

SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries

Aleksander Theo Strand



Thesis submitted for the degree of
Master in Applied Computer and Information Technology (ACIT)
30 credits

Department of Computer Science
Faculty of Technology, Art and Design

OSLO METROPOLITAN UNIVERSITY

Spring 2024

SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries

Aleksander Theo Strand

© 2024 Aleksander Theo Strand

SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries

<http://www.oslomet.no/>

Printed: Oslo Metropolitan University

Abstract

The rapid growth and increasing availability of sports multimedia and metadata, demands advanced information retrieval systems capable of efficiently processing vast amounts of multimodal data. This paper introduces SoccerRAG, an innovative framework that leverages Retrieval Augmented Generation (RAG), SQL agents, and Large Language Models (LLMs) to extract soccer-related information through natural language queries. By utilizing a multimodal dataset, SoccerRAG enables database querying, automatic data validation, and enhanced user interaction, making sports archives more accessible. Our evaluations demonstrate that SoccerRAG effectively handles complex queries and improves the accuracy of traditional natural language to SQL query, systems. The results highlight the potential of RAG, agents, and LLMs in sports analytics, paving the way for future innovations in the accessibility and real-time processing of sports data.

Acknowledgments

I would like to express my heartfelt gratitude to my supervisors at SimulaMet, Pål Halvorsen, Sushant Gautam and Cise Midoglu for always having the time to discuss interesting topics and challenging my ideas and implementations. This work is a testament to their mentorship, and I'm forever grateful for their guidance and support. I would also like to thank my girlfriend and family for putting up with me during this stressful time of research, writing, and job hunting.

Contents

Abstract	i
Acknowledgments	iii
1 Introduction	1
1.1 Motivation	1
1.2 Problem Statement	2
1.3 Scope	3
1.4 Research Methods	4
1.5 Ethical Considerations	5
1.6 Main Contributions	5
1.7 Thesis Outline	6
2 Background and Related Work	7
2.1 Terminology	7
2.2 Soccer Basics	9
2.3 SoccerNet Dataset	9
2.4 Large Language Models	10
2.4.1 Multimodality	12
2.4.2 Information Retrieval	13
2.4.3 LangChain	16
2.4.4 ChainLit	16
2.5 Chapter Summary	17
3 Methodology and Implementation	19
3.1 Data Management	20
3.1.1 Source Dataset	20
3.1.2 Context Retrievers	25
3.1.3 Database	29
3.2 Proposed Pipeline	32
3.2.1 Choosing a Language Model	34

3.2.2	Feature Extraction	35
3.2.3	Feature Validation	37
3.2.4	SQL Agent	43
3.3	Proposed Evaluation	46
3.3.1	Metrics for Evaluation	46
3.3.2	Queries for Evaluation	47
3.4	Proposed User Interface	49
3.5	Chapter Summary	53
4	Experiments and Results	55
4.1	Query Complexity Analysis	55
4.2	Per-Component Evaluation	57
4.2.1	Extractor-Validator	57
4.2.2	SQL Agent and RAG	58
4.2.3	Ablation Study	60
4.3	End-to-End Pipeline Evaluation	61
4.3.1	Execution Time	61
4.3.2	Cost	61
4.3.3	Robustness	62
4.3.4	Model Laziness	63
4.3.5	LLM capability	63
4.4	Chapter Summary	63
5	Discussion	65
5.1	Competitor Analysis	65
5.2	Model Laziness	67
5.3	Importance of Cleaned Input	68
5.4	Modular Implementation of Extractor-Validator Module	68
5.5	New Models	69
5.6	Potential Use Cases	69
5.6.1	SoccerNet Use Cases	70
5.6.2	Natural Language SQL Retrieval	70
6	Conclusion	73
6.1	Insights	73
6.2	Addressing The Research Questions	74
6.3	Limitations and Future Work	76
6.4	Contributions	77

A Results	83
A.1 Ablation Study Results	83
A.2 Extractor - Validator Results	91
A.3 SQL Complexity Calculations	100
A.4 Execution Time	110
A.5 SQL Rag	116
B Publications	129
B.1 SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries	130
B.2 Demo: Soccer Information Retrieval via Natural Queries using SoccerRAG	138

List of Figures

3.1	Pipeline Concept	19
3.2	Directory Structure in Dataset	20
3.3	Dataset - Event Type Distribution from Labels-caption.json	22
3.4	Dataset - Event Type Distribution from Labels-v2.json	23
3.5	Aggregated Labels per Game, for Game 1 to 126 - From Labels_v2.json	24
3.6	RAG using Embedding Models and Vector Databases	25
3.7	RAG using SQL Agents	27
3.8	Database ER Diagram	30
3.9	Proposed Pipeline	33
3.10	Extractor Module - Part 1 of Proposed Pipeline	35
3.11	Extractor Concept Figure	37
3.12	Validation Module - Part 2 of Proposed Pipeline	37
3.13	Cleaning of Extracted Data	41
3.14	Conceptual Overview of SQL Agent	44
3.15	Proposed UI	50
3.16	GUI Example 1	51
3.17	SoccerRAG GUI Example 2	52
4.1	Hit Rate for SQLRag Based on Test Results	59
4.2	Total Execution Time for Questions 1-10	62
5.1	Screenshot of Vanna.ai	66
5.2	Total Execution Time for Questions 1-10 for GPT4o	69
A.1	Execution Time for Question 1 with GPT-3.5-Turbo and GPT-4-Turbo . .	110
A.2	Execution Time for Question 2 with GPT-3.5-Turbo and GPT-4-Turbo . .	111
A.3	Execution Time for Question 3 with GPT-3.5-Turbo and GPT-4-Turbo . .	111
A.4	Execution Time for Question 4 with GPT-3.5-Turbo and GPT-4-Turbo . .	112
A.5	Execution Time for Question 5 with GPT-3.5-Turbo and GPT-4-Turbo . .	112
A.6	Execution Time for Question 6 with GPT-3.5-Turbo and GPT-4-Turbo . .	113
A.7	Execution Time for Question 7 with GPT-3.5-Turbo and GPT-4-Turbo . .	113
A.8	Execution Time for Question 8 with GPT-3.5-Turbo and GPT-4-Turbo . .	114

A.9 Execution Time for Question 9 with GPT-3.5-Turbo and GPT-4-Turbo . .	114
A.10 Execution Time for Question 10 with GPT-3.5-Turbo and GPT-4-Turbo .	115

List of Tables

2.1	Function Calling Leaderboard - 09.05.2024 (Yan et al., 2024)	11
3.1	Labels-caption.json	24
3.2	Labels-v2.json	24
3.3	Pricing for Different Models (OpenAI, 2023c)	34
3.4	Team Names and Abbreviation	40
4.1	Query Complexity Analysis	57
4.2	Extractor-Validator Evaluation	58
4.3	Combined Hit Rate Q1-Q60 - FS=1, FS=2, FS=3, FS=5	59
4.4	Ablation Study Results for Question 1 to 10	60
4.5	The cost for Q21 with GPT-3.5-Turbo and GPT-4.0-Turbo	62
5.1	Comparison of SQL Services	67
A.1	Question 1: Is Manchester United in the database? - GPT 3.5-Turbo	83
A.2	Question 1: Is Manchester United in the database? - GPT 4-Turbo	83
A.3	Question 2: Give me the total home goals for Bayern M in the 2014-15 season - GPT 3.5-Turbo	84
A.4	Question 2: Give me the total home goals for Bayern M in the 2014-15 season - GPT 4-Turbo	84
A.5	Question 3: Calculate home advantage for Real Madrid in the 2015-16 season - GPT 3.5-Turbo	85
A.6	Question 3: Calculate home advantage for Real Madrid in the 2015-16 season - GPT 4-Turbo	85
A.7	Question 4 - How many goals did Messi score in the 15-16 season? - GPT 3.5-Turbo	86
A.8	Question 4 - How many goals did Messi score in the 15-16 season? - GPT 4-Turbo	86
A.9	Question 5 - How many yellow-cards did Enzo Perez get in the 15-2016 season? - GPT 3.5-Turbo	87
A.10	Question 5 - How many yellow-cards did Enzo Perez get in the 15-2016 season? - GPT 4-Turbo	87

A.11 Question 6 - List all teams that played a game against Napoli in 2016-17 season in seriea? Do not limit the number of results - GPT 3.5-Turbo	87
A.12 Question 6 - List all teams that played a game against Napoli in 2016-17 season in seriea? Do not limit the number of results - GPT 4-Turbo	88
A.13 Question 7 - Give all the teams in the league ucl in the 2015-2016 season? - GPT 3.5-Turbo	88
A.14 Question 7 - Give all the teams in the league ucl in the 2015-2016 season? - GPT 4-Turbo	88
A.15 Question 8 - Give me all games in epl with yellow cards in the first half in the 2015-2016 season - GPT 3.5-Turbo	89
A.16 Question 8 - Give me all games in epl with yellow cards in the first half in the 2015-2016 season - GPT 4-Turbo	89
A.17 Question 9 - What teams and leagues has Adnan Januzaj play in? - GPT 3.5-Turbo	89
A.18 Question 9 - What teams and leagues has Adnan Januzaj play in? - GPT 4-Turbo	90
A.19 Question 10 - List ALL players that started a game for Las Palmas in the 2016-2017 season? Do NOT limit the number of results - GPT 3.5-Turbo	90
A.20 Question 10 - List ALL players that started a game for Las Palmas in the 2016-2017 season? Do NOT limit the number of results - GPT 4-Turbo	90

Chapter 1

Introduction

1.1 Motivation

The rapid growth of digital sports content has created a demand for efficient retrieval systems that can understand and process natural language queries (Gautam, 2023; Gautam et al., 2022a). Such systems can enable more people to access the ever-growing data. The motivation behind this research is to address the challenges in retrieving specific content from textual data generated from an extensive sports video libraries using intuitive, natural language requests. This approach simplifies user interaction and enhances the accessibility of sports video archives.

The first step in this research is retrieving data from a database using natural language, leveraging the power of LLMs. A problem with the current solutions is handling misspellings or abbreviations in user input. It seems that most research focuses on the retrieval of data using proper language and full names of entities of interest. This is, however, not how a "normal" user of an information retrieval system would behave. To increase user satisfaction, systems should be able to understand what a user is interested in and, if not, ask for additional input.

The focus of this research is to address these challenges and contribute to the development of efficient retrieval systems that can simplify user interaction and enhance the accessibility of sports video archives. By leveraging the power of Natural Language Processing (NLP), traditional algorithms and machine learning techniques, this research seeks to enable users to retrieve specific sports content using intuitive, natural language requests, such as "show me the first goal scored by Team A" or

"did Player B get a yellow card?". This approach has the potential to change the way users interact with sports content, making it more accessible, engaging, and useful for a wide range of applications.

1.2 Problem Statement

Current sports statistics retrieval systems often rely on manual tagging or basic keyword searches, which can be time-consuming and imprecise. The problem this research addresses is the development of an advanced retrieval system that leverages NLP and multimodal data fusion to interpret and fulfill complex natural language queries accurately.

The main research question this thesis will answer is:

Can we effectively query complex datasets using natural language?

As the numbers of tables and columns increase in a database, the complexity of the queries needed to retrieve data also increases. In this thesis, we want to explore if LLMs can be used to retrieve data from these sources.

This question encompasses several sub-questions and challenges, including:

Sub-question 1: How does the system handle spelling mistakes and other input errors?

One of the critical factors in enabling natural language querying is the ability to handle imperfect inputs, such as spelling mistakes, typographical errors, or ungrammatical constructions. An effective retrieval system should be robust enough to identify and correct these errors, ensuring that the intended meaning of the query is accurately captured. This sub-question explores the techniques and algorithms employed to achieve this level of fault tolerance.

Sub-question 2: How dynamic is the system? Can a change in the underlying database necessitate code changes?

In real-world scenarios, databases constantly evolve, with new data added, updated, or removed. An ideal retrieval system should be adaptable and flexible, capable of

accommodating changes to the underlying database without requiring extensive code modifications or system overhauls. This sub-question investigates the system's ability to adapt to changes in the data structure and content dynamically, ensuring seamless operation and minimizing maintenance efforts.

Sub-question 3: Is it possible to extract detailed statistics from the database using natural language?

When working with datasets about real events, we often want to understand why an event occurs and look for patterns to predict similar events in the future. The suitability of a solution like this to provide statistics and structured lists is based on the system's ability to retrieve complete information sets in order to answer the question.

Sub-question 4: What role does different LLMs play in the quality of the system?

Is the capability of the LLM a decisive factor when querying data, or can good data preparation reduce the complexity that the LLM needs to handle?

By addressing these sub-questions, the research aims to develop a comprehensive and robust natural language querying system that can efficiently handle complex queries, accommodate input errors, and adapt to changes in the underlying data sources.

1.3 Scope

This research focuses on creating a database retrieval system, integrating LLMs, machine learning, and algorithms. The scope includes developing data indexing and efficient retrieval of statistics found in the dataset.

The main focus of the research will be on ensuring that the interaction between the user and the database is as streamlined as possible. This will enable other researchers and developers to adopt this solution and integrate it into their own systems, enhancing the overall user experience. By making it easier for developers to incorporate natural language querying into their designs, this research will pave the way for a more universal approach to database querying, where databases can be queried using natural language.

One important goal is to make the solution as generalized as possible, meaning one could easily apply the created framework to databases for other domains.

The scope do not cover implementation of open source LLM, nor video retrieval directly. Production deployment is also outside the scope, as this is a research thesis.

Since our main focus is on creating a basic framework that demonstrates the idea, only a small set of 20 test questions was used to test the frameworks' performance. It's important to note that this small sample might not give us a complete understanding of how well the framework performs in all situations. However, the goal of this research thesis was to show that the basic idea works. In the future, more testing and improvements can be done to make it better and use it on a larger scale.

1.4 Research Methods

Computer science and AI research covers a diverse range of methodologies. According to the ACM Education Board, the field of computing can be categorized into three primary paradigms: theory, abstraction, and design (Denning et al., 1989).

Our research aligns with the design paradigm, which involves a systematic approach to constructing a system that solves a specific problem. This approach consists of four distinct stages:

1. Elucidating requirements: defining the problem domain and identifying the needs of the system.
2. Specifying the system: outlining the functional and non-functional specifications of the system.
3. Designing and implementing the system: creating a detailed design and developing the system according to the specifications.
4. Testing the system: verifying the system's performance and evaluating its efficacy.

The primary motivation behind this thesis is to design a system capable of retrieving data from a database using natural language queries. This requires identifying

suitable tools for the task, defining specifications, and implementing the pipeline. To validate the system's performance, we will formulate a set of test questions to be executed using the pipeline.

Furthermore, to isolate the contribution of each component in our pipeline, we will conduct ablation studies, which will provide valuable insights into the system's functionality and effectiveness.

1.5 Ethical Considerations

As LLM becomes more capable and widely accepted, it's important to consider the ethical implications of deploying systems using these models. LLM has a bias stemming from the data it's trained on, which can affect the output. It is also important to remember that LLMs is not a knowledge base, they are generative models that predict the next token based on patterns observed in their training data. The ethical consequences for this can be enormous, where those who develop the best LLM, becomes the owner of truth. In this regard, it is important to give the LLM supplemented data to ensure that it answers based on our facts.

Another critical ethical consideration in AI is privacy. To address this, our research focuses on publicly available data published by SoccerNet (Giancola et al., 2018), published under an MIT license, for research purposes. Balancing the innovative benefits of AI with the fundamental right to privacy is crucial for public trust and ensuring ethical AI development and deployment (Y. Zhang et al., 2021). The use of AI and machine learning models will be critically evaluated to prevent biases and ensure fairness in retrieval results.

Ethical disclaimer: During the writing of this thesis, Grammarly was used to help with spelling and, in some cases, sentence building.

1.6 Main Contributions

The primary contributions include the development of a novel sports statistics retrieval system that accepts natural language queries, enabling users to retrieve data from sports metadata using everyday language, which enhances user-friendliness and

accessibility.

The extraction-validation process we propose, is a novel method, making it possible to enhance the process of querying data using natural language, regardless of the data domain.

The integration of multimodal data for comprehensive search results, which merges various data types such as video metadata, audio, and text, resulting in a more complete and in-depth search experience that provides more accurate and relevant results.

The enhancement of universal design through the use of natural language to query databases. This capability promotes inclusivity and accessibility by making information more accessible to a wider range of users, regardless of their age, disability, or other factors.

1.7 Thesis Outline

In Chapter **2 Background and Related Work**, we introduce key terminology, provide information about SoccerNet and soccer basics, and present an overview of current research in the field of (LLMs) and context retrieval systems.

In Chapter **3 Methodology and Implementation**, different ways of implementing data retrieval are discussed, the dataset is described, and we present the pipeline for the framework. Additionally, we propose various experiments and implement a user interface for the application.

In Chapter **4 Experiments and Results**, the framework is tested and evaluated before we conduct an ablation study to assess its components. We then test the pipeline end-to-end, examining metrics such as execution time and costs.

In Chapter **5 Discussion**, we start by looking at competing solutions, look at current issues with LLMs, and the importance of validated data.

In Chapter **6 Conclusion**, we address the research questions posed in Section 1.2. We then discuss potential use cases, limitations, and future work and highlight the contributions this paper makes to the field.

Chapter 2

Background and Related Work

In this chapter, key terminology will be introduced, together with information about SoccerNet and soccer basics. We will also present an overview of topics discussed later in the thesis.

2.1 Terminology

Words can take on different meanings depending on the context in which they are presented. To ensure clarity, it is important to define key concepts and the terminology used to describe them in this research. By establishing a common understanding of these terms, we can achieve effective communication and prevent misinterpretations throughout the discussion.

- **Events** are used for in-game events, such as corners, cards, goals, and fouls shot on and off target.
- **Shot on target** is an event when a player shoots the ball, and the goalkeeper or defender makes a save, preventing the ball from entering the goal.
- **Shot off target** is an event when a player shoots the ball but misses the goal, either wide, high, or low, without the goalkeeper or defender making a save.
- **LLMs** are machine learning models that can understand and generate text in human language. In this thesis, we will mostly be talking about OpenAI's GPT

3.5-Turbo('GPT-3.5-Turbo', 2024) and 4.0-Turbo('GPT-4', 2024) models when using the term, but there are several other LLMs on the market.

- **Tokens** are what LLMs use to predict the next word. A token can be a single character, a part of a word, a whole word, and, in some cases, multiple words (Song et al., 2021). When calculating the cost of API usage regarding LLMs, we calculate input and output tokens.
- **Input tokens** refer to the tokens that are provided as input to the language model, representing the text or sequence of tokens that the model will process and generate predictions for.
- **Output tokens** refer to the tokens generated by the language model as output, based on the input tokens and the model's learned patterns and knowledge.
- **Chains** are a concept in several frameworks designed around LLMs. In this context, chains are pipelines that perform one task after another before arriving at the desired output. A chain consists of a sequence of predetermined steps, where each step is executed in a predefined order (Langchain, 2024a).
- **Agents** are a similar concept as chains, and also involve LLMs. Agents use a language model as a reasoning engine to determine which actions to take and in which order rather than following a hard-coded sequence of steps. Agents have the freedom and flexibility to dynamically decide on the next function or action to execute based on the given inputs and intermediate results, using reasoning and a decision-making process powered by the language model (Langchain, 2024a).
- **Context Retriever** is a term used about modules that can retrieve extra context to help an LLM answer a question. This could be a search engine plugin, an agent that has access to databases, or RAG solutions.
- **Hallucinations** is text generated by a LLM that is not grounded in factual information. The LLM will, however, present the false information as it's correct (Ji et al., 2022).
- **Properties** or **Entities** when talking about extraction is things like player name, team name, in-game-events, and so forth.

2.2 Soccer Basics

Soccer is one of the biggest sports in the world. At its core, soccer is a simple game, where two teams of eleven players each are competing in scoring the most goals. However, beneath the simple objective of the game, there are several layers of tactics, athleticism, teamwork, and rules. These rules, or the Laws of the Game as defined by The International Football Association Board (IFAB), establish the framework within which the game operates.

Most countries where soccer is played have national leagues. Here, several teams play against each other twice a year, gaining 3 points for winning, 1 point for a draw, and 0 points for losing. At the end of the year, the team with the most points wins the league, and the teams with the least points might be transferred to a lower division.

A team consists of eleven starting players, but the coach has the opportunity to substitute up to three players at any given time. This allows for tactical adjustments, fresh legs, and the opportunity to respond to the changing dynamics of the game.

When a player shoots at the goal, we can put the shot into one of three categories: goal, shot on target, or shot off target. If the player scores, we record the shot as a goal. If the goalkeeper saves the shot, it's recorded as a shot on target, and if the shot misses the goal, it's recorded as a shot off target. These statistics provide valuable insights into a team's attacking efficiency and the performance of individual players.

Additionally, the game features various types of fouls and disciplinary actions, such as free kicks, yellow cards and red cards, which can have a significant impact on the flow and outcome of the match. Referees are responsible for ensuring the rules are followed and maintaining the integrity of the game (International Football Association Board, 2023).

2.3 SoccerNet Dataset

SoccerNet is a large-scale dataset for soccer video understanding, consisting of 550 recordings of broadcasts across the top European leagues, such as the Premier League, La Liga, and Bundesliga. To foster research and innovation in the field of

sports analytics, the team behind SoccerNet hosts yearly competitions where teams of researchers compete to solve tasks like Ball Action Spotting and Game State Reconstruction.

The SoccerNet dataset and its associated challenges have attracted a growing community of researchers from diverse backgrounds, including computer vision, machine learning, and sports analytics. The yearly competitions provide a platform for teams to showcase their innovative approaches and push the boundaries of what is possible in the realm of soccer video understanding (Giancola et al., 2018).

2.4 Large Language Models

LLMs have become vital in the generative AI revolution, transforming our interaction with and processing of language. The development of LLMs originated in the 1950s and 1960s with rule-based models. These early models, reliant on hand-crafted linguistic rules (symbolic AI), were inadequate for the complexities of NLP (Hadi et al., 2023).

Generative AI is a type of unsupervised machine learning that enables the creation of multimodal content, such as text, images, videos, and 3D models, that can be highly realistic and often indistinguishable from human-generated content. The technology uses neural networks, such as GANs and Transformers, to generate new data that can be used in a wide range of applications, from art and entertainment to education and science (Fui-Hoon Nah et al., 2023).

The first chatbot capable of simulating human conversation, Eliza, was developed in 1966 by MIT researcher Joseph Weizenbaum. Eliza employed a pattern-matching methodology, using predefined rules to identify keywords in user input and formulate responses (Weizenbaum, 1966). Although limited in functionality, Eliza represented a significant milestone in NLP research, laying the groundwork for future advancements.

In the 1990s, statistical language models, based on probabilistic methods, improved upon rule-based models (Jelinek, 1998). These models were better equipped to manage larger datasets and offered enhanced accuracy, yet they struggled with semantic understanding and contextual language processing (Hadi et al., 2023).

A major breakthrough occurred in the mid-2010s with the introduction of neural language models, notably the recurrent neural network language model (RNNLM) in 2010. Subsequent developments, such as the Google Neural Machine Translation (GNMT) system in 2015, marked significant progress in modeling language context and generating natural-sounding text (Hadi et al., 2023).

The 2017 introduction of the Transformer model was a huge advancement in the field. It facilitated learning of longer-term language dependencies and enabled parallel training on multiple GPUs (Vaswani et al., 2023). This advancement allowed for the training of larger models and the release of OpenAI’s GPT-1 in 2018, with its transformer-based architecture, was a notable advance, leading to subsequent models like GPT-3.5-Turbo (‘GPT-3.5-Turbo’, 2024) and GPT-4.0-Turbo (‘GPT-4’, 2024). These models demonstrated exceptional proficiency in producing coherent and natural text. Other significant developments include Meta’s Llama, Google’s Bard, Amazon’s Alexa AI features, and Huawei’s Pangu models (Hadi et al., 2023).

There is currently an arms race between companies to create new and improved LLMs. To navigate this ever-changing field, "leaderboards" such as Berkeley Function-Calling Leaderboard (Yan et al., 2024) In table 2.1 functions with the suffix (FC) can be plugged into function calling frameworks.

