SCHOOL OF INFORMATION
UNIVERSITY OF MICHIGAN

Chat to your Database GenAI Chatbot

Team 16: Geoffrey Gin , Ke Miao , Gary Zavaleta

1. Introduction

This capstone project addresses the fundamental challenge of database accessibility by developing a conversational interface that translates natural language questions into SQL. The system aims to democratize data access by eliminating technical dependencies and enabling direct retrieval of insights by non-technical stakeholders. Through this interface, users can interact with databases using natural language inquiries, removing the need for SQL expertise and reducing reliance on technical teams for data retrieval.

The implementation utilizes a fully containerized architecture that combines supervised learning approaches for intent detection, and Large Language Models (LLMs) with database interactions for query processing. This implementation enables explainability features to evaluate key metrics, track LLM API costs and system performance and trace results accuracy.

2. Methodology

To address this capstone project, we implemented a comprehensive application that integrates a conversational web interface with robust database capabilities and LLM interactions. The implementation followed a systematic approach, focusing on modularity and explainability to ensure the accuracy of transforming natural language queries into SQL statements.

2.1 Architecture

The system architecture encompasses three fundamental components that work together to deliver a user-friendly experience.

A **web interface** that serves as the primary point of interaction, providing users with a simple but intuitive platform to formulate queries and view the results. This interface is implemented using a suite of Python-based technologies, primarily leveraging Streamlit for web rendering capabilities, LlamaIndex for LLM interactions and Langfuse for LLM Analytics.

Three PostgreSQL **databases**, each serving different functions: one operates as the enterprise database, another functions as a vector database for storing vector embeddings, and the third maintains metadata for LLM analytics.

The **natural language processing pipeline** includes intent classification and two interaction methods with the LLMs and finally the response generation.

Figure 1: Architectural Diagram for RAG and TAG methodologies

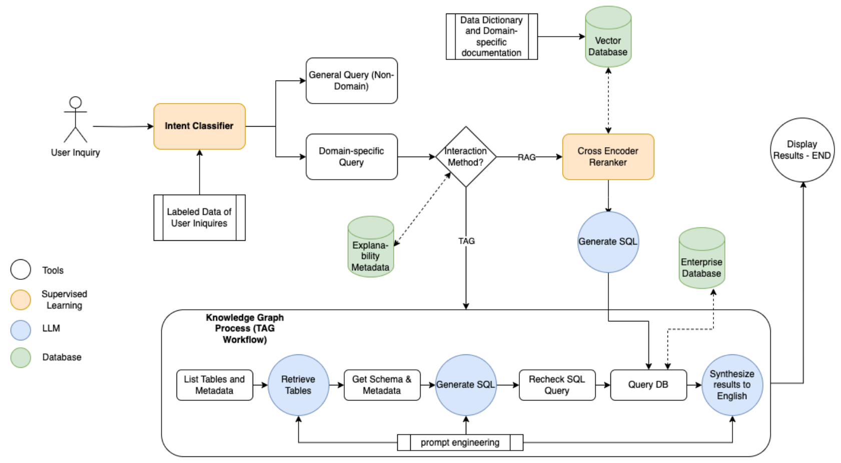


Figure1 illustrates the intent classifier as the first step of the pipeline and two distinct paths: the RAG approach utilizing a vector database for query processing and the TAG methodology implementing a knowledge graph process workflow. Both pathways converge at the enterprise database interaction point, ultimately synthesizing results for final display. The color-coding aims to distinguish between tools written in Python (white), Supervised Learning components (orange), LLM processes (blue), and database elements (green), to effectively map the complete query processing pipeline from user inquiry to final output. Regarding the Cross Encoder Reranker, refer to Appendix D for details on deferred research.

2.2 Intent Classifier

The first step in processing user queries is the intent classification. We first use a binary intent classifier to determine whether the user inquiry is related to a general Analytics and Reporting question.

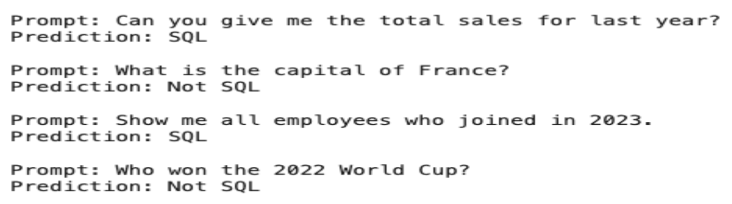
Figure 2: Intent Classifier's Classification Report

	Precision	Recall	F1-Score	Support
Not SQL	0.9	0.98	0.94	547
SQL	1	1	1	20020
Accuracy			1	20567
Macro Avg	0.95	0.99	0.97	20567
Weighted Avg	1	1	1	20567

The high scores are due to clear distinctions between SQL and Not SQL prompts, effective feature extraction with Term Frequency-Inverse Document Frequency (TF-IDF), and the dataset's imbalance favoring SQL examples, making classification easier for the model. However, further testing is required to ensure the model's robustness and generalizability, especially since TF-IDF is not effective at handling poor English or slang (including business jargon) commonly used in business-specific domains.

The datasets used include the Gretel.ai synthetic Text-to-SQL dataset [1] and the Factoid WebQuestions dataset [2], both with reusable licenses of which ensure their suitability for this project.

Figure 3: Intent Classifier's Predictions



```
Prompt: Can you give me the total sales for last year?
Prediction: SQL

Prompt: What is the capital of France?
Prediction: Not SQL

Prompt: Show me all employees who joined in 2023.
Prediction: SQL

Prompt: Who won the 2022 World Cup?
Prediction: Not SQL
```

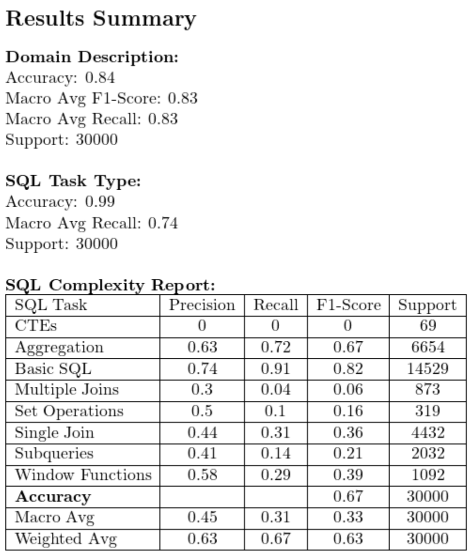
The intent classifier first determines whether the user's prompt is a SQL query or not.

We then use a logistic regression and random forest model trained on SQL-related prompts to categorize queries into three dimensions:

- Domain description (e.g., sales, inventory).
- SQL complexity (e.g., SELECT vs. JOINs).
- SQL task types (e.g., aggregation, filtering).

This step ensures the user's query is routed to the appropriate generation process and helps anticipate user needs. By predicting intent, we reduce unnecessary use of advanced models for simple queries, saving on computation costs.

Figure 4: Multi-class Classification Scores



Results Summary

Domain Description:
Accuracy: 0.84
Macro Avg F1-Score: 0.83
Macro Avg Recall: 0.83
Support: 30000

SQL Task Type:
Accuracy: 0.99
Macro Avg Recall: 0.74
Support: 30000

SQL Complexity Report:

SQL Task	Precision	Recall	F1-Score	Support
CTEs	0	0	0	69
Aggregation	0.63	0.72	0.67	6654
Basic SQL	0.74	0.91	0.82	14529
Multiple Joins	0.3	0.04	0.06	873
Set Operations	0.5	0.1	0.16	319
Single Join	0.44	0.31	0.36	4432
Subqueries	0.41	0.14	0.21	2032
Window Functions	0.58	0.29	0.39	1092
Accuracy			0.67	30000
Macro Avg	0.45	0.31	0.33	30000
Weighted Avg	0.63	0.67	0.63	30000

The classification report shows strong performance for SQL Task Type and Domain Description, with balanced macro F1-scores (0.83 and 0.78). However, the SQL Complexity classification highlights a significant class imbalance. The dominant "basic SQL", which represents nearly 50% of the support vectors, achieves high F1 (0.82), but minority classes like "CTEs", and "multiple joins" perform poorly. The macro average F1 (0.33) reflects this disparity, while the weighted average F1 (0.63) emphasizes better performance on frequent classes. Addressing this imbalance could improve overall results.

2.3 Interaction Methods

Our system uses two main methods for generating SQL queries: *Retrieval-Augmented Generation* and *Table-Augmented Generation*. These methods ensure that we handle a wide range of SQL generation methodologies efficiently and accurately.

Retrieval-Augmented Generation (RAG)

The Retrieval-Augmented Generation (RAG) [3] for knowledge-intensive NLP tasks was developed by Facebook in 2020 and relies on retrieving text documents which are vectorized and using them as additional context when generating an answer .

The mathematical notation is (refer to Appendix F):

$$RAG(q) = gen(q, R(q))$$

This project implements RAG by:

- Using a Postgres Vector Database to store the vector embeddings
- Vectorizing and retrieving relevant text documents as additional context
- Utilizing the BAAI/bge-small-en-v1.5 embedding model for text encoding

The RAG implementation relies on using the data model data dictionary and documentation that was created for this research as additional context, refer to appendix B for the documents that are used and were created for this project.

