NORP LLM Chatbot Evaluation Report

Chetan Reddy Bojja
chetanreddy.b@gatech.edu

Chathurvedhi Talapaneni
ctalapaneni3@gatech.edu

November 30, 2024

We declare that we allow future students to utilize the project deliverables in future courses.

We extend our heartfelt gratitude to Professor Calton Pu, Anmol Agarwal and Desiree Dominguez for their continuous guidance and support throughout the course of this project.

Contents

1 Introduction

Nonprofit Organization Research Panel(NORP) is a university-based project researching nonprofit organizations in the United States. It is a collaborative project with computer and social scientists working on analyzing the data from several nonprofits. The social science research on NORP helps understand the trends in specific areas of work such as human service or arts and culture. Information on income size, administrative efficiency, annual budget, and sources of external funding for different types of nonprofits. The NORP panel invites various nonprofits around the United States and information is gathered using periodic surveys. The members who have participated in the panel have access to NORP analysis and these studies are published to professionals in the nonprofit field to grow further. NORP is a collaborative project between Georgia Institute of Technology, George Mason University, American University, and Urban Institute

1.1 Motivation

Social scientists often struggle with extracting the data and insights from the databases as they do not have the required SQL expertise. The current tools and framework of the database make it non-user-friendly for users without the necessary technical skills. This project aims to tackle this problem to enable social scientists to seamlessly extract the necessary information and insights from the database.

1.2 Our Approach

To deal with this challenge, we developed a system that translates a natural language query from the user to the necessary SQL command using popular Large Language Models(LLMs). Our project implements several prompt engineering methods on multiple LLM models to provide a comparative analysis of the LLM models available to us. This would make accessing the database as simple as requesting a simple worded query and the LLM model can return the necessary SQL query to extract the necessary data from Metabase and present visualizations. Our solution will improve the accessibility of the database for the researchers with simple database visualizations using natural language while maintaining correctness and efficiency.

2 Methodology

2.1 Action Plan

1. **Utilize LLMs:** Transform natural language queries into SQL using annotated datasets.
2. **Prompt Engineering:** Engineering the user input queries to improve the output generation for the LLM. Input user query enhanced with the database schema details for better context.
3. **Feedback Mechanism:** Implement an iterative feedback mechanism to improve the query quality by providing specific context.
4. **Evaluation:** Evaluate the SQL generation with NLP-based metrics along with manual evaluation on Metabase.

2.2 Goal

To select an efficient LLM model with sufficient precision in SQL query generation and necessary usability for the research scientists.

3 Tools and Technologies

3.1 LLMs Utilized

- Gemma1 7B
- Gemma2 9B
- Mistral 7B
- LFM 40B MoE
- LLaMA3 8B
- Phi-3 3.8B

3.2 Infrastructure

- Public-facing APIs and deployed open-source models on the PACE cluster.

3.3 Frameworks and Tools

- Hosted LLMs through open-source APIs.
- Executed generated SQL queries on Metabase for performance evaluation and visualization.
- Utilized SQL parsers and frameworks to validate database queries.

4 Prompt Engineering - An Overview

What is Prompt Engineering?

Prompt engineering is the process of structuring a user input to guide an LLM by providing the necessary context to improve the interpretation of the question and generate better outputs. It may involve phrasing the query in a different manner, providing context, specifying a certain style required, or assigning a role to the LLM. There are several such approaches to implement prompt engineering and the following approaches are the ones implemented in this project.

Approaches

- **Zero-Shot:** Simple query with none to minimal context
- **One-Shot:** Simple query along with one example for better context
- **Few-Shot:** Simple query with several examples for improved context

Resource Used

`www.promptingguide.ai`

5 Zero-Shot vs One-Shot vs Few-Shot

5.1 Zero-Shot Prompting

- **Prompting:** Providing only the natural language query from the user
- **Example:** "Show total shootings by month."
- **Challenges:** Minimal context leading to very inaccurate SQL commands

5.2 One-Shot Prompting

- **Prompting:** Providing the natural language query along with the schema details necessary
- **Example:** "Show total shootings by month. Schema: Sales (month, total)."
- **Outcome:** Significantly improves the generated SQL commands but still struggles in edge cases

5.3 Few-Shot Prompting

- **Prompting:** Providing the natural language query, the schema details, and a few row examples for the schema
- **Example:** "Show total shootings by month. Schema: Sales (month, total). Example rows: ('Jan', 1000), ('Feb', 2000)."

- **Outcome:** Best performing technique of the three. The examples given improve the query generation as the information as to how the data is stored in the column is present.

5.4 Performance Comparison

- **Zero-Shot:** Fastest but terrible accuracy
- **One-Shot:** Significant improvement over Zero-Shot
- **Few-Shot:** Overall best results from the three chosen techniques.

6 Evaluation Metrics

6.1 NLP Metrics

- **BLEU:** Evaluates the precision of the sequence in the generated queries. The metric signifies the number of contiguous sequences of words in the generated output that match the contiguous sequence of words of the reference text. The score ranges from 0 to 1, with the higher value signifying closer text.
- **ROUGE:** Evaluates the recall of the sequence in the generated queries. The metric signifies the number of contiguous sequences of words in the reference text that match the contiguous sequence of words of the generated output. The score ranges from 0 to 1, with the higher value signifying closer text.
- **Levenshtein Distance:** Evaluates the difference between the generated output and the reference text by comparing them as two strings. The metric counts the number of edits(substitutions, deletions, or insertions) to be made to convert one string to another. This metric is modified to provide the Levenshtien similarity which ranges from 0 to 1 with the higher value signifying closer text.
- **Jaccard Similarity:** Compares the generated and reference text by tokenizing them into words or phrases and measures the size of the intersection over the size of the union.
- **Cosine Similarity:** Converts the generated and reference text into their vectors and measures the cosine of the angle between the two vectors in a multi-dimensional space. Ranges from -1 to 1, where 1 means identical, 0 means completely dissimilar, and -1 means opposite meaning.

