Building Natural Language Interface to Databases using Large Language Models and LangChain to evaluate Business Intelligence Applications

Chelcie De Almeida
Northwestern University
Chelciealmeida2023@u.northwestern.edu
https://github.com/ChelcieDeAlmeida/JinnyDB

Executive Summary

This research investigates the potential of leveraging Large Language Models (LLMs), specifically OpenAI's GPT 3.5 Turbo, through the LangChain framework to create a modern Natural Language Interface to Databases (NLIDB) application. The objective is to autonomously handle business and data queries by converting natural language into SQL, enhancing efficiency. The study assesses GPT 3.5 Turbo's effectiveness in executing SQL queries on AdventureWorks and Bookings databases. Initial results demonstrate a 75% accuracy rate with notable challenges in contextual understanding, semantic complexity, and complex query handling. GPT 3.5 Turbo's strengths lie in its precision and potential for real-time interactions, while future work could focus on reinforcing domain knowledge, refining prompts, addressing data modeling challenges, and exploring hybrid approaches. In summary, this research showcases LLMs' potential in reshaping NLIDB systems while highlighting areas for improvement in understanding and handling complex queries for enhanced business value.

Abstract

This research investigates the potential of utilizing Large Language Models (LLMs), specifically OpenAI's GPT 3.5 Turbo to develop a modern Natural Language Interface to Databases (NLIDB) application via the LangChain framework. Through the transformation of natural language to Structured Query Language (SQL), the aim is to autonomously facilitate business and data requests of varying complexity, enhancing accessibility and efficiency. The literature review traces the evolution of NLIDB and highlights the recent advancements in LLMs, especially in Text-to-SQL tasks, emphasizing their contributions to database interaction and semantic search. Using the AdventureWorks and Bookings databases for evaluation, the paper demonstrates the effectiveness and initial shortcomings of GPT 3.5 Turbo in executing SQL queries across a range of complexities. The evaluation results, which are categorized by action, thought, and observation, contribute to the broader discourse on the capabilities and limitations of LLMs in database interfaces. The findings of this research can potentially unblock both customers and data experts by automating data requests.

Keywords Natural language Processing (NLP) • Text-to-SQL • Natural Language Interface to Database (NLIDB) • LangChain • Structured Query Language (SQL) • LLMs • GPT 3.5

Problem Statement & Introduction

In the absence of a dedicated data team, employees engaged in business activities often lack the requisite skillset to extract data that holds the potential for actionable insights. The development of a Natural Language Interface to Database (NLIDB) system could mitigate the influx of data requests directed at analytics teams and empower business operations to swiftly access data, thereby addressing challenges that demand enhanced comprehension. Currently, the construction of such systems is beset with challenges, primarily attributed to the intricate nature of comprehending, and processing human language within the framework of database queries. Through the utilization of Large Language Models (LLMs) and LangChain, our endeavor is to assess the feasibility of employing intelligent tools in the creation of an NLIDB system characterized by a high degree of precision.

Database systems as we know it have experienced several technological advancements and iterations since their inception in 1970. As organizational challenges grew in complexity, systems catapulted to answer some of the most urgent needs in the form of scalability, velocity, volume, access, and variety. Despite almost 60 years of evolution, businesses still turn to data professionals for their data requests because business users do not comprehend Structured Query Languages or SQL. LUNAR – a Natural Language Interface to Databases (NLIDB) – was the first system built to query a database in natural language which returned tabular results. LUNAR used two repositories, one dedicated to chemical analysis and the other, to literature references using an Augmented Transition Network and procedural semantics. It is noted that the system had a 78% success rate which grew to 90% when errors were handled. Despite favorable outcomes, the system deprecated due to linguistic constraints as stated by (Neelu et al., 2011) in their paper titled *Natural Language Interface for Database: A brief review*. Since then, there

have been great strides in consumable artificial intelligence frameworks, and by extension, intelligent systems. Most notably, transformer architecture, on which popular NLP Large Language Models (LLMS) like ChatGPT, BARD, and Llama are built on top of. As such, the aim of this paper is to evaluate the effectiveness and efficacy of using LangChain – a framework built to create applications using LLMs – with OpenAI's GPT 3.5 to create a text-to-SQL backend to build a modern rendition of an NLIDB application. That is, a system that can provide business value in real time and unblock both the customer and data expert by facilitating requests that range in complexity autonomously.

Literature Review

There have been many propositions to build text-to-sql tools or suggestions on how to fine-tune for optimal performance. *Natural Languages Interface to Database: A brief overview* focuses on the significance of databases in our daily lives and the subsequent need for efficient and user-friendly interfaces. It discusses the evolution of Intelligent Database Systems (IDBS), emphasizing the importance of Natural Language Interface to Databases (NLIDB). The text suggests that the advent of NLIDB marks a significant step towards improving user interactions with databases (Neelu et al., 2011).

In contrast, *Towards Semantic Search* centers on the application of NLP and semantics in improving search capabilities. It posits that the intersection of machine learning, data, and task design is crucial to advancing the state of the art in NLP-based search. (Epaminodas et al., 2008) emphasizes the role of complex models and large volumes of annotated data in implementing machine learning techniques effectively.

SQL-PALM: IMPROVED LARGE LANGUAGE MODEL ADAPTATION FOR TEXT-TO-SQL presents the emergence of LLMs in code generation, including Structured Query

Language (SQL). The text introduces SQL-PaLM, a Text-to-SQL model, which has shown promising results in both in-context learning and fine-tuning settings. It highlights the effectiveness of execution-based self-consistency prompting approach in achieving superior test-suite accuracy and robustness across multiple variants of Spider Challenge benchmark which is large scale semantic parsing and text-to-sql dataset with multiple complexities as outlined by (Ruoxi et al., 2023).

Evaluating the Text-to-SQL Capabilities of Large Language Models offers an empirical evaluation of the Text-to-SQL capabilities of the Codex language model. The text suggests that Codex serves as a strong baseline for the Spider benchmark, even without any fine-tuning. Interestingly, (Rajkumar et al.,2022) finds that the addition of a small number of in-domain examples in the prompt can significantly improve the Codex's performance, surpassing state-of-the-art models [6].

In their white paper titled PHOTON: A Robust Cross-Domain Text-to-SQL System, Salesforce Research and The Chinese University of Hong Kong describe the PHOTON system, a Natural Language Interface to Databases (NLIDB), designed to facilitate user access to relational data. It addresses challenges arising from differences between natural language and programming, including ambiguous questions and queries outside the system's semantic scope. PHOTON comprises a strong neural semantic parser, a human-in-the-loop question corrector, a SQL executor, and a response generator. Notably, the question corrector utilizes a neural sequence editor to identify confusion spans in input questions and suggests rephrasing to achieve translatable queries. The system's efficacy is demonstrated through experiments on simulated data, showcasing its ability to enhance the robustness of text-to-SQL systems against challenging user input (Zeng et al., 2020).

The collection of sources discussed in this literature review collectively contribute significant insights to the endeavor of using Large Language Models (LLMs) to construct Natural Language Interface to Database (NLIDB) systems. These sources underscore the importance of improving user interactions with databases through efficient and user-friendly interfaces. The evolution of Intelligent Database Systems (IDBS) and the subsequent emergence of NLIDB are highlighted as key steps in enhancing this interaction (Neelu et al., 2011). Furthermore, the intersection of machine learning, data, and task design is emphasized as pivotal for advancing NLP-based search capabilities (Epaminodas et al., 2008).