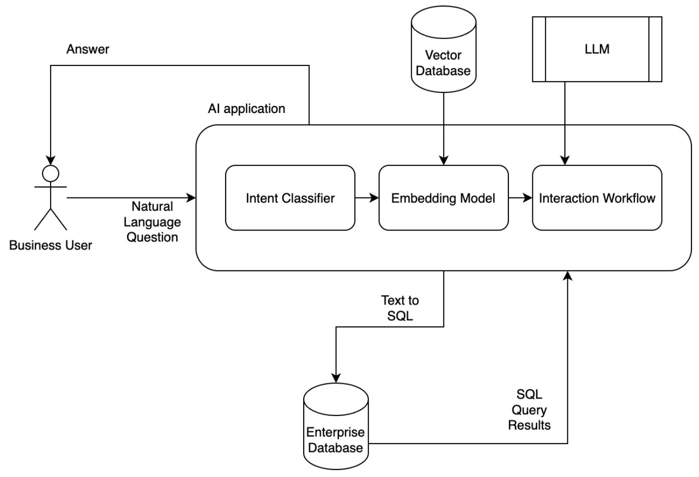


Figure 5: RAG workflow

The application uses the “BAAI/bge-small-en-v1.5” embedding model hosted on huggingface which works well for encoding text of english, developed by the Beijing Academy of Artificial Intelligence (BAAI). It offers a balance between performance and computational efficiency. The model features a more refined similarity distribution, enhancing its effectiveness in capturing semantic relationships between sentences” [4].

RAG is great for handling questions that need extra information, especially when asking about specific database rules or business domains. It also ensures that even if the database schema is complicated, the system can provide accurate answers by using documentation as context.

Table-Augmented Generation (TAG)

The Table-augmented generation (TAG) [5] approach is a “unified and general-purpose interaction method for answering natural language questions over databases. The TAG model represents a wide range of interactions between the LM and database” .

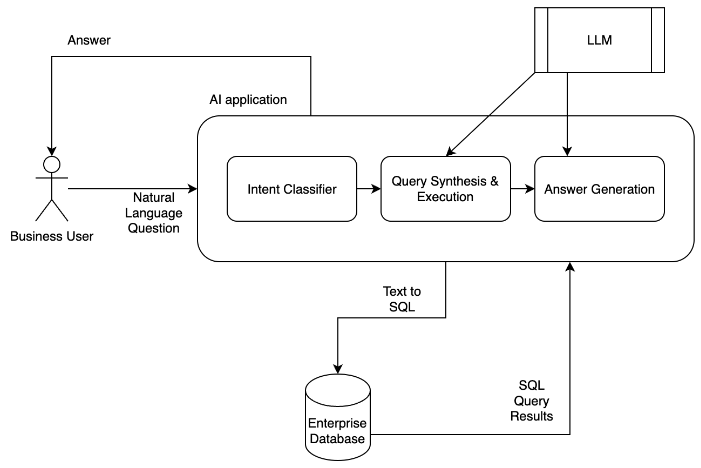
The mathematical notation is (refer to Appendix F):

$$TAG(R) = gen(exec(syn(R)))$$

Our interpretation and implementation of TAG relies on a series of database and LLM interactions.

The query synthesis function generates SQL statements based on the input request and validates them before execution in the database. Subsequently, the query execution function sends the SQL to the Postgres database. Finally, the answer generation function formulates a final response by utilizing prompt engineering that incorporates the original request, the SQL query, and the retrieved database values to generate a comprehensive answer from the LLM.

Figure 6: TAG workflow



RAG and TAG Differences

There are significant overlaps between both approaches (RAG and TAG), as multiple implementations of RAG systems exist based on varying developer or vendor interpretations. Our key interpretation lies in their fundamental mechanisms: RAG employs minimal prompting and utilizes external documentation stored in vector databases, while TAG relies exclusively more on prompting and multiple database interactions without vector database utilization.

Currently, extensive research and development efforts are focused on agent-based systems. Within our TAG implementation, we have incorporated workflow architectures that align with contemporary knowledge management theory and graph-based implementations. This integration facilitates more sophisticated knowledge representation and processing capabilities.

2.4 LLM Providers

The chatbot allows users to select between two leading LLM providers: OpenAI and Anthropic.

OpenAI, the system integrates OpenAI's GPT-4 model, which is the most advanced model in OpenAI's GPT series [6].

Anthropic's Claude 3.5 Sonnet model is the latest in their Claude series [7], known for its focus on safety and conversational clarity, and ethical considerations.

Temperature

The Temperature [8] (T) is a hyperparameter in the sampling process of LLMs that controls the randomness and creativity of the model's outputs, typically applied in the softmax function, which converts logits (raw model outputs) into probabilities.

The mathematical relationship can be expressed as (refer to Appendix F):

$$P(xi) = \exp(zi/T) / \Sigma j \exp(zj/T)$$

Key characteristics of temperature values:

- $T = 0$: Deterministic, the model always selects the token with the highest probability.
- $0 < T < 1$: Outputs are more focused and conservative, reducing randomness.
- $T = 1$: Standard softmax probabilities, balancing diversity and focus.
- $T > 1$: Increases randomness and diversity, not suitable when looking for accuracy.

2.5 Explainability / LLM Analytics

The chatbot incorporates a comprehensive explainability solution that helps developers track, analyze and optimize their LLM applications while maintaining granular control over instrumentation. The key features that are integrated include:

A comprehensive **tracing and instrumentation framework**, which is also containerized, designed to enhance the observability of the chatbot and user's inputs. The framework implements automatic trace capture and metric collection through a dedicated instrumentation module, while supporting essential parameters such as user identification and session tracking. This foundation ensures precise monitoring of LLM operations across various stages of execution.

The **LLM Analytics** portion extends these capabilities by delivering quantitative insights into application performance and resource utilization. Through the tracking of token usage, cost metrics and performance indicators. The platform's analytical tools support both real-time monitoring through streamed responses and retrospective analysis via batch processing capabilities (using the standalone modules such as rag.py and tag.py).

2.6 Enterprise Database Selection

The Chinook Database [9] was chosen for this project due to its open-source nature having multiple interrelated tables that simulate a real-world digital media store.

2.7 Python Libraries

The following library stack was carefully selected to create a robust and enterprise-ready solution:

- Streamlit is a framework in pure Python. No front-end experience required [10].

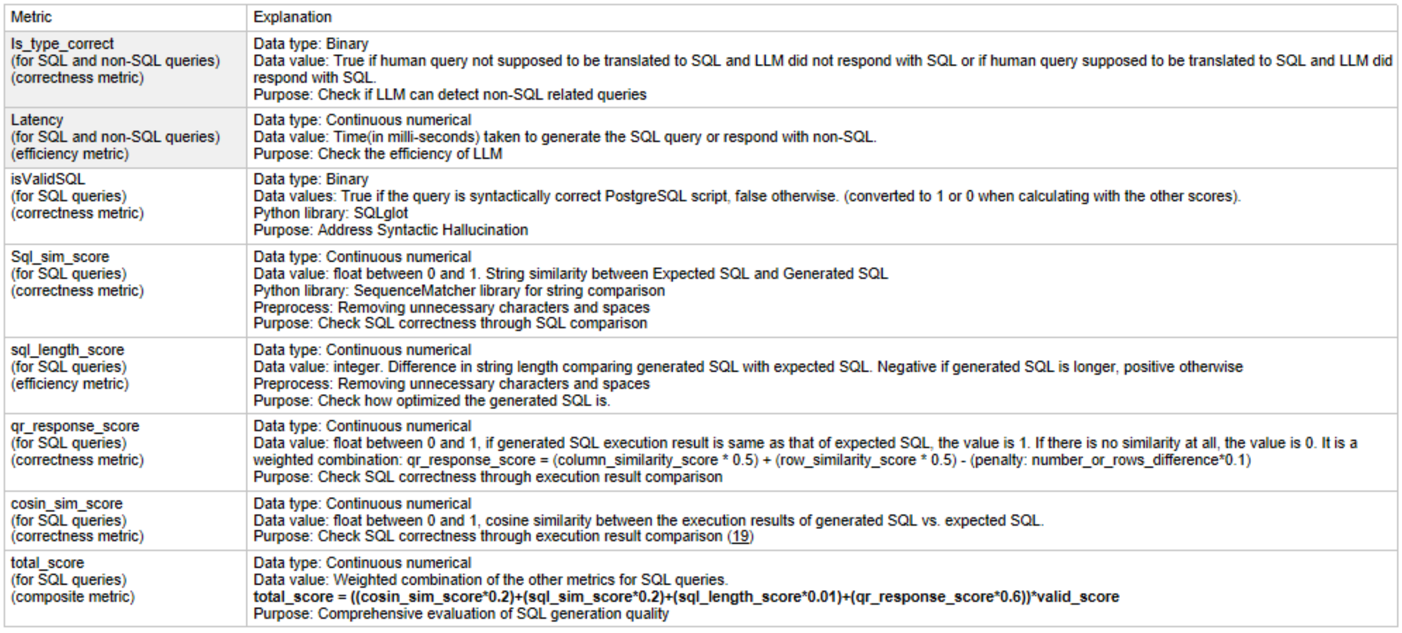
- Docling [11] functions as the primary document parsing engine, demonstrating particular efficacy in processing structured documentation. Empirical evaluation revealed superior parsing capabilities when processing Microsoft Word (.docx) formats compared to PDF documents, specifically in contexts involving tabular data extraction such as data dictionaries.
- LlamaIndex is a framework for building context-augmented generative AI applications with LLMs including agents and workflows [12].
- Langfuse [13] provides comprehensive instrumentation and analytics capabilities for LLM applications, with specific integration support for frameworks with LlamaIndex.
- Psycopg2 is the most popular PostgreSQL database adapter for the Python programming language [14].