6.2 Semantic Parsers

- Utilized to check the structure of the generated SQL queries.
- Makes sure that the generated query aligns with the intended schema and logic
- Utilized in removing segments of the generated output to separate the necessary SQL command from the LLM output.

6.3 Metabase Evaluation

- Execute generated SQL queries on the database using Metabase.
- Compare the results of generated queries against the expected outputs.
- Direct evaluation of the usability and the precision required to implement the LLM in the workflow.

7 Evaluation

7.1 Performance Analysis

The LLM models were evaluated over two datasets:

- Simple: Consisted of simple SQL command request
- Complex: Requesting more involved SQL commands with information retrieved from several columns in different forms

The following two tables represent the performance of the LLM models chosen over the NLP metrics described in the previous section.

Model	Levenshtein Similarity	Jaccard Similarity	Cosine Similarity	BLEU Score	ROUGE-1	ROUGE-2	ROUGE-L
LFM MoE	0.217	0.165	**0.129**	**0.026**	**0.518**	**0.305**	**0.462**
Gemma	**0.251**	**0.166**	0.122	0.016	0.515	0.291	0.460
Phi-3	0.112	0.076	0.070	0.008	0.319	0.176	0.259
Gemma 2	0.097	0.067	0.076	0.006	0.296	0.156	0.244
Mistral 8x7B	0.102	0.070	0.057	0.004	0.304	0.164	0.244
Llama 3.2	0.120	0.020	0.021	0.001	0.147	0.054	0.123

Table 1: Performance Metrics for Various Models (Simple Queries)

Model	Levenshtein Similarity	Jaccard Similarity	Cosine Similarity	BLEU Score	ROUGE-1	ROUGE-2	ROUGE-L
LFM MoE	0.094	**0.117**	**0.179**	**0.008**	**0.451**	**0.217**	**0.332**
Gemma	**0.122**	0.112	0.143	0.006	0.427	0.203	0.328
Phi-3	0.071	0.069	0.126	0.005	0.357	0.153	0.237
Gemma 2	0.061	0.076	0.109	0.007	0.366	0.162	0.238
Mistral 8x7B	0.066	0.071	0.093	0.006	0.346	0.161	0.229
Llama 3.2	0.094	0.021	0.033	0.001	0.128	0.045	0.096

Table 2: Performance Metrics for Various Models (Complex Queries)

Model	Zero Shot	One Shot	Few Shot
LFM MoE	Worse	Average	Good
Gemma	Worse	Average	Good
Phi-3	Worse	Poor	Average
Gemma 2	Worse	Poor	Average
Mistral 8x7B	Worse	Poor	Poor
Llama 3.2	Worse	Worse	Poor

Table 3: Prompt Engineering Performance for Various Models

Observations:

- Gemma1 7B and LFM MoE 40B models are the best-performing models out of the LLM models chosen
- Out of the chosen models, LFM MoE model had shown better performance in programming tasks and has the corresponding best performance in the generation of the SQL queries
- The Llama3 8B model shows significantly poor performance in comparison to the other models chosen which was unexpected.
- Before the analysis, we expected Gemma2 9B to outperform Gemma1 7B and also take longer to generate the responses. Contrary to out expectation, it is shown that Gemma1 model outperforms the Gemma2 model and we can also see from the next section, Gemma2 also takes longer to generate the responses when compared to Gemma1

7.2 Latency Analysis

We perform the latency analysis comparison of Gemma1, Gemma2, and LFM over the entire dataset of 88 simple query data points and 40 complex query data points:

- **Gemma 1:** 296.7061 seconds
- **Gemma 2:** 430.8373 seconds
- **LFM:** 683.7302 seconds

We can see that along with Gemma2 showing worse performance in comparison to Gemma1, it also takes longer to generate the response which was unexpected. The LFM model although performs better than the Gemma1 model by a slight amount, takes more than double the time to generate the response.

We also show the average latency comparison over the LFM and Gemma1 models over the different complexities of the queries.

Model	Simple(88)	Complex(40)	Complete(128)
LFM	5.04	6.32	5.44
Gemma1	2.08	2.77	2.29

Table 4: Comparison of average latency for response generation

7.3 Performance vs Latency Analysis over Query Complexity

The below graph determines the performance(Levenshtein similarity) vs latency analysis choosing the Gemma1 model as the evaluation model.

Observations:

- It is seen that most simple query latency is less than equal to 3 seconds.
- The complex query latency is generally higher and also has a few taking up to 7 seconds.
- It is also noticed that most of the red points(complex queries) have lower Levenshtein similarity scores with all of them less than equal to 0.3 and the blue points(simple queries) reach up to 0.7 similarity score.