Rank	Overall Acc	Model	Organization	License
1	87	GPT-4-0125-Preview (Prompt)	OpenAI	Proprietary
2	86.47	Claude-3-Opus-20240229 (Prompt)	Anthropic	Proprietary
3	84.35	GPT-4-1106-Preview (FC)	OpenAI	Proprietary
4	84.24	GPT-4-turbo-2024-04-09 (Prompt)	OpenAI	Proprietary
5	83.29	Gorilla-OpenFunctions-v2 (FC)	Gorilla LLM	Apache 2.0
6	83.12	GPT-4-0125-Preview (FC)	OpenAI	Proprietary
7	82.12	Meta-Llama-3-70B-Instruct (Prompt)	Meta	Meta Llama 3 Community
25	62.47	GPT-3.5-Turbo-0125 (FC)	OpenAI	Proprietary

Table 2.1: Function Calling Leaderboard - 09.05.2024 (Yan et al., 2024)

The table shows that proprietary models outperform open-source models on function-calling tasks, but models like Gorilla-OpenFunctions-v2 (Patil et al., 2023) show promising results. The overall accuracy is calculated as a mean of nine metrics the leaderboard uses.

The validation of systems employing LLMs can indeed pose challenges due to their inherently non-deterministic characteristics, as described by Ouyang et al., 2023.

However, LLMs has a laziness issue. The laziness manifests in the LLM not completing a given task or providing an example of how the user can complete the task. OpenAI has stated that their new GPT-4.0-Turbo model is less lazy, but

feedback from regular users of the model is mixed (IVG, 2024; Mauran, 2023; OpenAI, 2023b).

2.4.1 Multimodality

The drive to enhance model versatility for handling various inputs, such as images, sound, and video, has made multimodal data fusion a key research area. Big data, characterized by its high volume, velocity, variety, and veracity, often comprises structured, semi-structured, and unstructured formats from diverse sources. Each modality carries unique information. For example, a sports news webpage might combine images and text, representing different modalities of the same event. Integrating these multimodal data types is essential for a comprehensive understanding, particularly when individual modalities offer partial information. Multimodal data fusion integrates disparate data sources, types, and distributions into a unified representation, enabling exploration of both inter-modality and cross-modality relationships (Gautam et al., 2022b). This fusion yields insights superior to those from any single modality. Consequently, multimodal models have emerged as a powerful tool in AI, paving the way for more comprehensive and intelligent systems. Central to these multimodal models are LLMs (Gao et al., 2020; Wu et al., 2023).

These models, capable of complex reasoning across various tasks, offer user-friendly interfaces instructable with natural language (Touvron et al., 2023). However, the static nature of LLMs, exemplified by the extensive training of models like Llama-2-13b, poses a limitation. Training this model required 368640 hours on Nvidia A100 GPUs, highlighting the challenge (Touvron et al., 2023). To address this, LoRA (Low-Rank Adapters) and QLoRA (Quantized Low-Rank Adapters) have been developed, reducing computational expenses. LoRA, by altering a small subset of the model's weights, yields significant improvements with minimal updates. QLoRA further reduces memory requirements through quantization methods (Dettmers et al., 2023). These methods make it possible to fine-tune a model for a specific use case. Although effective for specific fields, these methods are limited in dynamic scenarios. Fine-tuning is a method where an unsupervised trained model is trained with supervision (Ziegler et al., 2020). This approach is mainly about giving the model a sense of direction on how to respond, but it also has the added benefit of helping the model learn new information, such as a new language (Strand et al., 2024).

2.4.2 Information Retrieval

For external information retrieval, RAG is a more suitable approach. RAG is a method for open-domain question answering that combines retrieval of relevant documents with a generation model to produce accurate answers. The RAG method first vectorizes the input query and uses that to retrieve a set of K documents from a vector database. This will return the K documents most likely to contain the information to the question. The retrieved documents are then passed to a LLM, which generates the final answer based on the content of the documents and the question (Yu et al., 2022).

Vector databases are used to store and retrieve documents efficiently for RAG systems. In a vector database, each document is represented as a dense vector in a high-dimensional space. These vectors are generated using pre-trained embedding models, which convert the text of the document into a fixed-length vector that captures its semantic meaning. When a question is posed, the RAG model generates a query vector for the question using the same embedding technique. The vector database then retrieves the documents whose vectors have the highest cosine similarity to the query vector, indicating that they are most likely to contain relevant information (Lewis et al., 2021).

By using a vector database to store and retrieve documents, RAG models can efficiently access a large corpus of text to find the most relevant information for a given question. This approach allows RAG models to answer questions that require knowledge beyond the scope of the training data, making them well-suited for open-domain question-answering tasks (Lewis et al., 2021). A vector database is designed for the efficient storage and retrieval of vector data, setting it apart from conventional databases that organize data in tables and columns. Unlike traditional systems, vector databases are specifically tailored to handle high-dimensional vector representations of data, optimizing both storage and query capabilities.

FAISS and ChromaDB stand out as leading local vector databases. While they primarily operate as local vector management libraries and not strictly as databases, they offer seamless integration with AI tools like LangChain and OpenAI's embedding models. This integration enhances the efficiency of embedding searches (Jeong, 2023).

However, as the database and complexity of documents grow, and especially in multi-modal cases where the system needs to handle information from several different

documents, standard RAG methods has shown some weaknesses. Solutions like MuRAG (Chen et al., 2022), LLaVA (Liu et al., 2024), and Video-Llama (H. Zhang et al., 2023) have been proposed to incorporate additional modalities. However, they are not applicable to this research because metadata from sources such as video, audio, and images has already been extracted and presented in a text format. Structuring this type of data can be effectively managed with databases. With the advancement of models like GPT-4, it is possible to develop "agents" capable of converting natural language into SQL queries for use in production environments. These agents will then retrieve information from the database based on the user's input. There are several ways to create these agents, and the two most commonly used frameworks are Langchain and Llama-index.

In the LangChain framework, agents are important components for constructing LLM applications. These agents, categorized into Action Agents and Plan-and-Execute Agents, determine the appropriate tools and actions based on user input. Action Agents decide on the next action by considering the outputs of all previous actions. At the same time, Plan-and-Execute Agents devise a comprehensive sequence of steps based on user input and execute them sequentially. This approach is essential for developing flexible and dynamic LLM applications (Topsakal & Akinci, 2023). Agents interact with the world through tools, which are interfaces facilitating specific actions aligned with the agent's functional objectives. These tools are purposefully selected based on the agent's intended purpose. For instance, an agent interfacing with a SQL database would need tools for executing queries and inspecting tables.

Additionally, LangChain introduces the concept of toolkits. These are aggregated collections of synergistic tools assembled for particular use cases, designed to consolidate tools that function effectively together for distinct tasks. Toolkits also encompass convenience methods for easy loading, enhancing the integration process of various tools into the agent's workflow. This architecture allows for a high degree of flexibility and customization in LangChain. Agents can be tailored to specific applications by selecting the right tools and toolkits, enabling them to perform a wide range of tasks efficiently and interact with diverse data sources and environments (Topsakal & Akinci, 2023).

It is important to understand that tools are not run on the LLM, but locally or the application serving the user interface.

In the paper, *Can LLM Already Serve as A Database Interface?* (Li et al., 2024), the authors evaluate the performance of LLMs on a novel text-to-SQL benchmark called

BIRD (BIg bench for laRge-scale Database grounded in text-to-SQLs). The BIRD benchmark aims to mimic real-world scenarios by incorporating large databases with complex schemas, noisy data values, and the need for external knowledge reasoning. The authors test several state-of-the-art LLMs, including GPT-4, on this challenging benchmark. Their experimental results demonstrate that even the most powerful model, GPT-4, achieves only 54.89% execution accuracy, significantly lagging behind human performance of 92.96%. This highlights the limitations of current LLMs in handling text-to-SQL tasks involving large, noisy databases and the need for external knowledge grounding. The paper addresses a crucial issue, as many existing text-to-SQL solutions are designed and evaluated on clean, relatively small databases, which do not accurately reflect the complexities encountered in real-world applications. The authors encourage further research to address the challenges presented in the BIRD benchmark, paving the way for more robust and practical text-to-SQL systems (Li et al., 2024).

It is worth noting that this paper does not include any open-source models. This might be because open-source models currently do not have advanced function-calling abilities. In *NexusRaven: a Commercially-Permissive Language Model for Function Calling* (Srinivasan et al., 2023), the authors claim that their model surpasses even GPT-4 in function calling. However, the model "*primarily emphasizes single-round interactions with humans through natural language instruction*". The model's focus on single-round interactions may limit its applicability in scenarios requiring multi-step reasoning or extended dialogue. However, the techniques introduced, such as multi-step data refinement and demonstration retrieval augmentation, could potentially benefit other language models and approaches aimed at more interactive and multi-turn tool usage.

To properly understand the limitations of language models with respect to functions or tools, it is essential to grasp the fundamental concept of functions. A function, in the context of language models, is analogous to functions in programming languages. It expects an input, processes the input according to its defined logic, and returns an output or result. When using functions with LLMs like GPT-3.5 or GPT-4, the user must provide the model with a description of the function's purpose and the expected input format. Based on this information and its internal reasoning capabilities, the model can then decide whether to call or invoke the function. If the model chooses to call the function, the function's execution occurs on the client side (i.e., outside the model), and the resulting output is returned to the model (OpenAI, 2023a).

2.4.3 LangChain

In the LangChain framework, one can create either a chain of functions, calling one or more predefined functions sequentially before providing the user with a final answer or output, or create an Agent that has a set of tools (functions). The Agent provides a lot more flexibility and adaptability, as it can do internal reasoning and decide on what tools, and how many tools, should be invoked to give the final answer as described in (Langchain, 2024a).

The team behind LangChain has also created LangSmith. LangSmith is a comprehensive platform designed to help you manage your LLM applications with ease. With its powerful tools, you can efficiently monitor, calculate costs, and trace API usage from your applications. The platform's intuitive dashboard provides essential information, such as the origin of the call, input and output details, start time, total tokens, and the cost of an API call ('Getting Started with LangSmith', 2024).

By utilizing database property verification prior to executing the LLM agent, the speed and quality of the solution can be significantly improved. A feasible approach to achieve this is to employ the Langchains extractor chain (Langchain, 2024b), which can extract properties from a given prompt into a schema. This chain will always return data in the same format, making it possible for a program to process it. Subsequently, a conventional algorithm can be applied to identify the closest matches. One such algorithm is the Ratcliff/Obershelp pattern recognition (Gestalt Approach), first introduced in 1988 (Ratcliff & Metzener, 1988). This algorithm performs well at finding the best match between two sets of data, taking into account the sequence and structure of the data, rather than relying solely on exact matches. This approach effectively detects similarities and differences, making it highly valuable in applications such as spell checking, text analysis, and natural language processing, where recognizing related or similar words is essential. Other methods explored in this field are the longest common sub-sequence algorithms Bergroth et al., 2000 and the Levenshtein algorithm (Schulz & Mihov, 2002).

2.4.4 ChainLit

Chainlit is an open-source Python package that enables developers to build production-ready conversational AI applications. It provides a range of features that

make it an attractive choice for building chatbots and virtual assistants. Chainlit's key features include building conversational AI applications quickly by integrating seamlessly with an existing codebase or starting from scratch in minutes. The platform also provides tools for visualizing multi-step reasoning. Chainlit comes with a set of integrations with popular libraries and frameworks, such as OpenAI and LangChain, making it a flexible and versatile choice for building LLM based applications. Furthermore, Chainlit allows developers to build custom frontends, providing them with the ability to integrate their own React-based user interfaces with the Chainlit backend (Chainlit, 2023).

2.5 Chapter Summary

In this chapter, we have discussed the terminology used in the paper, looked briefly at soccer and the dataset SoccerNet.

We also discussed some key concepts related to LLMs and how they're used in the context of agents and tools. LLMs are impressive machine learning models that can understand and generate text in human languages. One popular example of an LLM is OpenAI's GPT models. To help these models predict the next word in a sentence, they use tokens. Tokens can represent single characters, parts of words, whole words, or even multiple words, depending on the context.

We also explored two important concepts: chains and agents. Chains are a series of tasks that are performed one after another to achieve a specific goal. They follow a fixed sequence of steps, and each step is executed in a predefined order. On the other hand, agents are more flexible and adaptive. They use a LLM to reason and determine the best actions to take and the order in which to take them. This allows agents to make decisions based on the inputs they receive and any intermediate results that might be generated.

Another essential concept is tools. Tools are functions that agents can use to accomplish tasks. Each tool has an input schema that tells the language model what parameters are needed to call the tool, and a function that actually runs the tool. Tools can provide operations such as RAG, feature extracting and SQL-querying.

In order to be able to retrieve data from an external source, we rely on these agents with their tools. However, tools are not able to verify that entities asked for exist in the

dataset at their disposal. The dataset schema is also important to improve in retrieval accuracy.

We will therefore in the next chapter look at different retrieval schemas, and explore options to verify that entities in the user's question exists in the dataset.

Chapter 3

Methodology and Implementation

In this chapter, we delve into the various implementation options for our framework. We will explore the design of the dataset structure, the retrieval schema, and the experiments. In addition to this, we will discuss and implement a User Interface (UI) for the framework.

The vision behind this framework is to give non-technical users the ability to quickly and accurately get information from a mixture of text files using only natural language. We aim for the framework to work as a layer between users and data sources, as illustrated in figure 3.1.

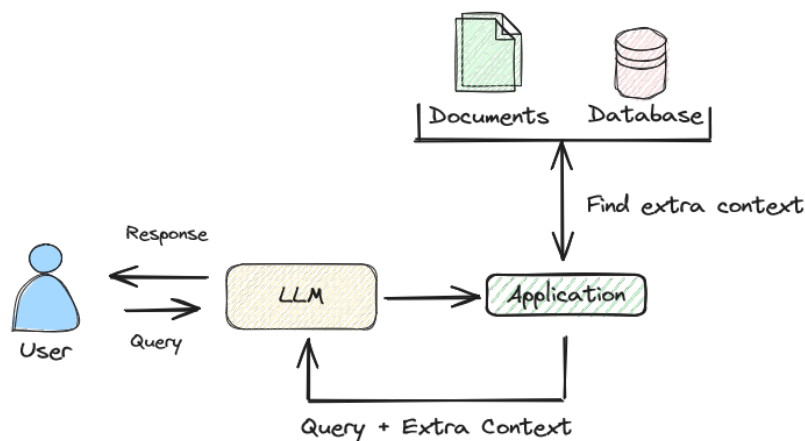


Figure 3.1: Pipeline Concept

The pipeline aims to simplify the interaction as much as possible, making interaction streamlined. One of the major components of the solution is the context retriever. Context retrievers have the ability to search for extra information in documents,

databases, and even online. This data is then paired with the user’s question to give the LLM with updated and accurate information.

3.1 Data Management

3.1.1 Source Dataset

The dataset used in this framework consists of files from SoccerNet, and are organized in a hierarchy of directories and JSON files as shown in Figure 3.2, and ASR files from the paper SoccerNet-Echoes (Gautam et al., 2024). For the game between Brunley and Liverpool on 20-08-2017, the files can be found in the following path: "Dataset/SN-ASR_captions_and_actions/england_epl/2016-2017/2016-08-20 - 17-00 Burnley 2 - 0 Liverpool". This directory contains four files: 1_half-ASR.json, 2_half-ASR.json, Labels-caption.json, and Labels-v2.json.

The **1_half-ASR.json** and **2_half-ASR.json** files contain transcribed commentary using Whisper (Radford et al., 2022) for voice-to-text. Each transcribed line is segmented, with start time, end time, and transcribed text.

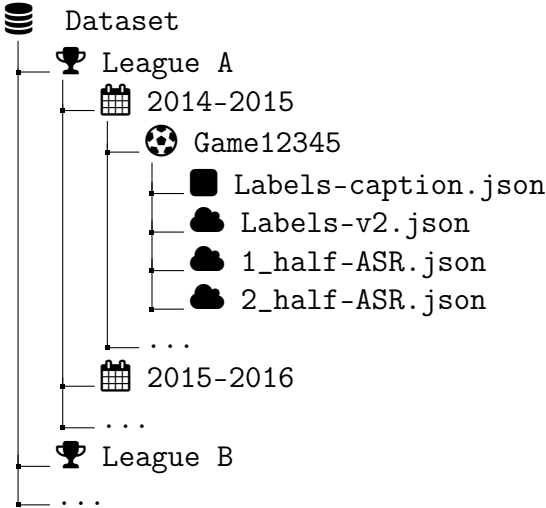


Figure 3.2: Directory Structure in Dataset

One can download the files from SoccerNet by running these lines of code:

```
from SoccerNet.Downloader import SoccerNetDownloader
Downloader = SoccerNetDownloader(LocalDirectory="data/dataset/SoccerNet")
```

```
Downloader.downloadGames(files="Labels-v2.json",
split=["train", "valid", "test"])
Downloader.downloadGames(files="Labels-caption.json",
split=["train", "valid", "test"])
```

The **Labels-caption.json** file has two main parts. Part one is general information about the game. Here we find

- **Timestamp** - Unix time format for game start
- **Score** - Final score of the game
- **Round** - The round of the season
- **Teams** - The teams that played in that game

We also find game **lineup** in part one. In the lineup, we get information about players, both starting and substitutes, with name, shirt number, country of birth, and a lineup number. The lineup number can be translated to position on the field. In addition, we find a item called **facts** for each player. This item contains information about goals, yellow and red cards, assists, and substitutions for the given player.

The dataset then includes game-specific facts, such as

- **Referee** - Name of the referee in charge of the match
- **Venue** - Where the game was played
- **Game date** - The date and time of the game presented in DD/MM/YYYY - HH:MM format.

In part two of the file, the dataset contains **Annotations**. This is information about the game, created by clean commentary from the game. The annotations contain eight items for each record, but only six of them are interesting for our use case. These are

- **Important** - A boolean flag that say if a event is important or not
- **gameTime** - Time of the event, presented as PERIOD - TIME

- **Label** - Describe the event, such as corner, substitution, soccer-ball (goal), funfact, y-card and so on. All event labels and the total occurrence in the database is shown in Figure 3.3
- **Description** - Textual description of the event, extracted from game commentary
- **Visibility** - If the event was shown on camera or not
- **Position** - At what frame of the video did the event happened.

To standardize the dataset, and make it easier for the SQL Agent to navigate, soccer-ball was renamed to goal, while y- and r-card were renamed to Yellow and Red card.

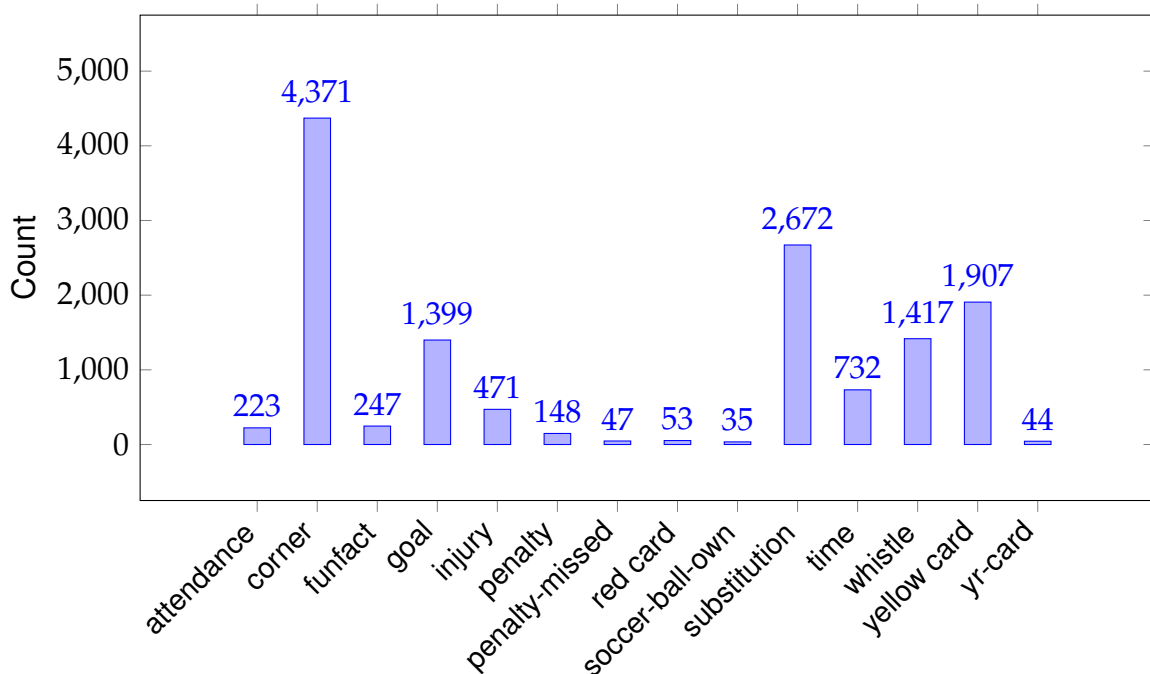


Figure 3.3: Dataset - Event Type Distribution from Labels-caption.json

The Labels-v2.json file is a more compact and detailed version of the annotations in Labels-captions.json. This file includes

- **gameTime** - Time of the event, presented as PERIOD - TIME
- **Label** - Description of the event. This dataset contains a lot more events, such as Shots on/off target, Throw-in, Clearance, Foul, in addition to what is found in Labels-captions.json. All event labels and the total occurrence in the database is shown in Figure 3.4

- **Position** - At what frame of the video did the event happen.
- **Team** - What team the event is for. (Home, Away, Not applicable)
- **Visibility** - Is the event shown on camera, translated from a string value of "visible" and "not shown" to a boolean true/false value.

By examining the different events in the dataset, we discovered that Labels-v2.json had more events recorded than Labels-captions.json, as shown in Figure 3.4 and Figure 3.3. Therefore, event names in Labels-v2.json was chosen as our standard, and labels in Labels-captions.json was renamed to follow this standard. Since Labels-captions.json also include a description of the event, an effort was made trying to connect the two tables using the gameTime column. Sadly, the gameTime of these two dataset is out of sync, making it hard to connect an event across the two files.

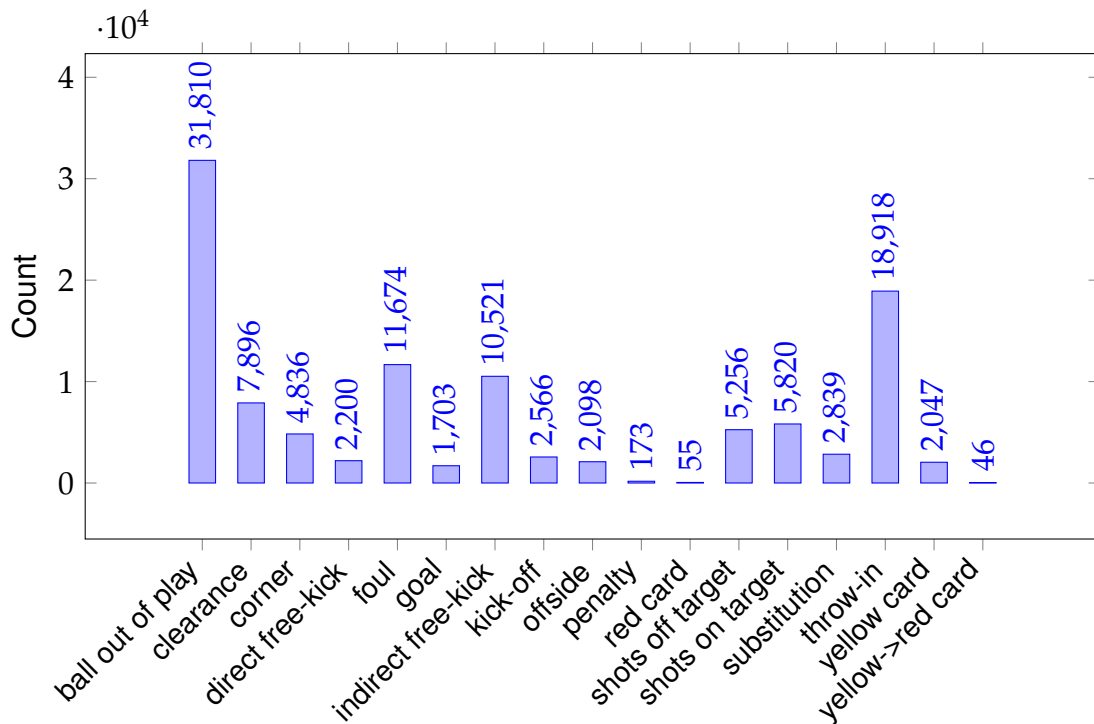


Figure 3.4: Dataset - Event Type Distribution from Labels-v2.json

As shown in 3.1 and 3.2, both describing the first goal in the match between Chelsea and Burnley on 02.02.2015, both gameTime and position (frame in video) is out of sync. This makes it impractical to tie the two files together. Thus, it was decided to put the information in two different tables.