The role of LLMs, particularly in code generation and Structured Query Language (SQL), is elucidated through studies such as SQL-PALM, which introduces execution-based self-consistency prompting as an effective technique for improving the accuracy and robustness of Text-to-SQL models (Ruoxi et al., 2023). Additionally, the evaluation of the Codex language model's Text-to-SQL capabilities showcases its potential as a strong baseline for the Spider benchmark, even without fine-tuning. The significance of in-domain examples in enhancing Codex's performance is demonstrated, further emphasizing the potential of LLMs in this domain (Rajkumar et al., 2022).

The practical application of NLIDB systems is exemplified by the PHOTON system, designed to address challenges in translating natural language queries to database queries. Through its neural semantic parser, human-in-the-loop question corrector, SQL executor, and response generator, PHOTON enhances the robustness of text-to-SQL systems against challenging user input (Zeng et al., 2020). This collection of sources collectively illuminates the potential of LLMs in building NLIDB systems that bridge the gap between natural language

communication and database queries, contributing to more efficient and effective interactions with databases.

Data

The research for this experiment comprised of two databases, AdventureWorks and a demo Bookings database that were both curated for use with Postgres SQL. AdventureWorks was originally built by Microsoft but has since been deprecated. Developer, Lorin Thwaits, revived AdventureWorks to its original structure. AdventureWorks contains 68 tables that model a fictitious bike parts wholesaler distributed across 5 schemas that resemble the components of an end-to-end business i.e., Human resources, Production, Purchasing, Sales, and Person. To simplify this research, and considering token constraints of GPT 3.5-16K, the AdventureWorks database was migrated to a single schema called Prod which is where part of the following analysis as illustrated in this paper was conducted.

The Bookings demo database was built by Postgres Pro and comprises of a single schema with 8 tables and four views that simulate flight data for three months. The main entity is the bookings table which joins to a tickets – and by extension – ticket flights, flights, airports, aircrafts, seats, and boarding passes objects as shown in Figure 1A. While the Airlines database is smaller by design, most of the tables hold between 60,000-2,000,000 records as shown in Figure 1B below.

table_schema name	table_name name	rows_n integer
bookings	ticket_flights	2360335
bookings	boarding_passes	1894295
bookings	tickets	829071
bookings	bookings	593433
bookings	flights_v	65664
bookings	flights	65664
bookings	seats	1339
bookings	routes	710
bookings	airports	104
bookings	airports_data	104
bookings	aircrafts_data	9
bookings	aircrafts	9

(Figure 1B: Bookings db table count)

During preliminary analysis, LangChain's SQL Parser – which uses the SQLAlchemy library – outputted error messages regarding the 'Coordinates' column that is stored in the airports_data, airport, and routes tables due to its Point data type which is generally better handled in geospatial specific databases such ArcGIS and PostGIS. To ensure we bypassed this initial hurdle, the Coordinates column was converted to a text data type. It's important to note that none of the test cases prepared for this analysis are dependent on the Coordinates column except that its treatment was a necessity to move forward with this research without further incident.

Methods

The evaluation of this experiment is based on OpenAI's GPT 3.5 Turbo which is generative language model trained on countless sources on the internet. The model will have its temperature set to 0 and 0.2 to lower the chances of randomness to produce more predictable outputs as the higher the temperature, the more liberties the model will take in trying to generate responses. Given the nature of the experiment, precision is critical. We conducted this research on two databases that vary in complexity and size to model real world scenarios where mature organizations consist of large data repositories with a diverse set of database objects. By contrast, smaller or newer organizations will have a much more tamed data infrastructure in comparison. In both use cases, the availability of a data team or individuals is crucial to propelling business objectives forward.

To determine efficacy, a list of prompts (Figure 2A-B) that range from easy to hard levels of complexity have been prepared with accompanying SQL scripts that were written to validate in Postgres against the output each model will reproduce. Photon – one of the most accurate prototyped NLIDBs developed by Salesforce Research and The Chinese University of Hong Kong – used a method called Static SQL Correctness Check as stated in their white paper called *PHOTON: A Robust Cross-Domain Text-to-SQL System* – to validate the system's output against expected results. This quality check satisfies the following criteria:

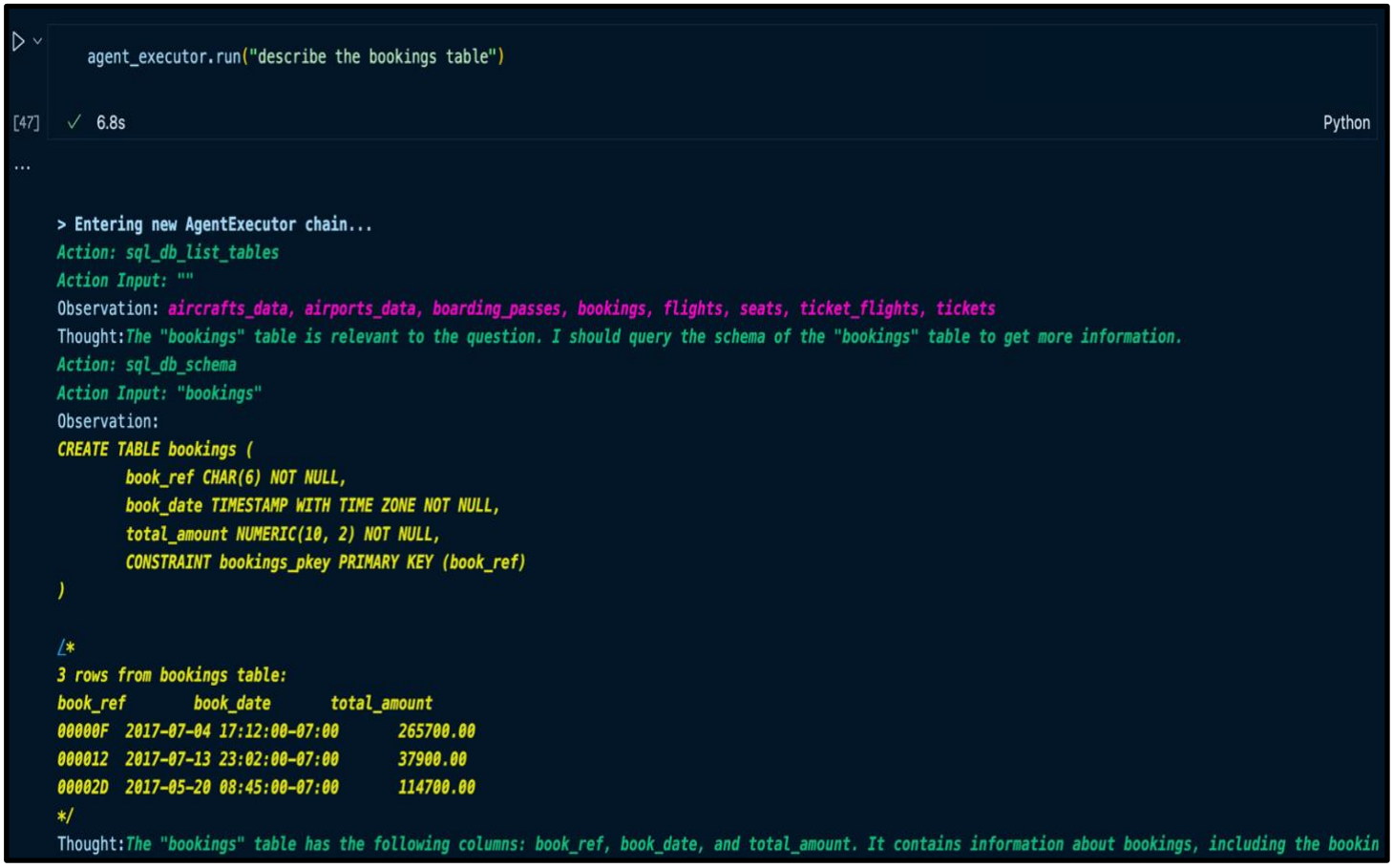
1. The SQL Query is syntactically correct.
2. The SQL query satisfies schema consistency. (J. Zeng et. al, 2020).