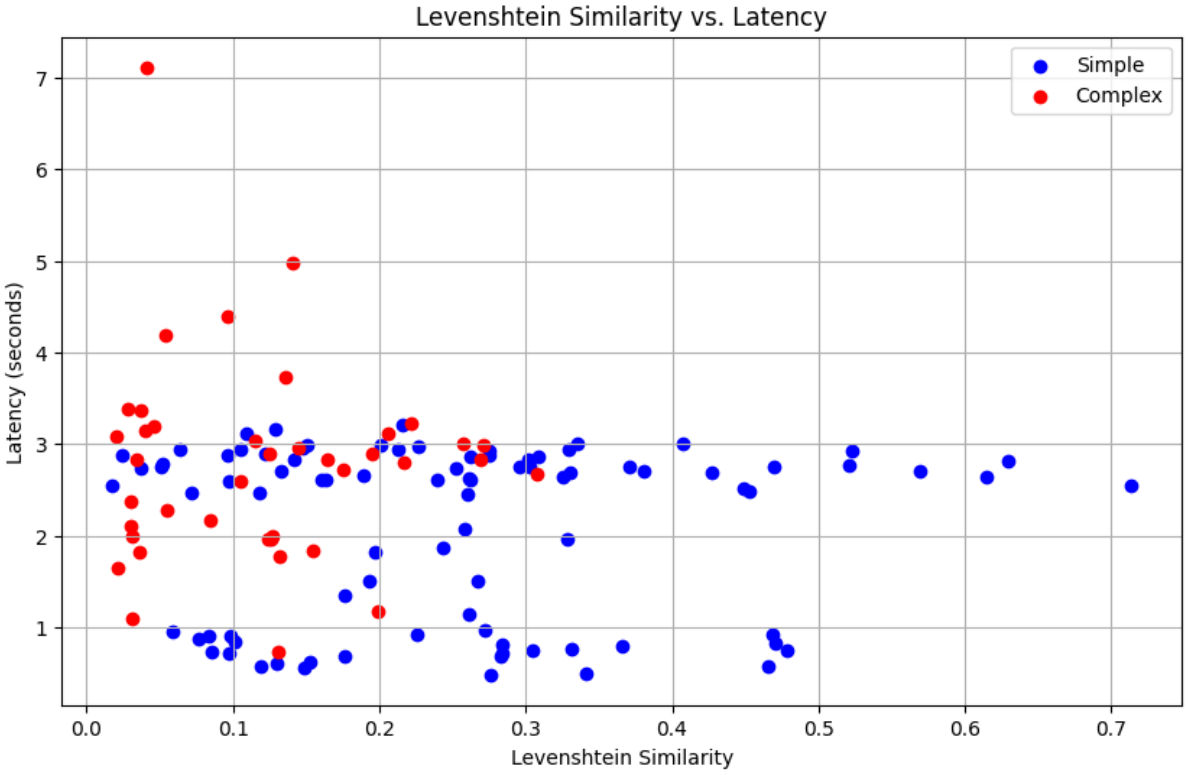

Figure 1: Levenshtein Similarity vs. Latency for Gemma 1

The below graph is the performance(ROUGE) vs latency analysis choosing the Gemma1 model as the evaluation model.

Observations:

- In terms of the ROUGE metric, it is harder to highlight the differences between the simple and complex queries.
- However, there are a few outliers in simple queries with more than 0.9 score and in complex queries with less than 0.2 score showing that few simple queries are generated very accurately.

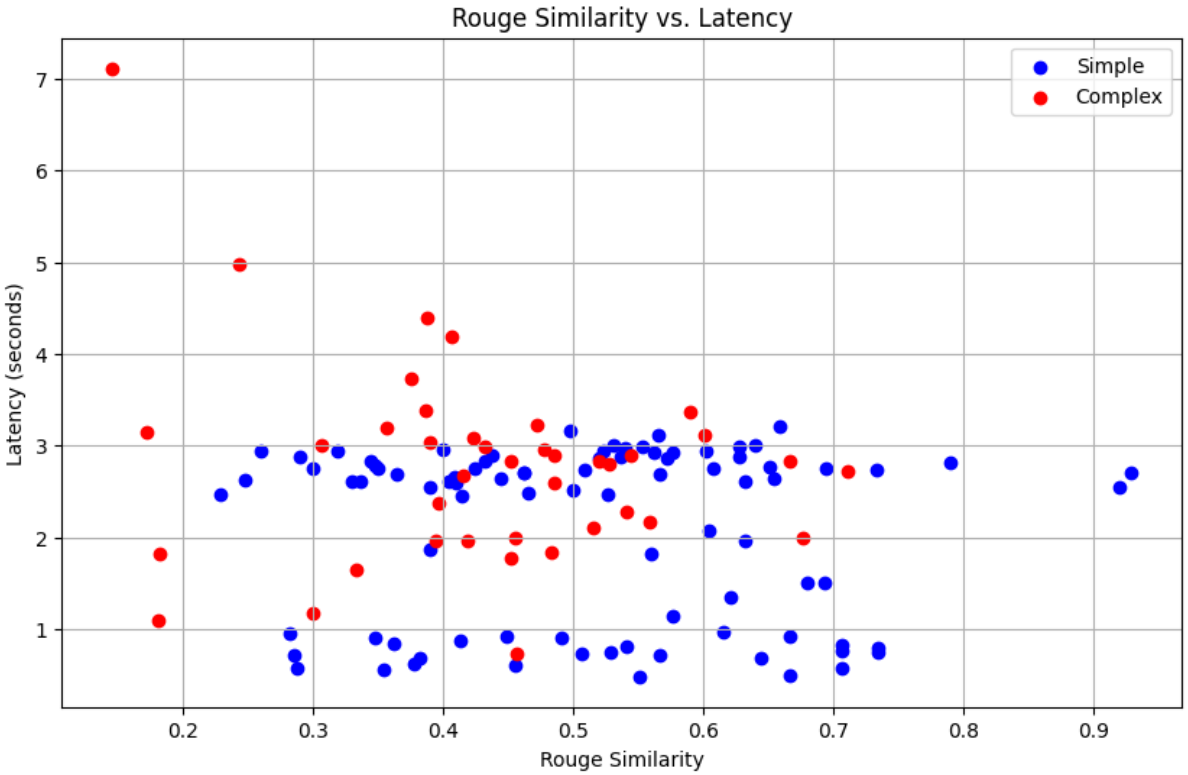

Figure 2: ROUGE Similarity vs. Latency for Gemma 1

7.4 Performance vs Latency Analysis over Model

We also perform the analysis to understand the difference between the two best performing models chosen in this project: LFM and Gemma1