Table 3.1: Labels-caption.json

Key	Value
important	true
gameTime	1 - 13:36
label	soccer-ball
description	Goal! Eden Hazard provides Branislav Ivanovic (Chelsea) with a nice pass inside the box. It allows him to finish with a precise effort into the bottom right corner. 1:0.
visibility	shown
position	816000

Table 3.2: Labels-v2.json

Key	Value
gameTime	1 - 13:10
label	Goal
position	790722
team	home
visibility	visible

In figure 3.5 the total number of events for game 1 to 126 is shown. As one could expect, ball out of play is the most common event, while events such as goal and red card are rarer.

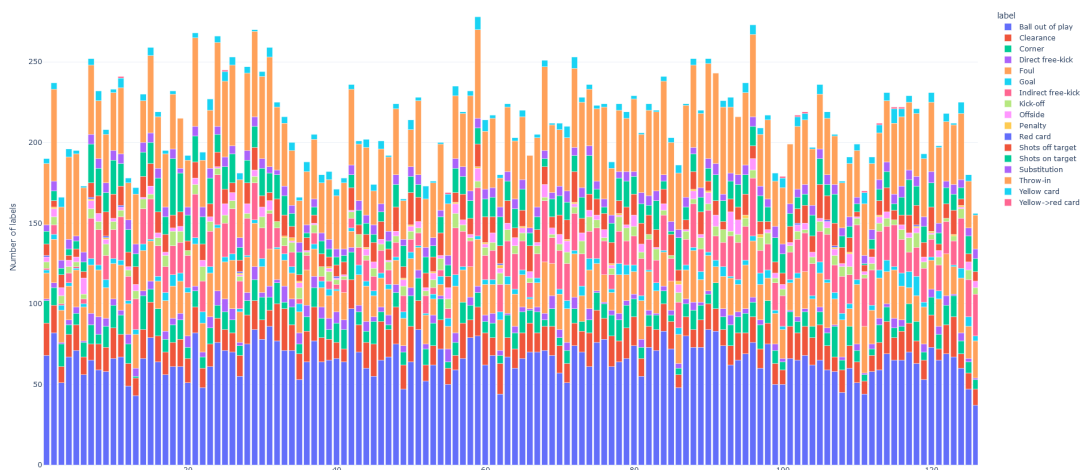


Figure 3.5: Aggregated Labels per Game, for Game 1 to 126 - From Labels_v2.json

3.1.2 Context Retrievers

In Chapter 2, we saw that there are several methods to provide additional context to the LLM. One such method is using embedding models and vector databases to find similar contexts, as illustrated in figure 3.6.

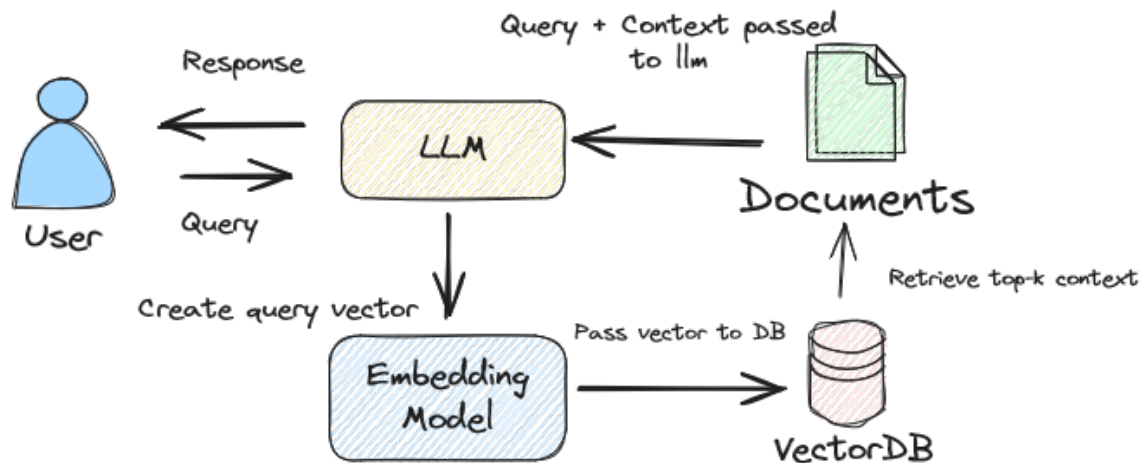


Figure 3.6: RAG using Embedding Models and Vector Databases

As outlined in Section 3.1.1, the data provided for each game is organized across multiple files. Given the project's multi-modal scope, it's likely that additional modalities, and consequently, more files, will be introduced later. Although this system can retrieve all documents associated with a given game, it's not assured that the LLM will effectively interpret the context returned in Figure 3.6. This challenge arises because some events are linked through timestamps while others provide general information about the game, among other distinctions. A crucial aspect of generating embeddings involves dividing the document into several chunks. This process will further fragment the data, complicating the task for the LLM to analyze the text efficiently.

If this RAG based system is asked to answer a question such as question 2 in Chapter 3.3.2, where the system is asked to find the total goals scored by a team within a season, the RAG module would have to process all documents in the dataset to what games the team played in. It would then need to extract and aggregate the goals for each game. This would take a lot of computing, and the system would be unusable as the database grows.

To address this issue, it was decided to explore the benefits of introducing SQL agents that can translate text input into SQL queries and then retrieve the context

directly from the SQLite database. This process is similar to the one described in Figure 3.6, but it introduces agents that implement a chain of thought reasoning process. As seen in Figure 3.7, the agent can perform several loops, where it justifies its reasoning and splits a question from the user into several different queries to the database.

The introduction of SQL-Agents offers several advantages:

1. **Structured Data Retrieval:** By translating natural language queries into SQL statements, the agents can directly access and retrieve relevant information from the structured database, eliminating the need to sift through and interpret unstructured text data.
2. **Context Preservation:** The database stores the data in a well-organized and context-preserving manner, ensuring that related events and information are linked together. This enables the agents to retrieve and combine relevant context more effectively.
3. **Modular Approach:** The SQL-Agents can be designed to handle specific types of queries or data modalities, allowing for a modular and extensible system as new data sources or modalities are introduced.
4. **Chain of Thought Reasoning:** By implementing a chain of thought reasoning process, the agents can break down complex queries into smaller, more manageable steps, justifying their reasoning at each stage. This approach can improve the overall quality and interpretability of the generated responses.
5. **Scalability:** With the data stored in a structured database, the system can more easily scale to handle larger datasets or higher query volumes, as database management systems are optimized for efficient data retrieval and querying.

The framework can retrieve information from the database based on the user's question. After information retrieval, the agent assesses the information and decides on future actions. These actions could involve querying for more data, providing data directly from the database, or analyzing the result and providing the user with a textual answer.

Additionally, there is an opportunity to create a RAG system, which would give the SQL agent access to sample SQL queries. For instance, a user could ask, "Give me the name of all teams in the database," and the corresponding SQL query would be

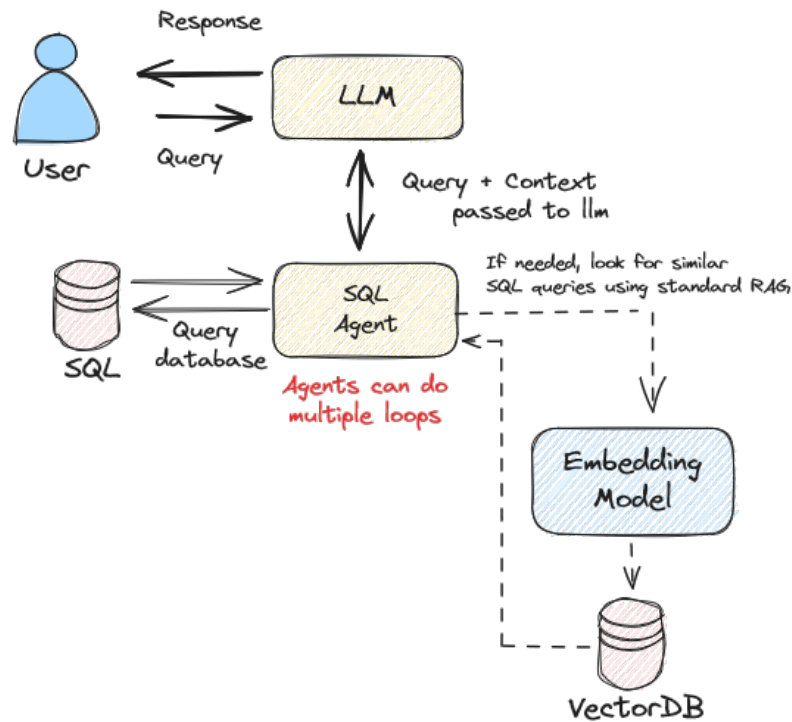


Figure 3.7: RAG using SQL Agents

SELECT NAME FROM TEAMS; This approach would enable us to provide the agent with additional tools to tackle tasks it may not be performing well on initially and also provide users with a flexible tool to modify the application to their needs by simply adding a line of text in the `sqli.json` file.

Another potential approach is to design a multi-agent system, where one agent interacts with the database while another agent searches the web for additional information or generates statistics. However, this method raises two significant concerns. Firstly, the agent would need to match the entity names in the database exactly, which could be problematic if the agent resolves a term like "Barca" to "Barcelona FC" instead of the expected "Barcelona". Secondly, the agent may find extra information online that does not match the data in the database, which could lead to inconsistencies. This approach's lack of user control over the data's origin is a major issue. Therefore, we decided to explore alternative solutions.

A common problem we experienced was mistakes in input. For example, a user might ask, "How many goals did Messi score for Barca in the 15-16 season?" This question raises three key challenges:

1. First, the player name given by the user is Messi, but in the database, Messi

is stored as Lionel Messi. The same is the case with Barca, which is stored as Barcelona. An agent set up to search the web, might resolve Barca to Barcelona, but it might just as well resolve it to Barcelona FC.

2. The seasons in the database are stored with full years, so 15-16 would need to be 2015-2016 to find the correct values. One can not expect the web agent to resolve. Additionally, the season's structure will change from league to league. In Norway, we would only use one year to describe a season.
3. The SQL Agent will, in some cases, be able to figure this out by itself by performing multiple queries searching for information. However, this process is slow and will not work in all cases.

One possible solution to address the issue of ambiguous user queries might be to instruct users to use full names in their queries. However, this approach would compromise the vision of using natural language to query the database. A more effective approach would be to introduce a query-building stage in the chain.

This stage would involve the agent analyzing the user's query to identify potential ambiguities or mismatches between the user's terminology and the database schema. By leveraging additional context or information sources, such as knowledge bases or entity resolution techniques, the agent could attempt to resolve these issues. For instance, the agent could recognize that "Messi" and "Barca" refer to "Lionel Messi" and "Barcelona" respectively, and that "15-16 season" should be translated to "2015-2016" based on the database's date format.

By incorporating this query-building stage, the agent can effectively bridge the gap between natural language queries and the structured database schema. This allows accurate and efficient retrieval of the desired information while still allowing users to query the database in a natural and intuitive way.

The query-building stage can also be designed to handle more complex queries, such as those involving multiple entities, dates, locations, and other specific details. This approach enables a more natural and user-friendly interaction with the system while still leveraging the structured nature of the database for precise and efficient information retrieval.

By combining the strengths of SQL agents and a structured database approach, our system aims to achieve significant improvements in accuracy, efficiency, and scalability when retrieving and combining relevant context for multi-modal data.

Furthermore, the chain-of-thought reasoning process enhances the interpretability of the generated responses, providing a more transparent and trustworthy interaction.

However, in order to make the SQL agent work, the dataset has to be translated into a structured database.

3.1.3 Database

As discussed in section 3.1.2, using a conventional RAG system for this unstructured data would be impracticable. However, transforming this data into an SQL database presents its own challenges. A script utilizing SQLAlchemy was developed to tackle this issue, with a user-friendly SQLite database as the chosen database system.

This script converts the JSON files into a structured format, creating tables and columns based on a predefined database schema. This process ensures that all relevant information is preserved and easily accessible to the SQL Agent. By structuring the data in this way, information retrieval and generation from the dataset becomes more efficient and accurate. Figure 3.8 illustrates the entity relation diagram of the database.

The script also includes a feature that standardizes variable names, enhancing consistency across the database. The basic structure of the database is as follows:

- teams
- players
- leagues
- games
- game_lineup
- events
- player_events
- captions
- commentary

- player_events
- player_events_labels
- agumented_teams
- agumented_leagues

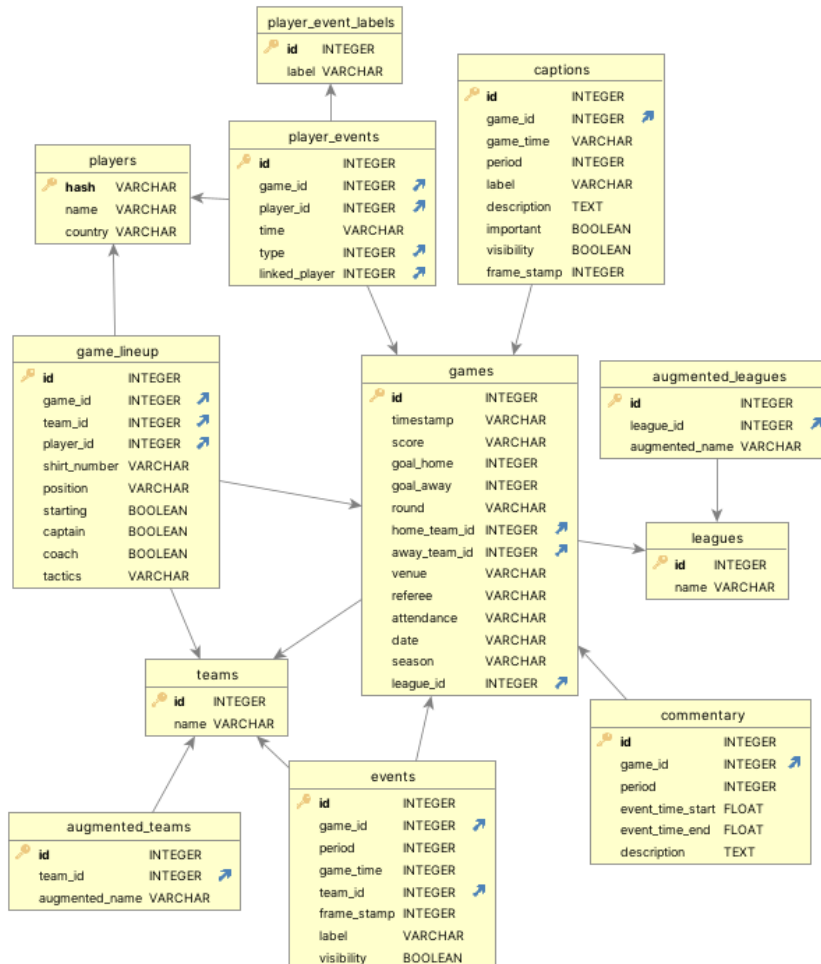


Figure 3.8: Database ER Diagram

The helper tables, `augmented_teams`, and `augmented_leagues`, are created from CSV files containing pairs of augmentations and full names. This allows the framework to perform lookups on the team *Manchester United* by using *ManU* as an abbreviation.

The data processing script systematically extracts information from JSON files and stores it in a structured database. The steps involved in the process are outlined below:

1. Extract the **league** and **season** information based on the folder structure.

2. From the `label-caption.json` file, extract the following game details:
 - Home and away team names
 - Score
 - Round
 - Venue
 - Referee
 - Date and timestamp of the game
 - Attendance
3. Check if the home and away teams are present in the database; if not, create new entries for them.
4. Add the game details to the database as a new game entry.
5. From the `label-caption.json` file, also extract the game lineup information.
6. For each player in the lineup, get the following details:
 - Player number
 - Captain status
 - Starting status
 - Country
 - Position on the field
 - Player-specific facts (if any)
7. Check if the player is already in the database; if not, create a new player entry.
8. Add the player to the `game_lineup` table, linking them with the current game.
9. Parse player-specific facts and add these as entries in the `player_event` table associated with the current game.
10. Process all the events listed in the `label-caption.json` file, renaming events as necessary.
11. Add each event to the `caption` table, using the game as a foreign key (FK).
12. From the `label-v2.json` file, extract event data and add it to the `Event` table, again using the game as a FK.

13. From files named `1_half-ASR.json` and `2_half-ASR.json`, add commentary data to the `commentary` table with details such as period, event start time, event end time, and the description of the event.

During the development of the SQL chain, it became evident that descriptive table and column names played a crucial role in improving the LLM's understanding and ability to provide accurate solutions. As a result, we had to revisit and refine the database design multiple times before settling on a clear and intuitive structure for both users and the LLM.

One of the key considerations was the naming convention for tables and columns, as the SQL Chain relies on these names to comprehend the data and determine the appropriate queries to retrieve the desired information. For instance, in the early stages of the project, the table containing event information was named "annotations," while the table now known as "captions" was called "events." Although both tables contained event-related data, the "annotations" table had more comprehensive labeling, making it a better choice when the user requested information about specific events like shots, fouls, or corners.

By iterating on the database design and ensuring clear and descriptive naming conventions, we were able to improve the LLM's ability to understand the data structure and formulate accurate queries. This, in turn, enhanced the overall performance and reliability of the RAG system, allowing for more precise and relevant information retrieval and generation.

3.2 Proposed Pipeline

Querying a database using natural language involves multiple steps to translate human input into an optimized form for LLMs. The proposed pipeline, as illustrated in Figure 3.9, consists of several stages designed to ensure accurate and efficient retrieval of information. Creating a robust chain can also have the extra benefit that it allows a weaker LLMs to run the chain and get the correct result. This can both reduce costs and improve the overall speed of the system. This can also open the path for using open-source models at a later time.

The three main stages is: **Feature Extraction**, **Feature Validation**, and **SQL Querying**.

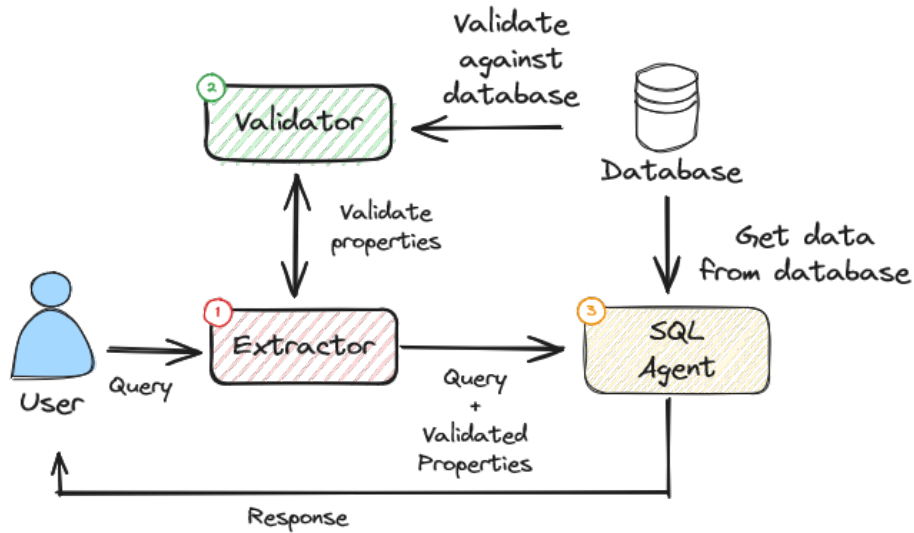


Figure 3.9: Proposed Pipeline

1. **Feature Extraction:** The user's input is sent to an LLM along with the properties schema and a system prompt describing the properties the LLM should extract from the question. The schema should be designed to be as dynamic as possible. The LLM then returns a list of extracted properties relevant to the query.
2. **Feature Validation:** In this stage, each extracted feature is checked against the appropriate table in the database using string-matching algorithms. This step aims to correct spelling mistakes and abbreviations. Once a value is found, both the value and its primary key are added to the extracted value.
3. **SQL Querying:** The cleaned user prompt, along with system-specific prompts, is combined to guide the LLM in generating SQL queries that will answer the user's question. The constructed query is then passed to the SQL chain, which designs and executes the SQL queries against the underlying database. The SQL chain handles communication between the system and the database, retrieves the requested data, and prepares the results for presentation to the user.

By addressing potential issues such as spelling mistakes, abbreviations, or context-dependent interpretations and by including primary keys for items in the database during the feature extraction and validation stages, the system can better understand the user's intent and prepare the query for the subsequent stages. The query construction stage leverages the cleaned user prompt and system prompts to guide the LLM in generating SQL queries that accurately represent the user's information

needs. Finally, the SQL chain bridges the gap between natural language and structured data retrieval, executing the query and presenting the results to the user.

3.2.1 Choosing a Language Model

In the field of LLMs, new models are emerging on a weekly basis, with many of them available in semi-open source formats, making it possible to host them on one's own infrastructure. While this approach is interesting and can be cost-effective, as well as enabling better customization, open-source models like Llama and Mistral do not yet support advanced function calling as required by the LangChain framework. Gorilla-openfunctions-v2 (Patil et al., 2023) do work with functions, but during initial testing, it would most times stop mid-chain. It has been considered to fine-tune a model to handle the feature extraction part of the chain. While this is a viable option, it demands a substantial amount of data, is time-consuming, and would shift focus away from the overall goal of this thesis. Therefore, it was decided to focus on OpenAI's GPT-4.0-Turbo and GPT-3.5-Turbo models.

From table 3.3, one can see that there is a huge increase in the cost of using GPT 4 over GPT 3.5. For both input and output tokens, the cost increase is 1900%. This makes it clear that one should aim to use GPT 3.5 whenever possible (OpenAI, 2023c).

We aim to investigate how the two models work with the SQL chain. By optimizing the chain and making it easier for the LLM to perform its task efficiently, we can also pave the way for the use of locally hosted open source LLMs once they achieve function calling abilities.

Model	Input	Output
gpt-4-0125-preview	\$10.00 / 1M tokens	\$30.00 / 1M tokens
gpt-3.5-turbo-0125	\$0.50 / 1M tokens	\$1.50 / 1M tokens

Table 3.3: Pricing for Different Models (OpenAI, 2023c)

While thorough experiments and validation has been conducted with GPT-3.5-Turbo and GPT-4-Turbo, it is important to note that the landscape of LLMs is rapidly evolving, with new models and advancements emerging at a rapid pace. As a result, it is crucial to periodically re-evaluate the available options and consider incorporating newer models or techniques that may offer improved performance, capabilities, or cost-effectiveness for a specific use case.

3.2.2 Feature Extraction

The feature extraction module is an important step when it comes to improving the overall quality of natural language to SQL queries and in our proposed pipeline, this module is step one. During our prototyping phase, it became clear that user will not fully type all names, they will not know how events are labeled in the database, and even when having this information spelling mistakes might happen.

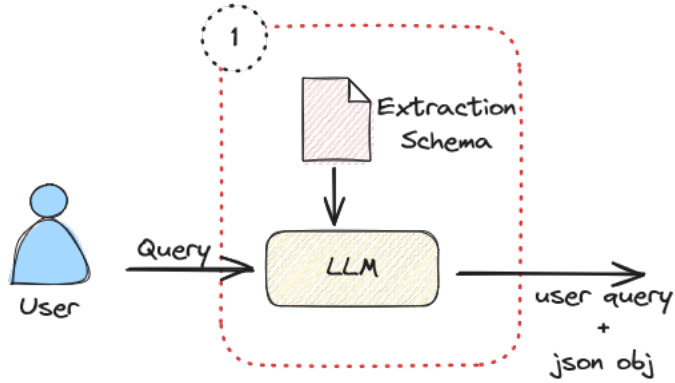


Figure 3.10: Extractor Module - Part 1 of Proposed Pipeline

In order to improve the LLM's comprehension of user queries and to ensure the input of complete names, the extractor was integrated into the chain. This extractor, which utilizes the LangChain library, functions by sending the query to the LLM and receiving structured output, as illustrated in Figure 3.10 and 3.11. The obtained data is then cross-referenced with the database to expand abbreviations, and seasons are adjusted using a custom function.

This procedure is crucial because the SQLchain often encounters difficulties in managing typos and abbreviations. In the best-case scenario, it may necessitate executing additional queries, while in the worst-case scenario, it may yield incorrect or no information at all.

A potential approach to address this issue could involve pre-evaluating the prompt with the LLM before starting the SQL chain. Although this introduces an additional step, it would streamline the process for instances where the LLM already possesses the required information, eliminating the need for a database lookup. However, a challenge arises when the LLM uses a different name for a team than what is stored in the database. For example, the team 'Leicester City FC' is recorded as 'Leicester City' in the dataset, but the LLM refers to it as 'Leicester City FC', resulting in a mismatch. Consequently, alternative solutions were explored, such as using vector databases and traditional algorithms.