We conducted similar quality check exercises on both, the AdventureWorks and Bookings databases and recorded the results. It is important to note that complexity, as determined in this experiment, is subjective and modeled based on the thought process an analyst or data scientist

would take to produce the data results being asked of them. Furthermore, these test cases were designed to meet a similar standard and criteria to that of the Spider Challenge. The spider challenge as stated on their site is:

> Spider is a large-scale *complex and cross-domain* semantic parsing and text-to-SQL dataset annotated by 11 Yale students. The goal of the Spider challenge is to develop natural language interfaces to cross-domain databases. It consists of 10,181 questions and 5,693 unique complex SQL queries on 200 databases with multiple tables covering 138 different domains. In Spider 1.0, different complex SQL queries and databases appear in train and test sets. To do well on it, systems must *generalize well to not only new SQL queries but also new database schemas* (YALE-LILLY,2020).

Finally, both models are passed through LangChain's SQL agent which evaluates the prompt provided by the user to produce observations by way of *action*, *thought* and *observations*. *Action* denotes the executable step LangChain will have to take to understand the prompt i.e., list tables. *Observation* will return a list of tables, while *thought* is represented in contemplation like that of humans i.e., "the 'Bookings' table is relevant to the question. I should query the schema of the 'bookings' table to get more information" as illustrated in Figure 3 below.



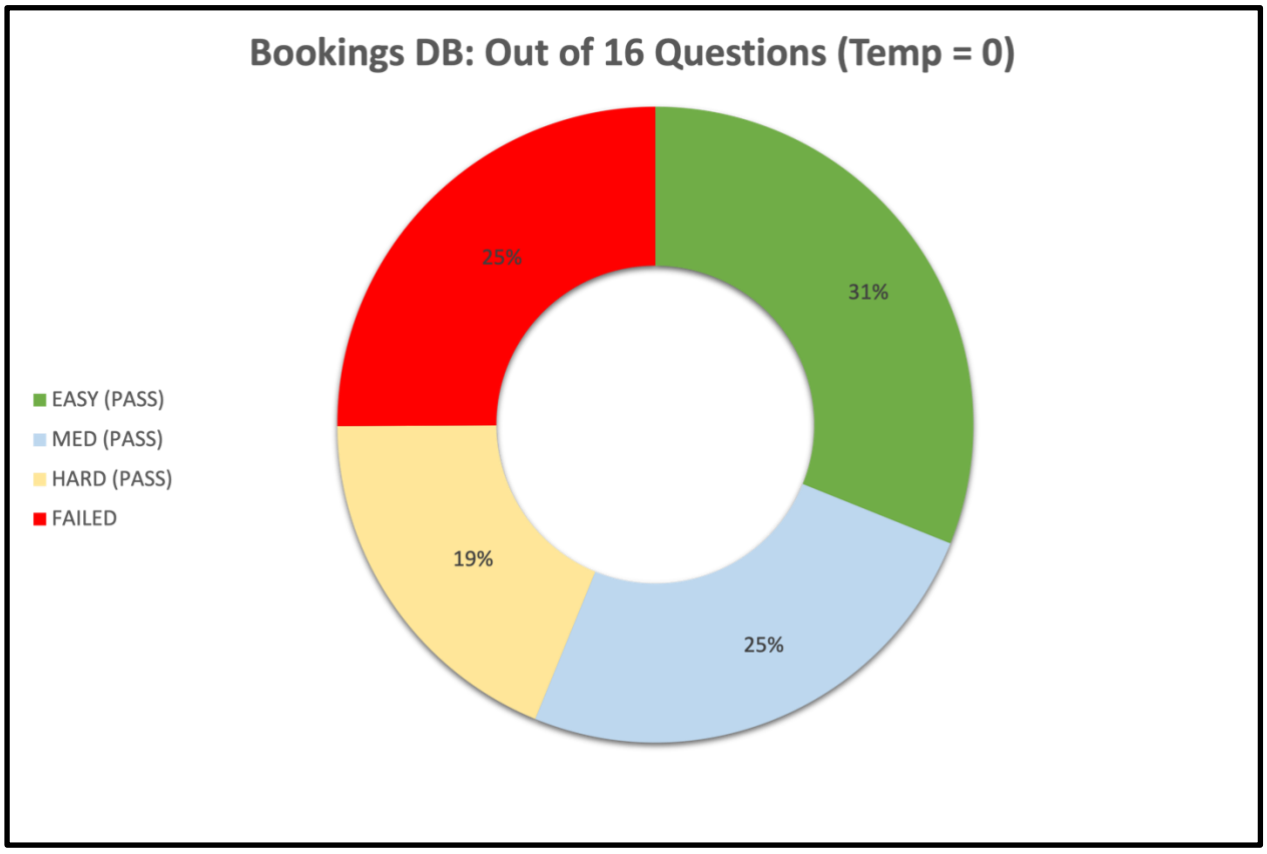
(figure 3: Langchain's logical process)

Results

As stated earlier in the paper, this experiment was conducted on questions that vary in degrees of complexity and while not as extensive as the spider challenge, the following results will be used to compare against spider's leadership board which is currently the Text-to-SQL authority on effective and potentially implementable NLIDB systems. While the spider benchmark evaluates against a number of test cases to determine a model's robustness, this

