

Fanfinder: An LLM-based Data Analysis Chatbot for Nonprofit Insights

From Natural Language Questions to Automated SQL Analysis

IN A PARK

1. Project Overview

- Project Name: LLM-driven Data Analysis Chatbot for Nonprofit
- Skill: Text-to-SQL, LLM (GPT-4o mini), LangChain, BigQuery
- Project Duration: 6 months (2024.10 - 2025.03)
- Achievement: Achieved 90% SQL generation accuracy on 100 test queries and 98% intent alignment precision.

2. Problem Statement

Non-profit organizations often possess valuable data but lack the analytical skills or SQL literacy required to extract insights. Consequently, their decisions remain report-driven rather than data-driven, preventing them from performing deeper, ad-hoc analyses. To address this limitation, we designed an automated question-to-report system powered by LLMs.

3. Project Goal

This project aims to develop an LLM-based automated data analysis system that allows non-technical users to ask questions in natural language and receive accurate, data-driven answers in real time.

3.1 Workflow overview

This project streamlines the process of transforming a user's question into a data-driven insight. It focuses on automating three key steps — interpreting intent, generating SQL queries, and delivering readable answers.

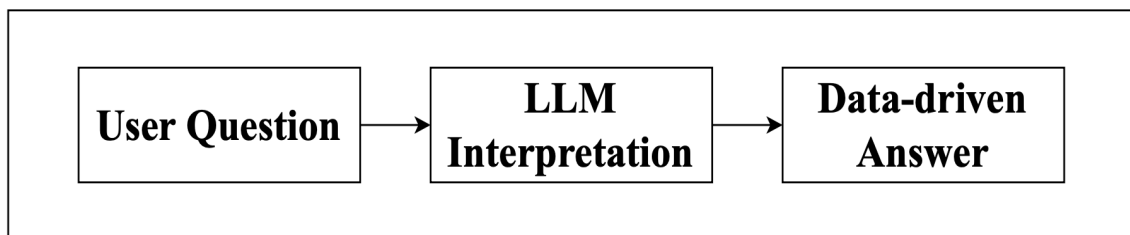


Figure 1. Simplified goal-centered workflow of the LLM-powered Text-to-SQL system.

4. Architecture

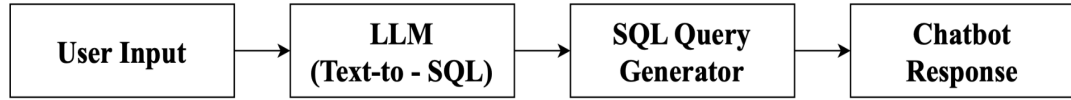


Figure 2. Overview of the Text-to-SQL process.

This diagram shows our system architecture. First, the user asks a question. LLM will convert the question into a query. Then it will execute on the database. Finally a result comes out.

To guide each stage of this process, the system uses designed prompts. A prompt is an instruction that tells an LLM what task it needs to perform. Since a user’s question alone often doesn’t provide enough context for the model, the prompt includes guidance such as the model’s role, the desired output format, and any constraints.

For example, when natural language is converted into SQL, the prompt might include “Convert the user’s question into an SQL query” or “Return only the SQL statement as the output.” The prompt clearly defines the LLM’s purpose. In this project, we have 4 types of prompts; User Intent Classification prompt, Table Classification prompt, Text-to-SQL prompt and Answer Generation.

The overall system pipeline consists of six sequential modules, from user input to final answer generation. Each module corresponds to a specific prompt type designed to guide the model’s reasoning process. The first step in this workflow is understanding what the user intends to ask — which is handled by the intent classification prompt.

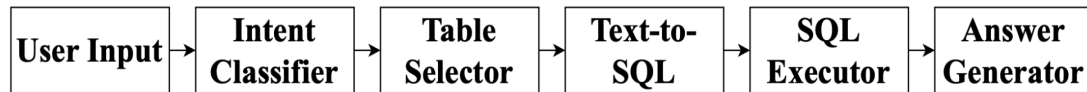


Figure 3. End-to-end pipeline of the Nuguna Reporter system. Each component represents a distinct prompt stage: intent classification, table selection, SQL generation, and answer construction.