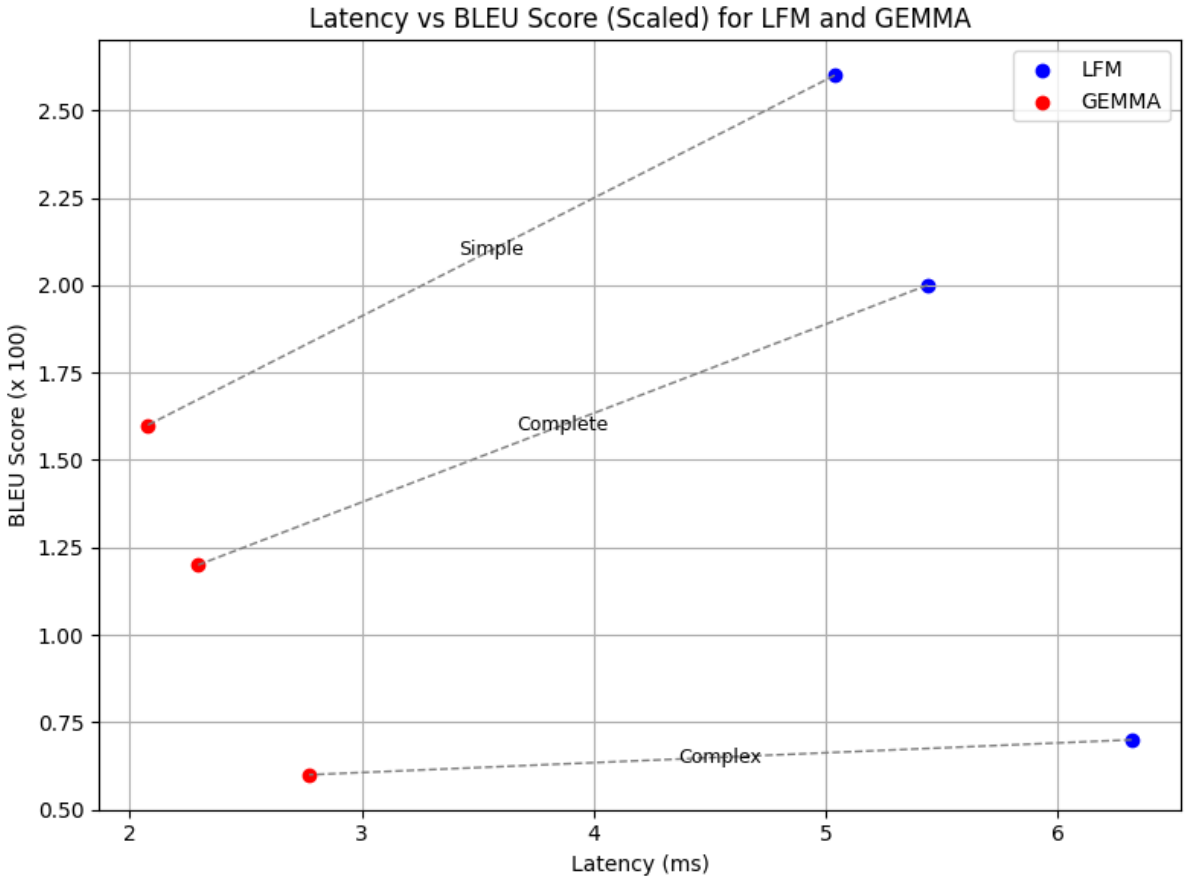

Figure 3: Latency vs BLEU Score (Scaled) for LFM and Gemma

Observations:

- The difference between the models over simple queries is quite clear with LFM taking longer and generating better responses.
- The interesting observation is that although the latency on complex queries goes from 2.8s to 6.3s going from Gemma1 to LFM, the performance barely increases. This shows that both the models perform particularly badly on the complex queries.

7.5 Metabase Evaluation

We evaluated the accuracy of generated SQL queries against the Metabase database.

7.5.1 Zero-Shot Prompting

Prompt:

"Give the total count of each crime in LA."

```
SELECT crime_category, COUNT(*) FROM la_crime_data;
```

Figure 4: Generated SQL Query: Zero-Shot Prompting

```
SELECT
    la_crime_data.Crime_Type AS Crime_Type,
    COUNT(*) AS count
FROM
    la_crime_data
GROUP BY
    la_crime_data.Crime_Type
ORDER BY
    la_crime_data.Crime_Type ASC;
```

Figure 5: Ideal SQL Query: Zero-Shot Prompting

Performance: Zero-Shot prompting resulted in an incomplete SQL query. This lacked grouping and ordering causing it to be incomplete against the Metabase.