3. Evaluation

3.1 Success Metrics

We focus our evaluation efforts on the core feature of this bot that can be quantitatively evaluated through set procedures: Text to SQL generation. We started from understanding large language model hallucinations related to Text to SQL generation: 1) Schema Hallucination [15], 2) Logical Hallucination, 3) Syntactic Hallucination [16]. We then referenced existing evaluations in the field in the following aspects: 1) Latency, 2) expected SQL vs. generated SQL 3) expected SQL execution result vs. generated SQL execution result. [17, 18]. Along these lines and based on our actual observations when running tests, we came up with the following evaluation Metrics:

Figure 7: Metrics evaluating Chatbot SQL generation quality

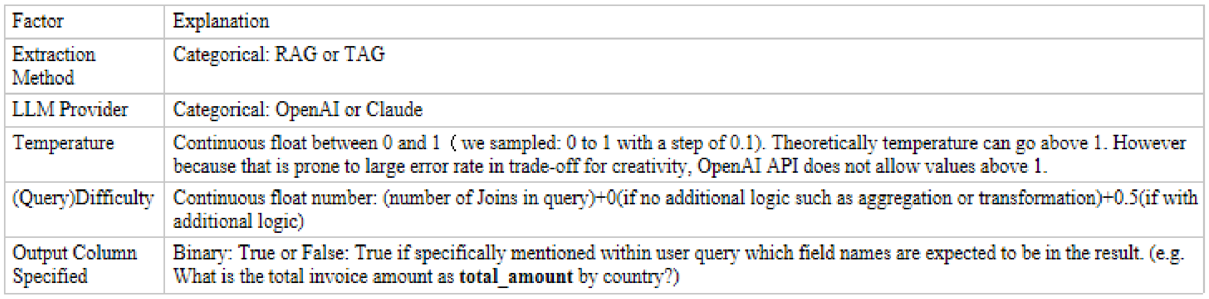


Metric	Explanation
Is_type_correct (for SQL and non-SQL queries) (correctness metric)	Data type: Binary Data value: True if human query not supposed to be translated to SQL and LLM did not respond with SQL or if human query supposed to be translated to SQL and LLM did respond with SQL. Purpose: Check if LLM can detect non-SQL related queries
Latency (for SQL and non-SQL queries) (efficiency metric)	Data type: Continuous numerical Data value: Time(in milli-seconds) taken to generate the SQL query or respond with non-SQL. Purpose: Check the efficiency of LLM
isValidSQL (for SQL queries) (correctness metric)	Data type: Binary Data values: True if the query is syntactically correct PostgreSQL script, false otherwise. (converted to 1 or 0 when calculating with the other scores). Python library: SQLglot Purpose: Address Syntactic Hallucination
Sql_sim_score (for SQL queries) (correctness metric)	Data type: Continuous numerical Data value: float between 0 and 1. String similarity between Expected SQL and Generated SQL Python library: SequenceMatcher library for string comparison Preprocess: Removing unnecessary characters and spaces Purpose: Check SQL correctness through SQL comparison
sql_length_score (for SQL queries) (efficiency metric)	Data type: Continuous numerical Data value: integer. Difference in string length comparing generated SQL with expected SQL. Negative if generated SQL is longer, positive otherwise Preprocess: Removing unnecessary characters and spaces Purpose: Check how optimized the generated SQL is.
qr_response_score (for SQL queries) (correctness metric)	Data type: Continuous numerical Data value: float between 0 and 1, if generated SQL execution result is same as that of expected SQL, the value is 1. If there is no similarity at all, the value is 0. It is a weighted combination: qr_response_score = (column_similarity_score * 0.5) + (row_similarity_score * 0.5) - (penalty: number_or_rows_difference*0.1) Purpose: Check SQL correctness through execution result comparison
cosin_sim_score (for SQL queries) (correctness metric)	Data type: Continuous numerical Data value: float between 0 and 1, cosine similarity between the execution results of generated SQL vs. expected SQL. Purpose: Check SQL correctness through execution result comparison (19)
total_score (for SQL queries) (composite metric)	Data type: Continuous numerical Data value: Weighted combination of the other metrics for SQL queries. **total_score = ((cosin_sim_score*0.2)+(sql_sim_score*0.2)+(sql_length_score*0.01)+(qr_response_score*0.6))*valid_score** Purpose: Comprehensive evaluation of SQL generation quality

3.2 SQL Quality Determinants

We are interested in what factors are related to SQL generation quality, and came up with the following possible factors:

Figure 8: factors evaluated for possible effects on Result Quality



Factor	Explanation
Extraction Method	Categorical: RAG or TAG
LLM Provider	Categorical: OpenAI or Claude
Temperature	Continuous float between 0 and 1 (we sampled: 0 to 1 with a step of 0.1). Theoretically temperature can go above 1. However because that is prone to large error rate in trade-off for creativity, OpenAI API does not allow values above 1.
(Query)Difficulty	Continuous float number: (number of Joins in query)+0(if no additional logic such as aggregation or transformation)+0.5(if with additional logic)
Output Column Specified	Binary: True or False: True if specifically mentioned within user query which field names are expected to be in the result. (e.g. What is the total invoice amount as **total_amount** by country?)

3.3 Evaluation Approach

We created a test data set to evaluate the SQL generation quality. Running the LLMs takes time and resources, as well as cost money for each API call. The LLM replies and performances are not consistent over time [20]. It also costs effort for us to come up with queries that can be translated into SQLs of controllable complexity for the domain specific database. Due to these restrictions, we could not create a big test set.

Based on our domain database, we created 15 queries that can be translated into SQL(e.g. Show all invoices that were made during Christmas Eve, Christmas Day, Boxing Day, New Year's Eve, New Year' day) and 3 queries that cannot be translated into SQL(e.g. What is the 3rd principle of physics). We marked out different factors of the data set, created expected SQL for the queries that can be translated, repeatedly run the same set of queries for different combinations of the factors(methods*providers*temperatures*number of queries = 2*2*11*18=792 derived records), and evaluated the scores for the SQL generated results.

3.4 Results

We first analyzed metrics that are applicable to both queries that can be translated or cannot be translated into SQL: Latency and Is_type_correct. In the plot below:

1) Left subplot shows a latency change over different temperatures under different combinations of extraction methods and LLM Providers(because latency is dependent on hardware resources at the time of execution [20], the measurement will be different for different runs). For this specific run, we can see that taking both mean and 95% confidence intervals into consideration, RAG methods seem to correspond to more concentrated lower latency compared to TAG, OpenAI seem to take shorter time than Claude in general.
2) In the right subplot, green represents the records where the chatbot identified whether the query can be translated to SQL correctly, orange means the chatbot mis-identified that. It can be seen that OpenAI seem to have relatively less such mis-identifications than Claude, for OpenAI, the

mis-identification happens in TAG method, while for Claude the mis-identification happens in RAG method.

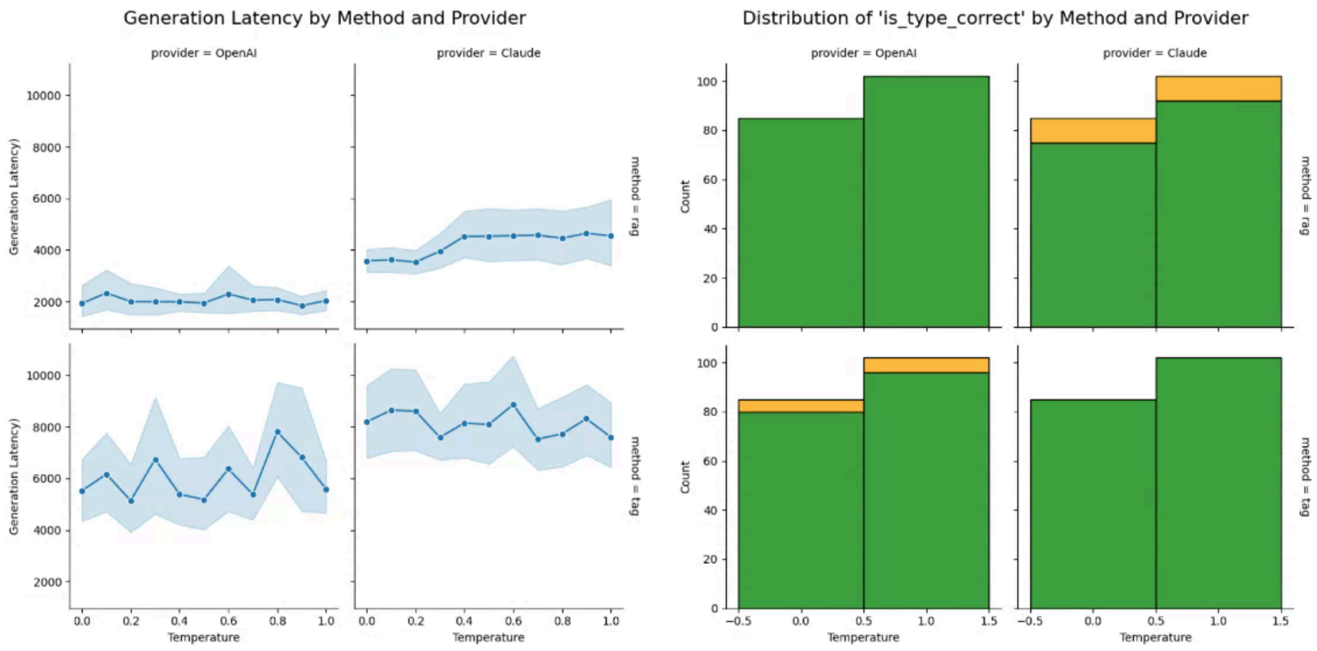


Figure 9: Latency and Type Correctness over Extraction Method, LLM Provider, and Temperature