1) User Intent Classification Prompt

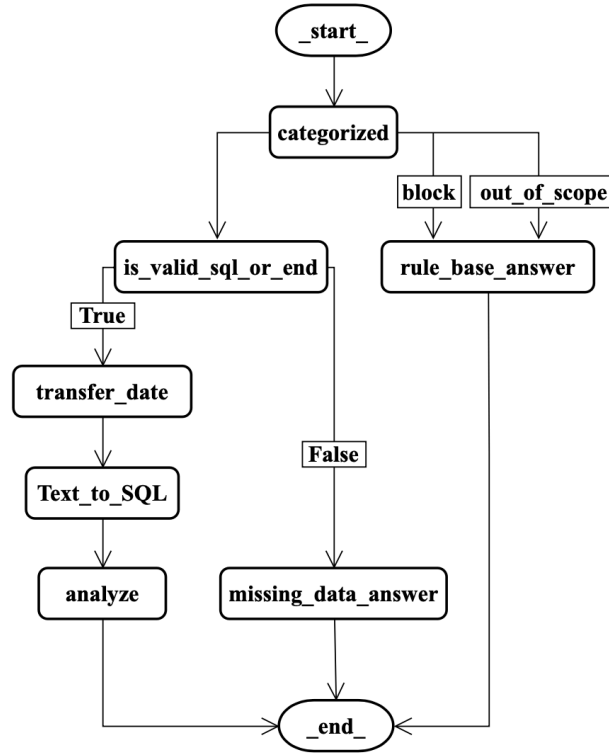


Figure 4. Workflow of user intent classification showing categorization, validation, and SQL conversion stages.

This prompt helps the LLM determine whether the question is valid or appropriate to answer. Once the user's question comes in, the LLM processes the question by the following step in the above diagram.

- categorized** : The LLM determines whether the question is allowed to be converted into an SQL query, blocked, or is outside of the chatbot's response scope. For instance, if the question contains abusive language, it is classified as either block or out_of_scope. Then the chatbot returns a rule-based answer, such as "This question is inappropriate to respond." This mechanism ensures that only valid and relevant questions proceed to the next step of processing.
- is_valid_sql_or_end** : If the question is answerable, it classifies it as True. Some questions can be processed as False, if the required element is missing. If a required element is missing, the system returns a 'missing_data_answer'. Required elements refer to key domain components, such as 'date' or 'metric', that must appear in the question. For example, if a user asks a question without specifying a date, the question will be classified as False, triggering a missing data response. This approach ensures that the chatbot can offer essential data before generating an accurate and contextually valid response.
- transfer_date - Text_to_SQL - analyze** : The question will be converted into an SQL statement using the date value and then the LLM will start analyzing.

2) Table Classification Prompt

The goal of this step is to select the most relevant database table for each user query. To achieve this, table information and few-shot examples are embedded in the prompt, allowing the LLM to determine which table should be used for SQL generation.

- Improving Classification Accuracy

The model leverages four complementary components to improve table selection reliability:

- a) Table descriptions: To help the LLM understand the database schema, each table is accompanied by a concise schema description, including a brief description of each column, a data type and, null values
- b) User question patterns: Understanding user query patterns is essential for anticipating common question types. Since the chatbot is designed for non-profit organizations, typical user questions often follow two main structures such as dimension (e.g., product) and metric (e.g., sales, revenue). Recognizing these patterns helps the model associate relevant metrics with their corresponding tables.
- c) Analysis scenario: Non-profit data analysts tend to explore data through follow-up questions. To capture this behavior, we incorporated realistic scenarios provided by a non-profit client, Nuguna. These examples helped the model understand practical question patterns, such as:
 - Do you often extract this data during analysis?
 - Have you seen any improvements in operations or performance as a result of this analysis?
- d) Main information: Each table includes “main information” tags that define its key attributes. For example, if a user asks the following two questions.
 1. “What product categories did customers order?”
→ The LLM focuses on the keyword category and identifies the Product table as relevant.
 2. “What’s the average delivery time for the products ordered?”
→ The model identifies delivery time as the key variable and links it to the Order table.

```
from langchain_core.prompts import PromptTemplate

prompt = PromptTemplate.from_template("""
You are an assistant that determines which database table best matches a user's question.

Context:
- Three available tables with predefined schemas
(e.g., event_report, page_report, source_report)
- Each table includes columns such as dates, events, and user attributes.

Your task:
1. Identify the key entities or metrics mentioned in the user's question.
2. Select ONE table that contains those entities.
3. Respond only with the table name.

Example:
Question: "Which page had the most visits last month?"
Output: page_report

{input}
""")
```