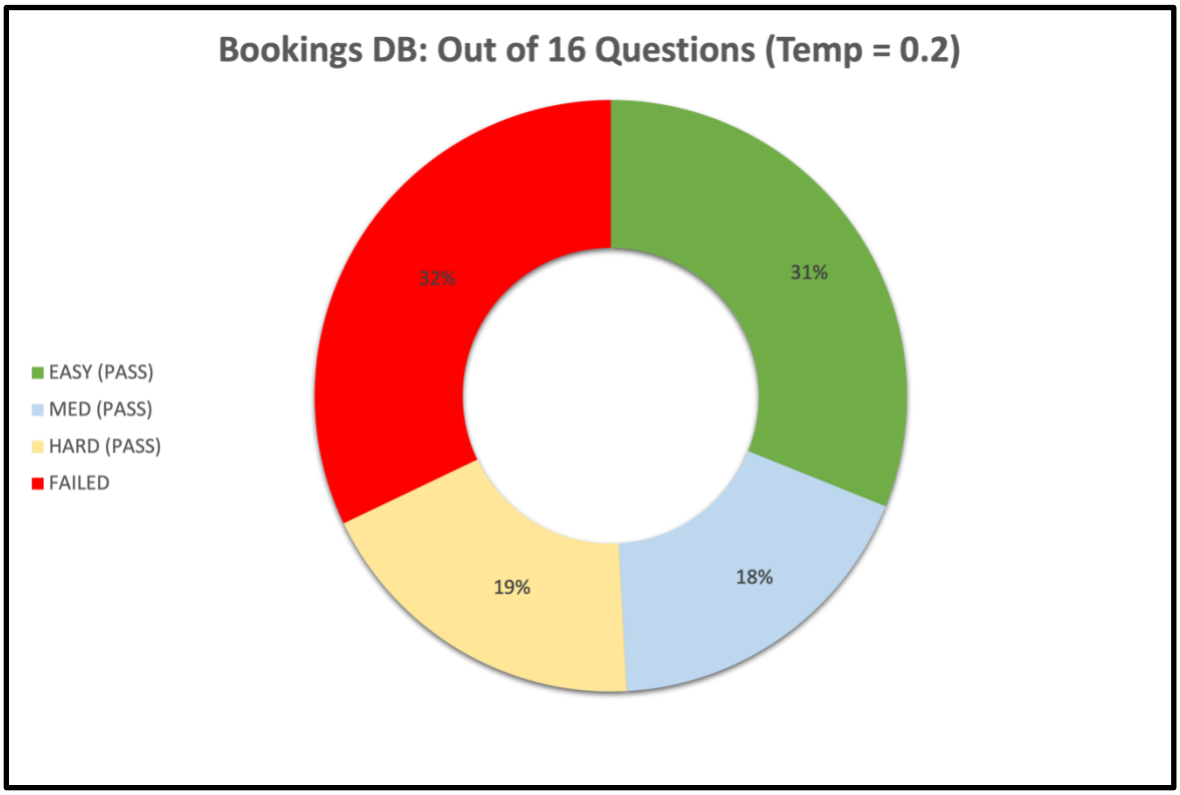
research will simply focus on the accuracy of the SQL queries GPT produces in comparison to previously designed queries and expected outputs. First, we'll examine the results produced using the Bookings database, followed by the results using AdventureWorks.

Using the Bookings database with the model temperature set to 0, the GPT 3.5 model got a 75% overall score or 12 out of 16 questions correctly as shown below in figure 4. More specifically, 5 out of the 6 easy questions passed the test cases which is 31% of the overall question set. 4 out of the 6 medium questions passed the test cases which is 25% of the overall question set. 3 out of the 4 hard questions passed the test cases which is 19% of the overall question set.



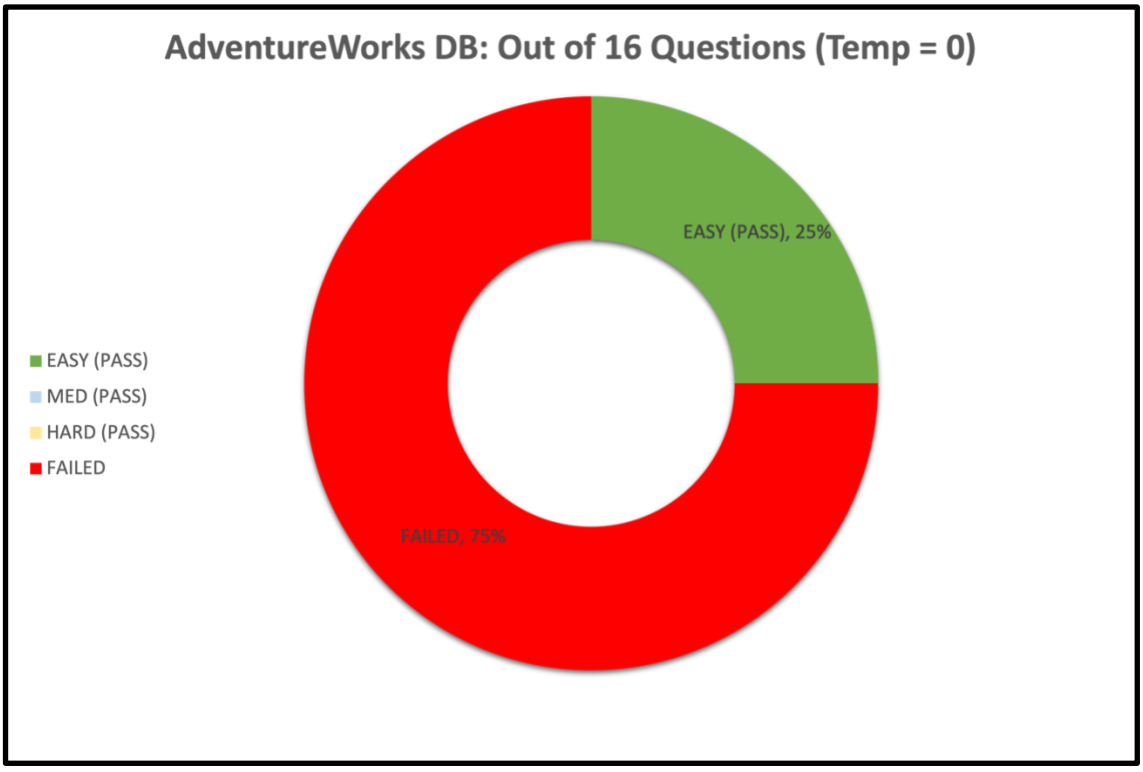
(Figure 4: Bookings DB 16 Question test case at model temp = 0)

The overall score dropped to 69% which is 11 out of 16 passes once the model temperature was set to 0.2. 5 out of the 6 easy questions passed the test cases which is 31% of the overall question set. 3 out of the 6 medium questions passed the test cases which is 18% of the overall question set. 3 out of the 4 hard questions passed the test cases which is 19% of the overall question set as shown below in Figure 5.