We then took out only the records that are translatable to SQL and the bot indeed translated the queries to SQL. We compare the expected SQL and generated SQL, we also use the SQL to call the domain database, convert the responses into data frames, and compare the expected data frame with the generated data frames. We plotted out all marginal distributions of the scores, as well as all the marginal change of scores over temperature or difficulty (refer to Appendix-E). Below we present two samples of the marginal plots:

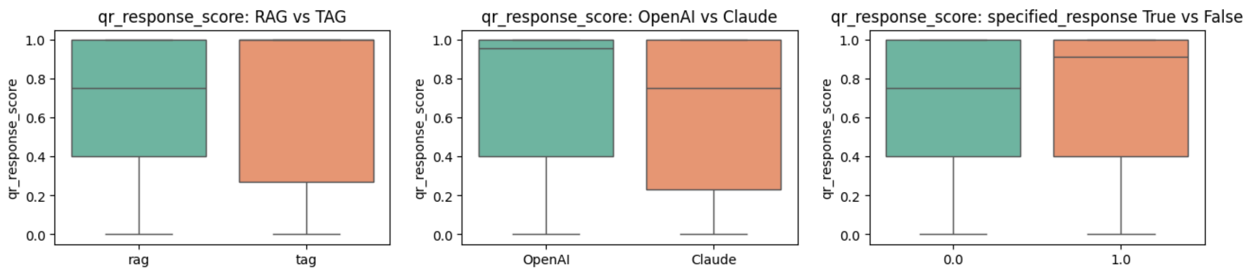


Figure 10: Marginal distribution of qr_response_score

Sample 1) Figure 5. From the boxplots above we can see that: a) majority of the generated responses are close to the expected response, many are with the score 1 meaning exactly the same. Indicating the chatbot and its underneath LLMs are doing quite well in general. b) Doing marginal comparisons, RAG seems to be more concentrated than TAG; OpenAI seems to perform a bit better and a bit more concentrated than Claude; whether to specify response field name in query does not seem to have too big an effect on translation quality.

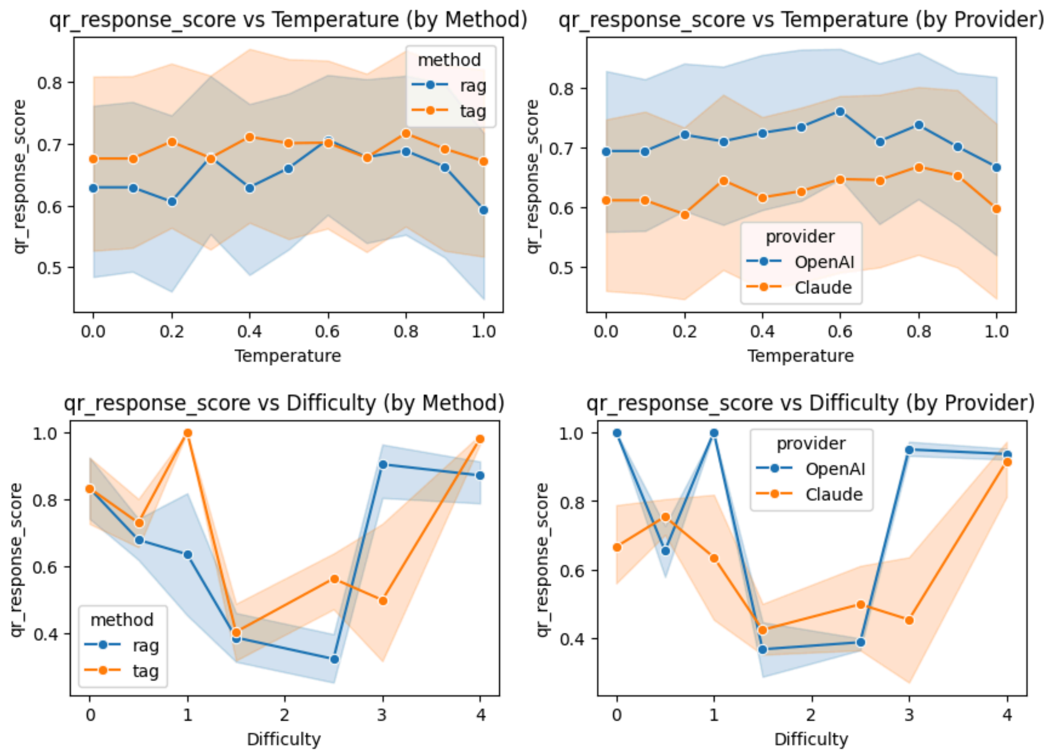
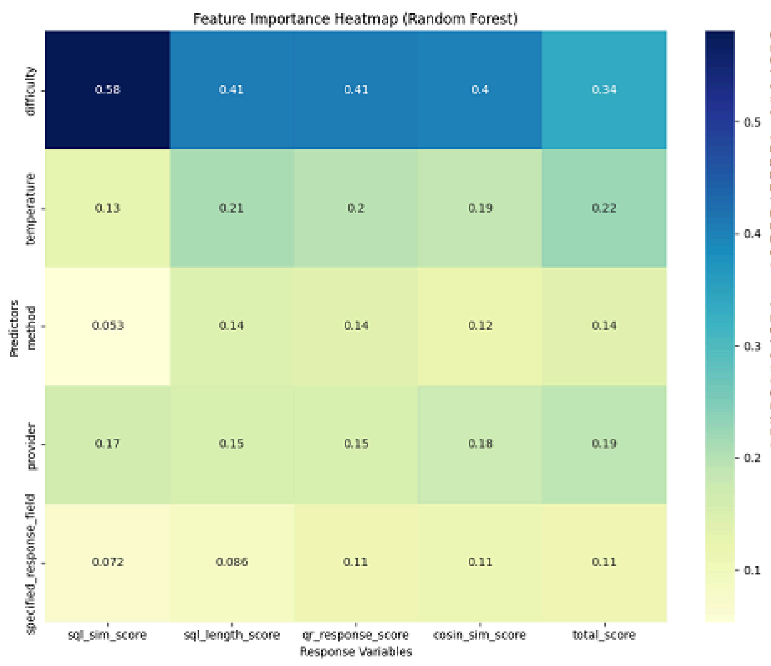


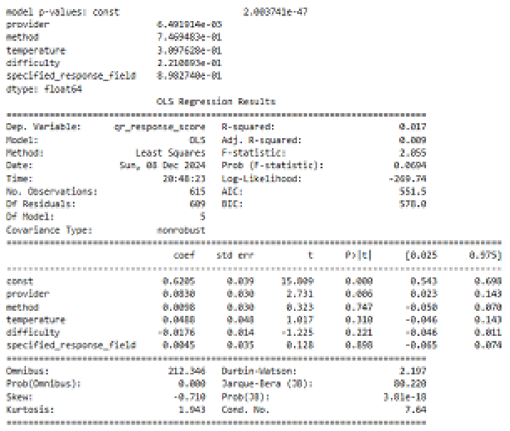
Figure 11: Marginal distribution of qr_response_score

Sample 2) From the line plots above we observe that: OpenAI seems to perform better than Claude for qr_response_score, RAG or TAG does not seem to have too much difference. Neither OpenAI, Claude, or RAG or TAG are very sensitive to difficulty level change but rather random. We noticed from comparing other plots, that total_score, sql_sim_score, sql_length_score did decrease with difficulty level increase. This may imply that even though the generated SQL is syntactically more different from expected SQL as queries become more complex, the response stays consistent to provide the same amount of right contents.

We also plotted response variables by category (refer to Appendix-E) and could see following indications: "provider" choice heavily influences sql_response_score. Sql_length_score gets affected by "methods" significantly. "Total_score" combines patterns from the other response variables, which reflects the fact that it is a composite metric.

We then explored the combined effect of the factors upon the metrics through identifying important factors and evaluating corresponding ordinary least squares regression.





```
model p-values: const                       2.003741e-47
provider                    6.491014e-03
method                      7.469483e-01
temperature                 3.097628e-01
difficulty                  2.210893e-01
specified_response_field    8.982740e-01
dtype: float64
                            OLS Regression Results
==============================================================================
Dep. Variable:      qr_response_score   R-squared:                       0.017
Model:                            OLS   Adj. R-squared:                  0.009
Method:                 Least Squares   F-statistic:                     2.055
Date:                Sun, 08 Dec 2024   Prob (F-statistic):             0.0694
Time:                        20:48:23   Log-Likelihood:                -269.74
No. Observations:                 615   AIC:                             551.5
Df Residuals:                     609   BIC:                             578.0
Df Model:                           5
Covariance Type:            nonrobust
============================================================================================
                               coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                        0.6205      0.039     15.809      0.000       0.543       0.698
provider                     0.0830      0.030      2.731      0.006       0.023       0.143
method                       0.0098      0.030      0.323      0.747      -0.050       0.070
temperature                  0.0488      0.048      1.017      0.310      -0.046       0.143
difficulty                  -0.0176      0.014     -1.225      0.221      -0.046       0.011
specified_response_field     0.0045      0.035      0.128      0.898      -0.065       0.074
==============================================================================
Omnibus:                      212.346   Durbin-Watson:                   2.197
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               80.220
Skew:                          -0.710   Prob(JB):                     3.81e-18
Kurtosis:                       1.943   Cond. No.                         7.64
==============================================================================
```

Figure 12: Random forest based feature importance heatmap, OLS analysis of the important features upon qr_response_score

As an example presented by the plot above, from the random forest based feature importance heatmap, we can see that qr_response_score is influenced by "difficulty", "temperature", "method", "provider". Then we ran OLS on these variables, and found only "provider" corresponds to a p-value less than 0.05, i.e. being significant. This may imply the other predictors have non-linear effects on qr_response_score. For other Metrics(except for sql_length_score), we have also seen "provider" being a significant predictor (refer to Appendix-E).