The random nature of LLMs also introduced some extra complexity. Sending this question to the extractor chain: *"In what game did Messi score 3 or more goals"* will, in most cases, return a list of

```
{"person_name": ["Messi"], "in_game_event": ["goals"]}
```

However, this response could also return

```
{"person_name": ["Messi"], "in_game_event": ["game", "goals"]}
```

As one can see in the second response, the word "game" has been extracted as an in-game event rather than being recognized as part of the question.

This randomness was present across both GPT-3.5-Turbo and GPT-4.0-Turbo. While there is a slight improvement when using GPT-4.0-Turbo, the increase in time consumption and cost of using GPT-4.0-Turbo suggest that GPT-3.5-Turbo is the correct model for this task. This assumption was tested in Section 4.2.1.

Wrongly extracted features were an issue throughout much of the prototyping phase, and the result could change from run to run. This inconsistency was likely due to the inherent randomness and variability in the way LLMs processes and understands natural language inputs. Even with the same input, the models might sometimes interpret the text slightly differently, leading to variations in the extracted features.

A stricter system prompt was crafted to mitigate this, providing more explicit guidance and context to the language model. This aimed to reduce the potential for misinterpretations and improve the consistency of the extracted features. Additionally, we decided to create unit tests for the module. This approach makes it possible to verify that the extractor was providing correct results at a given time by comparing its outputs to a set of predefined, expected outputs.

Unit testing is a commonly used software development practice that helps ensure the correctness and reliability of individual components or functions. By establishing a suite of test cases that cover various input scenarios and expected behaviors, developers can automatically validate the extractor's performance and catch any regressions or deviations from the desired functionality. However, in most cases, one creates mock API calls, meaning that the test does not call the actual API endpoint but calls a function that returns a pre-defined value. Since our issue was with the response from the API itself, this would not be helpful.

Input

How many goals did Messi score for Barca in the 15-16 season?

Schema

```
schema = {  
  "properties": {  
    "person_name": {"type": "string"},  
    "team_name": {"type": "string"},  
    "in_game_event": {"type": "string"},  
    "season": {"type": "string"},  
  },  
  "required": (),  
}
```

→ {"person_name": "Messi", "team_name": "Barca", "in_game_event": "goal", "season": "15-16"}

Function

```
"name": "information_extraction",  
"description": "Extracts information from the passage.",  
"parameters": {"properties"}
```

Figure 3.11: Extractor Concept Figure

Because of testing against a live API endpoint, there is a cost associated with each unit test, and it was therefore decided not to implement the test in a CI/CD pipeline.

3.2.3 Feature Validation

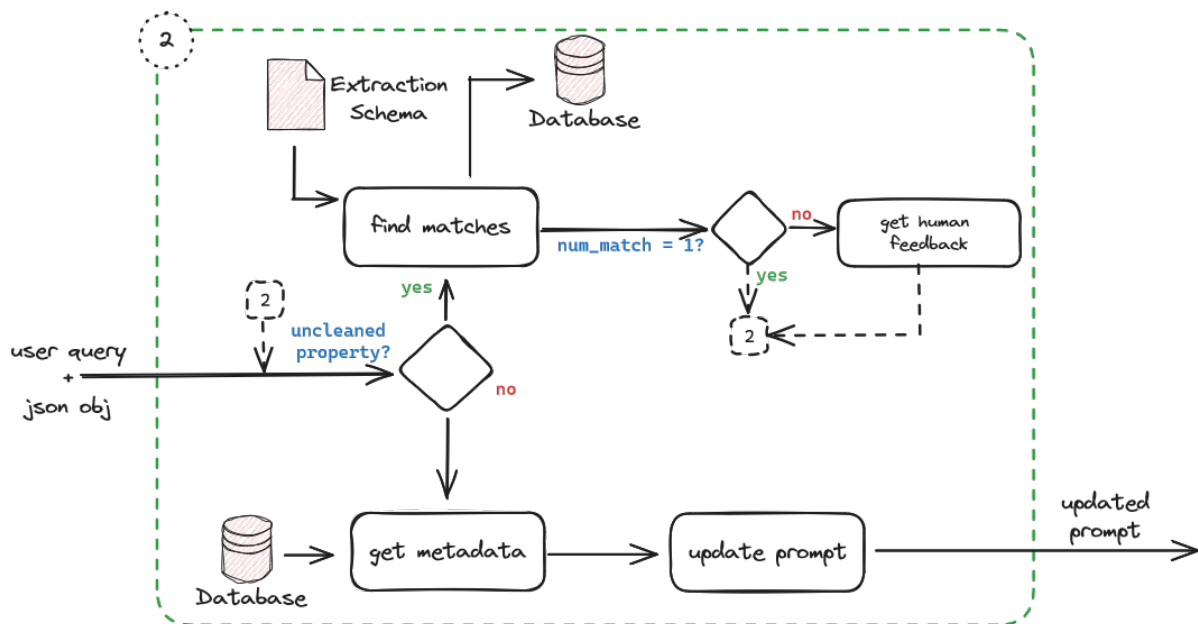


Figure 3.12: Validation Module - Part 2 of Proposed Pipeline

The initial approach to feature validation involved the use of a vector database combined with RAG techniques. However, despite the potential benefits of this approach, the results proved to be unreliable, especially when extracting short abbreviations like "Man" or "ManU". Another potential issue arises when the user only provides the first or last name of a player. There are several players with "Henry" as

a first name, and with no last name provided, the framework was unable to decide which Henry to validate against,

In light of the unsatisfactory performance of the vector database combined with RAG, the focus shifted toward exploring traditional lookup and word-matching methods. This move was driven by the need to identify a more robust and reliable solution for handling user prompts and queries. Among the potential solutions that emerged from this investigation were the Ratcliff-Obershelp algorithm, Levenshtein Distance, and Longest Common Subsequence (LCS).

The Ratcliff-Obershelp algorithm, which calculates the similarity between two strings based on the number of matching characters, offered promising results. Similarly, the Levenshtein Distance, a measure of the minimum number of single-character edits required to transform one string into another, also demonstrated acceptable accuracy. After careful evaluation and comparison of these two options, the choice ultimately fell on Levenshtein Distance as the preferred method for handling user prompts and queries.

This transition from complex embedding models to a traditional algorithm serves as a valuable reminder of the importance of revisiting established solutions when addressing challenges in AI. While cutting-edge AI techniques can be compelling, it is crucial not to overlook the potential benefits of more conventional methods. In some cases, advancements in AI can be discovered by re-examining how problems were previously tackled and adapting those solutions to the current context. This underscores the need for a balanced approach that considers both innovation and proven techniques in the pursuit of AI development.

Figure 3.13 showcases the process of extracting the name 'Messi' from the user-generated prompt. This extraction is an essential step, as it allows the system to identify and isolate the key information required for further processing. Once the value is extracted, it is employed to perform a lookup against the database, with the objective of finding the entry that most closely corresponds to the name. This comparison is based on string similarity measures, which evaluate the likeness between the extracted value and the entries stored in the database.

A threshold-based approach was implemented to optimize user experience and enhance process efficiency. If a match is found with a score above a threshold and there are no competing matches with the same score, the system automatically selects the highest scoring value as the correct match. This eliminates the need for

user intervention in cases where the system can confidently determine the correct match.

However, there may be instances where no match achieves a score higher than the threshold or multiple matches ties with a score over the threshold. In such cases, the system cannot confidently determine the correct match without additional input. To resolve this problem, the function will generate a list of the N most similar names to the extracted value and present this list to the user. The user is then asked to provide feedback by selecting the most accurate value from the list. This feedback not only helps the system determine the correct match for the current query but could also contribute to the system's learning and improvement for future queries.

Once the correct value is confirmed, either through automatic selection or user feedback, the process is repeated for each property associated with the user prompt. This iterative process ensures that all properties are accurately identified and matched with their corresponding entries in the database. Upon completing this process, a new property dictionary is created, which stores the confirmed values for all properties. This dictionary serves as the basis for further processing and analysis, enabling the system to provide accurate and relevant responses to the user's query.

An abbreviations table was incorporated into the database to provide additional support and enhance the functionality of the system. This table serves as a repository for multiple abbreviations associated with a team, as demonstrated in Table 3.4. To create abbreviation tables for teams and leagues, we provide the files

```
data/dataset/augmented_teams.csv  
data/dataset/augmented_leagues.csv
```

in the project's GitHub repo. The design of the extractor validator schema allows for abbreviation tables to be defined as needed. This ensures a dynamic and scalable design for this module. It also allows this module to be used on a completely unrelated database. Listing 1 shows the schema entry for the team name extraction. On line 5 - 7 the table and columns that one should check against is defined. On lines 9 - 11, the augmented table is defined together with its respective columns. These lines are optional, and if not included, there will not be any abbreviation check for the property.

The introduction of this table necessitates some manual effort to populate it with the correct values. However, the benefits of this addition significantly outweigh the

initial investment of time and effort, as it contributes to improved system speed and accuracy.

```

1  "team_name": {
2      "type": "array",
3      "items": {
4          "type": "string",
5          "db_table": "teams",
6          "db_column": "name",
7          "pk_column": "id",
8          "numeric": false,
9          "augmented_table": "augmented_teams",
10         "augmented_column": "augmented_name",
11         "augmented_fk": "team_id"
12     }
13 },

```

Listing 1: JSON Structure for the "team_name" Field

Team Name	Abbreviation Name
Manchester United	ManU
Manchester United	Man U
Manchester United	ManUnt
Manchester City	ManC
Liverpool	LFC

Table 3.4: Team Names and Abbreviation

The improvement in speed can be attributed to the priority given to the abbreviations table during the lookup process. When a user prompt is received, the function first checks for matches from the abbreviation table before proceeding with other lookup methods. If a match is discovered in the abbreviation table, the function can immediately return the full team name, bypassing the need for further lookup. This speeds up the overall process, enabling the system to provide faster responses to user queries.

The abbreviations table also offers end users the opportunity to customize the abbreviations for the entities they frequently engage with. This personalization feature not only caters to individual user preferences but also enhances the accuracy of the system. By allowing users to define their own abbreviations, the system can better understand and interpret user prompts, resulting in more accurate and relevant responses. This combination of speed and accuracy improvements ultimately leads to an enhanced user experience and increased satisfaction with the system. This module makes up part 2 of our proposed pipeline and is conceptually designed as shown in Figure 3.12.

Once all properties are correct, the user prompt is updated with the correct names and their respective primary keys where this makes sense. By applying this cleaning step, the SQL chain gets a lot more information, and one can omit the exploring steps of the chain. This will also guarantee that the framework gets the information requested.

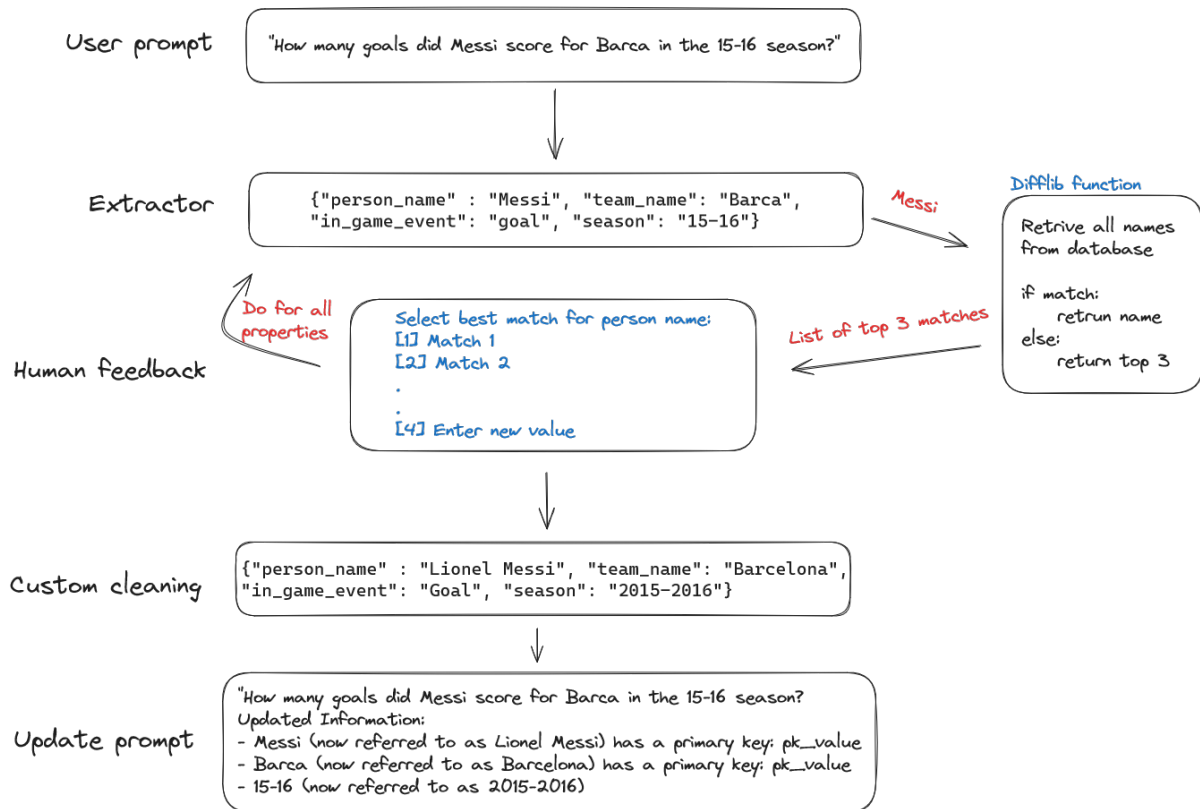


Figure 3.13: Cleaning of Extracted Data

During experiments conducted by sending users' prompts directly versus cleaning them first, it was observed that the cleaned prompts more frequently led to the correct answer. This highlights the significance of preprocessing user inputs to enhance response accuracy.

To delve deeper into the underlying issues, let's examine some examples. We can identify patterns and common challenges in the original prompts by analyzing specific instances where preprocessing made a difference.

- **User prompt:** "Did Real or Man U win the most games in the 2015 season?"

Output: Swansea won the most games in the 2014-2015 season with a total of 37 wins.

- **Cleaned prompt:** "Did Real or Man U win the most games in the 2015 season?"

Updated Information:

- Real (now referred to as Real Madrid) has a primary key: 30.
- Man U (now referred to as Manchester United) has a primary key: 7.
- 2015 (now referred to as the 2015-2016 season).

Output: In the 2015-2016 season, Real Madrid won 26 games, while Manchester United won 5 games. Therefore, Real Madrid won the most games in that season.

This analysis reveals that the initial user prompt incorrectly retrieved Swansea as the team with the most wins for the 2014-2015 season. This error occurred because the system could not resolve the references to "Man U" or "Real," and thus defaulted to querying for the team with the most wins in any season. Furthermore, the system incorrectly defaulted to the 2014-2015 season without considering alternatives.

In contrast, the cleaned prompt provided the system with clear references for teams and the specific season to look for, enabling it to deliver the correct answer based on the information found in the database.

Another example of this, with an even worse outcome, can be seen using this prompt:

- **User prompt:** "Who (and when) got the first yellow card in the game between Chelsea and Swansea in the 15-16 season?"

Output: The first yellow card in the game between Chelsea and Swansea in the 15-16 season was given at the following time:

- Description: A yellow card was given to a Chelsea player for a foul.
- Event Time: 2700 seconds (45 minutes into the game).

- **Cleaned prompt:** "Who (and when) got the first yellow card in the game between Chelsea and Swansea in the 15-16 season? Updated Information:

- Swansea (now referred to as Swansea) has a primary key: 6.
- 15-16 (now referred to as the 2015-2016 season).
- First yellow card (now referred to as yellow card).

Output: The first yellow card in the game between Chelsea and Swansea in the 2015-2016 season was received by Jonjo Shelvey.

The initial observation reveals an incorrect spelling of "Swansea," a mistake used in the uncleaned version of the prompt, leading to a failed query that returns an empty string. Consequently, the LLM begins to generate inaccurate information, fabricating events that do not exist.

When looking at the cleaned prompt, it's clear that it returns more accurate results, successfully identifying the first of three yellow cards and extracting commentary data that describe the yellow card. However, it falls short of providing comprehensive details, failing to specify the player's team, or at what time of the game the event happened.

The last example highlights a fascinating challenge. Here, the user requests Lionel Messi's full name, a task the LLM could easily accomplish without resorting to a SQL chain. However, due to the operational design of the chain, it attempts a database lookup. Using the user's input, it fails to find an exact match and tells the user this.

- **User prompt:** "What is Messi's full name?"

Output: I am unable to retrieve Messi's full name from the database.

- **Cleaned prompt:** "What is Messi's full name? Updated Information:

- Messi (now referred to as Lionel Messi) has a primary key: vgOOdZbd.

Output: Messi's full name is Lionel Messi.

The cleaned prompt illustrates another intriguing aspect: even though the prompt provides Messi's full name, the system fetches this information using a database query instead of using the directly provided name.

During the development of this feature, a benchmark set was created to track progress across iterations more easily. The benchmark was designed to verify that the extractor extracted all the expected values and that the values could be found in the database.

3.2.4 SQL Agent

The final step of the pipeline is the SQL agent provided by the LangChain framework. Figure 3.14 presents a conceptual overview of this agent. Extracted and validated

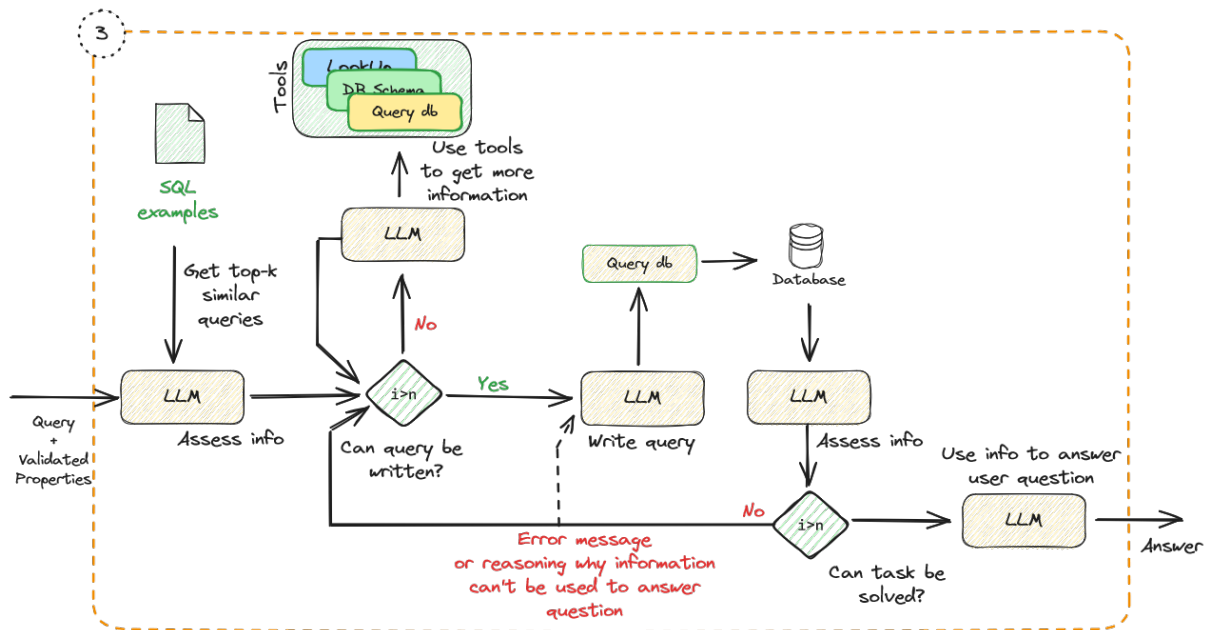


Figure 3.14: Conceptual Overview of SQL Agent

properties are used in a pipeline aimed at retrieving information from the database to help answer the user's question.

The first step uses vector search on the query to get the top k similar queries from the SQL examples file. The LLM then assesses if it can answer the question based on this information alone. If not, it utilizes internal tools like "get_db_schema" or "describe_table" to retrieve additional information about the database schema and contents. This schema exploration can be repeated multiple times before proceeding. If the query is highly similar to one or more examples, the agent may choose to skip this exploration step.

Next, the agent writes an SQL query intended to solve the task. It then runs this query against the database using the "query_db" tool. Note that the LLM can potentially write queries that alter or drop tables, so care must be taken when querying production databases.

Once the query returns results, the LLM re-assesses whether this information can answer the original question. If so, a final answer is composed. Otherwise, the agent cycles back to retrieve more schema information or revise the query.

The flow follows a cycle of retrieving relevant information, writing queries, executing queries, and evaluating whether the results are sufficient to answer the question. This iterative process allows the agent to come up with a response through incremental knowledge gathering and query improvements. In cases where the Agent can't come

up with a good answer, it is instructed to return *"I don't know"* by the system prompt.

The system prompt is an instruction that can be sent to a LLM to tell it how it should act. The system prompt used by our system is based on LangChain's (Langchain, 2024a) base prompt and is as follows:

```
You are an agent designed to interact with a SQL database.
Given an input question, create a syntactically correct {dialect} query to run,
then look at the results of the query and return the answer.
Unless the user specifies a specific number of examples they wish to obtain,
always limit your query to at most {top_k} results.
You can order the results by a relevant column to return the most interesting
examples in the database.
Never query for all the columns from a specific table, only ask for the
relevant columns given the question.
You have access to tools for interacting with the database.
Only use the given tools. Only use the information returned
by the tools to construct your final answer. You MUST double check
your query before executing it.
If you get an error while executing a query, rewrite the query and try again.

DO NOT make any DML statements (INSERT, UPDATE, DELETE, DROP etc.) to the
database.

If the question does not seem related to the database, just return
'I don't know' as the answer.
DO NOT include information that is not present in the database in your answer.
```

Here are some examples of user inputs and their corresponding SQL queries. They are tested and works. Use them as a guide when creating your own queries:

The prompt starts by defining the role of the LLM and sets guidelines, such as the SQL dialect to use and the number of results to return. It then informs the system what tools are available for communication with x|the database. To lower the chance of hallucinations (generating text that is not grounded in factual information), the LLM is asked only to use information returned from the tools in the final answer.

An interesting line in the prompt is the "DO NOT make any DML statements.." This

line aims to prevent the LLM from altering the database in any way or form.

The prompt then instructs the system on what it should say if it cannot find relevant information.

The final paragraph of the prompt is where SQL examples are provided. These examples act as guidelines to help the LLM design a query to answer the question.

The prompt aims to provide a structured environment for the LLM to interact with the database while also setting clear boundaries and guidelines to ensure the integrity and security of the database.

3.3 Proposed Evaluation

3.3.1 Metrics for Evaluation

Extractor-validator For the extractor-validator module, we propose to use questions 1-20, paired with ground truth for these questions, and evaluate the extractor and the validator as separate modules. We ask one question for each module:

1. Did the extractor successfully extract all the expected items? Options: Yes, No or Partial
2. Did the validator successfully validate all the extracted items? Options: Yes, No, Partial

This scoring system, with the options of Yes (●), No (▲), and Partial (■) for each component, allows for a quantitative assessment of the extractor-validator chain's ability to accurately identify and validate the relevant information from the given questions. The overall performance could then be evaluated by analyzing the distribution of scores across all ten questions. The questions contain some spelling mistakes, and the chain leverages the aberration tables for both team names and league names.

Since these two modules are interconnected, we chose to test them in the same chain instead of doing module-by-module.

SQLRAG The RAG module plays a pivotal role in guiding the framework to solve the task effectively. To evaluate its performance, we suggest an additional 40 test questions, so we, in total, got 60 questions, each manually paired with a corresponding SQL query from `/src/conf/sqls.json`. The pairing should be done using experience.

To measure the efficiency of the RAG solution, we propose testing with four different numbers of *few-shot*, FS of 1,2,3 and 5. To calculate the hit rate, we see if the returned list of few-shots contains the SQL query mapped to the question. If yes, record a hit, if no, mark it as a miss.

Ablation To fully understand the contributions of the different parts of our pipeline, we suggest an extensive ablation study where all possible configurations of our pipeline were tested with Q1-Q10.