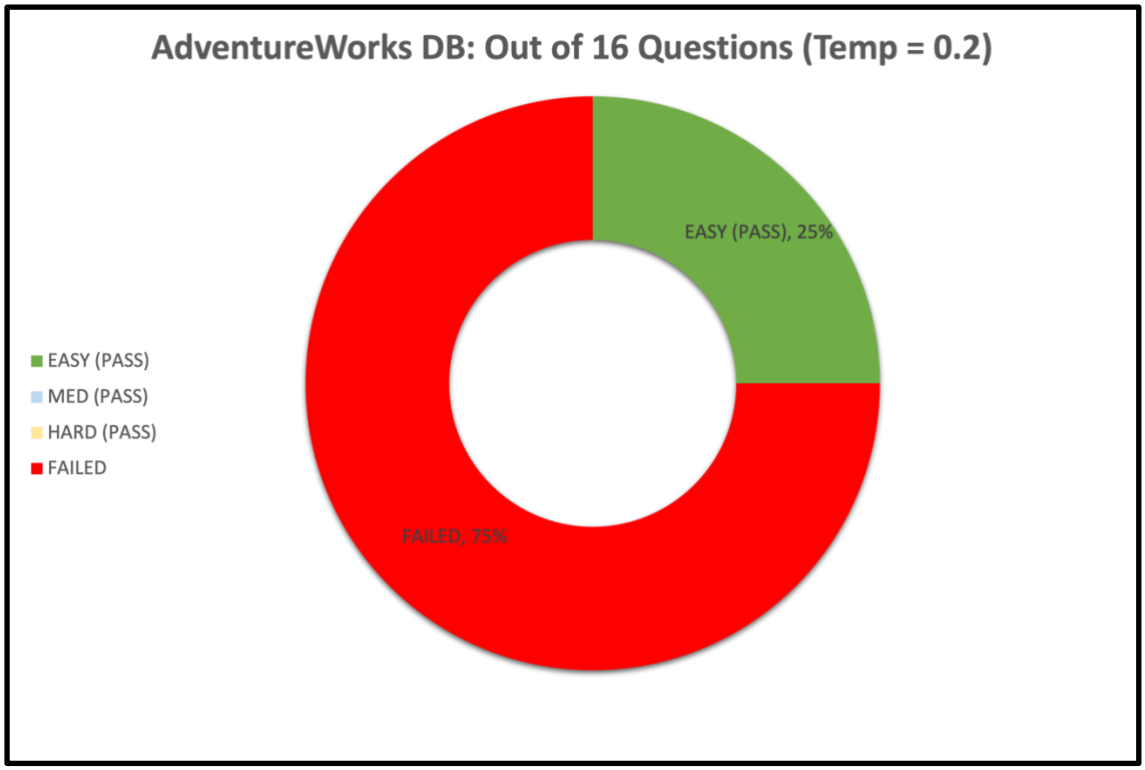


(Figure 5: Bookings DB 16 Question test case at model temp = 0.2)

Conversely, the AdventureWorks database performed the same in both instances of the test cases irrespective of model temperature as shown below in Figure 6 and 7. Specifically, 4 out of the 6 easy questions passed the test cases while the remaining questions failed the test cases.



(Figure 6: AdventureWorks DB 16 Question test case at model temp = 0)



(Figure 7: AdventureWorks DB 16 Question test case at model temp = 0.2)

Analysis and Interpretation

I interpreted the outcome of the results, or generally the failed test cases, and categorized them based on 3 main challenges: Contextual Understanding, Semantic Complexity, and Handling Complex Queries. While NLIDB systems face many more drawbacks, the 3 challenges seem to be the biggest issues during this experiment. That is, contextual understanding is the implicit context that must be inferred for accurate interpretation. Semantic complexity is the semantic meaning of words and phrases within the context of the database query requires a deep understanding of a word can mean something different to the database and another to the language used. Finally, handling complex queries can involve complex operations such as aggregations, joins, and subqueries. Translating these operations from natural language to SQL

requires advanced query generation mechanisms. There is a fourth challenge which is scalability and performance, but we'll reserve that explanation when we interpret the results of the AdventureWorks test cases.

Generally, GPT grasped concepts well. For instance, question 6 which asked it "How many distinct flights departed from Moscow?' GPT identified Airports that had the 'DME' airport code which is one of the airports based out of Moscow. The query composition was incorrect but the context and thought process were like that of analyst who is getting familiar with a database they have never worked before. Similarly, when asked "which cities have more than one airport? Return the airport code, airport name, and city," it used a table that had the data in a more simplified manner opposed to the query I had prepared that had to process the data in specific columns with jsonb data types. However, handling complex queries was a challenge for GPT as this question required a nested query or a combination of nested query, window function, and aggregate functions to work and it did not understand how it could accomplish the task. An example of both semantic complexity and query handling is once I asked the system to "get me the passenger's name, phone number, and email of the 10 passengers order by passenger name in ascending order. Make sure to only retrieve data of passengers that have an email." It ignored previously stored information and hallucinated tables and columns that didn't exist. Passenger contact information is stored as a jsonb in a column called "contact_data" which stores phone numbers and emails. GPT, however, was explicitly looking for columns called phone_number and email. Once it failed to find them, it began to hallucinate those objects in another table.

Finally, the AdventureWorks interpretation – based on this assessment – on performance and scalability. Most of the questions were catered around 5 tables as an attempt to minimize the amount of work GPT would have to do. However, because most questions didn't specify which

database object, we wanted GPT to apply its focus to, it scanned an enormous amount of information is an issue known to LLMs commonly known as context window. That is, the number of tokens a model can handle which in the case of GPT 3.5-16K, it's precisely 16,000 tokens. Once I asked GPT to describe the schema once, it returned the correct table names and columns. However, once I began going through each test case, it hallucinated most of the columns and tables. Albeit, close to the original naming conventions of specific tables such as 'department' vs 'departments', it was still hallucinated. This is why I think performance was the greatest hinderance to the AdventureWorks test cases. After all tests were run, we re-ran every failed by using prompt engineering and input refinement such pointing GPT to the right table, giving it column descriptions and code suggestion. It passed every test it had previously failed.

Conclusions and Directions for Future Work

In summation, this paper delves into the critical issue of enabling efficient data access and utilization in the absence of dedicated data teams. The challenge of extracting actionable insights from data often hinders business operations due to a lack of requisite skills among business-facing employees. The introduction of a Natural Language Interface to Database (NLIDB) system is proposed as a solution to alleviate the burden on analytics teams by facilitating rapid data retrieval and comprehension for addressing intricate business challenges.

The paper highlights the existing challenges in constructing NLIDB systems, primarily attributed to the complexity of translating human language into database queries. This complexity necessitates the deployment of advanced techniques, such as Large Language Models (LLMs) and LangChain, to enhance the precision and effectiveness of NLIDB systems. Through the

integration of LLMs, the aim is to explore the viability of intelligent tools in the creation of NLIDB systems that can autonomously respond to user queries.

The application of LangChain's SQL agent in conjunction with OpenAI's GPT 3.5 Turbo offers insights into the effectiveness of NLIDB systems, particularly in addressing challenges related to contextual understanding, semantic complexity, and complex query handling. The results highlight the model's performance across various levels of complexity and provide valuable insights into its strengths and limitations.

Overall, this paper advances the understanding of NLIDB systems and their potential to revolutionize the way businesses interact with data. By incorporating LLMs and advanced frameworks, the paper opens avenues for building NLIDB systems that are more accessible, efficient, and capable of autonomously responding to complex user queries. The experimental findings underscore the need for continuous refinement and development in this field, as NLIDB systems hold the promise of unlocking substantial business value and enhancing decision-making processes.

The direction of future works will focus on Training on the data its executing on, reinforcement learning, domain knowledge, and data modeling. As mentioned at the end of the interpretation and analysis section, after fine tuning and refining the prompts, GPT passed all the failed test cases which illustrates that once it learns useful information that pertains to the domain and data model, it can produce impressive results. It’s my estimation that context windows issues can be minimized if the model has existing information of the data its executing against therefore part of the future work will be to build a data dictionary to reinforce the model’s understanding of the domain it is querying against.