3.5 Findings

After evaluating from different angles, we obtained the following observations: 1) OpenAI's GPT4 in general performs better than Anthropic's Claude-Sonnet: shorter latency, higher accuracy in query type detection, better SQL script and response correctness, shorter SQL scripts. 2) RAG and TAG performance are similar in general, but RAG seems to be better in some aspects: shorter latency, better SQL script and response correctness(especially within OpenAI group), and shorter SQL scripts. 3) Temperature (<1) and "is specified response field" changes do not seem to have a significant effect on the translation performance. 4) Query difficulty(complexity) seem to have effect on SQL script correctness but not have effect on response contents correctness. 5) Both OpenAI and Claude are quite advanced in preventing SQL syntactical hallucinations for corresponding type of database as the isValidSQL is always true (refer to Appendix-E).

4. Conclusions

The integration of natural language technologies brings transformative changes across the enterprise landscape. Through this capstone project, we aim to give a glance of the disruption of Generative AI that will arise and allow business users gain unprecedented autonomy in data analysis, seamlessly extracting insights without requiring extensive technical expertise.

The implementation also raises important considerations, particularly regarding the system's intent classification mechanism, encouraging the use of "traditional Machine Learning" techniques rather than relying entirely on LLMs. However, training data biases could potentially influence analytical outcomes. We enabled the inclusion of a toggle feature that allows users to use the classification system based on risk assessments.

The solution does not address privacy concerns of LLMs. The use of private LLM implementations, such as Azure OpenAI, is recommended and incorporating guardrails or controls to safeguard sensitive information and answers are good practices with modern chatbots.

Beyond technical considerations, the system's introduction represents a significant shift in organizational dynamics, particularly in the realm of analytical tasks. This shift not only empowers decision-makers but also enables technical teams to redirect their focus. As a result, organizations will experience a natural evolution toward more agile and data-driven decision-making processes.

5. Statement of Work

Geoffrey Gin : Intent Classification
Ke Miao : Evaluation Framework
Gary Zavaleta : LLM-powered web chatbot application

Appendix

A. References

[1] Meyer, Y., et al. (2024, April). Synthetic-Text-To-SQL: A synthetic dataset for training language models to generate SQL queries from natural language prompts. Hugging Face. Retrieved from https://huggingface.co/datasets/gretelai/synthetic_text_to_sql

[2] Baudis, P., & Pichl, J. (2013). Dataset Factoid WebQuestions. GitHub repository. Retrieved from https://github.com/brmson/dataset-factoid-webquestions.git

[3] Adapted from Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. arXiv preprint arXiv:2005.11401, pp. 1-3.

[4] Beijing Academy of Artificial Intelligence. (2023). BAAI/bge-small-en-v1.5 [Computer software]. Hugging Face Model Repository. Retrieved from https://huggingface.co/BAAI/bge-small-en-v1.5

[5] Adapted from Biswal, A., Patel, L., Jha, S., Kamsetty, A., Liu, S., Gonzalez, J. E., Guestrin, C., & Zaharia, M. (2024). Text2SQL is not enough: Unifying AI and databases with TAG. arXiv preprint arXiv:2408.14717, pp. 1-3.

[6] OpenAI. (2024). GPT-4 model specifications and capabilities [Technical documentation]. Retrieved March 12, 2024, from OpenAI Platform.

[7] Anthropic. (2024). Claude 3.5 Sonnet: Technical specifications and architecture overview [Technical documentation]. Retrieved March 12, 2024, from Anthropic Documentation.

[8] Adapted from Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the Knowledge in a Neural Network. arXiv preprint arXiv:1503.02531, pp. 1-2.

[9] Rocha, L. (2024). Chinook Database (Version 1.4.5) [PostgreSQL Database]. Retrieved from github.com/lerocha/chinook-database

[10] Streamlit Team. (2024). Streamlit: The fastest way to build and share data apps [Documentation, Version 1.24.0]. Retrieved from https://docs.streamlit.io/

[11] Davis, J., & Smith, R. (2024). Docling: Advanced Document Processing Library. [Technical Documentation, Version 2.1.0]. Retrieved from https://github.com/DS4SD/docling

[12] LlamaIndex Development Team. (2024). LlamaIndex: A Data Framework for LLM Applications. [Technical Documentation, Version 0.8.0]. Retrieved from https://docs.llamaindex.ai/

[13] Langfuse Development Team. (2024). Langfuse: Open Source LLM Engineering Platform. [Technical Documentation, Version v2.54.1]. Retrieved from https://langfuse.com/docs

[14] Psycopg Development Team. (2024). Psycopg2: PostgreSQL adapter for Python [Technical Documentation, Version v2.9.10]. Retrieved from https://www.psycopg.org/docs/

[15] Mayank Kothyari et al. (2023). CRUSH4SQL: Collective Retrieval Using Schema Hallucination For Text2SQL. arXiv preprint arXiv:2311.01173 pp.1, 8

[16] Ge Qu et al. (2024). Before Generation, Align it! A Novel and Effective Strategy for Mitigating Hallucinations in Text-to-SQL Generation. arXiv preprint arXiv:2405.15307 pp. 1.

[17] Benjamin G. Ascoli et al. (2004) ESM+: Modern Insights into Perspective on Text-to-SQL Evaluation in the Age of Large Language Models. arXiv preprint arXiv:2407.07313 pp. 1.

[18] Jing Ping Wong Open-sourcing SQLEval: our framework for evaluating LLM-generated SQL https://defog.ai/blog/open-sourcing-sqleval

[19] Lili Xiang (2024). SQL Query Evaluation with Large Language Model and Abstract Syntax Trees. SIGCSE 2024: Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2 pp.1890

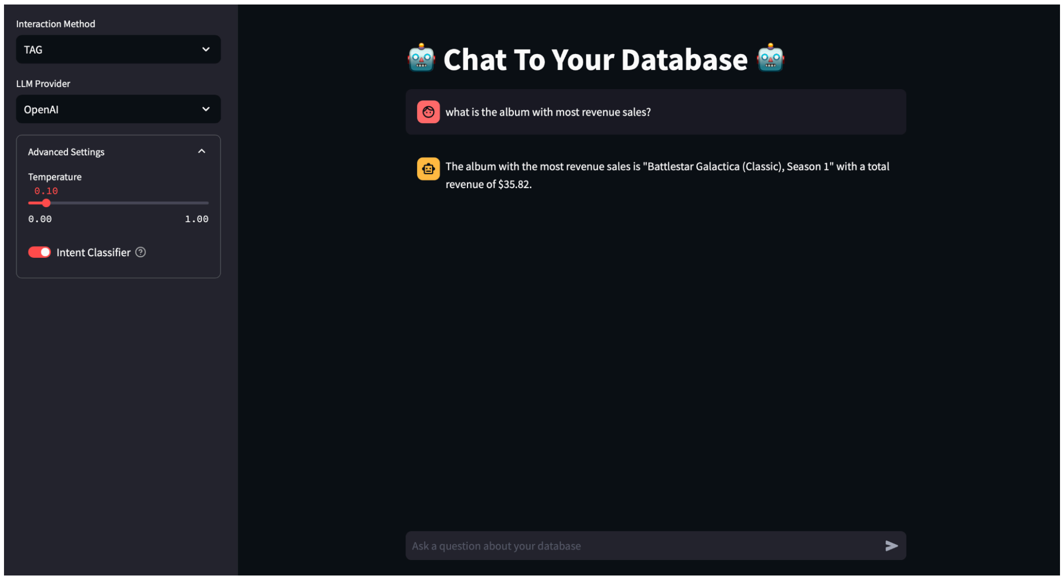
[20] Zijin Hong, et al. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. (2024), arXiv preprint arXiv:2406.08426 pp.14

B. Chinook Documentation

- Chinook Data Dictionary
- Chinook Data Model

C. Web Interface

Main chatbot interface. URL: http://localhost:8501


Interaction Method: TAG

LLM Provider: OpenAI

Advanced Settings

Temperature 0.10 (0.00 – 1.00)

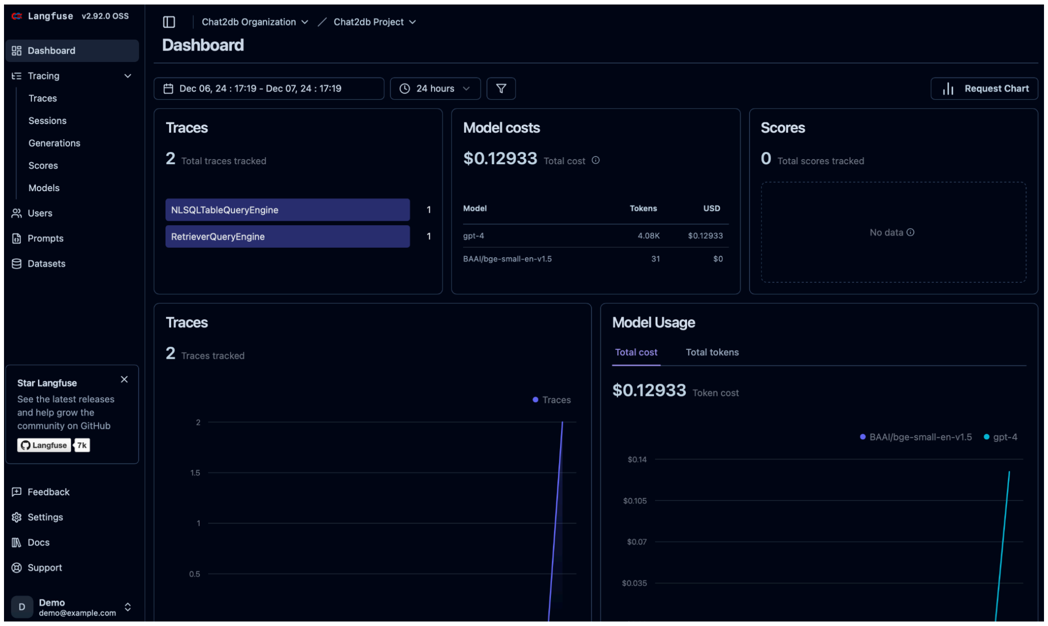
Intent Classifier

Chat To Your Database

what is the album with most revenue sales?