1. **Only SQL Agent:** Sending the prompt directly to the SQL agent without RAG. This is the last step in every test.
2. **SQL Agent with RAG:** Sending the prompt directly to the SQL agent with RAG
3. **Extractor only:** Sending the prompt with extracted values to the SQL agent
4. **Extractor and SQL RAG:** Sending the prompt with extracted values to the SQL agent. The SQL agent uses RAG to get examples
5. **Extractor and Validation:** Sending the prompt with extracted and validated values to the SQL agent
6. **Extractor, Validation, and SQL RAG:** Sending the prompt with extracted and validated values to the SQL agent. The SQL agent uses RAG to get examples

By testing the components in these different configurations, it was possible to quantify the contribution of each component to the final solution. This information is crucial for understanding the strengths and weaknesses of the various parts of the system and identifying areas for potential improvement.

3.3.2 Queries for Evaluation

In order to evaluate the framework's capabilities effectively, we have carefully chosen questions that encompass a wide variety of scenarios. These questions have been

strategically designed to progress in difficulty, starting from Q1, which simply requires finding an entity in the database, to Q20, demanding identification and connection of five distinct entities.

Our selection also includes questions that involve aggregation and list-based queries to push the boundaries of the framework. Furthermore, we have intentionally incorporated questions like Q18, which are specifically crafted to challenge the framework, particularly the context awareness and accuracy of the extractor-validator component.

We define a set of 21 questions to be used during the experiments. The questions are split into several categories, such as

1. Existence and membership queries (Q - 1,9,13,17,20)
 2. Statistical and aggregation queries (Q - 2,3,11,12,16,18)
 3. Record retrieval and listing queries (Q - 6,7,10,21)
 4. Event dependent queries (Q - 4,5,8,14,15,19)
- **Question 1:** Is Manchester United in the database?
 - **Question 2:** Give me the total home goals for Bayern M in the 2014-15 season
 - **Question 3:** Calculate home advantage for Real Madrid in the 2015-16 season
 - **Question 4:** How many goals did Messi score in the 15-16 season?
 - **Question 5:** How many yellow-cards did Enzo Perez get in the 15-2016 season?
 - **Question 6:** List all teams that played a game against Napoli in 2016-17 season in seriea? Do not limit the number of results
 - **Question 7:** Give all the teams in the league ucl in the 2015-2016 season?
 - **Question 8:** Give me all games in epl with yellow cards in the first half in the 2015-2016 season
 - **Question 9:** What teams and leagues has Adnan Januzaj play in?
 - **Question 10:** List ALL players that started a game for Las Palmas in the 2016-2017 season? Do NOT limit the number of results

- **Question 11:** Did Ajax or Manchester United win the most games in the 2014-15 season?
- **Question 12:** How many yellow and red cards were given in the UEFA Champions League in the 2015-2016 season?
- **Question 13:** Are Messi and C. Ronaldo in the database?
- **Question 14:** How many goals did E. Hazard score in the game between Bournemouth and Chelsea in the 2015-2016 season?
- **Question 15:** How many yellow cards were given in the game between Bayern Munich and Shakhtar Donetsk in the 2014-15 UEFA Champions League, and did anyone receive a red card?
- **Question 16:** Make a list of when corners happened in the English Premier League (EPL) 2015-2016 season. Aggregate by a period of 15 minutes.
- **Question 17:** What league is Manchester United, Arsenal, Bournemouth, Real Madrid, Chelsea, and Liverpool in?
- **Question 18:** How many players have "Aleksandar" as their first name in the database, and how many goals have they scored in total?
- **Question 19:** What did the commentary say about the game between Arsenal and Southampton in the 2016-17 season?
- **Question 20:** Have Mesut Ozil, Pablo Insua, or Alex Pike played for West Ham or Barcelona?
- **Question 21:** How many games has Real Madrid and Malaga played against each other? List each game with the date and score. Also say who was the home team. Output should be: home_team_name, away_team_name, date, home_team_score, away_team_score

3.4 Proposed User Interface

To further enhance user experience and effectively demonstrate the framework's capabilities, we propose the integration of a user inference module built upon CainLit. This incorporation yields two significant advantages. Firstly, CainLit's built-in step function enables the visualization of the data extraction and validation process,

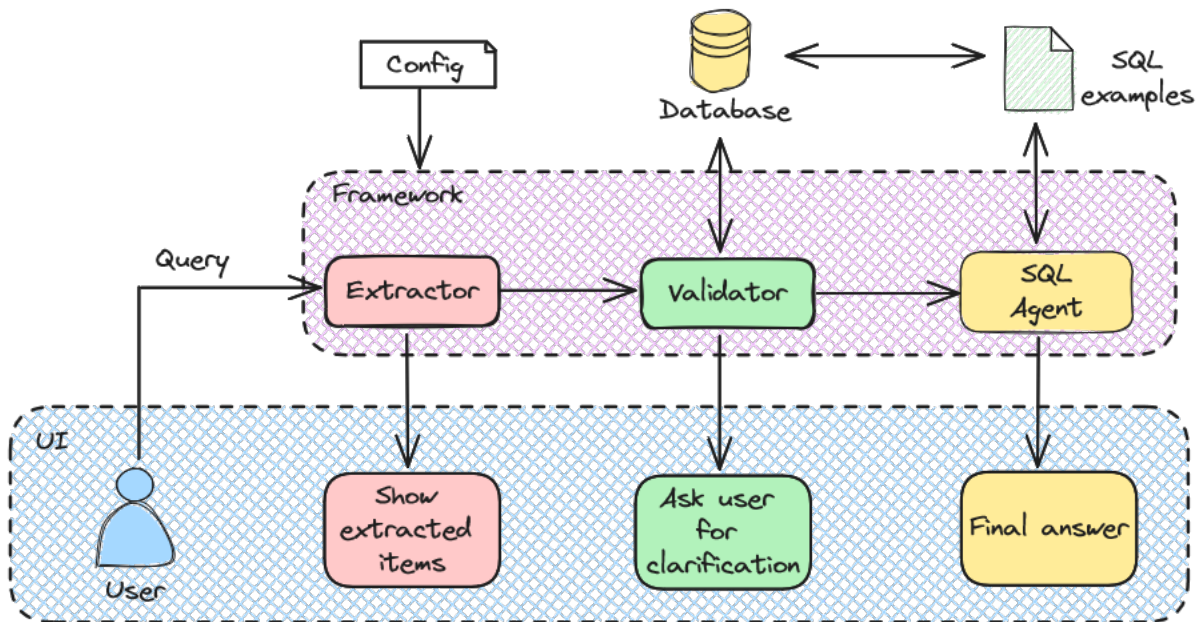


Figure 3.15: Proposed UI

providing users with a clear understanding of the framework’s operations. In addition, the validation function can present users with buttons to select the correct option in instances where the module is uncertain. To solve the potential issues with wrongly extracted features, we can implement a *“No update”* function, allowing users to discard these values and improve both the updated prompt and the output.

As shown in Figure 3.15, the user first asks the system a question. The extractor module extracts the results and feeds the extracted values back to the UI and to the validation module. If the validator needs feedback, the user will be asked to choose from N buttons to clarify what entity they want. The validated properties are then sent to the SQL Agent, which will query the database and provide the final answer to the UI.

Secondly, CainLit’s native support for Markdown formatting enables the generation of visually appealing and well-structured output from the LLM. This feature enhances the overall aesthetic of the framework’s output, making it more engaging and easier to comprehend for users. By leveraging these capabilities, the framework’s usability and effectiveness can be significantly improved, providing users with a more intuitive and informative experience.

A UI was designed using the proposed framework ChainLit and shared as open source in our GitHub repository (*‘SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries’*, 2024). By running the command

```
pip install -r requirements.txt
python setup.py
chainlit run app.py
```

from the root of the project, a web page with the UI is presented. In figure 3.16, the user asks, "Can you calculate Real Madrids [sic] home advantage for the 2015/16 season". The extractor module then extracts and presents the properties extracted from that prompt. We can also see that the validator validated the properties automatically in this instance. The user is then presented with the updated prompt, which is then sent to the SQL agent.



Figure 3.16: GUI Example 1

At the end of the chain, the entity "ChatBot" presents the user with the answer to the question.

This demonstrates that the framework is able to grasp soccer-related concepts such as home and away matches.

In Figure 3.17a, the user asks "Create a list of all games Lionel got a yellow card / Make the list in markdown with following columns [sic] / GamelId, HomeTeam, AwayTeam, Score, Date" Both player name, "Lionel" and in-game event, "yellow card"

is extracted. However, the database contains two players with the first name Lionel, and the validator module is unable to determine which one to select. Therefore, the system presents the user with a choice between the two options, plus an additional "No Update" option, ensuring that the correct information is retrieved.

If the user chooses "No Update", the property will not be checked against the database. This is a useful feature if the framework extracts labels of no interest.

Once the user makes a selection, the prompt is updated, and the output is presented in Figure 3.17b. In the final cell of this figure, the framework seamlessly converts the markdown list into a list object, thereby making it easier for the user to read the information and enhancing the overall user experience.

For a complete guide to running the framework, see Appendix B.2 or our GitHub repository ('SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries', 2024).

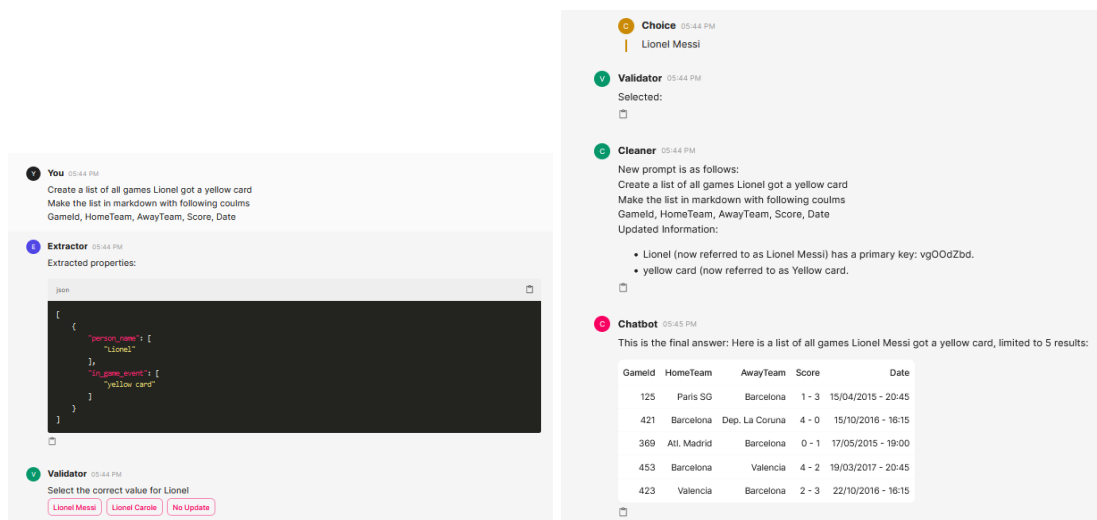


Figure 3.17: SoccerRAG GUI Example 2

It is worth noting that every extracted item from a question will try to auto-validate first, but if the similarity score is too low or two items have a matching score, the user will be asked to clarify.

For demo purposes, we deployed an instance of the application¹. This application requires the user to add an OpenAI API key. Since this is purely to demonstrate the framework, it's not set up with multiple sessions. That means you should remove your

¹<https://simulamet-host-soccerrag.hf.space/>

API key once it is done, or it will leak to other user sessions. For continued usage, it's suggested that the GitHub repository be cloned and the application hosted locally.

3.5 Chapter Summary

In this chapter, we explored a regular RAG solution before deciding on a SQL-based retrieval schema. The importance of providing the framework with the correct data structure is an important step in improving overall performance. The dataset was then transformed from folders with JSON files to a structured SQL database. A solution was proposed for the validation of entities, and we designed a pipeline consisting of Feature Extraction, Feature Validation and SQL Querying with a few-shot RAG technique. The need for abbreviation tables was discussed, and some simple experiments were done to explore this need.

Furthermore, we looked at the pricing difference between OpenAI's LLMs, and identified a potential saving by utilizing GPT-3.5-Turbo instead of GPT-4.0-Turbo. This is another motivation to construct a better framework around the SQL Agent, as a better-structured input demands less of the LLM used.

A UI for the framework was proposed and implemented. This component is important since it makes it easier for non-technical users to interact with the framework. We also proposed a set of experiments to evaluate our framework.

In the next chapter, we will conduct the experiments proposed to evaluate the framework.

Chapter 4

Experiments and Results

In this chapter, we test the framework to verify the justification of the different modules added to the framework. Firstly, we will calculate query complexities, to be able to get a difficulty score for each of the test questions used in the ablation study. We will then perform a per-component evaluation and an ablation study, before evaluating the framework end-to-end in terms of execution time, cost, robustness, and model laziness.

4.1 Query Complexity Analysis

For each question from the user, the system will generate and run an SQL query against the database. These SQL queries has different complexity based on the question the user ask. For example, "Is Manchester United in the database?", can be solved with:

```
SELECT t.*  
FROM teams AS t  
WHERE t.name = 'Manchester United';
```

Whereas the question, "List all teams that played a game against Napoli in 2016-17 season in seriea? Do not limit the number of results" needs a more complex query, such as:

```

SELECT DISTINCT
CASE
WHEN t1.name = 'Napoli' THEN t2.name
ELSE t1.name
END AS opponent_team_name
FROM
games g
JOIN
leagues l ON g.league_id = l.id
JOIN
teams t1 ON g.home_team_id = t1.id
JOIN
teams t2 ON g.away_team_id = t2.id
WHERE
l.name = 'italy_serie-a'
AND g.season = '2016-2017'
AND (t1.name = 'Napoli' OR t2.name = 'Napoli')
ORDER BY
opponent_team_name;

```

In order to quantify the complexity, we chose to look at the Halstead Metrics (Halstead, 1977), and specifically the "Difficulty" calculation. These calculations are expected to give insights into how difficult a question would be to answer for the framework. It is worth noting

$$D = \frac{n_1}{2} \times \frac{N_2}{n_2}$$

Where

- D is the difficulty of understanding the query.
- n_1 is the number of distinct operators.
- n_2 is the number of distinct operands.
- N_2 is the total number of operands.

Operators are symbols or keywords that specify the type of computation to be performed. They represent actions like addition, subtraction, multiplication, division,

Table 4.1: Query Complexity Analysis

Query Complexity Metric	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Number of distinct operators, n_1	5	6	8	8	8	10	10	9	8	9
Number of distinct operands, n_2	5	14	31	26	26	31	20	36	28	25
Total number of operators, N_1	5	11	33	18	18	22	16	27	19	19
Total number of operands, N_2	5	16	110	26	26	38	22	41	31	26
Vocabulary, v	10	20	39	34	34	41	30	45	36	34
Length, N	10	27	143	44	44	60	38	68	50	45
Volume, V	33.22	116.69	755.81	223.85	223.85	321.45	186.46	373.45	258.50	228.94
Difficulty, D	2.50	3.43	14.19	4.00	4.00	6.13	5.50	5.12	4.43	4.68
Effort, E	83.05	400.09	10727.66	895.39	895.39	1970.20	1025.54	1913.91	1144.77	1071.42
Time to understand (s), T	4.61	22.23	595.98	49.74	49.74	109.46	56.97	106.33	63.60	59.52

comparison, logical operations (AND, OR, NOT), and more. In SQL, operators also include keywords that control the flow of queries, such as SELECT, JOIN, WHERE, and others that dictate how data should be fetched, joined, or filtered.

Operands are the entities on which operators act. They can be constants, variables, columns, or more complex expressions. In essence, operands are the values or references that are manipulated or examined by the operations defined by the operators. In an SQL query like `SELECT t.* FROM teams AS t WHERE t.name = 'Manchester United', t.*, teams, t and 'Manchester United' are operands, while SELECT, FROM, WHERE, AS and = are operators.`

The results of the calculations are presented in Table 4.1, and show that Q3 is ranked as the most complex query, with a difficulty rating of 14.19. By analyzing the calculations in Appendix A.3, we see that the length of the query is what contributes the most to this complexity score.

It is worth noting that the queries used in the calculations were hand-made, and tested to confirm they return the data needed to answer the question. We can however not guarantee that the SQL Agent generate the exact same SQL query when answering the questions, as there is the possibility that more than one query can return the needed data.

4.2 Per-Component Evaluation

4.2.1 Extractor-Validator

Following the setup in Chapter 3.3.1, we tested Q1-20 on the extractor-validator module using GPT-3.5-Turbo. The experiments were initially planned to run on both GPT-3.5-Turbo and GPT-4.0-Turbo. However, after completing them on GPT-3.5-

Turbo, a decision was made to re-run the test only for the two questions that did not receive a perfect score.

The results, presented in Table 4.2 and Appendix A.2, show that GPT-3.5-Turbo only had problems with Q1, Q15 and Q18. In Q1, it extracted an extra entity without causing any issues within the framework. In Q15, GPT-3.5-Turbo missed the red card, and in Q18 it extracted the player name.

When using GPT-4.0-Turbo, the framework gave a correct answer for Q15. Nonetheless, it still extracted the player name for Q18. This observation suggests that while GPT-4.0-Turbo performs better on more complex tasks, the extractor chain is context-unaware, irrespective of the underlying language model.

Table 4.2: Extractor-Validator Evaluation

Module Evaluation	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Extractor	■	●	●	●	●	●	●	●	●	●
Validator	●	●	●	●	●	●	●	●	●	●
Overall	●	●	●	●	●	●	●	●	●	●
	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
Extractor	●	●	●	●	▲ / ●	●	●	■	●	●
Validator	●	●	●	●	● / ●	●	●	●	●	●
Overall	●	●	●	●	■ / ●	●	●	●	●	●

From the experiments, it can be assumed that the extractor-validator chain can be run with GPT-3.5-Turbo without significant loss of accuracy. Knowing this, all experiments following this experiment, uses this model for feature extraction, regardless of what's being used in other part of the solution.

4.2.2 SQL Agent and RAG

We conducted four rounds of testing, varying the few-shot (FS) variable, which controls the number of examples returned by the RAG module. Our objective was to get a balance between minimizing the number of samples returned while maintaining high accuracy levels.

However, as shown in Figure 4.1, reducing the FS variable to 1 had a severely negative impact, and resulted in a hit rate of 63.33%. Setting FS to 2 gave 80%, FS to 3, 85% and FS to 5 gave 90%.

There are some negative effects of including too many examples. Firstly, more examples mean more tokens, making the framework both slower and more costly. In

Table 4.3: Combined Hit Rate Q1-Q60 - FS=1, FS=2, FS=3, FS=5

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
xxxx	xvvv	vvvv	vvvv	xvvv	xxvv	xxvv	vvvv	xvvv	xxvv
Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
xxxx	vvvv	vvvv	vvvv	xxxx	xxxx	xxvv	xxxx	vvvv	vvvv
Q21	Q22	Q23	Q24	Q25	Q26	Q27	Q28	Q29	Q30
xxxx	vvvv	vvvv	vvvv	vvvv	vvvv	vvvv	vvvv	vvvv	vvvv
Q31	Q32	Q33	Q34	Q35	Q36	Q37	Q38	Q39	Q40
vvvv	vvvv	xvvv	vvvv	vvvv	vvvv	vvvv	xvvv	xxxx	xvvv
Q41	Q42	Q43	Q44	Q45	Q46	Q47	Q48	Q49	Q50
vvvv	xvvv	vvvv	vvvv	vvvv	vvvv	xvvv	vvvv	vvvv	vvvv
Q51	Q52	Q53	Q54	Q55	Q56	Q57	Q58	Q59	Q60
vvvv	xvvv	xvvv	vvvv	vvvv	vvvv	xxvv	vvvv	vvvv	vvvv

addition to this, including more queries, can confuse the LLM, since it's not always clear what query it should use as a base. Therefore, we set FS=2 as the default for the framework, and the results for this setting, together with the additional 40 questions, are presented in Appendix A.5.

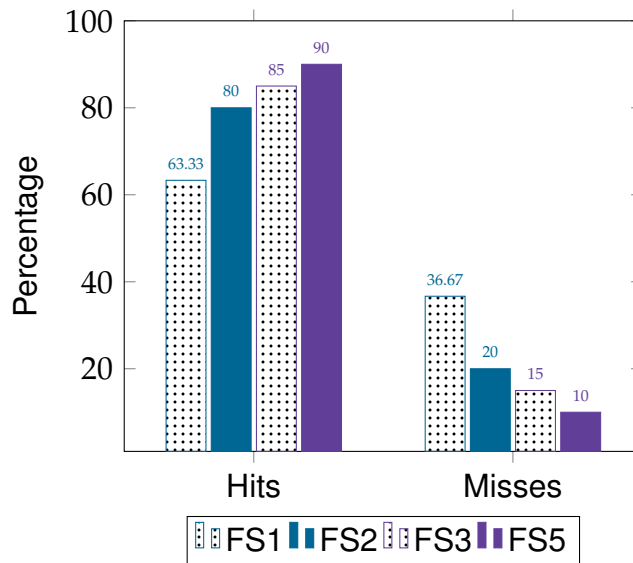


Figure 4.1: Hit Rate for SQLRag Based on Test Results

In Table 4.3 we see that some questions fail for all settings of FS. These insights can be used to either add new SQL examples or modify the description of the examples given. There are also some cases when no good SQL examples are present, and we chose what we deemed to be the closest match. This choice is done from personal preference, and might not always line up with what a vector search would pick.

4.2.3 Ablation Study

For the ablation study, we decided to test with both GPT-3.5-Turbo and GPT-4.0-Turbo to investigate the difference in quality, speed, and cost between the two models.

From table 4.4, we see that with the full pipeline, GPT-3.5-Turbo and GPT-4.0-Turbo have comparable performance. On question 7, GPT-4.0-Turbo fails as a result of laziness, not due to a wrong query. It is also interesting to see that all pipelines for both models failed on question 8. From table 4.1, we see that it's not the most difficult query, but the query returns a lot of data. This creates a situation where the LLM gets overwhelmed and struggles with processing it in a meaningful way.

From the experiment, we can also conclude that the extractor and validation module play a vital role in aiding the SQL Chain. It is also clear that GPT-4.0-Turbo is a more capable model, and manages to solve 60% of the questions without using SQL Rag.

An interesting observation in regard to the results presented in table 4.1, is that Q8, is only the 4th most difficult query. However, the query has the highest vocabulary (distinct operands + distinct operators), thus the most difficult for a LLM to construct.

This suggests that one can optimize the way query complexity is calculated in the age of LLMs, as the Halstead Metrics were proposed in 1977 and targeting human understanding.

Table 4.4: Ablation Study Results for Question 1 to 10

LLM	Pipeline	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
GPT-3.5-Turbo	1	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	2	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗
	3	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	4	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
	5	✗	✓	✓	✗	✗	✓	✗	✗	✗	✗
	6	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
GPT-4.0-Turbo	1	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗
	2	✓	✗	✓	✗	✗	✗	✗	✗	✓	✓
	3	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗
	4	✓	✗	✗	✗	✗	✗	✗	✗	✓	✓
	5	✓	✗	✓	✗	✓	✓	✓	✗	✗	✓
	6	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓

4.3 End-to-End Pipeline Evaluation

4.3.1 Execution Time

During the development process, we observed that OpenAI's response latency varied depending on the time of day. We hypothesized that this latency fluctuation was related to server load, likely influenced by working hours in the United States. To confirm this hypothesis, we decided to run questions 1 through 10 five times during peak hours (when server load was expected to be high, defined as 8AM-5PM EST, or 2PM-11PM CET) and five times during off-peak hours (when server load was expected to be low, defined as 5PM-8AM EST, or 11PM-2PM CET).

The experiment reveals that there are peak usage hours when utilizing the OpenAI language models, resulting in increased execution times. Figure 4.2 illustrates that GPT-4.0-Turbo experiences the most significant impact during peak hours, with a 46.15% increase in execution time compared to non-peak hours. For GPT-3.5-Turbo, the execution time increase is 22.56%. The contribution of each individual question to the overall execution time is provided in Appendix A.4.

During peak usage hours, the demand for computing resources from the OpenAI language models is higher, leading to longer queuing times and potentially slower response times. This phenomenon is more pronounced for GPT-4.0-Turbo, the more advanced and computationally intensive model, which experiences a larger relative increase in execution time compared to GPT-3.5-Turbo.

It is worth noting that the execution time increase observed in this experiment may vary depending on factors such as the specific workload, the complexity of the prompts, and the overall system load on the OpenAI infrastructure at the time of the experiment.

4.3.2 Cost

Our framework makes several calls to OpenAI's APIs, each having a small cost. After concluding that GPT-3.5-Turbo provides acceptable results for the extractor chain, we decided on using that in all chains to cut both costs and execution time. However, for more complex questions, GPT-4.0-Turbo provides better quality answers.