References

[1] Lorin Thwaits. 2016. AdventureWorks-for-Postgres. https://github.com/lorint/AdventureWorks-for-Postgres. (2023)

[2] Postgres Pro. NA. Demo Database “Airlines”. https://postgrespro.com/docs/postgrespro/10/demodb-bookings. (2023)

[3] Nihalani, Neelu, Sanjay Silakari, and Mahesh Motwani. *Natural language Interface for Database: A Brief review* 8, no. 2 (March 2011): 600–608. https://ijcsi.org/papers/IJCSI-8-2-600-608.pdf.

[4] Kapetanios, Epaminondas, Vijayan Sugumaran, and Myra Spiliopoulou. “Towards Semantic Search.” Essay. In Natural Language and Information Systems. Springer, 2008.

[5] Sun, Ruoxi, Sercan Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. SQL-PALM: IMPROVED LARGE LANGUAGE MODEL ADAPTATION FOR TEXT-TO-SQL, June 25, 2023. https://doi.org/arXiv:2306.00739v3.

[6] Rajkumar, Nitarshan, Raymond Li, and Dzmitry Bahdanau. Evaluating the Text-to-SQL Capabilities of Large Language Models, March 15, 2022. https://doi.org/arXiv:2204.00498v1 .

[7] Zeng, Jichuan, Xi Victoria Lin, Caiming Xiong, Richard Socher2, Michael R Lyu, Irwin King, and Steven C.H Hoi. “PHOTON: A Robust Cross-Domain Text-to-SQL System.” http://victorialin.net/, 2020. http://victorialin.net/pubs/photon-acl2020.pdf.

[8] Yale-LILY. "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task." Yale University - LILY. Accessed [August 24, 2023]. https://yale-lily.github.io/spider.

Appendix

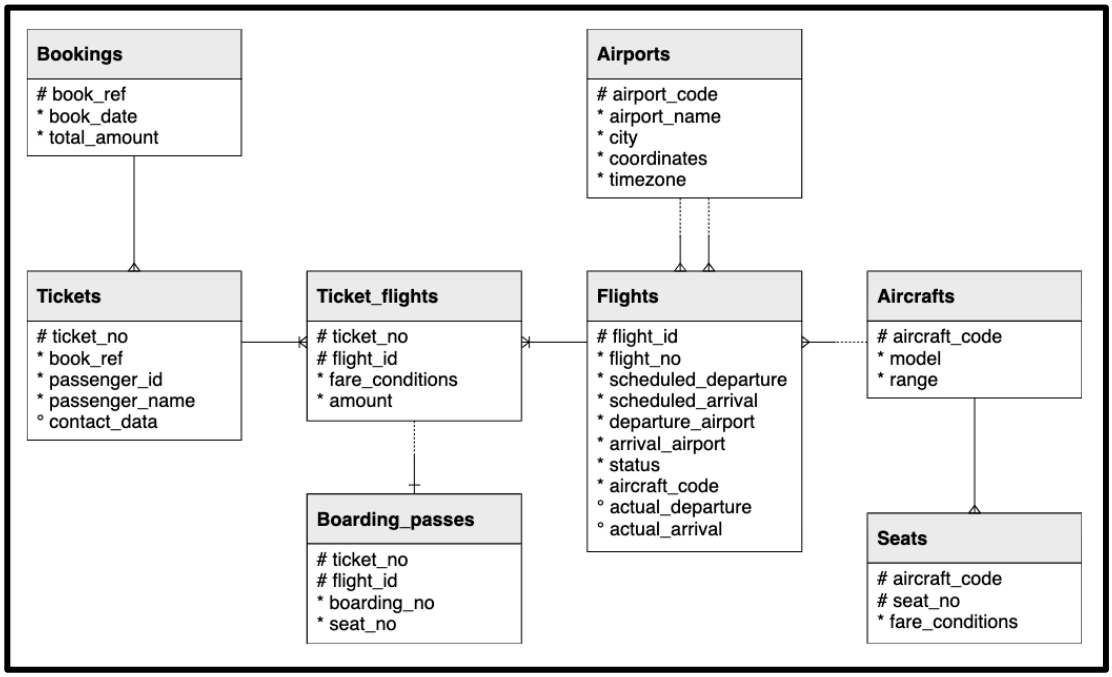


Figure 1A: Airlines Database Booking Schema Diagram

table_schema name	table_name name	rows_n integer
bookings	ticket_flights	2360335
bookings	boarding_passes	1894295
bookings	tickets	829071
bookings	bookings	593433
bookings	flights_v	65664
bookings	flights	65664
bookings	seats	1339
bookings	routes	710
bookings	airports	104
bookings	airports_data	104
bookings	aircrafts_data	9
bookings	aircrafts	9

Figure 1B: Airlines Booking Database Table/View record counts.

ID	Prompts	Complexity	Dataset
1	How many rows are in the bookings table	EASY	Bookings
2	How many rows are in the aircrafts table	EASY	Bookings
3	find the top ten most expensive flights and order total amounts in descending order	EASY	Bookings
4	How many ticket bookings does passenger Antonina Kuznecova have?	EASY	Bookings
5	How many ticket bookings does passenger Antonina Kuznecova have (testing for case sensitivity)?	EASY	Bookings
6	How many distinct flights departed from Moscow? (testing to determine if langchain can Identify db views)	EASY	Bookings
7	Get me the count of each flight status grouped by status and also provi	MEDIUM	Bookings
8	Get me the count of each flight status grouped by status. Return only t	MEDIUM	Bookings
9	Get me the list of seats and the fare conditions or class where those sea	MEDIUM	Bookings
10	Get me the passenger name that had the most flights that had a status	MEDIUM	Bookings
11	Get me the the passenger name, phone number, and email of 10 passe	MEDIUM	Bookings
12	Which cities have more than 1 airport? Return the airport code, airpo	MEDIUM	Bookings
13	Get me the ticket number, flight id ,fare conditions, seat number, boarding number, passenger name, and amount for flight 4686 where the status is Arrived. Order the results by amount in descending order	HARD	Bookings
14	Which flight segments are included into ticket number 00054326619	HARD	Bookings
15	Which flight segments are included into ticket number 00054326619	HARD	Bookings
16	Get me the aircraft code and the fare conditions along with each fare conditions count for each aircraft code. For example, aircraft code 319 would have a fare conditions of business(20), Economy(96) where 20 and 96 represent the number of seats in those conditions. The output should have the aircraft_code, and fare_conditions with their seat count in the same record.	HARD	Bookings