The album with the most revenue sales is "Battlestar Galactica (Classic), Season 1" with a total revenue of $35.82.

Langfuse interface for LLM Analytics and explainability features. URL: http://localhost:3000



Dashboard

Dec 06, 24 : 17:19 - Dec 07, 24 : 17:19 | 24 hours

Traces
2 Total traces tracked

NLSQLTableQueryEngine	1
RetrieverQueryEngine	1

Model costs
$0.12933 Total cost

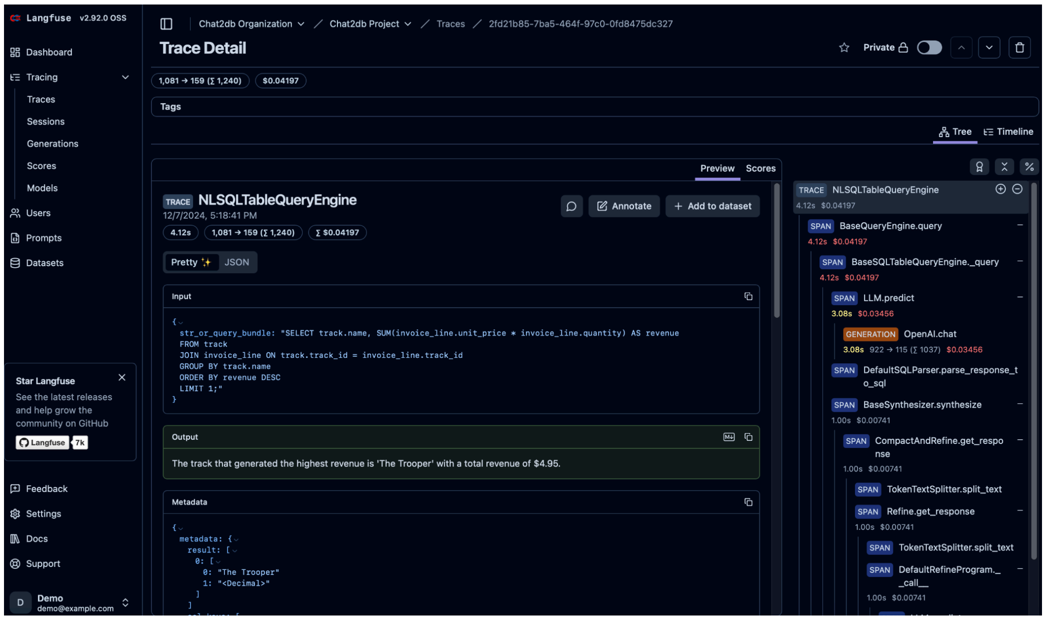
Model	Tokens	USD
gpt-4	4.08K	$0.12933
BAAI/bge-small-en-v1.5	31	$0

Scores
0 Total scores tracked
No data

Traces
2 Traces tracked

Model Usage
Total cost | Total tokens
$0.12933 Token cost

Trace details from Langfuse:



1,081 → 159 (∑ 1,240) $0.04197

TRACE NLSQLTableQueryEngine
12/7/2024, 5:18:41 PM
4.12s 1,081 → 159 (∑ 1,240) ∑ $0.04197

Input

```
{
  str_or_query_bundle: "SELECT track.name, SUM(invoice_line.unit_price * invoice_line.quantity) AS revenue
  FROM track
  JOIN invoice_line ON track.track_id = invoice_line.track_id
  GROUP BY track.name
  ORDER BY revenue DESC
  LIMIT 1;"
}
```

Output

The track that generated the highest revenue is 'The Trooper' with a total revenue of $4.95.

Metadata

```
{
  metadata: {
    result: [
      0: [
        0: "The Trooper"
        1: "<Decimal>"
      ]
    ]
```

```
TRACE NLSQLTableQueryEngine
4.12s $0.04197
  SPAN BaseQueryEngine.query
  4.12s $0.04197
    SPAN BaseSQLTableQueryEngine._query
    4.12s $0.04197
      SPAN LLM.predict
      3.08s $0.03456
        GENERATION OpenAI.chat
        3.08s 922 → 115 (∑ 1037) $0.03456
      SPAN DefaultSQLParser.parse_response_to_sql
      SPAN BaseSynthesizer.synthesize
      1.00s $0.00741
        SPAN CompactAndRefine.get_response
        1.00s $0.00741
          SPAN TokenTextSplitter.split_text
          SPAN Refine.get_response
          1.00s $0.00741
            SPAN TokenTextSplitter.split_text
            SPAN DefaultRefineProgram.__call__
            1.00s $0.00741
```

D. Deferred Research

The use of embedding model semantic ranking for the retrieval portion in RAG was initially considered.

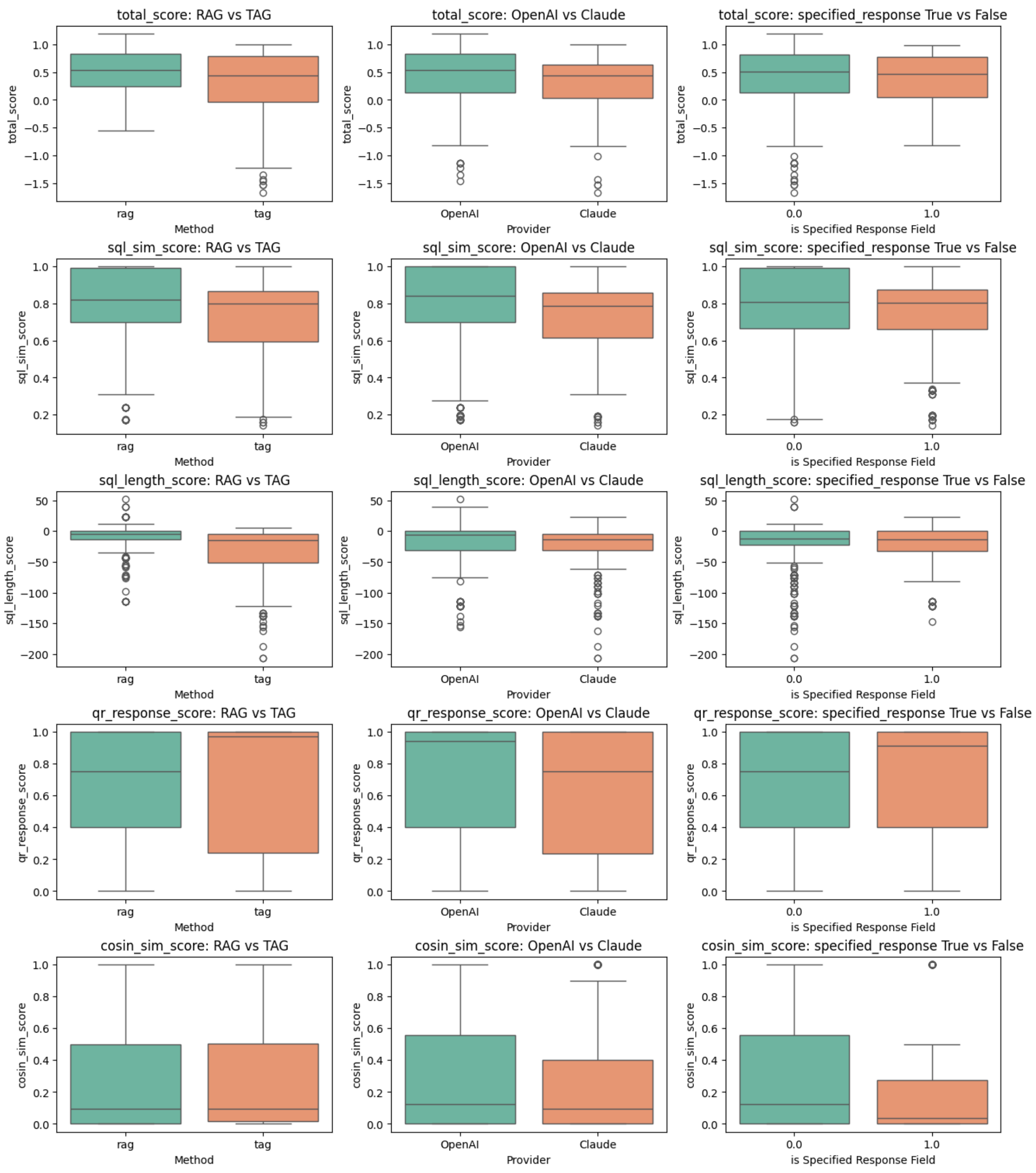
The Cross Encoder Rerankers utility depends on the complexity of the semantic matching required and the need for high-precision relevance assessment. They perform full attention between query and document tokens, enabling them to capture complex semantic relationships that might be missed by simpler retrieval methods.