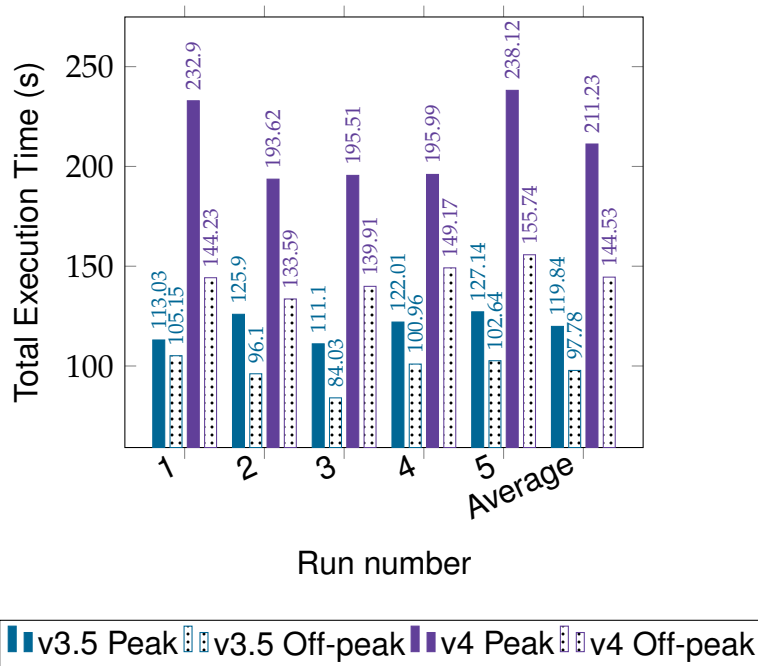


Figure 4.2: Total Execution Time for Questions 1-10

When deciding between the two models, both cost and time are important factors. As discussed in chapter 3.2.1, GPT-4.0-Turbo is 1900% more expensive than GPT-3.5-Turbo.

Model	Extractor	SQL Agent	Total
gpt-4-0125-preview	\$0.00347	\$0.05919	\$0.063
gpt-3.5-turbo-0125	\$0.000196	\$0.00288	\$0.003

Table 4.5: The cost for Q21 with GPT-3.5-Turbo and GPT-4.0-Turbo

We utilized LangSmith to track costs of each API call made by the framework and from Table 4.5 we see that for Q21 the total cost was approx 2000% more expensive with GPT-4.0-Turbo than GPT-3.5-Turbo.

4.3.3 Robustness

Due to the non-deterministic nature of LLMs, the robustness of the framework is a critical concern. The inherent stochasticity in LLMs, comes from the probabilistic foundations and can lead to variability in performance across different runs, even when the input remains the same. This variability poses a challenge for ensuring the reliability and stability of the framework in practical applications.

There is not much that can be done to combat this issue, except from creating stricter prompts.

4.3.4 Model Laziness

Model laziness is an issue where GPT-3.5-Turbo and GPT-4.0-Turbo either stop the answer or the chain prematurely. We see this across several answers, especially when the framework is asked to return a list of items.

An example of this can be seen in Table A.13 and A.14. When using the full pipeline, both models write and execute the correct SQL query, but GPT-4.0-Turbo only returns the first 5 teams. This happens even if the model is instructed to limit the response to 10 items, if not anything else is specified.

4.3.5 LLM capability

Throughout the experiments, we see that GPT-4.0-Turbo perform better than GPT-3.5-Turbo, when helper functions are removed. For complex questions, asking about more than one statistic, such as "List all players that got a yellow card in game X. Also list players that got a red card in that game", GPT-4.0-Turbo seems like a more capable model.

4.4 Chapter Summary

The experiments conducted in this chapter show that the Extractor-Validator module plays an important part in enhancing the overall performance of the framework. The reason for this is that even if an LLM would try to fix spelling mistakes and abbreviations, it does not know the naming convention within the database. If a player asked for statistics on a player called Alessandro M, the agent would not be able on its own to decide if the user was asking for Alessandro Mastalli or Alessandro Matari (two players in the dataset).

In this regard, the introduction of augmentation tables also greatly help. A good example for this is league names. The league names come in a short form from SoccerNet, where the English Premier League is named `england_epl`. Without the extraction and validation, the SQL Agent would have to query for all league name, and then make a qualified guess of what league the user was asking for.

It was also discovered that the difficulty parameter of the Halstead Metrics did not, as we expected, give the best estimate of query *difficulty*, and one should instead look at grading questions around the *vocabulary* parameter. However, further research should be done to find an optimal metric for query complexity in regard to LLMs.

The results from the ablation study also suggest that our framework can greatly improve GPT-3.5-Turbo, making the pipeline both faster and cheaper to use.

Chapter 5

Discussion

5.1 Competitor Analysis

Cloud providers like AWS¹ and Azure² are providing their own solutions for natural language to SQL. These solutions provide both the query framework and hosting of the database. A drawback of these solutions is the complexity and that you become provider dependent, as they want their users to host every part of the chain, from database, training data, and LLM within their ecosystem. That might be a perfect solution for an organization already using the ecosystem, but can act as a limiting factor within research or academic organizations. Another issue is that it's hard to verify how well the solution works. As for AWS, they mention *self-correcting queries*, but the documentation does not explain what happens when spelling mistakes or abbreviations are introduced. In table 5.1 we therefore gave AWS a medium (■) score on Feature validation. There are also several open-source frameworks that aim to solve the task of SQL querying using natural languages. These frameworks provide the option to bring your own LLMs and databases. One notable mention is Vanna.ai. They have a mature user inference and an optimized pipeline. By training a specific RAG solution for a database, they improve the quality of the SQL generation. One notable drawback of this solution is the setup complexity. To be able to train the RAG layer, the user has to provide high-quality training data consisting of queries, database structure, and general information about the dataset or company.

¹ <https://aws.amazon.com/blogs/machine-learning/build-a-robust-text-to-sql-solution-generating-complex-queries-self-correcting-and-querying-diverse-data-sources/>

² <https://techcommunity.microsoft.com/t5/analytics-on-azure-blog/revolutionizing-sql-queries-with-azure-open-ai-and-semantic/ba-p/3913513>



Figure 5.1: Screenshot of Vanna.ai

Our analysis of the Vanna.ai framework revealed significant limitations in handling misspelled words, abbreviations, and even straightforward queries. In stark contrast to the polished "demo" solution advertised, the framework consistently faltered on simple inputs, rendering it unusable without extensive retraining efforts. Refining the model to meet practical requirements would necessitate a substantial investment of time and resources, calling into question the viability of Vanna as an off-the-shelf solution for real-world applications.

However, as shown in Figure 5.1, the solution has a polished framework able to plot data and read text out loud for the users. This UI is based on the Streamlit framework, a similar framework as our UI is built on.

Service	Open source	Scalability	Modality	Feature Validation	Cost	DB Connection	Can run local	UI
Ai2SQL	▲	●	▲	▲	\$9/mo	▲	▲	●
text2sql	▲	●	▲	▲	\$7/mo	▲	▲	●
AWS	▲	●	●	■		●	▲	●
Azure	▲	●	●	▲		●	▲	●
Vanna	●	●	●	▲	\$0/mo	●	●	●
SoccerRAG	●	●	●	●	\$0/mo	●	●	●
LangChain	●	●	●	▲	\$0/mo	●	●	▲

Table 5.1: Comparison of SQL Services

In addition to the mentioned applications, there are also tools like Ai2SQL and text2sql that enable users to define their database schema and then respond to natural language queries by generating a corresponding SQL query. However, a significant drawback of these methods is that they require users to manually query the database themselves.

LangChain is the base framework that SoccerRAG is based on. It's a modal and open framework, enabling a high degree of freedom and customization. Using LangChain directly will, however, require designing a wrapper for its core functions.

From what we found, no current framework but ours applies the extraction and validation of properties.

5.2 Model Laziness

When querying for larger lists, as in question 8 of the experiments, we can observe that model laziness plays a role in not returning the correct answer. This suggests that queries aimed at creating long lists of data or calculating statistics from 20 or more entities in the database are currently unsolvable with the existing models. As such, these types of queries should be left out of the scope of this project. An example of this behavior can be seen by asking, *"List ALL players that started a game for Las Palmas in the 2016-2017 season? Do NOT limit the number of results"*. This will write the correct query and return a list of 19 players, but the system gives this answer: *"The query returned the names of players who started a game for Las Palmas in the 2016-2017 season. Here are some of the players who started the games:"* before proceeding with listing 10 of them. During experimentation, this behavior would come and go, suggesting that there is no apparent reason for it to happen.

However, it is crucial to acknowledge that issues like model laziness are likely to

arise, and with OpenAI confirming this issue, we can expect models addressing this problem to be released in the future. Therefore, while we must exclude such queries from the current project's scope, it is essential to remain aware of these limitations and be prepared to revisit them as model capabilities improves.

5.3 Importance of Cleaned Input

The ablation study in Section 4.2.3 demonstrates the critical role of accurate input prompts when retrieving information from a database. While LLMs can sometimes correct mistakes in the prompts, there are situations where their ability to do so is insufficient, or the prompt is corrected to what the LLM finds most likely. This finding also raises the question of whether there is a need to provide clear guidance and instructions to users of such applications on how to formulate prompts correctly to achieve the desired results.

Inaccurate or poorly designed prompts can lead to irrelevant or incorrect information being retrieved, undermining the effectiveness of the system. By making users aware of the importance of precise and unambiguous language in their prompts, as well as any specific formatting or structure requirements, might improve the overall user experience. Providing examples of well-crafted prompts and common mistakes to avoid can be valuable in ensuring that users interact with the system effectively.

5.4 Modular Implementation of Extractor-Validator Module

A key objective of this thesis is to develop a highly generalizable solution. To achieve this, we have designed the extractor-validator module to be modular and reusable. Currently, it can be imported as a standalone component, allowing for easy integration with other systems. To adapt the module to a new database, users need to modify the *src/conf/schema.json* file to match the new database schema. While providing the module as a standalone package would further improve its modularity, this was beyond the scope of this thesis due to time constraints.

The implications of this is that competing frameworks can easily implement this

feature to improve their solution, thus contributing to the development of better frameworks for text-to-SQL frameworks.

5.5 New Models

One day before the deadline for this thesis, OpenAI released GPT-4o (OpenAI, 2024). Due to time constraints, we have not run extensive testing, but our preliminary results show that GPT-4o matches the speed of GPT-3.5-Turbo, as shown in Figure 5.2, while providing the same quality as GPT-4.0-Turbo. GPT-4o also cost half as much as GPT-4.0-Turbo, but still 10 times more expensive as GPT-3.5-Turbo.

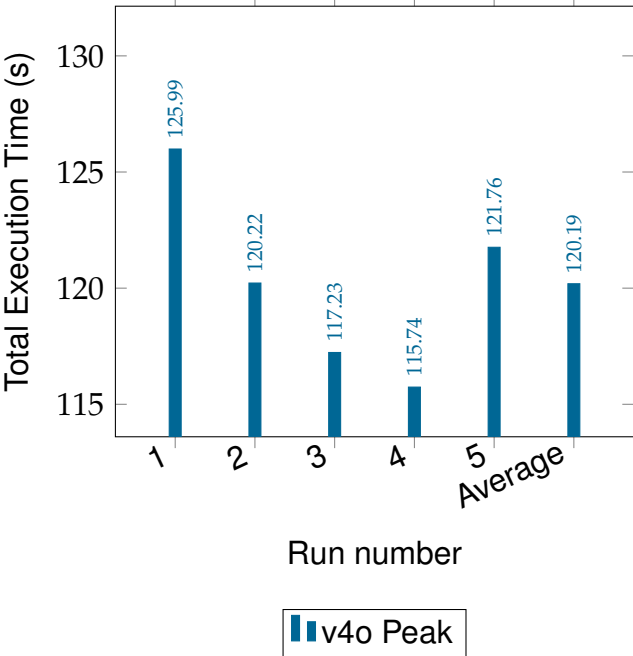


Figure 5.2: Total Execution Time for Questions 1-10 for GPT4o

Due to the modular setup of the framework, we were able to test this model just by changing the "OPENAI_MODEL" parameter in the .env file. This paves the way for easy updating of new models as they emerge. Easy updating of the model is key to keeping up with the ever-changing LLM field.

5.6 Potential Use Cases

The framework has several use cases, and we can be divided into two subcategories.

5.6.1 SoccerNet Use Cases

Since we use SoccerNet data while building this framework, we propose some use cases for this data.

1. Game summary

Using SoccerRAG as a module, a system can be developed to generate informative summaries of games, leagues, and player performances. This could enable users to quickly grasp key insights and trends from large datasets.

By retrieving statistics using the SQL Agent, it would then be possible to develop a traditional RAG based system, that would only have access to relevant data.

This has the potential to greatly reduce the search space for the RAG module, and thus improve the accuracy of the system.

2. Video retrieval

This framework is an important step towards being able to retrieve specific video segments, such as highlights, goals, and other events, from metadata.

By building upon SoccerRAG, systems can be created to provide user tailored video segments, providing valuable insights and new perspectives.

SoccerRAG can be instructed to return timestamps for events and the `player_event` tables. It is then possible to design a new module that extract the timestamps returned, and pass them to a function retrieving video clips from the timestamps.

3. Injury analysis

By combining video retrieval with injury-related metadata, a system could be developed to analyze the situations where player injuries happens, providing insights into the causes and potential prevention strategies. This could lead to improved player safety and reduced injury rates.

5.6.2 Natural Language SQL Retrieval

1. Enhancing Existing Frameworks

Our innovative approach to extracting properties and validating them against the database can significantly improve the accuracy of natural language search against databases. The modular design of our framework enables integration with existing text-to-SQL frameworks, potentially enhancing their capabilities.

2. Streamlining development with open-source UI

We have made our UI code available as open-source software 'SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries', 2024. This resource can accelerate the development of other applications built around LLMs, facilitating the creation of user-friendly interfaces for natural language SQL retrieval.

Chapter 6

Conclusion

This thesis shows that by introducing new elements to a standard SQL agent, one can greatly improve the performance of the system. Furthermore, we see that the stochastic nature of LLMs can create issues in a production environment since the same input given right after one another can result in different results. Deploying open-source language models instead of relying on the OpenAI API can mitigate this in the future.

6.1 Insights

- **Traditional algorithms still work.** In the early stage of development, we decided to use a LLM to clean entities before querying the database. This worked for some abbreviations, but for others, the LLM resolved the name to something else than we had in the database. We then went on to use vector search in an effort to validate the items without the help of the LLM, without an acceptable accuracy. It was not until we applied the Levenshtein Distance algorithm to the problem that we found the results to be acceptable.

This stands as a great reminder that traditional methods still play an important part in creating intelligent systems.

- **Things do not always work as advertised.** There are several existing frameworks claiming to be able to query databases. However, when we took a closer look, it was impossible to recreate their accuracy with custom datasets. This was a major setback for this thesis, as the main focus had to

pivot from video retrieval to creating a working natural language to SQL retrieval framework.

- **We need more capable open source LLMs.** While GPT-4.0-Turbo may appear to be the undisputed champion of LLMs at first glance, its limitations become apparent upon closer inspection. Since the model is only accessible through an API, we lack direct control over how it processes data. Furthermore, our experience has shown that OpenAI's models do not produce the same answers when prompted with the same prompt, which can be a significant issue in production environments.

The unpredictability of these models can lead to unreliable application performance, as it is impossible to ensure that the application will work as expected. In many cases, a less capable model that consistently produces the same output from the same input may be a better fit. This would allow for the creation of tailored helper functions to address the model's weaknesses, enabling the framework to function as expected.

6.2 Addressing The Research Questions

In Section 1.2 we asked one main question and four sub-questions. Here we will address these questions.

How does the system handle spelling mistakes and other input errors? As seen in section 4.2.3, the Extractor and Validator play a crucial role in providing the correct answer. This suggests that the SQL Agent doesn't handle spelling mistakes or abbreviations well on its own. When constructing the questions for the experiments, the aim was to write them as directly as possible, without considering that a LLM would need to understand them later. However, some extra instructions were needed in some questions, such as question 10, where *"Do NOT limit the number of result"* was added to combat model laziness.

Overall, we conclude that the system handle spelling mistakes satisfying, and that using a commanding natural language works well when asking questions.

How dynamic is the system? Can a change in the underlying database necessitate code changes? From experiments, it is clear that data added to the database will not necessitate code changes. However, when adding new tables to

the database, one has to update the SQL Rag document with new examples covering the new tables. If one omits this step, there is no guarantee that the new table will be used by the system.

To increase the likelihood of new tables or columns being used by the framework, a change in *src/conf/sqls.json* is needed. This change could either be updating existing queries or introducing new example queries.

Is it possible to extract detailed statistics from the database using natural language? As shown in Table 4.4, the framework provides good answers when asked to generate limited statistics. However, when the user asks questions demanding a long list of information, model laziness often comes into play, resulting in a wrong answer.

Because of this, we must conclude that it is possible to generate statistics, but questions like "Count the numbers of goals ...", will perform better than "List the goals ...", simply because it's less likely that model laziness will be an issue.

What role does different LLMs play in the quality of the system? Our framework's design significantly enhances the performance of the GPT-3.5-Turbo model. As shown in Table 4.4, both models show similar quality when all components of the framework are utilized. However, it is evident that GPT-4.0-Turbo achieves higher accuracy in pipelines 1-5, indicating that it's a stronger model. This suggests that GPT-4.0-Turbo may improve the quality of answers in scenarios where the framework doesn't find a matching SQL example.

Moreover, we are pleased to observe that the top FC model in Table 2.1 performs similarly to the 25th-ranked model when the full framework is employed. This finding highlights the importance of creating helper functions, which can greatly improve the capabilities of text-to-SQL systems.

Based on this, we must conclude that different LLMs do play a role, but the impact of using a lower quality model can be greatly reduced when using SoccerRAG.

Can we effectively query complex datasets using natural language? Based on the findings in this thesis, we can conclude that it is possible to query complex datasets using a mixture of data preparation, such as converting the dataset into a database, applying a pipeline designed for the problem, and providing an intuitive UI.

6.3 Limitations and Future Work

Question complexity: As questions become more complex, the framework struggles to provide SQL queries that answer the question. The primary reason for this limitation is the limited number of examples in the SQL RAG file (*src/conf/sqls.json*). After submitting B.1 *SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries*, no additional examples were added to the file. While theoretically, we could design SQL queries that enable the framework to achieve a perfect score on most questions, we chose to focus on improving the framework with a small set of queries rather than overfitting to our test questions.

In the future, the results presented in Table 4.3 should be addressed by adding new input-query pairs to the *sqls.json* file, improving the system's performance for real-world applications.

Video retrieval: Initially, our framework was intended to include custom video and media retrieval using natural language. However, given the current state of the text-to-SQL framework, and our limited time, we prioritized improving retrieval methods over adding a video retrieval module.

In the future, an additional module could be developed to utilize the *frame_stamp* or *game_time* parameters in the events table to return videos, such as game highlights. To ensure accurate input for this module, new examples should first be included in *sqls.json*. The system's output would then be processed through a new extraction pipeline to extract parameters like *game_time* and other relevant information.

Game summarization: When the dataset first got examined, a hypothesis was made that one could generate detailed match summaries using a mix of transcribed commentary, events, and other metadata. This has not been explored fully due to time constraints. Designing SQL queries aimed at fetching important information from the commentary table, and processing information in a meaningful way is something we expect will be done in our future works.

Model laziness: Throughout the process, we were limited by model laziness, where the LLM only returned a partial list when asked to retrieve "all" "entities" from a table. However, this was a problem that would randomly appear, and it's therefore problematic to design good solutions for this issue while relying on a third-party API.

LLM constraints: With the current framework, we are constrained to use OpenAI's

API endpoint. GPT-4.0-Turbo is the strongest LLM on the market, but both other commercial and open source models are catching up and can surpass GPT-4.0-Turbo in isolated tasks. As seen in 2.1, Gorilla-v2, could be one such model. Due to time constraints, we have not explored these models, but since our framework is built on top of LangChain, changing the model is a trivial task. This could also mitigate the laziness and randomness of the framework. More research should be done by finding suitable open source LLMs for the framework.

6.4 Contributions

Scientific contributions: We introduce the extraction-validation schema, making it possible to validate properties before the LLM query the database. The schema has been shown to provide a great improvement in accuracy for systems using natural language to query databases.

Our additional public dissemination and artifacts include:

- **Software:**
 - We present **SoccerRAG** as open source software ('SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries', 2024). This is an important contribution as it enables further research and improvements in the field.
 - Additional **code** is released as open-source ('SoccerRAG: Thesis Repo', 2024)
- **Demo:**
 - **Proof-of-concept framework deployment:** An interactive demo is deployed to provide easy access to the framework. This contribution makes it possible to evaluate the solution without a technical background. The demo is available at <https://simulamet-host-soccerrag.hf.space/>.
- **Datasets:**
 - **agumented_leauges:** A dataset containing augmented names and abbreviations for leagues ('SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries', 2024).

- **augmented_teams**: A dataset containing augmented names and abbreviations for teams ('SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries', 2024).

These datasets are valuable as they can be reused by other researchers working with soccer data in general.

- **Publications:**

- *SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries*
B.1
- *Demo: Soccer Information Retrieval via Natural Queries using SoccerRAG*
B.2

We chose to publish two papers to effectively communicate our research, which constitutes a scientific contribution to the field. By dividing our work into two separate papers, we ensured that the core concepts and methodology were presented clearly and concisely. The first paper provides a detailed examination of the underlying principles and ideas. In contrast, the second paper is a practical guide, demonstrating how to apply the framework through a command-line interface and an interactive UI.