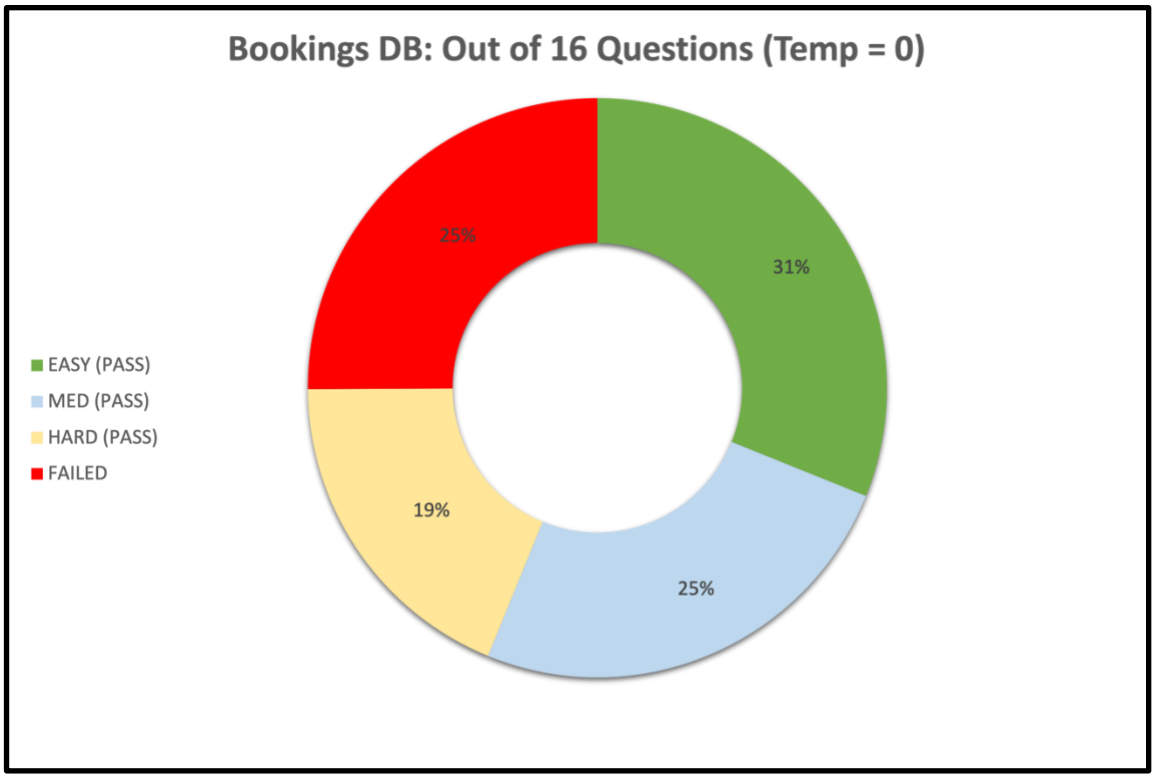
Figure 2A: Bookings Database Prompts with level of complexity (in revision)

ID	Prompts	Complexity	Dataset
1	How many rows are in the employee table	EASY	AdventureWorks
2	Get me the jobtitle, maritalstatus, gender, and vacationhours of the top 5 employees with the most vacation hours	EASY	AdventureWorks
3	Get me the jobtitle, maritalstatus, gender, and vacationhours of the top 5 employees with the least vacation hours	EASY	AdventureWorks
4	Get me the number of employees that are salaried	EASY	AdventureWorks
5	how many males and females are employees?	EASY	AdventureWorks
6	Get all the employee records for the 10 oldest employees and calculate their individual age	MEDIUM	AdventureWorks
7	How many years has the employee with the job title Chief Executive Officer been working at the company?	MEDIUM	AdventureWorks
8	How many employees that have Marketing in the job title are salaried?	MEDIUM	AdventureWorks
9	How many employees that have Marketing in the job title are not salaried?	MEDIUM	AdventureWorks
10	How many vacation hours, in days, do the top 10 salaried employees with the highest vacation hours have?	MEDIUM	AdventureWorks
11	How many vacation hours, in days, do the top 10 not salaried employees with the highest vacation hours individually have?	MEDIUM	AdventureWorks
12	How many vacation hours, in days, do the top 10 not salaried employees with the highest vacation hours individually have? Only return the Job title and vacation days	MEDIUM	AdventureWorks
13	How many employees does each department have? Return the department group name and employee count for each department group name	HARD	AdventureWorks
14	How many active employees does each department have? Return the department group name and employee count for each department group name	HARD	AdventureWorks
15	How many active employees work in each shift type? Return the count and shift type only	HARD	AdventureWorks
16	Which departments have active employees that work the night shift and how many employees are there?	HARD	AdventureWorks

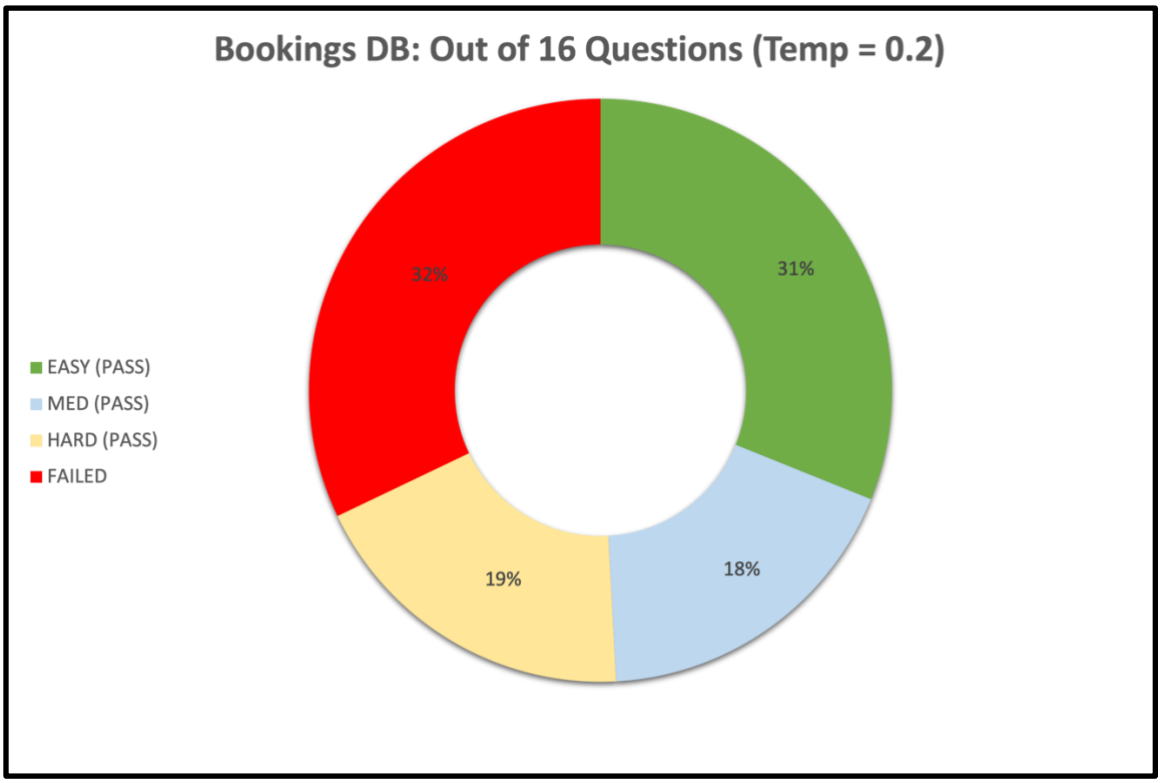
Figure 2B: AdventureWorks Database Prompts with level of complexity (in revision)

```
agent_executor.run("describe the bookings table")

[47]  ✓ 6.8s                                                    Python

...

> Entering new AgentExecutor chain...
Action: sql_db_list_tables
Action Input: ""
Observation: aircrafts_data, airports_data, boarding_passes, bookings, flights, seats, ticket_flights, tickets
Thought:The "bookings" table is relevant to the question. I should query the schema of the "bookings" table to get more information.
Action: sql_db_schema
Action Input: "bookings"
Observation:
CREATE TABLE bookings (
        book_ref CHAR(6) NOT NULL,
        book_date TIMESTAMP WITH TIME ZONE NOT NULL,
        total_amount NUMERIC(10, 2) NOT NULL,
        CONSTRAINT bookings_pkey PRIMARY KEY (book_ref)
)

/*
3 rows from bookings table:
book_ref        book_date       total_amount
00000F  2017-07-04 17:12:00-07:00       265700.00
000012  2017-07-13 23:02:00-07:00       37900.00
00002D  2017-05-20 08:45:00-07:00       114700.00
*/
Thought:The "bookings" table has the following columns: book_ref, book_date, and total_amount. It contains information about bookings, including the bookin
```