In our project's context with only two documents used as context in the RAG method, the semantic matching capabilities of cross-encoders could still provide value in certain scenarios.

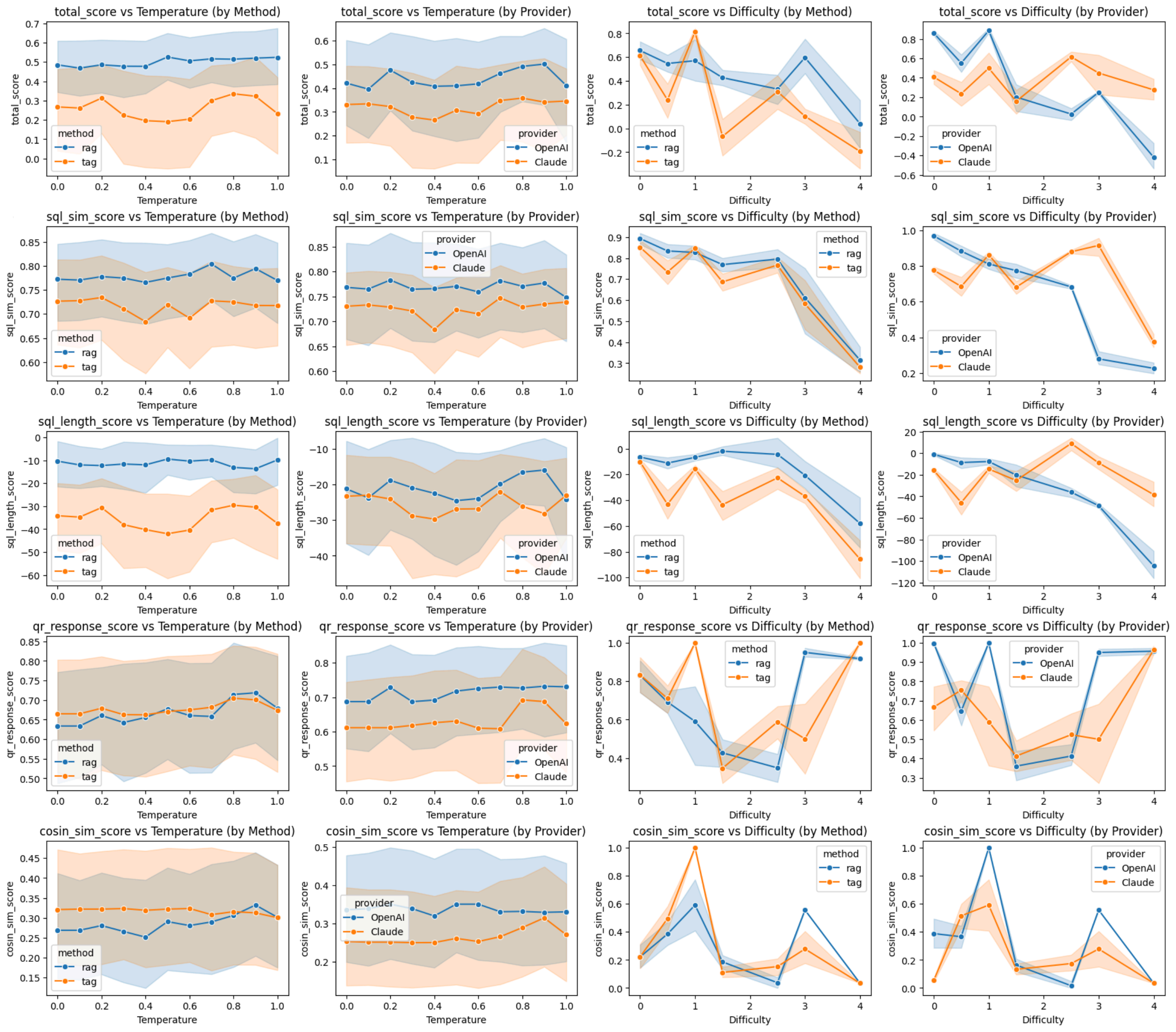
However, given the computational and storage overhead (for good models) and the limited document set, using LlamaIndex's default retrieval methods is likely sufficient, unless the specific requirements for extremely precise semantic matching or encounter cases where the default retrieval methods consistently fail to identify the most relevant document, which was not the case in our evaluation test cases.

E. More Evaluation Plots

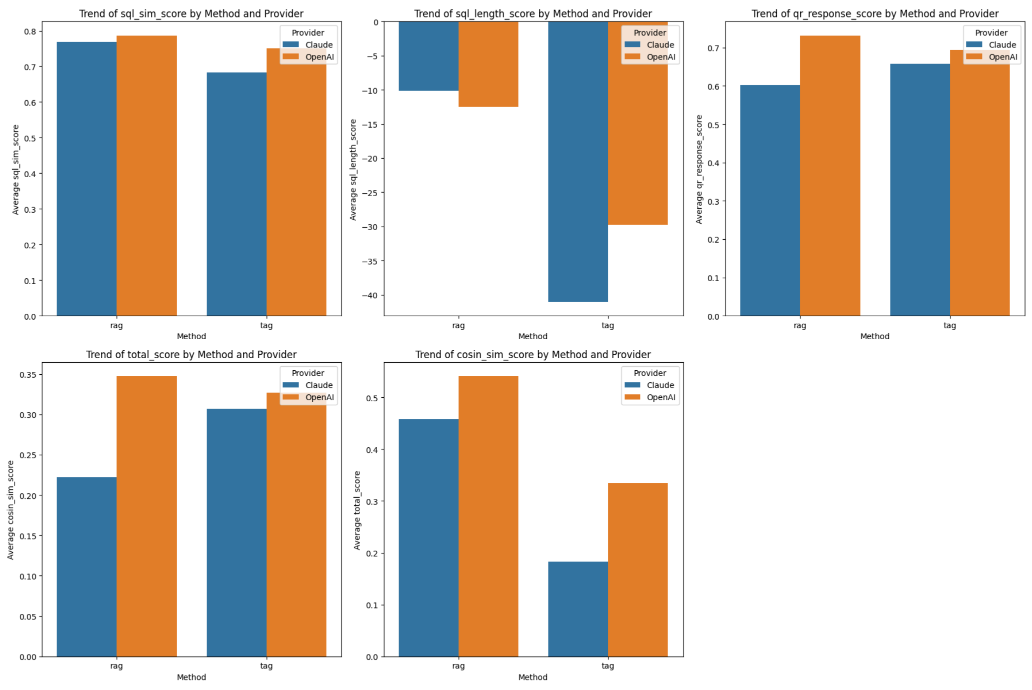
Marginal effects of Method or Provider upon Response Variables



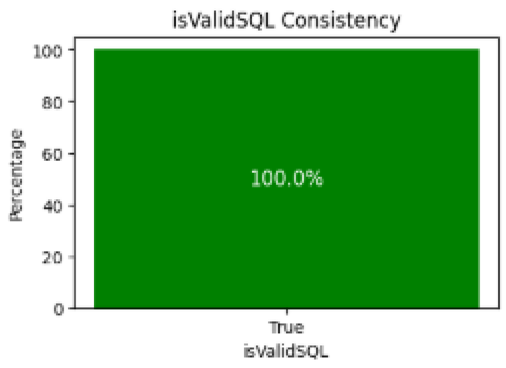
Change of Response Variables Over Changes in Temperature and Difficulty Within Provider and Methods Groups



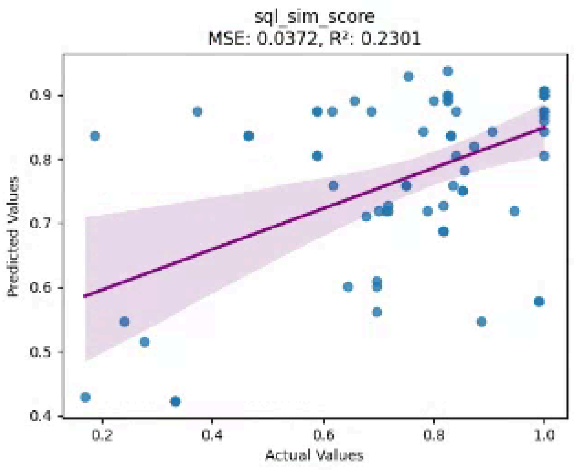
Trend of response variables by categories of predators



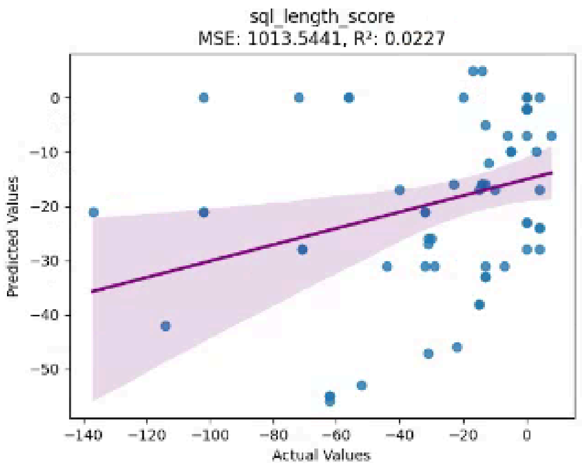
Generated SQL are all valid PostgreSQL SQL queries