Bibliography

- Bergroth, L., Hakonen, H., & Raita, T. (2000). A survey of longest common subsequence algorithms. *Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000*, 39–48. <https://doi.org/10.1109/SPIRE.2000.878178>
- Chainlit. (2023). Chainlit [Accessed: 2024-05-03]. *Chainlit*.
- Chen, W., Hu, H., Chen, X., Verga, P., & Cohen, W. W. (2022). Murag: Multimodal retrieval-augmented generator for open question answering over images and text. *arXiv preprint arXiv:2210.02928*.
- Denning, P., Comer, D., Gries, D., Mulder, M., Tucker, A., Turner, J., & Young, P. (1989). Computing as a discipline. *Computer*, 22, 63–70. <https://doi.org/10.1109/2.19833>
- Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms.
- Fui-Hoon Nah, F., Zheng, R., Cai, J., Siau, K., & Chen, L. (2023). Generative ai and chatgpt: Applications, challenges, and ai-human collaboration.
- Gao, J., Li, P., Chen, Z., & Zhang, J. (2020). A Survey on Deep Learning for Multimodal Data Fusion. *Neural Computation*, 32(5), 829–864. https://doi.org/10.1162/neco_a_01273
- Gautam, S. (2023). Bridging Multimedia Modalities: Enhanced Multimodal AI Understanding and Intelligent Agents. *ICMI '23: Proceedings of the 25th International Conference on Multimodal Interaction*, 695–699. <https://doi.org/10.1145/3577190.3614225>
- Gautam, S., Midoglu, C., Shafiee Sabet, S., Kshatri, D. B., & Halvorsen, P. (2022a). Assisting soccer game summarization via audio intensity analysis of game highlights. *Proceedings of 12th IOE Graduate Conference*, 12, 25–32. <https://doi.org/10.13140/RG.2.2.34457.70240/1>
- Gautam, S., Midoglu, C., Shafiee Sabet, S., Kshatri, D. B., & Halvorsen, P. (2022b, October 10). Soccer Game Summarization using Audio Commentary, Metadata, and Captions. In *NarSUM '22: Proceedings of the 1st Workshop on*

- User-centric Narrative Summarization of Long Videos* (pp. 13–22). Association for Computing Machinery. <https://doi.org/10.1145/3552463.3557019>
- Gautam, S., Sarkhoosh, M. H., Held, J., Midoglu, C., Cioppa, A., Giancola, S., Thambawita, V., Riegler, M. A., Halvorsen, P., & Shah, M. (2024). SoccerNet-Echoes: A Soccer Game Audio Commentary Dataset. *arXiv*. <https://doi.org/10.48550/arXiv.2405.07354>
- Getting started with langsmith [Accessed: 2024-04-18]. (2024).
- Giancola, S., Amine, M., Dghaily, T., & Ghanem, B. (2018). SoccerNet: A Scalable Dataset for Action Spotting in Soccer Videos. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (pp. 18–22). IEEE. <https://doi.org/10.1109/CVPRW.2018.00223>
- GPT-3.5-Turbo [[Online; accessed 12. Apr. 2024]]. (2024, April). <https://platform.openai.com/docs/models/gpt-3-5-turbo>
- GPT-4 [[Online; accessed 12. Apr. 2024]]. (2024, April). <https://openai.com/gpt-4>
- Hadi, M. U., tashi qasem al, q. a., Qureshi, R., Shah, A., muneer amgad, a., Irfan, M., Zafar, A., Shaikh, M. B., Akhtar, N., Wu, J., & Mirjalili, S. (2023). A survey on large language models: Applications, challenges, limitations, and practical usage. <https://doi.org/10.36227/techrxiv.23589741.v1>
- Halstead, M. H. (1977). *Elements of Software Science (Operating and programming systems series) | Guide books*. Elsevier Science Inc. <https://dl.acm.org/doi/abs/10.5555/540137>
- International Football Association Board. (2023). Laws of the game 2023/24 [Accessed: 2024-03-30]. <https://www.theifab.com/laws-of-the-game-documents/>
- IVG. (2024). How to deal with lazy gpt-4 [Accessed: 2024-03-20]. <https://community.openai.com/t/how-to-deal-with-lazy-gpt-4/689286/12>
- Jelinek, F. (1998). *Statistical methods for speech recognition*. MIT Press. <https://books.google.no/books?id=1C9dzcJTWoC>
- Jeong, C. (2023). A study on the implementation of generative ai services using an enterprise data-based llm application architecture. *arXiv preprint arXiv:2309.01105*.
- Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y., Madotto, A., & Fung, P. (2022). Survey of hallucination in natural language generation. *CoRR*, *abs/2202.03629*. <https://arxiv.org/abs/2202.03629>
- Langchain. (2024a). Docs [Accessed: 2024-03-28]. <https://python.langchain.com/docs/>
- Langchain. (2024b). Extraction [Accessed: 2024-02-16]. https://python.langchain.com/docs/use_cases/extraction

- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2021). Retrieval-augmented generation for knowledge-intensive nlp tasks.
- Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., et al. (2024). Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.
- Liu, H., Li, C., Wu, Q., & Lee, Y. J. (2024). Visual instruction tuning. *Advances in neural information processing systems*, 36.
- Mauran, C. (2023). Chatgpt's laziness fixed in openai's upgrade to gpt-4 turbo [Accessed: 2024-03-20]. *Mashable*. <https://mashable.com/article/chatgpt-laziness-openai-upgrade-gpt-4-turbo>
- OpenAI. (2023a). Chat api reference - functions. <https://platform.openai.com/docs/api-reference/chat/create%5C#chat/create-functions>
- OpenAI. (2023b). Gpt-4 and gpt-4 turbo documentation [Accessed: 2024-03-20]. <https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo>
- OpenAI. (2023c). Openai pricing [Accessed: 2024-03-20]. <https://openai.com/pricing>
- OpenAI. (2024). Gpt-4 and gpt-4 turbo documentation [Accessed: 2024-05-14]. <https://platform.openai.com/docs/models/gpt-4o>
- Ouyang, S., Zhang, J. M., Harman, M., & Wang, M. (2023). Llm is like a box of chocolates: The non-determinism of chatgpt in code generation.
- Patil, S. G., Zhang, T., Wang, X., & Gonzalez, J. E. (2023). Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.
- Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Sutskever, I. (2022). Robust speech recognition via large-scale weak supervision.
- Ratcliff, J. W., & Metzener, D. E. (1988). Pattern matching: The gestalt approach. *Dr. Dobb's*. <https://www.drdoobs.com/database/pattern-matching-the-gestalt-approach/184407970?pgno=5>
- Schulz, K. U., & Mihov, S. (2002). Fast string correction with levenshtein automata. *International Journal on Document Analysis and Recognition*, 5(1), 67–85. <https://doi.org/10.1007/s10032-002-0082-8>
- SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries [[Online; accessed 12. Apr. 2024]]. (2024, April). <https://github.com/simula/soccer-rag>
- SoccerRAG: Thesis Repo [[Online; accessed 14. May 2024]]. (2024, May). <https://github.com/simula/forzify>
- Song, X., Salcianu, A., Song, Y., Dopson, D., & Zhou, D. (2021). Fast wordpiece tokenization.
- Srinivasan, V. K., Dong, Z., Zhu, B., Yu, B., Mosk-Aoyama, D., Keutzer, K., Jiao, J., & Zhang, J. (2023). Nexusraven: A commercially-permissive language model

- for function calling. *NeurIPS 2023 Foundation Models for Decision Making Workshop*.
- Strand, T. A., Balla, F., Kumar, N., Lotfigolian, M., & Willoch, S. H. (2024). How we created rutergpt — ai-lab at ruter — part 3. *Medium: Ruter - Produktutvikling og teknologi*. <https://medium.com/ruter-produktutvikling-og-teknologi/how-we-created-rutergpt-ai-lab-at-ruter-part-3-8e2961f9449e>
- Topsakal, O., & Akinci, T. C. (2023). Creating large language model applications utilizing langchain: A primer on developing llm apps fast. *Proceedings of the International Conference on Applied Engineering and Natural Sciences, Konya, Turkey*, 10–12.
- Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., . . . Scialom, T. (2023). Llama 2: Open foundation and fine-tuned chat models.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). Attention is all you need.
- Weizenbaum, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1), 36–45. <https://doi.org/10.1145/365153.365168>
- Wu, J., Gan, W., Chen, Z., Wan, S., & Yu, P. S. (2023). Multimodal large language models: A survey.
- Yan, F., Mao, H., Ji, C. C.-J., Zhang, T., Patil, S. G., Stoica, I., & Gonzalez, J. E. (2024). Berkeley function calling leaderboard.
- Yu, W., Zhu, C., Li, Z., Hu, Z., Wang, Q., Ji, H., et al. (2022). A Survey of Knowledge-enhanced Text Generation. *ACM Comput. Surv.*, 54(11s), 1–38. <https://doi.org/10.1145/3512467>
- Zhang, H., Li, X., & Bing, L. (2023). Video-llama: An instruction-tuned audio-visual language model for video understanding. *arXiv preprint arXiv:2306.02858*.
- Zhang, Y., Wu, M., Tian, G. Y., Zhang, G., & Lu, J. (2021). Ethics and privacy of artificial intelligence: Understandings from bibliometrics. *Knowledge-Based Systems*, 222, 106994. <https://doi.org/10.1016/j.knosys.2021.106994>
- Ziegler, D. M., Stiennon, N., Wu, J., Brown, T. B., Radford, A., Amodei, D., Christiano, P., & Irving, G. (2020). Fine-tuning language models from human preferences.

Appendix A

Results

A.1 Ablation Study Results

The ablation study results, presented in Chapter 4.2.3, contain the data used. The accepted parameter is set subjectively.

Table A.1: Question 1: Is Manchester United in the database? - GPT 3.5-Turbo

Answer	Time	Accepted
I cannot confirm if Manchester United is in the database...	3,8	0
Yes, Manchester United is in the database.	7,8	1
...However, the team "Manchester United" is not listed in the database....	5,3	0
...Therefore, Manchester United is not in the database.	4,4	0
Yes, Manchester United is in the database.	5,9	1
Yes, Manchester United is in the database.	7,3	1

Table A.2: Question 1: Is Manchester United in the database? - GPT 4-Turbo

Answer	Time	Accepted
Yes, Manchester United is in the database..	14,8	1
Yes, Manchester United is in the database.	11,1	1
Yes, Manchester United is in the database.	20,3	1
Yes, Manchester United is in the database.	10,4	1
Yes, Manchester United is in the database.	14,4	1
Yes, Manchester United is in the database with a primary key of 7.	9,9	1

Table A.3: Question 2: Give me the total home goals for Bayern M in the 2014-15 season - GPT 3.5-Turbo

Answer	Time	Accepted
The total home goals for Bayern Munich in the 2014-15 season is 1.	5,6	0
The total home goals scored by Bayern M in the 2014-15 season is 1.	8,1	0
The total home goals for Bayern M in the 2014-15 season is 0.	10,4	0
The total home goals for Bayern Munich in the 2014-2015 season were 25.	5,5	1
The total home goals scored by Bayern M in the 2014-15 season is 0.	17,0	0
The total home goals scored by Bayern Munich in the 2014-2015 season is 25.	16,2	1

Table A.4: Question 2: Give me the total home goals for Bayern M in the 2014-15 season - GPT 4-Turbo

Answer	Time	Accepted
The total home goals for Bayern M in the 2014-15 season is not available in the database.	11,1	0
The total home goals for Bayern Munich in the 2014-15 season is not available in the database.	24,1	0
The total home goals for Bayern M in the 2014-15 season is not available in the database.	28,1	0
The total home goals for Bayern Munich in the 2014-2015 season was 25.	15,5	1
The total home goals for Bayern M in the 2014-15 season is not available in the database.	13,8	0
The total home goals for Bayern Munich in the 2014-2015 season was 25.	18,2	1

Table A.5: Question 3: Calculate home advantage for Real Madrid in the 2015-16 season - GPT 3.5-Turbo

Answer	Time	Accepted
The query results show the games where Real Madrid played at home in the 2015-2016 season. Here are some of the games:	11,4	0
It seems that there might be no data available for Real Madrid in the specified season.	17,3	0
I don't have the specific information needed to calculate the home advantage for Real Madrid in the 2015-16 season.	7,3	0
Here are some of the games played by Real Madrid at home during that season:	9,6	0
The database contains tables for "games" and "teams" with the following columns:	11,3	0
In the 2015-2016 season, Real Madrid played a total of 18 home games. Out of these games:	14,7	1

Table A.6: Question 3: Calculate home advantage for Real Madrid in the 2015-16 season - GPT 4-Turbo

Answer	Time	Accepted
This indicates a home advantage, as their win rate was higher in home games compared to away games.	22,6	1
This data suggests that Real Madrid had a stronger performance at home compared to away games during the 2015-16 season, indicating a home advantage.	82,8	0
I'm unable to calculate the home advantage for Real Madrid for that specific season.	134,3	0
For Real Madrid in the 2015-2016 season, the home advantage statistics are as follows:	46,2	1
It appears there was an issue with the data retrieval, resulting in null	76,9	0
... 2015-2016 season, with a slightly better win rate in home games	87,3	1

Table A.7: Question 4 - How many goals did Messi score in the 15-16 season? - GPT 3.5-Turbo

Answer	Time (s)	Accepted
Messi scored 2 goals in the 15-16 season.	6.21	0
Messi scored 0 goals in the 15-16 season.	7.77	0
I have queried the database to find the number of goals Messi scored in the 15-16 season, but the query did not return any results. It seems that there may be a discrepancy in the data or the way the events are recorded in the database.	7.39	0
Lionel Messi scored goals in the 2015-2016 season. The number of goals he scored can be retrieved by querying the specific goal information for Lionel Messi in that season.	7.72	0
Messi did not score any goals in the 15-16 season based on the data in the database.	15.58	0
Lionel Messi scored a total of 3 goals in the 2015-2016 season.	10.93	1

Table A.8: Question 4 - How many goals did Messi score in the 15-16 season? - GPT 4-Turbo

Answer	Time	Accepted
Messi scored 0 goals in the 15-16 season according to the database. s.	28.76	0
Messi scored 0 goals in the 15-16 season according to the database.	25.40	0
Messi scored 0 goals in the 15-16 season according to the database. This result might not reflect actual historical data, as it depends on the completeness and accuracy of the database's records.	28.34	0
Lionel Messi scored 38 goals in the 2015-2016 season.	24.44	0
Messi scored 0 goals in the 15-16 season according to the database.	58.41	0
Lionel Messi scored 3 goals in the 2015-2016 season.	26.49	1

Table A.9: Question 5 - How many yellow-cards did Enzo Perez get in the 15-2016 season? - GPT 3.5-Turbo

Answer	Time	Accepted
Enzo Perez received 2 yellow cards in the 2015-2016 season.	6.72	0
Enzo Perez received a total of 3 yellow cards in the 2015-2016 season.	9.377	1
Enzo Perez received 2 yellow cards in the 2015-2016 season.	5.99	0
I don't have the specific column name for yellow cards in the player_events table..	11.35	0
Enzo Perez received 0 yellow cards in the 2015-2016 season.	20.11	0
Enzo Perez received a total of 3 yellow cards in the 2015-2016 season.	13.41	1

Table A.10: Question 5 - How many yellow-cards did Enzo Perez get in the 15-2016 season? - GPT 4-Turbo

Answer	Time	Accepted
Enzo Perez received 0 yellow cards in the 15-2016 season.	29.03	0
Enzo Perez received 0 yellow cards in the 15-2016 season.	22.53	0
Enzo Perez received 0 yellow cards in the 15-2016 season.	24.60	0
Enzo Perez received 3 yellow cards in the 2015-2016 season.	37.05	1
Enzo Perez received 0 yellow cards in the 15-2016 season.	30.08	0
Enzo Perez received 3 yellow cards in the 2015-2016 season.	27.68	1

Table A.11: Question 6 - List all teams that played a game against Napoli in 2016-17 season in seriea? Do not limit the number of results - GPT 3.5-Turbo

Answer	Time	Accepted
I don't know	5.17	0
The query has been executed successfully.	8.9	0
I have queried the database to find all teams that played a game against..	7.7	0
The teams that played a game against Napoli in the 2016-2017 season..	10.54	1
The query has been executed, and here are the results:	12.1	0
The teams that played a game against Napoli in the 2016-2017 season in..	20.5	1

Table A.12: Question 6 - List all teams that played a game against Napoli in 2016-17 season in seriea? Do not limit the number of results - GPT 4-Turbo

Answer	Time	Accepted
The query did not return any results,	33.73	0
This could be due to a lack of specific data for that season or team in the database.	58.69	0
I couldn't find any teams that played a game against Napoli in the 2016-17	27.38	0
The teams that played a game against Napoli in the 2016-2017 season in the Italy Serie A	39.28	1
There are no teams listed that played against Napoli in the 2016-17	36.28	0
The teams that played a game against Napoli in the 2016-2017 season in the Italy Serie A are	40.93	1

Table A.13: Question 7 - Give all the teams in the league ucl in the 2015-2016 season? - GPT 3.5-Turbo

Answer	Time	Accepted
I have queried the database, but it seems there was no data returned for the given question	6.21	0
The database contains tables for "games", "leagues", and "teams".	16.18	0
The teams in the UEFA Champions League (UCL)...	8.184	0
The team in the UEFA Champions League (UCL) in the 2015-2016 season is Burnley.	7.92	0
The teams in the league "ucl" (UEFA Champions League) in the season "2015-2016" are	12.21	0
The teams in the "europe_uefa-champions-league"	20.26	1

Table A.14: Question 7 - Give all the teams in the league ucl in the 2015-2016 season? - GPT 4-Turbo

Answer	Time	Accepted
The teams in the UEFA Champions League for the 2015-2016 season..	36.14	1
The teams in the UEFA Champions League for the 2015-2016 season, limited to 5 examples	35.54	0
The teams in the UEFA Champions League for the 2015-2016 season were:	44.46	1
The teams in the UEFA Champions League for the 2015-2016 season were:	26.19	1
The teams in the UEFA Champions League for the 2015-2016 season were: Limit 10	38.86	0
This is a limited list of the first 5 teams alphabetically. (Lazy)	31.21	0

Table A.15: Question 8 - Give me all games in epl with yellow cards in the first half in the 2015-2016 season - GPT 3.5-Turbo

Answer	Time	Accepted
List 10 games	10.3	0
I have re-executed the query, but it seems there are no results matching the criteria..	15.28	0
List 3 games	9.79	0
List 10 games	14.38	0
List 3 games	11,67	0
List 3 games	10.89	0

Table A.16: Question 8 - Give me all games in epl with yellow cards in the first half in the 2015-2016 season - GPT 4-Turbo

Answer	Time	Accepted
List 10 games	50.69	0
There are no games in the English Premier League (EPL) during the 2015-2016 season with yellow cards..	42.4	0
List 5 games	35.93	0
List 10 games	48.74	0
,There are no games in the English Premier League (EPL) during the 2015-2016 season with yellow cards..	41.34	0
List 5 games	81.91	0

Table A.17: Question 9 - What teams and leagues has Adnan Januzaj play in? - GPT 3.5-Turbo

Answer	Time	Accepted
Adnan Januzaj has played for the team ""Manchester United"" in the database	7.51	0
Dortmund, Manchester United	17.31	0
I'm unable to retrieve the information ... due to a missing column	10.94	0
I dont't know	5.92	0
List all three teams and leagues	24.38	1
List all three teams and leagues	10.27	1

Table A.18: Question 9 - What teams and leagues has Adnan Januzaj play in? - GPT 4-Turbo

Answer	Time	Accepted
... it seems there was an issue retrieving the data...	42.7	0
List 3 teams and leagues	36.22	1
Only Manchester	40.60	0
... returned no values ...	30.03	0
List all three teams and leagues	35.99	1
List all three teams and leagues	27.26	1

Table A.19: Question 10 - List ALL players that started a game for Las Palmas in the 2016-2017 season? Do NOT limit the number of results - GPT 3.5-Turbo

Answer	Time	Accepted
I don't know	4.95	0
Las Palmas is not present ...	13.87	0
List 10 players	9.67	0
... no results returned for Las Palmas...	8.01	0
List 5 players	9.83	0
List 19 players	26.19	1

Table A.20: Question 10 - List ALL players that started a game for Las Palmas in the 2016-2017 season? Do NOT limit the number of results - GPT 4-Turbo

Answer	Time	Accepted
.. it seems there was an issue retrieving the data..	42.7	0
List 19 players	36.22	1
List 18 players	40.60	0
List 18 players	30.03	0
List 19 players	35.99	1
List 19 players	27.26	1

A.2 Extractor - Validator Results

The extractor-validator results show the performance of the extractor and validator module presented in Chapter 4.2.1. The scoring parameter in the results is either "Perfect" or if any issues happened, comment on what went wrong.

Question 1: Is Manchester United in the database?

- Extracted values:
 - Team Name: Manchester United
- Cleaned prompt Is Manchester United in the database? **Updated Information:**
 - Manchester United has a primary key: 7.
- Score - Perfect

Question 2: Give me the total home goals for Bayern M in the 2014-15 season

- Extracted values:
 - Team Name: Bayern M
 - Year Season: 2014-15
 - In Game Event: home goals
- Cleaned prompt Give me the total home goals for Bayern M in the 2014-15 season **Updated Information:** - Bayern M (now referred to as Bayern Munich) has a primary key: 33. - 2014-15 (now referred to as 2014-2015). - home goals (now referred to as Goal).
- Score - Should not have extracted home goals

Question 3: Calculate home advantage for Real Madrid in the 2015-16 season

- Extracted values:
 - Team Name: Real Madrid
 - Year Season: 2015-16

- Cleaned prompt Calculate home advantage for Real Madrid in the 2015-16 season **Updated Information:** - Real Madrid has a primary key: 30. - 2015-16 (now referred to as 2015-2016).
- Score - Perfect

Question 4: How many goals did Messi score in the 15-16 season?

- Extracted values:
 - Person Name: Messi
 - Year Season: 15-16
 - In Game Event: goals
- Cleaned prompt How many goals did Messi score in the 15-16 season? **Updated Information:** - Messi (now referred to as Lionel Messi) has a primary key: vgOOdZbd. - 15-16 (now referred to as 2015-2016). - goals (now referred to as Goal).
- Score - Perfect

Question 5: How many yellow-cards did Enzo Perez get in the 15-2016 season?

- Extracted values:
 - Person Name: Enzo Perez
 - In Game Event: yellow-cards
 - Year Season: 15-2016
- Cleaned prompt How many yellow-cards did Enzo Perez get in the 15-2016 season? **Updated Information:** - Enzo Perez has a primary key: 2Prv8OYO. - yellow-cards (now referred to as Yellow card). - 15-2016 (now referred to as 2015-2016).
- Score - Perfect

Question 6: List all teams that played a game against Napoli in 2016-17 season in seriea? Do not limit the number of results

- Extracted values:
 - Team Name: Napoli
 - Year Season: 2016-17
 - League: seriea
- Cleaned prompt List all teams that played a game against Napoli in 2016-17 season in seriea? Do not limit the number of results **Updated Information:**
 - Napoli has a primary key: 61. - 2016-17 (now referred to as 2016-2017). - seriea (now referred to as italy_serie-a) has a primary key: 5.
- Score - Perfect

Question 7: Give all the teams in the league ucl in the 2015-2016 season?

- Extracted values:
 - League: ucl
 - Year Season: 2015-2016
- Cleaned prompt Give all the teams in the league ucl in the 2015-2016 season? **Updated Information:**
 - ucl (now referred to as europe_uefa-champions-league) has a primary key: 2.
- Score - Perfect

Question 8: Give me all games in epl with yellow cards in the first half in the 2015-2016 season. Its important that you list all of them!

- Extracted values:
 - In Game Event: yellow cards
 - Year Season: 2015-2016
 - League: epl
- Cleaned prompt Give me all games in epl with yellow cards in the first half in the 2015-2016 season. Its important that you list all of them! **Updated Information:**
 - yellow cards (now referred to as Yellow card). - epl (now referred to as england_epl) has a primary key: 1.

- Score - Perfect

Question 9: What teams and leagues has Adnan Januzaj play in?

- Extracted values:
 - Person Name: Adnan Januzaj
- Cleaned prompt What teams and leagues has Adnan Januzaj play in? **Updated Information:** - Adnan Januzaj has a primary key: 6qyeYCYO.
- Score - Perfect

Question 10: List ALL players that started a game for Las Palmas in the 2016-2017 season? Do NOT limit the number of results

- Extracted values:
 - Team Name: Las Palmas
 - Year Season: 2016-2017
- Cleaned prompt List ALL players that started a game for Las Palmas in the 2016-2017 season? Do NOT limit the number of results **Updated Information:** - Las Palmas has a primary key: 115.
- Score - Perfect

Question 11: Did Ajax or Manchester United win the most games in the 2014-15 season?

- Extracted values:
 - Team Name: Ajax, Manchester United
 - Year Season: 2014-15
- Cleaned prompt Did Ajax or Manchester United win the most games in the 2014-15 season? **Updated Information:** - Ajax has a primary key: 31. - Manchester United (now referred to as Manchester United) has a primary key: 7. - 2014-15 (now referred to as 2014-2015).

- Score - Perfect

Question 12: How many yellow and red cards were given in UCL in the 2015-2016 season?

- Extracted values:
 - In Game Event: yellow cards, red cards
 - League: UCL
 - Year Season: 2015-2016
- Cleaned prompt How many yellow and red cards were given in UCL in the 2015-2016 season? **Updated Information:** - yellow cards (now referred to as Yellow card). - red cards (now referred to as Red card). - UCL (now referred to as europe_uefa-champions-league) has a primary key: 2.
- Score - Perfect

Question 13: Is Messi and C Ronaldo in the database?

- Extracted values:
 - Person Name: Messi, C Ronaldo
- Cleaned prompt Is Messi and C Ronaldo in the database? **Updated Information:** - Messi (now referred to as Lionel Messi) has a primary key: vgOOdZbd. - C Ronaldo (now referred to as Cristiano Ronaldo) has a primary key: WGOY4FSt.
- Score - Perfect

Question 14: How many goals did E Hazard score in the game between Bournemouth and Chelsea in the 2015-2016 season?

- Extracted values:
 - Person Name: E Hazard
 - Team Name: Bournemouth, Chelsea

- Year Season: 2015-2016
- In Game Event: goals
- Cleaned prompt How many goals did E Hazard score in the game between Bournemouth and Chelsea in the 2015-2016 season? **Updated Information:**
 - E Hazard (now referred to as Thorgan Hazard) has a primary key: KOq0yhLo.
 - Bournemouth has a primary key: 21. - Chelsea has a primary key: 1. - goals (now referred to as Goal).
- Score - Perfect

Question 15: How many yellow cards were given in the game between Bayer Munich and S Donetsk in, and did anyone get a red card the 2014-15 ucl?

GPT-3.5-Turbo

- Extracted values:
 - Team Name: Bayer Munich, S Donetsk
 - In Game Event: yellow cards
 - League: ucl
 - Year Season: 2014-15
- Cleaned prompt How many yellow cards were given in the game between Bayer Munich and S Donetsk in, and did anyone get a red card the 2014-15 ucl? **Updated Information:** - Bayer Munich (now referred to as Bayern Munich) has a primary key: 33. - S Donetsk (now referred to as Shakhtar Donetsk) has a primary key: 40. - yellow cards (now referred to as Yellow card). - ucl (now referred to as europe_uefa-champions-league) has a primary key: 2. - 2014-15 (now referred to as 2014-2015).
- Score - Partial, missing the red card from extracted items. Corrects the grammatical error "where" to "were" in the cleaned prompt.