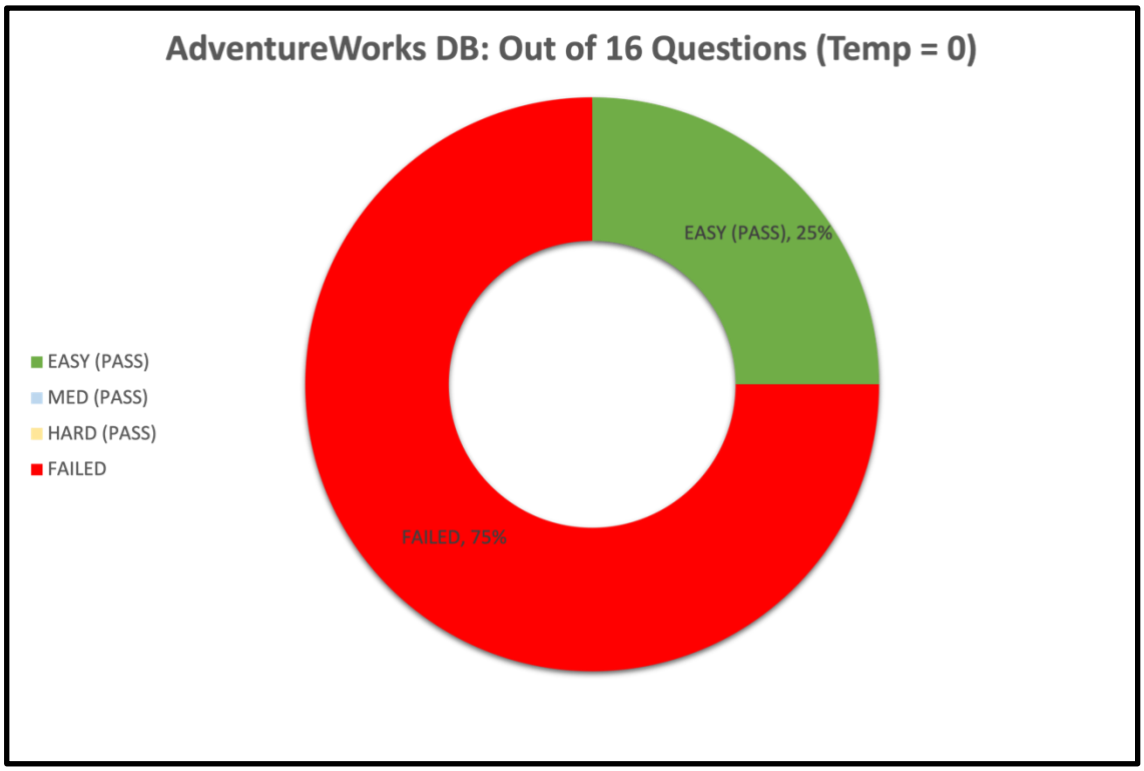
Figure 3: Langchain's logical process (Action, Observation, Thought) prompt evaluation



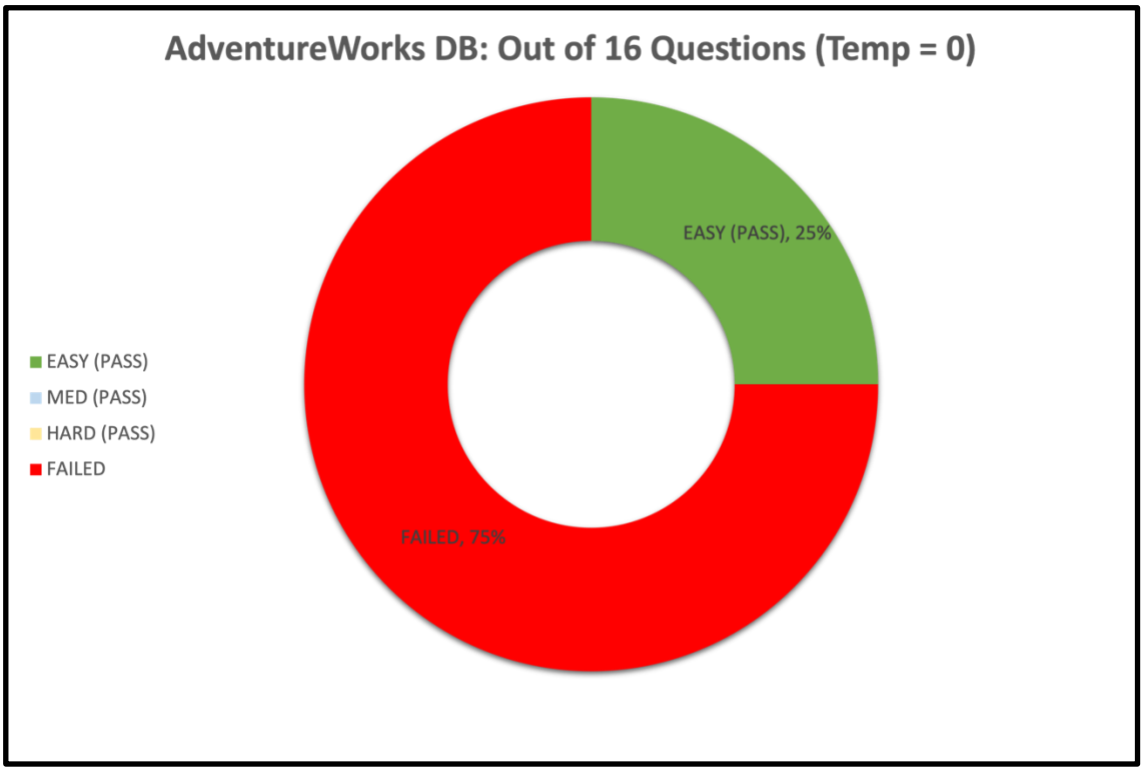
(figure 4: Bookings DB 16 Question test case at model temp = 0)



(figure 5: Bookings DB 16 Question test case at model temp = 0.2)



(Figure 6: AdventureWorks DB 16 Question test case at model temp = 0)



(Figure 7: AdventureWorks DB 16 Question test case at model temp = 0.2)