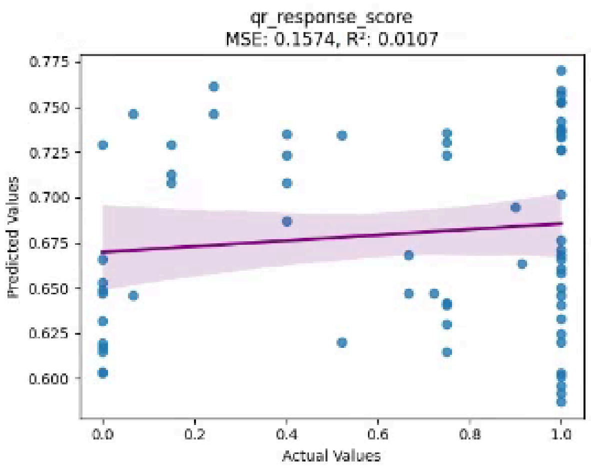
Regression plots



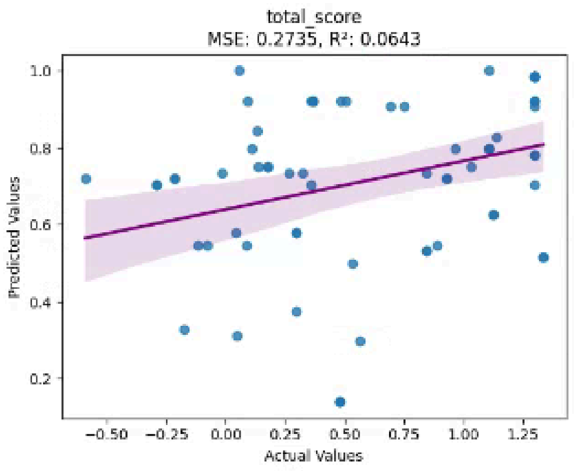
```
y = 41367956821974.15
(459360099042.89) * method_rag + (459360099042.85) * method_tag
(-41827316921016.10) * provider_Claude + (-41827316921016.13) * provider_OpenAI
(-0.12) * difficulty + (-0.01) * temperature
```



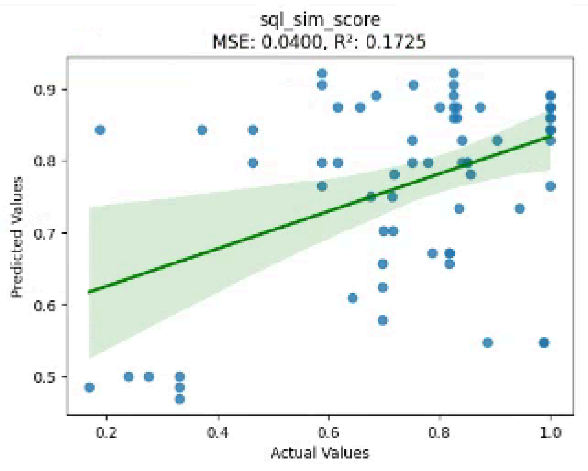
```
y = 7415334891784350.00
(82341726109064.59) * method_rag + (82341726109043.80) * method_tag
(-7497676617893410.00) * provider_Claude + (-7497676617893417.00) * provider_OpenAI
(-9.88) * difficulty + (0.28) * temperature
```

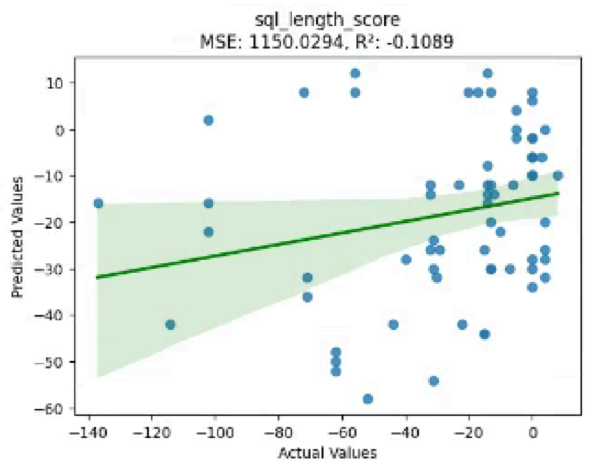


```
y = -3130446139429.08
(-34761253857.56) * method_rag + (-34761253857.54) * method_tag
(3165207393287.27) * provider_Claude + (3165207393287.37) * provider_OpenAI
(-0.02) * difficulty + (0.03) * temperature
```



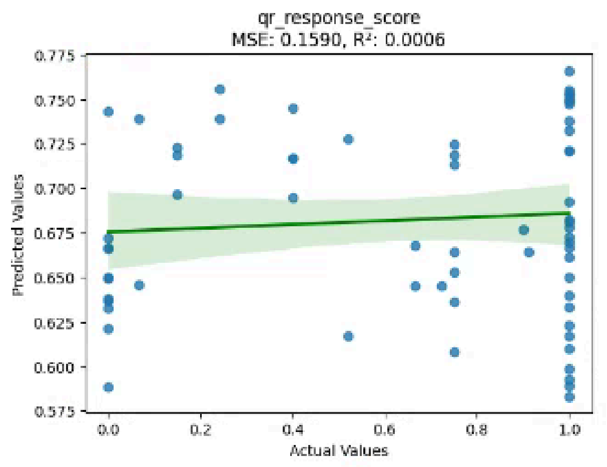
```
y = 83433289825006.80
(926464036945.95) * method_rag + (926464036945.76) * method_tag
(-84359753861951.75) * provider_Claude + (-84359753861951.77) * provider_OpenAI
(-0.16) * difficulty + (-0.00) * temperature
```

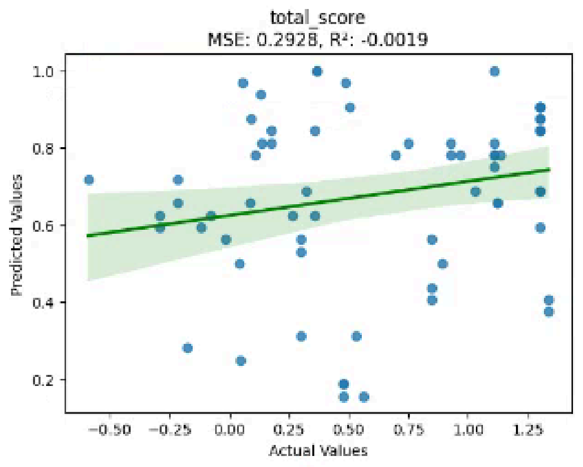




```
y = -720743112104 56.56
(-4039767361925.70) * method_rag + (-4039767361925.74) * method_tag
(76114078572383.20) * provider_Claude + (76114078572383.16) * provider_OpenAI
(-0.12) * difficulty + (-0.05) * temperature
(0.06) * specified_response_field
```

```
y = -126418890493537 08.00
(-7005782703560 10.00) * method_rag + (-700578270356031.00) * method_tag
(1335046731971053 2.00) * provider_Claude + (1335046731971051 6.00) * provider_OpenAI
(-11.66) * difficulty + (-7.70) * temperature
(6.61) * specified_response_field
```





```
y = 1117425381189.50
(62631727012.03) * method_rag + (62631727012.05) * method_tag
(-1180057108200.90) * provider_Claude + (-1180057108200.81) * provider_OpenAI
(-0.02) * difficulty + (0.03) * temperature
(0.02) * specified_response_field
```

```
y = -146923758475484.69
(-8235081185133.80) * method_rag + (-8235081185134.00) * method_tag
(155158839660619.50) * provider_Claude + (155158839660619.41) * provider_OpenAI
(-0.19) * difficulty + (-0.06) * temperature
(0.13) * specified_response_field
```

F. Mathematical Formulas

The mathematical notation of **RAG** is: $RAG(q) = gen(q, R(q))$

$R(q) = topk(sim(e(q), e(di)))\ for\ di \in D$

Where:

- q is the input query
- R(q) is the retrieval function.
- gen() is the generation function
- e() is the embedding function
- D is the knowledge base/document collection
- di represents individual documents/chunks
- sim() is the similarity function
- topk selects the k most relevant results

The mathematical notation of **TAG** is: $TAG(R) = gen(exec(syn(R)))$

syn(R) $\rightarrow$ Q
exec(Q) $\rightarrow$ T
gen(R,T) $\rightarrow$ A

Where:

- R is the natural language request
- Q is the generated database query
- T is the retrieved table data
- A is the generated answer
- syn() is the query synthesis function
- exec() is the query execution function
- gen() is the answer generation function

The mathematical notation of **Temperature** is: $P(xi) = exp(zi/T) / \Sigma j\ exp(zj/T)$

Where:

- T is the temperature parameter
- zi represents the logit score for token i
- tokens j represent all possible tokens in the model’s vocabulary
- zj represents all logit scores in the vocabulary
- Σj represents the sum over all possible tokens j
- P(xi) is the probability of selecting token i

Key characteristics of temperature values:

- $T = 0$: Deterministic, the model always selects the token with the highest probability.
- $0 < T < 1$: Outputs are more focused and conservative, reducing randomness.
- $T = 1$: Standard softmax probabilities, balancing diversity and focus.
- $T > 1$: Increases randomness and diversity, not suitable when looking for accuracy.

H. Feedback Incorporation

To meet the specification to incorporate feedback from the teaching team and classmates, we adopted Laura's feedback about choosing an multi-class classification model over a binary one.

Another suggestion from a classmate was to use charts instead of text for presenting the answers. However, we decided against this option as our primary focus was on evaluating the accuracy of the SQL generation statements rather than enhancing the presentation, which would have required additional development time by building a visualization engine. Nonetheless, it was a valuable suggestion.