7.5.2 One-Shot Prompting

Prompt:

> "Give the total count of each crime in LA. Schema: shootings (id, crime_type, date, location)."

```
SELECT
    crime_type,
    COUNT(id)
FROM
    la_crime_data
GROUP BY
    crime_type;
```

Figure 6: Generated SQL Query: One-Shot Prompting

```
SELECT
    la_crime_data.Crime_Type AS Crime_Type,
    COUNT(*) AS count
FROM
    la_crime_data
GROUP BY
    la_crime_data.Crime_Type
ORDER BY
    la_crime_data.Crime_Type ASC;
```

Figure 7: Ideal SQL Query: One-Shot Prompting

Performance: One-Shot prompting improved the structure over zero shot prompting by including schema details. But this still lacked ordering resulting in a bad SQL query.

7.5.3 Few-Shot Prompting

Simple Query Prompt:

"Give the total count of each crime in LA (with attached schema and top rows)"

```
SELECT
  Crime_Type,
  COUNT(*) AS TotalCount
FROM
  la_crime_data
GROUP BY
  Crime_Type;
```

Figure 8: Generated SQL Query: Few-Shot Prompting (Simple Query)

```
SELECT
    la_crime_data.Crime_Type AS Crime_Type,
    COUNT(*) AS count
FROM
    la_crime_data
GROUP BY
    la_crime_data.Crime_Type
ORDER BY
    la_crime_data.Crime_Type ASC;
```

Figure 9: Ideal SQL Query: Few-Shot Prompting (Simple Query)

Performance: Few-Shot prompting was able to generate the ideal query without unnecessary complexity. This shows LLMs perform well with straightforward queries with simple structures.

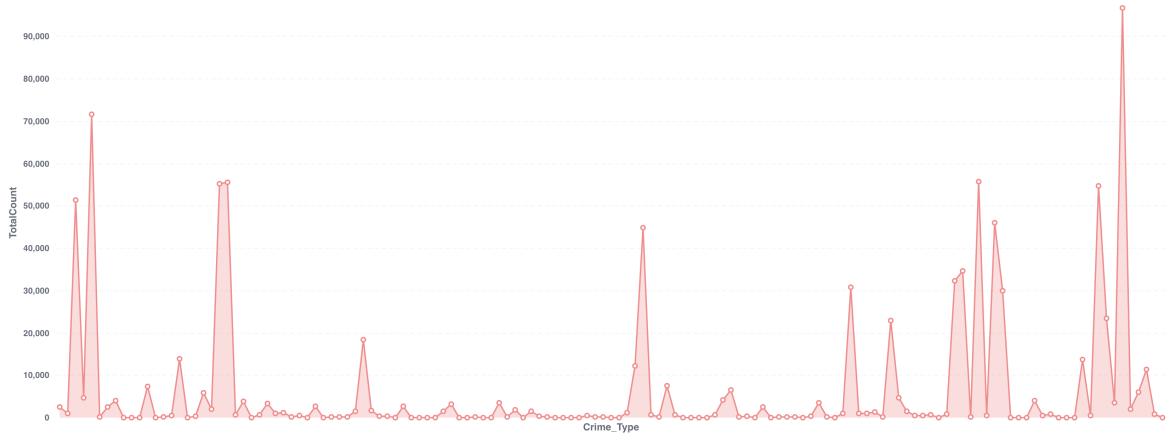

Figure 10: Metabase Generated SQL Query: Few-Shot Prompting (Simple Query)

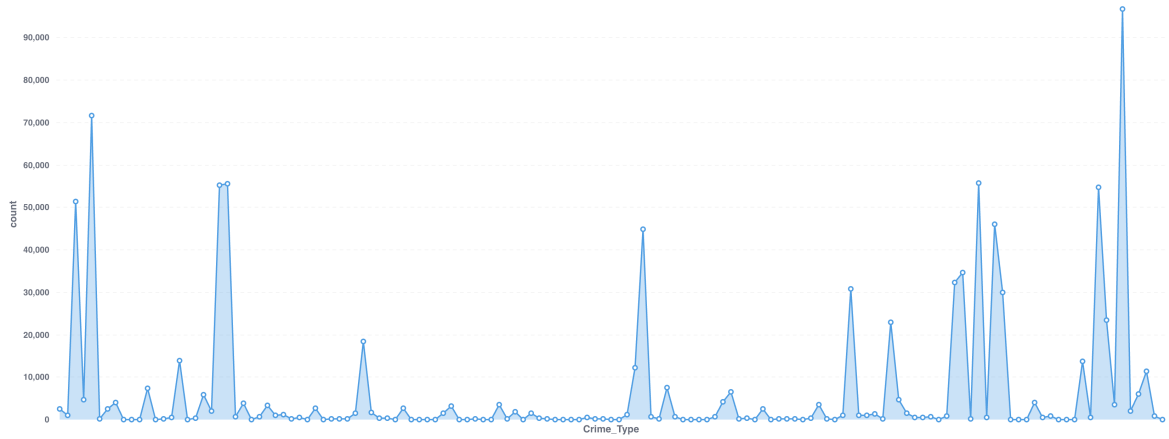

Figure 11: Metabase Ideal SQL Query: Few-Shot Prompting (Simple Query)

Complex Query Prompt:

"Give count of each crime type for each hour of the day in LA (with attached schema and top rows)"

Performance:

- LLM Limitations: The LLM limitations are noticeable with the use of complex SQL operations like joins, CASE statements and queries that need knowledge of the data.
- Lack of Semantic Understanding: LLMs lack a deep understanding of underlying data values or schema relationships. Due to this LLMs are unable to generate accurate and contextually appropriate queries for complex scenarios.
- Semantic vs. Syntactic Proficiency: LLMs excel at producing syntactically correct queries with the help of schema and example rows. But the issue is that they struggle with semantic comprehension and data-specific logic necessary for more advanced data manipulations.

```sql
SELECT
    Crime_Time,
    Crime_Type,
    COUNT(*) AS count
FROM la_crime_data
WHERE Area_Name = 'LA'
GROUP BY
    CAST(Crime_Time as TIME),
    Crime_Type
ORDER BY
    CAST(Crime_Time as TIME);
```

Figure 12: Generated SQL Query: Few-Shot Prompting (Complex Query)

```sql
SELECT
    FLOOR(lcd.Crime_Time / 100) + 1 AS Hour,
    CASE
        WHEN lcd.Crime_Type IN ('ASSAULT WITH DEADLY WEAPON', 'AGGRAVATED ASSAULT'
        -- Additional WHEN conditions omitted for brevity
        ELSE 'Other'
    END AS Crime_Type_Grouped,
    COUNT(*) AS Count
FROM
    la_crime_data lcd
GROUP BY
    Hour,
    Crime_Type_Grouped
ORDER BY
    Hour ASC,
    Crime_Type_Grouped ASC;
```

Figure 13: Ideal SQL Query: Few-Shot Prompting (Complex Query)

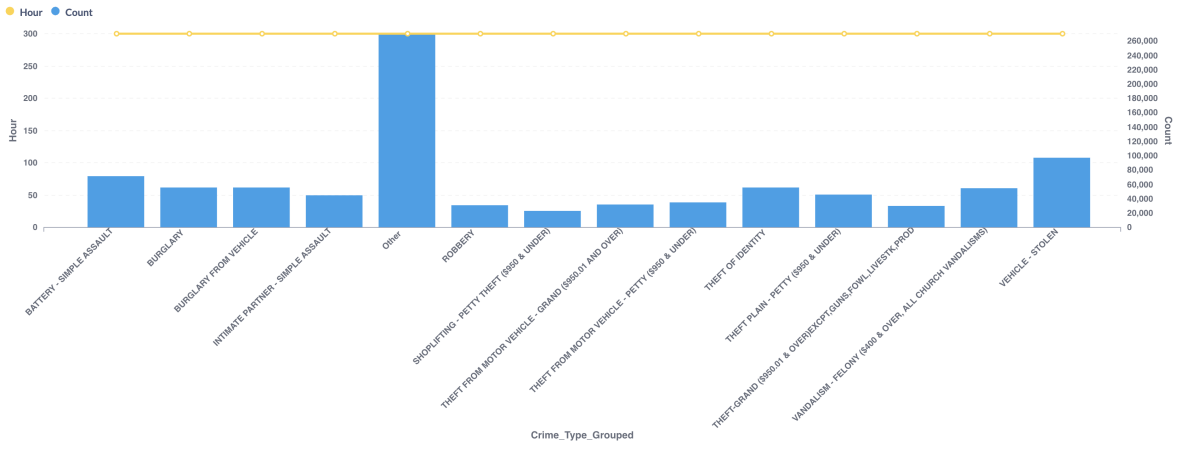

Figure 14: Metabase Ideal SQL Query: Few-Shot Prompting (Complex Query)

8 Challenges and Learnings

8.1 Challenges

- For analyzing the performance of the LLMs, we utilized NLP-based metrics which measure the similarity of sentence structure between two text segments. However, this metric is not appropriate in evaluating SQL correctness.
- Although we use few-shot engineering to provide context in terms of input query, the LLMs still struggle as they lack the necessary knowledge about the dataset entries and the type of request specific to the database.
- LLMs tend to struggle on complex queries even on few-shot prompting and require iterative refinement to achieve the desired response. This takes significantly more time and effort to implement in a practical sense.

8.2 Learnings

- The impact of prompt engineering is very evident in comparing zero-shot and few-shot training and shows the importance of context and priming the LLM for the necessary task at hand.
- It is seen that models which perform well on coding tasks such as LFM, also perform better on generating SQL queries. This can help in the search for better models and fine-tuning methods that can be used in the future directions of this project aiming at developing an LLM designed for SQL queries on Metabase.
- Although it is unfeasible in the current context of the aim of this project, the iterative feedback mechanism drastically improves the quality of the SQL response. The information that the LLM model seems to lack can provide insights into how the prompts can be engineered further to get better results in one query response.

9 Limitations and Future Direction

9.1 Limitations

- The models we have utilized in this project are all general-purpose models available on a public-facing API. The non-specialized model along with the limited annotated dataset prevents fine-tuning for schema-specific tasks.
- The is a clear need for a more appropriate metric for SQL evaluation than NLP-based metrics such as BLEU and ROUGE.

9.2 Future Directions

- **Advanced Prompt Engineering:** The goal of the project was to ascertain the performance of different LLMs, so we only chose fundamental prompt engineering techniques. Based on the results, the effect of prompt engineering techniques is apparent and future directions can look at better prompt engineering techniques such as incremental and schema-specific examples to improve accuracy.

- **Evaluation Metric:** A metric to evaluate the SQL commands is necessary and taking a look at other programming task evaluation metrics used in popular LLM training methods can be used as a starting point.
- **Specialized Models:** Using specialized models such as SQLCoder or fine-tuning programming focused LLMs while they are hosted on the PACE cluster can improve the performance as the model can be fine-tuned not only for SQL but also the databases on Metabase.
- **Iterative Refinement:** The preliminary understanding of what context must be sent to an LLM can be obtained from experimenting on multiple iterations on an LLM. The intermediate responses can provide information on what segments the LLM struggles with.

