	**PO 6**	**PO 7**	**PO 8**	**PO 9**	**PO 10**	**PO 11**	**PO 12**	**PSO 1**	**PSO 2**	**PSO 3**
CO1	2	2	2	1	2	2	2	1	1	1	1	2	3		
CO2	2	2	2		1	3	3	1	1		1	1		2	
CO3									3	2	2	1			3
CO4					2			3	2	2	3	2			3
CO5	2	3	3	1	2							1	3		
CO6					2			2	2	3	1	1			3

3/2/1: high/medium/low

Mapping	Justification
CO3- PSO1	Students will be able to identify, analyze, and develop solutions for complex engineering problems across multidisciplinary areas while creating a presentation on an academic document
CO3-PSO2	Students will demonstrate the ability to apply the fundamentals of computer science in competitive research by designing a presentation based on an academic document
CO4- PSO1	Students will be able to to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas while giving a presentation about an academic document
CO4- PSO3	Students will have the ability to apply the fundamentals of computer science in competitive research while giving a presentation about an academic document
CO5- PSO1	Students will be able to to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas while preparing a technical report