Listing 1. Simplified table-classification prompt

- 3) Text-to-SQL Prompt : This prompt is divided into two parts — a system prompt and a user prompt.
- System Prompt : It defines the chatbot’s core personality, tone, and operational behavior. It specifies parameters such as the model’s role (“SQL analyst assistant”), output format, and reasoning constraints to maintain consistent and interpretable query generation.
 - User Prompt : It provides detailed instructions on how to convert user questions into query statements. It instructs the model on how to identify key metrics, determine relevant columns, and construct SELECT–FROM–WHERE clauses based on user intent.

The Text-to-SQL module uses a dual-prompt structure consisting of a system and a human prompt. The system prompt defines query constraints and output rules, while the human prompt provides schema context and user-specific details for SQL generation.

```
SYSTEM_REPORT_TEMPLATE = """
You are an agent that generates SQL queries for a BigQuery database.
- Use the table name {TABLE_NAME}.
- Only include relevant columns, not all columns.
- Double check for common SQL errors before execution.
- Return the SQL query in one line without markdown or SQL tags.
"""

HUMAN_REPORT_TEMPLATE = """
Given the following schema:
Table: event_report
Description: A table containing information about user events and their interactions
columns :
- event_date (STRING): The date when the event occurred (in YYYYMMDD format) e.g., '20250106'
- event_bundle_sequence_id (INTEGER): The unique identifier for a bundle of events. Tracks the
sequence of events that occurred in the same user session e.g., -1402952294
- event_name (STRING): Column representing specific event names. e.g., 'pageview', 'click',
'purchase'

Write a SQL query to answer the user question below.
If no date is specified, assume yesterday as default.
User question: {input}
SQL query:
"""
```

Listing 2. Simplified Text-to-SQL prompt

4) Answer Generation Prompt : This prompt standardizes the format and tone of the chatbot’s final responses. For example, when provided input in the form “A is B,” the model outputs the result according to a fixed template, ensuring stylistic and structural consistency across different queries.

```

You are an AI assistant specialized in generating data-driven responses.
Based on the user query (input), executed SQL (sql), and returned data (data),
determine the most appropriate response type and generate a clear, concise answer.

Original Question: {input}
Executed SQL: {sql}
Query Result: {data}

Response Generation Guidelines
1. Identify the key metrics (e.g., visitors, conversion rate, revenue) and dimensions
(e.g., campaign, channel, landing page) from the question.
2. Reflect any time or date range mentioned in the query.
3. Determine the appropriate response type and apply one of the following formats:
  - Lookup: Retrieve a single value
  - Comparison: Describe increase/decrease or difference
  - Ratio: Calculate percentage or share
4. Provide a natural fallback phrase if no data is available.
5. Use a polite, conversational tone and include suitable emojis for readability.
6. Suggest 1-2 relevant follow-up questions to encourage continued analysis.

Response Format Example
[Date] [Source/Channel]’s [Metric] result is [Value].
Follow-up suggestion: “Would you like to compare it with another period?”

Sample Response
On December 1, 2024, the 2023GHR campaign attracted 111 visitors.
Follow-up suggestion: “Would you like to compare visitor numbers from other dates?”

```

Listing 3. Simplified Answer Generation prompt

5. QA

To verify the stability and logical consistency of the generated queries, a QA framework was implemented.

1) Automated QA Pipeline

Manual validation of SQL outputs was time-consuming and error-prone. To ensure reproducibility and scalability, I automated the QA pipeline using Google Spreadsheet API for data retrieval and OpenAI API for model interaction.