10 Conclusion

The project evaluates several LLM models such as Gemma, LFM, LLama, Phi, and Mistral for SQL query generation. Although this project evaluates each of these models and presents the best option for SQL generation, the obtained precision is still far off from the required precision to be integrated into the existing workflows. General-purpose LLMs cannot fully address the goal of generating precise SQL queries. This project is a good base to start off to understand what to consider while fine-tuning an LLM specific for SQL generation for the NORP database. Future advancements in dataset size, specialized models, and improved prompting techniques are key to overcoming the limitations shown in the project and achieving better SQL generation capabilities.

11 Skill Learning

In this project, we developed skills in 3 levels: L1, L2, and L3. We learned both general-level skills and technical skills. These skills are useful to us both at the research level and industry level once we graduate. These skills enabled us to complete this project successfully.

11.1 L1 Skills

General Skills

- **Reading and Comprehension:** Reading, making sense, and extracting useful information for the project from research papers, prompt engineering guides, and metabase documentation helped in guiding the project forward. Going through these papers and resources gave us an idea and structure for writing a clear and concise final project report.
- **Logical Reasoning:** We understood structuring the prompt to the LLM to generate the required SQL query from natural language input. This helped us in developing the logical reasoning to map the user intentions to correct SQL commands.

- **Fact vs. Opinion:** During this project, we faced many preconceived notions and assumptions. For example we assumed Gemma2 would perform better than Gemma1 in this SQL query generation task but with higher latency. However the latter did better during the project, proving that you have to believe in data-driven insights over subjective opinions.

Technical Skills

- **Programming Techniques:** We used Python mainly in this project to integrate various parts of this project together like running Large Language Models (LLMs), validating the generated SQL queries and computing the metrics. We automated the whole pipeline to generate the SQL query for various datasets (& simple and complex parts of it) and then metric computation. This saved time and reduced the manual effort & scope of errors. We were able to use this saved time to test our queries on Metabase and compare the models.

- **Tool Familiarity:** We learned how to integrate and use the open-source APIs to access the LLMs into the code with proper structure and then run the queries on Metabase. Using these tools helped us to effectively generate SQL queries from natural language queries helping us to do effective exploration and visualization.

- **SQL Basics:** We had to debug and validate the SQL queries generated by the models. We did this crucial task by using SQL parsers and validation frameworks. This ensured that the generated queries were syntactically correct to generate proper results.

11.2 L2 Skills

General Skills

- **Applying Theory in Practice:** Applying theoretical knowledge learned from NLP and machine learning (like metrics and comparison) to this project practically was a crucial step. Using LLMs like Gemma, Llama for real-world SQL generation in their API form helped us test our theoretical knowledge of LLMs and how to utilize it for this practical task.

- **Validating Assumptions:** We were testing our assumptions throughout the project. We assumed at the start that the NLP metrics like BLEU and Rouge would give an accurate estimate of the model's performance. However since they are just natural language metrics, they don't paint the full picture of the SQL query performance against the database. Only running the SQL query against the metabase would give a fuller picture of the model evaluation.

Technical Skills

- **Advanced Prompt Engineering:** Exploring and applying zero-shot, one-shot and few-shot prompting techniques helped us understand how context and examples affect the LLM performance and then helped in crafting the ideal prompt to generate the queries syntactically and contextually correct.

- **Innovative Synergy:** Combining advanced prompt engineering, automation and feedback loops helped in generating better SQL queries over time. We were able to refine the proper technique and get the results quicker using this automated pipeline.
- **Framework Integration:** Integrating LLMs for SQL generation with Metabase and evaluation frameworks into a single real-time system helps the NORP Research scientists a lot. This integration gives real-time data visualization and performance evaluation to validate the effectiveness of the generated SQL queries against the actual database.

11.3 L3 Skills

General Skills

- **Trend Recognition:** Noticing the recent adoption and application pull of LLMs into almost all the real-time systems fields and applying it to the NORP domain helped us stay updated with the current trends. Recognizing the shift towards specialized models underscored the necessity of adapting our approaches to leverage the strengths of domain-specific LLMs for enhanced performance.
- **Generalization/Specialization:** We understood the limitations of general-purpose LLMs like Gemma and Llama and the advantages of fine-tuned models that are aimed toward this specific task of code generation. The latter showed better results with accuracy and relevance as they were trained towards this which would be the way to move forward.
- **Systems Thinking:** We evaluated the trade-offs between centralized cloud systems and decentralized local deployments for deploying LLMs. We understood the limitations of local deployment by evaluating the resources at hand. This systems thinking approach helped us to make better decisions to finish the project in time with limited resources.

Technical Skills

- **Advanced AI/ML Understanding:** We observed the metrics and insights from model evaluation like comparing Gemma vs LFM performance and latency. This showed that models that perform better at code generation normally like LFM are showing better performance at generating the SQL query. This fine-tuned model benefit will help in choosing the right models to use in this project in the future.
- **LLM Deployment:** We detailedly understood resource allocation & LLM codebase and its structure. Using this knowledge we deployed the LLM on the PACE cluster. However with the allocated PACE compute resources, deploying the LLLM was too time-consuming to train and would take more than the time in hand. This long time for deployment and resource constraints made us realize that utilizing open-source API keys was the practical and correct solution for our project's needs.