Figure 5. Automated QA pipeline integrating Google Sheets and OpenAI API.

2) Implementation

a) Google Spreadsheet API

The Google Spreadsheet API was used to automate the collection of test cases. Each row in the spreadsheet represents a query–response pair, including the original user

input, expected SQL output, and validation result. This setup allowed the QA pipeline to dynamically retrieve test inputs and log evaluation outcomes, ensuring that all test data remained synchronized and reproducible across experiments.

b) OpenAI API

The OpenAI API served as the model interface for automated Text-to-SQL validation. The system programmatically sent each test input from the spreadsheet to the model, collected the generated SQL query, and compared it against the expected output. This integration reduced manual verification time and enabled large-scale regression testing to evaluate prompt stability and SQL accuracy.

```
def get_openai_response(user_input):
    response = client.chat.completions.create(
        model = "gpt-4o-mini",
        messages = [
            {"role": "system", "content": prompt},
            {"role": "user", "content": user_input}
        ]
    )
    return response.choices[0].message.content.strip()
```

3) Validation Framework

- a) SQL Syntax Validation : Ensures that the generated SQL statements are syntactically correct. We checked filter logic, aggregate functions, and complex CTEs to confirm that the queries execute successfully without errors.
- b) Intent Alignment : Verifies that the LLM understands the user’s question accurately. For example, for the query “Which channel had the highest number of acquisitions in November 2024?”, the model must recognize the date range, metric, and ranking intent.
- c) Business Logic Consistency : Checks whether the queries reflect domain-specific KPI definitions (e.g., donation performance) and use the appropriate tables and columns.

4) Results and Insights

- 1. Quantitative Outcomes : Through the automated QA pipeline, a total of 100 Text-to-SQL test cases were evaluated. The model achieved 90% SQL syntax accuracy and 98% intent alignment precision.
- 2. Error Analysis : To identify areas for improvement, we classified all failed SQL generations into 6 categories. The following table summarizes representative issues and their descriptions.

Category	Content
Temporal Reasoning Error	The model struggled with interpreting date ranges and aggregating period-based queries (e.g., “this month,” “last week”).
Null Value	In cases where numerical calculations involve null values, the query incorrectly returns null instead of preserving the valid numeric value
Over-generated Response	The model occasionally generates answers that exceed the intended scope of the user’s question

Restriction	The model automatically inserted a LIMIT clause into the SQL query, restricting the number of returned rows even though the user did not specify any output constraint.
Metric Calculation	The model incorrectly calculated key performance indicators by applying the wrong aggregation logic
Table selection	Incorrect mapping table

Summary Insights:

- The most frequent issues involved temporal reasoning error (50%) and metric calculation (40%).
- These patterns indicate that the model's weaknesses were concentrated in context-dependent date logic and limited exposure to domain-specific data.
- Prompt refinement reduced the overall error rate by 20% in the final phase.
- The categorized error analysis served as a foundation for iterative prompt tuning and error-aware evaluation in subsequent development cycles.

6. Final Result

The interface below shows the final implementation of the LLM chatbot. Users can ask questions in natural language, and the system automatically generates and executes SQL queries to deliver both textual and visual insights. The chatbot also supports follow-up questions, allowing continuous exploration of nonprofit data through conversational analysis.



Figure 6. Final chatbot interface converting user queries into automated SQL analysis and visualized results.

7. Reflection

Through this project, I found that building an effective LLM-driven system is not merely a technical task but a process of structured thinking. Designing questions for the model taught me the importance of framing - good analysis begins with good questions. By translating ambiguous user needs into clear, structured prompts, I learned how to think systematically about human intent and data representation.

I also gained a deeper understanding of the strengths and limitations of large language models. It became evident which parts of reasoning can be reliably delegated to the model and which still require human interpretation and control. This boundary awareness guided how I designed prompt structures and validation layers.

Finally, through numerous iterations of success and failure, I developed a habit of documenting experiments and tracing outcomes. Recording each test result - including causes of failure, prompt versions, and manual notes - provided valuable insight into prompt optimization and reproducibility. This disciplined approach to experimentation became one of the most meaningful lessons from the project.