12 References

- OpenAI Documentation: https://beta.openai.com/docs/
- Groq Documentation: https://groq.com/docs/
- Prompt Engineering Guide: www.promptingguide.ai
- SQLParse Documentation: https://sqlparse.readthedocs.io/en/latest/
- Metabase: https://www.metabase.com/
- NORP Metabase Georgia Tech: http://saopaulo.cc.gatech.edu/norpmetabase/
- Rouge-Score: https://github.com/google-research/google-research/tree/master/rouge
- Bert-Score: https://github.com/Tiiiger/bert_score
- Qingxiu Dong et al., "A Survey on In-context Learning": https://arxiv.org/pdf/2301.00234
- Pranab Sahoo et al., "A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications": https://arxiv.org/pdf/2402.07927
- Qinggang Zhang1 et al., "Structure Guided Large Language Model for SQL Generation": https://arxiv.org/pdf/2402.13284

13 Appendix

13.1 Sample SQL Queries

```sql
SELECT name FROM employees WHERE age > 30 ORDER BY name;
```

13.2 Code Snippets

```python
client = Groq(
    api_key="API_KEY",
)

groq_table = {
    "gemma_1":"gemma-7b-it",
    "gemma_2":"gemma2-9b-it",
    "llama":"llama3-groq-8b-8192-tool-use-preview",
    "mistral":"mixtral-8x7b-32768"
    "phi": "microsoft/phi-3-medium-128k-instruct:free",
    "lfm":"liquid/lfm-40b:free",
}

```

```python
# Generating Function
def generate_sql_query(model, natural_language_query, schema, top_entries, shot):
  if shot == 0:
    prompt = f"Write an SQL query to fulfill the following natural language request:\n\n" \
             f"Request: {natural_language_query}\n"
  elif shot == 1:
    prompt = f"Write an SQL query to fulfill the following natural language request:\n\n" \
             f"Request: {natural_language_query}\n" \
             f"Schema of the tables: {schema}\n"
  else:
    prompt = f"Write an SQL query to fulfill the following natural language request:\n\n" \
             f"Request: {natural_language_query}\n" \
             f"Schema of the tables: {schema}\n" \
             f"Top entries of the table:\n{top_entries}\n"

  response = client.chat.completions.create(
        messages=[
            {"role": "system", "content": "You are a helpful assistant that generates SQL queries."},
            {"role": "user", "content": prompt}
        ],
        model=groq_table[model]
  )

  generated_sql = response.choices[0].message.content.strip()

  return generated_sql
```

```python
import time

start_time = time.perf_counter()
# Function call
end_time = time.perf_counter()
latency = end_time - start_time
print(f"Time taken: {latency:.4f} seconds")
```

13.3 Project README file

This zip file consists of the following code and data files:

Code

- **RTS.ipynb:** Parent Jupyter notebook consisting of the entire code for all the analysis and experiments.
- **SQL_Gen_ORA.py:** Python script to generate SQL queries from the Open-Router.AI API source.

- **SQL_Gen_Groq.py:** Python script to generate SQL queries from the Groq API source.
- **SQL_Gen.py:** Python script to load data from SQL directories.
- **Metrics.py:** Python script to calculate the evaluation metrics.
- **Analysis.py:** Python script performing the analysis and generating the corresponding plots.
- **Presentation.py:** Python script to generate the plots and tables for the presentation.

Directories

- **Datasets:** Consists of simple and complex CSV files from three sources: Crime, Housing, and Shooting datasets.
- **Images:** Consists of all the plots generated during the analysis.
- **Metrics:** Consists of JSON files for the evaluation metrics for all datasets individually, as well as grouped simple and complex datasets.
- **SQL:** Consists of CSV files for the SQL-generated queries for all models, organized in dataset-named files, including the SQL queries for simple and complex datasets.
- **Results_ORA:** SQL-generated queries from the OpenRouter.AI source.
- **Results:** SQL-generated queries from the Groq source, combined with those from the OpenRouter.AI source.
- **Prompt_Analysis:** Consists of comparison datasets for the types of prompt engineering—Zero-shot, One-shot, and Few-shot—on the Crime dataset (Simple and Complex).

Other Files and Information

- **Report.pdf:** Detailed report of the analysis and results for submission.
- **py_requirements.txt:** Contains all the necessary libraries required for code execution.
- **presentation.pptx:** Presentation slides for the video presentation.
- **skill_learning.txt:** Consists of the skill learning for the chatbot evaluation
- **Presentation Link:** NORP LLM Chatbot Evaluation.
- **Execution Environment:** All the code was executed on Google Colab, with necessary libraries installed, and all the data loaded in the code was designed accordingly.

13.4 Python Library Requirements

- **Core Libraries:**
  - openai (for OpenRouter.AI API)
  - openai==0.28 (for Groq API)
  - pandas
  - numpy
  - os
  - chardet
  - shutil
  - time
  - matplotlib.pyplot
  - sqlparse
  - warnings
  - json
- **Libraries from Specific Modules:**
  - Groq (from Groq)
  - rouge_scorer (from rouge_score)
  - sentence_bleu (from nltk.translate.bleu_score)
  - score (from bert_score)
  - ngrams (from nltk.util)
  - collections (from Counter)

Installation Commands on Colab

- **Important Note:** Only install one openai version at a time.
- Commands:
  - `!pip install openai` (For OpenRouter.AI API)
  - `!pip install openai==0.28` (For Groq API)
  - `!pip install groq`
  - `!pip install rouge-score`
  - `!pip install bert-score`
  - `!pip install nltk`
  - `!pip install sqlparse`