GPT-4-Turbo

- Extracted values:
 - Team Name: Bayer Munich, S Donetsk

- Year Season: 2014-15
 - League: ucl
 - In Game Event: yellow cards, red card
- Cleaned prompt How many yellow cards were given in the game between Bayer Munich and S Donetsk in, and did anyone get a red card the 2014-15 ucl?
Updated Information: - Bayer Munich (now referred to as Bayern Munich) has a primary key: 33. - S Donetsk (now referred to as Shakhtar Donetsk) has a primary key: 40. - 2014-15 (now referred to as 2014-2015). - ucl (now referred to as europe_uefa-champions-league) has a primary key: 2. - yellow cards (now referred to as Yellow card). - red card (now referred to as Red card).
 - Score - Perfect

Question 16: Make a list of when corners happened in the EPL 2015-2016 season. Aggregate by a period of 15 minutes.

- Extracted values:
 - Year Season: 2015-2016
 - In Game Event: corners
 - League: EPL
- Cleaned prompt Make a list of when corners happened in the EPL 2015-2016 season. Aggregate by a period of 15 minutes. **Updated Information:** - corners (now referred to as Corner). - EPL (now referred to as england_epl) has a primary key: 1.
- Score - Perfect

Question 17: What league is ManU, Arsenal, Bournemouth, Real Madrid, Chelsea and Liverpool in?

- Extracted values:
 - Team Name: ManU, Arsenal, Bournemouth, Real Madrid, Chelsea, Liverpool

- Cleaned prompt What league is ManU, Arsenal, Bournemouth, Real Madrid, Chelsea and Liverpool in? **Updated Information:** - Manchester United has a primary key: 7. - Arsenal has a primary key: 5. - Bournemouth has a primary key: 21. - Real Madrid has a primary key: 30. - Chelsea has a primary key: 1. - Liverpool has a primary key: 9.
- Score - Perfect

Question 18: How many players has Aleksandar as first name in the database, and how many goals have they scored in total?

GPT-3.5-Turbo & GPT-4-Turbo

- Extracted values:
 - Person Name: Aleksandar
- Cleaned prompt How many players has Aleksandar as first name in the database, and how many goals have they scored in total?
- Score - It extracts the name Aleksandar correctly, but the question is asking about multiple Aleksandar's, thus extracting one player will confuse the agent.

Question 19: What did the commentary say about the game between Arsenal and Southampton in the 2016-17 season?

- Extracted values:
 - Team Name: Arsenal, Southampton
 - Year Season: 2016-17
- Cleaned prompt What did the commentary say about the game between Arsenal and Southampton in the 2016-17 season? **Updated Information:** - Arsenal (now referred to as Arsenal) has a primary key: 5. - Southampton (now referred to as Southampton) has a primary key: 8. - 2016-17 (now referred to as 2016-2017).
- Score - Perfect

Question 20: Have Mesut Ozil, Pablo Insua or Alex Pike played for West Ham or Barcelona?

- Extracted values:
 - Person Name: Mesut Ozil, Pablo Insua, Alex Pike
 - Team Name: West Ham, Barcelona
- Cleaned prompt Have Mesut Ozil, Pablo Insua or Alex Pike played for West Ham or Barcelona? **Updated Information:** - Mesut Ozil has a primary key: U30xq9EJ. - Pablo Insua has a primary key: 2PZuGsl. - Alex Pike has a primary key: MD4vSR3r. - West Ham has a primary key: 12. - Barcelona has a primary key: 32.
- Score - Perfect

A.3 SQL Complexity Calculations

To be able to gain insight on how difficult a question is for the framework to answer, we calculated the Halstead Metrics (Halstead, 1977). These metrics In order to calculate this metric, we designed for each question a query that would solve the task given. How these metrics were calculated is explained in Chapter 4.1.

Question 1: Is Manchester United in the database?

Correct query:

```
SELECT t.*
FROM teams AS t
WHERE t.name = 'Manchester United'
```

Number of distinct operators, $n_1 = 5$

Number of distinct operands, $n_2 = 5$

Total number of operators, $N_1 = 5$

Total number of operands, $N_2 = 5$

Vocabulary, $n = n_1 + n_2 = 10$

Length, $N = N_1 + N_2 = 10$ (A.1)

Volume, $V = N \times \log_2(n) = 33.22$

Difficulty, $D = \frac{n_1}{2} \times \frac{N_2}{n_2} = 2.50$

Effort, $E = D \times V = 83.05$

Time to understand (seconds), $T = \frac{E}{18} = 4.61$

Question 2: Give me the total home goals for Bayern M in the 2014-15 season

Correct query:

```
SELECT
    SUM(goal_home) AS TotalHomeGoals
FROM
    games
```

WHERE

```
home_team_id = (SELECT id FROM teams WHERE name = 'Bayern Munich')
AND season = '2014-2015';
```

Number of distinct operators, $n_1 = 6$

Number of distinct operands, $n_2 = 14$

Total number of operators, $N_1 = 11$

Total number of operands, $N_2 = 16$

Vocabulary, $n = n_1 + n_2 = 20$

Length, $N = N_1 + N_2 = 27$ (A.2)

Volume, $V = N \times \log_2(n) = 116.69$

Difficulty, $D = \frac{n_1}{2} \times \frac{N_2}{n_2} = 3.43$

Effort, $E = D \times V = 400.09$

Time to understand (seconds), $T = \frac{E}{18} = 22.23$

Question 3: Calculate home advantage for Real Madrid in the 2015-16

Correct query:

SELECT

```
'Home' AS Venue,
SUM(CASE WHEN goal_home > goal_away THEN 1 ELSE 0 END) AS Wins,
SUM(CASE WHEN goal_home = goal_away THEN 1 ELSE 0 END) AS Draws,
SUM(CASE WHEN goal_home < goal_away THEN 1 ELSE 0 END) AS Losses
```

FROM

```
games
```

WHERE

```
home_team_id = (SELECT id FROM teams WHERE name = 'Real Madrid')
AND season = '2015-2016'
```

UNION ALL

SELECT

```
'Away',
SUM(CASE WHEN goal_away > goal_home THEN 1 ELSE 0 END) AS Wins,
SUM(CASE WHEN goal_away = goal_home THEN 1 ELSE 0 END) AS Draws,
```

```

SUM(CASE WHEN goal_away < goal_home THEN 1 ELSE 0 END) AS Losses
FROM
  games
WHERE
  away_team_id = (SELECT id FROM teams WHERE name = 'Real Madrid')
  AND season = '2015-2016';

```

Number of distinct operators, $n_1 = 8$

Number of distinct operands, $n_2 = 31$

Total number of operators, $N_1 = 33$

Total number of operands, $N_2 = 110$

Vocabulary, $n = n_1 + n_2 = 39$

Length, $N = N_1 + N_2 = 143$ (A.3)

Volume, $V = N \times \log_2(n) = 755.81$

Difficulty, $D = \frac{n_1}{2} \times \frac{N_2}{n_2} = 14.19$

Effort, $E = D \times V = 10727.66$

Time to understand (seconds), $T = \frac{E}{18} = 595.98$

Question 4: How many goals did Messi score in the 15-16 season?

Correct query:

```

SELECT
  COUNT(*) AS goals_count
FROM
  player_events pe
JOIN
  players p ON pe.player_id = p.hash
JOIN
  games g ON pe.game_id = g.id
JOIN
  player_event_labels pel ON pe.type = pel.id
WHERE

```

```

p.name = 'Lionel Messi'
AND g.season = '2015-2016'
AND pel.label = 'Goal';

```

Number of distinct operators, $n_1 = 8$

Number of distinct operands, $n_2 = 26$

Total number of operators, $N_1 = 18$

Total number of operands, $N_2 = 26$

Vocabulary, $n = n_1 + n_2 = 34$

Length, $N = N_1 + N_2 = 44$ (A.4)

Volume, $V = N \times \log_2(n) = 223.85$

Difficulty, $D = \frac{n_1}{2} \times \frac{N_2}{n_2} = 4.00$

Effort, $E = D \times V = 895.39$

Time to understand (seconds), $T = \frac{E}{18} = 49.74$

Question 5: How many yellow-cards did Enzo Perez get in the 15-2016 season?

Correct query:

```

SELECT
    COUNT(*) AS yellow_cards_count
FROM
    player_events pe
JOIN
    players p ON pe.player_id = p.hash
JOIN
    games g ON pe.game_id = g.id
JOIN
    player_event_labels pel ON pe.type = pel.id
WHERE
    p.name = 'Enzo Perez'
    AND g.season = '2015-2016'
    AND pel.label = 'Yellow card';

```

Number of distinct operators, $n_1 = 8$

Number of distinct operands, $n_2 = 26$

Total number of operators, $N_1 = 18$

Total number of operands, $N_2 = 26$

Vocabulary, $n = n_1 + n_2 = 34$

Length, $N = N_1 + N_2 = 44$ (A.5)

Volume, $V = N \times \log_2(n) = 223.85$

Difficulty, $D = \frac{n_1}{2} \times \frac{N_2}{n_2} = 4.00$

Effort, $E = D \times V = 895.39$

Time to understand (seconds), $T = \frac{E}{18} = 49.74$

Question 6: List all teams that played a game against Napoli in 2016-17 season in seriea? Do not limit the number of results

Correct query:

```
SELECT DISTINCT
  CASE
    WHEN t1.name = 'Napoli' THEN t2.name
    ELSE t1.name
  END AS opponent_team_name
FROM
  games g
JOIN
  leagues l ON g.league_id = l.id
JOIN
  teams t1 ON g.home_team_id = t1.id
JOIN
  teams t2 ON g.away_team_id = t2.id
WHERE
  l.name = 'italy_serie-a'
  AND g.season = '2016-2017'
```



```

AND (t1.name = 'Napoli' OR t2.name = 'Napoli')
ORDER BY
    opponent_team_name;

```

Number of distinct operators, $n_1 = 10$

Number of distinct operands, $n_2 = 31$

Total number of operators, $N_1 = 22$

Total number of operands, $N_2 = 38$

Vocabulary, $n = n_1 + n_2 = 41$

Length, $N = N_1 + N_2 = 60$ (A.6)

Volume, $V = N \times \log_2(n) = 321.45$

Difficulty, $D = \frac{n_1}{2} \times \frac{N_2}{n_2} = 6.13$

Effort, $E = D \times V = 1970.20$

Time to understand (seconds), $T = \frac{E}{18} = 109.46$

Question 7: Give all the teams in the league ucl in the 2015-2016 season?

Correct query:

```

SELECT DISTINCT
    t.name AS team_name
FROM
    games g
JOIN
    leagues l ON g.league_id = l.id
JOIN
    teams t ON g.home_team_id = t.id OR g.away_team_id = t.id
WHERE
    l.name = 'europe_uefa-champions-league'
    AND g.season = '2015-2016'
ORDER BY
    t.name;

```

$$\begin{aligned}
&\text{Number of distinct operators, } n_1 = 10 \\
&\text{Number of distinct operands, } n_2 = 20 \\
&\text{Total number of operators, } N_1 = 16 \\
&\text{Total number of operands, } N_2 = 22 \\
&\text{Vocabulary, } n = n_1 + n_2 = 30 \\
&\text{Length, } N = N_1 + N_2 = 38 \qquad \qquad \qquad (\text{A.7}) \\
&\text{Volume, } V = N \times \log_2(n) = 186.46 \\
&\text{Difficulty, } D = \frac{n_1}{2} \times \frac{N_2}{n_2} = 5.50 \\
&\text{Effort, } E = D \times V = 1025.54 \\
&\text{Time to understand (seconds), } T = \frac{E}{18} = 56.97
\end{aligned}$$

Question 8: Give me all games in epl with yellow cards in the first half in the 2015-2016 season

Correct query:

```

SELECT DISTINCT
  g.id AS game_id,
  g.date AS game_date,
  t1.name AS home_team,
  t2.name AS away_team
FROM
  games g
JOIN
  leagues l ON g.league_id = l.id
JOIN
  teams t1 ON g.home_team_id = t1.id
JOIN
  teams t2 ON g.away_team_id = t2.id
JOIN
  events e ON g.id = e.game_id
WHERE
  l.name = 'england_epl'
  AND g.season = '2015-2016'

```

```

AND e.label = 'Yellow card'
AND e.period = 1
ORDER BY
  g.date;

```

Number of distinct operators, $n_1 = 9$

Number of distinct operands, $n_2 = 36$

Total number of operators, $N_1 = 27$

Total number of operands, $N_2 = 41$

Vocabulary, $n = n_1 + n_2 = 45$

Length, $N = N_1 + N_2 = 68$ (A.8)

Volume, $V = N \times \log_2(n) = 373.45$

Difficulty, $D = \frac{n_1}{2} \times \frac{N_2}{n_2} = 5.12$

Effort, $E = D \times V = 1913.91$

Time to understand (seconds), $T = \frac{E}{18} = 106.33$

Question 9: What teams and leagues has Adnan Januzaj play in?

Correct query:

```

SELECT DISTINCT
  t.name AS team_name,
  l.name AS league_name
FROM
  game_lineup gl
JOIN
  players p ON gl.player_id = p.hash
JOIN
  teams t ON gl.team_id = t.id
JOIN
  games g ON gl.game_id = g.id
JOIN
  leagues l ON g.league_id = l.id
WHERE

```

```

    p.name = 'Adnan Januzaj'
ORDER BY
    t.name, l.name;

```

Number of distinct operators, $n_1 = 8$

Number of distinct operands, $n_2 = 28$

Total number of operators, $N_1 = 19$

Total number of operands, $N_2 = 31$

Vocabulary, $n = n_1 + n_2 = 36$

Length, $N = N_1 + N_2 = 50$ (A.9)

Volume, $V = N \times \log_2(n) = 258.50$

Difficulty, $D = \frac{n_1}{2} \times \frac{N_2}{n_2} = 4.43$

Effort, $E = D \times V = 1144.77$

Time to understand (seconds), $T = \frac{E}{18} = 63.60$

Question 10: List ALL players that started a game for Las Palmas in the 2016-2017 season? Do NOT limit the number of results

Correct query:

```

SELECT DISTINCT
    p.name AS player_name
FROM
    game_lineup gl
    JOIN players p ON gl.player_id = p.hash
    JOIN teams t ON gl.team_id = t.id
    JOIN games g ON gl.game_id = g.id
WHERE
    t.name = 'Las Palmas'
    AND g.season = '2016-2017'
    AND gl.starting = 1
ORDER BY
    p.name;

```

Number of distinct operators, $n_1 = 9$

Number of distinct operands, $n_2 = 25$

Total number of operators, $N_1 = 19$

Total number of operands, $N_2 = 26$

Vocabulary, $n = n_1 + n_2 = 34$

Length, $N = N_1 + N_2 = 45$ (A.10)

Volume, $V = N \times \log_2(n) = 228.94$

Difficulty, $D = \frac{n_1}{2} \times \frac{N_2}{n_2} = 4.68$

Effort, $E = D \times V = 1071.42$

Time to understand (seconds), $T = \frac{E}{18} = 59.52$

A.4 Execution Time

The execution time for questions 1-10 combined is shown in Chapter 4.3.1 and in Figure 4.2. The following figures show the execution time for questions 1 - 10 individually. The experiment was conducted by prompting the OpenAI API five times for each question and calculating the mean response time. This was done for both GPT-3.5-Turbo and GPT-4.0-Turbo, and during two different times, called peak and off-peak.

After experimenting with response times during the development of the framework, peak time was set to 8AM-5PM EST and off-peak was set to 5PM-8AM EST.

The reason for using EST timestamps is because we identified that peak time was well aligned with US working hours.

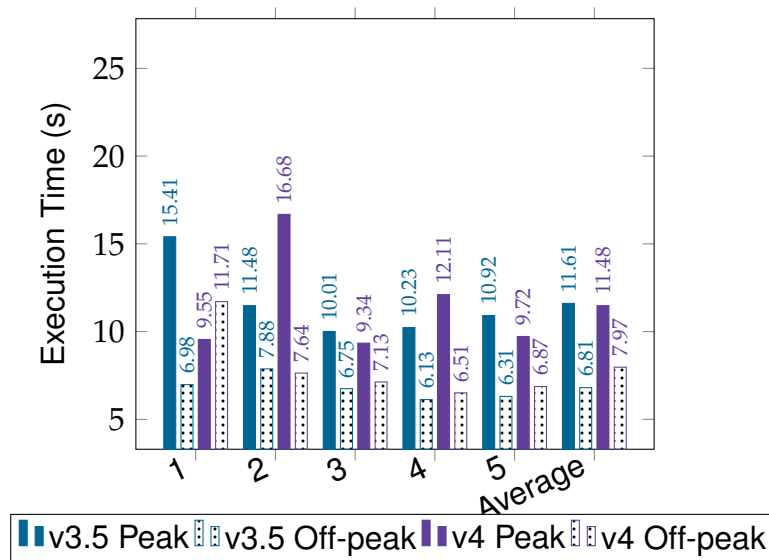


Figure A.1: Execution Time for Question 1 with GPT-3.5-Turbo and GPT-4-Turbo

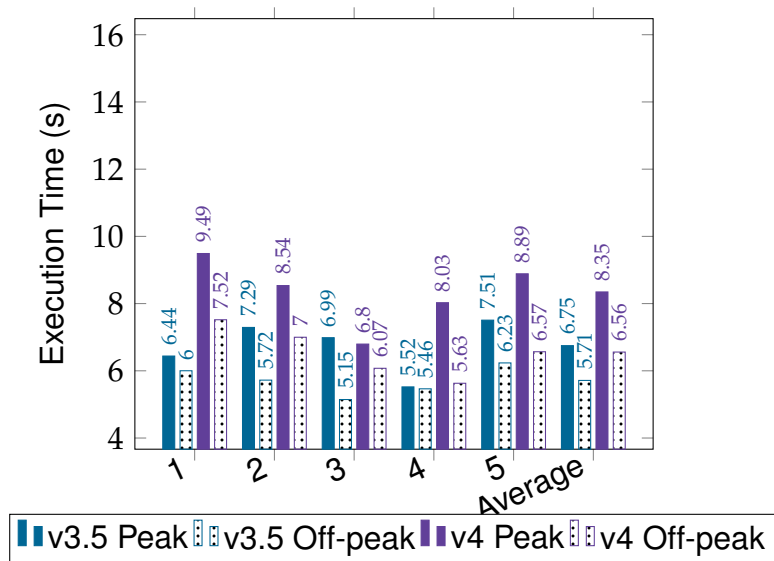


Figure A.2: Execution Time for Question 2 with GPT-3.5-Turbo and GPT-4-Turbo

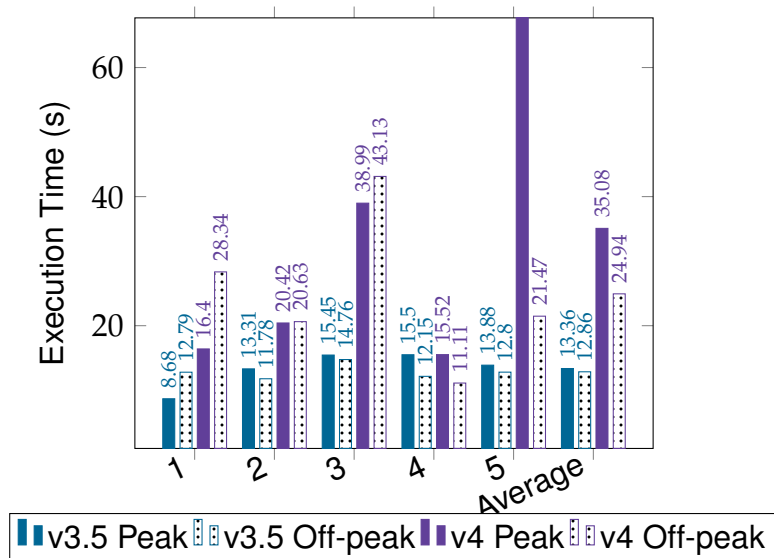


Figure A.3: Execution Time for Question 3 with GPT-3.5-Turbo and GPT-4-Turbo

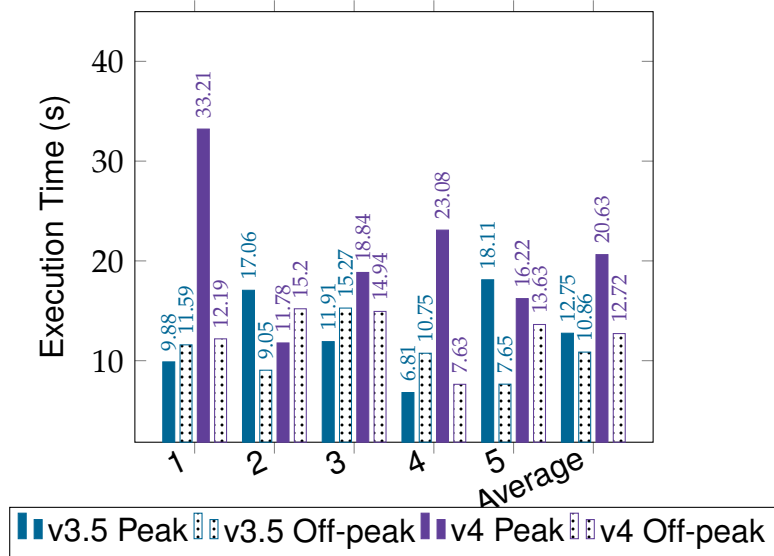


Figure A.4: Execution Time for Question 4 with GPT-3.5-Turbo and GPT-4-Turbo

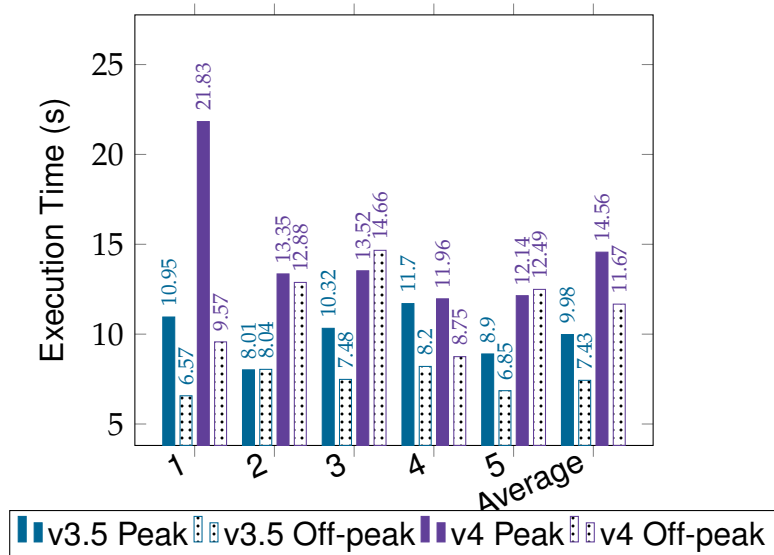


Figure A.5: Execution Time for Question 5 with GPT-3.5-Turbo and GPT-4-Turbo

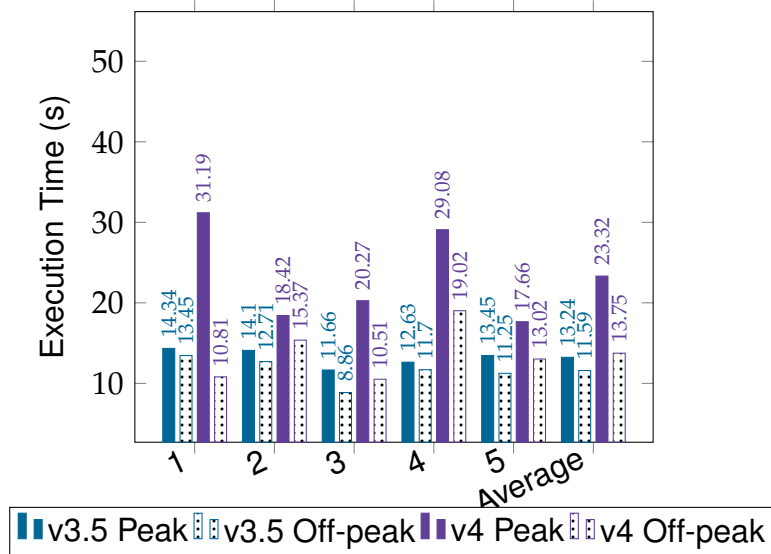


Figure A.6: Execution Time for Question 6 with GPT-3.5-Turbo and GPT-4-Turbo

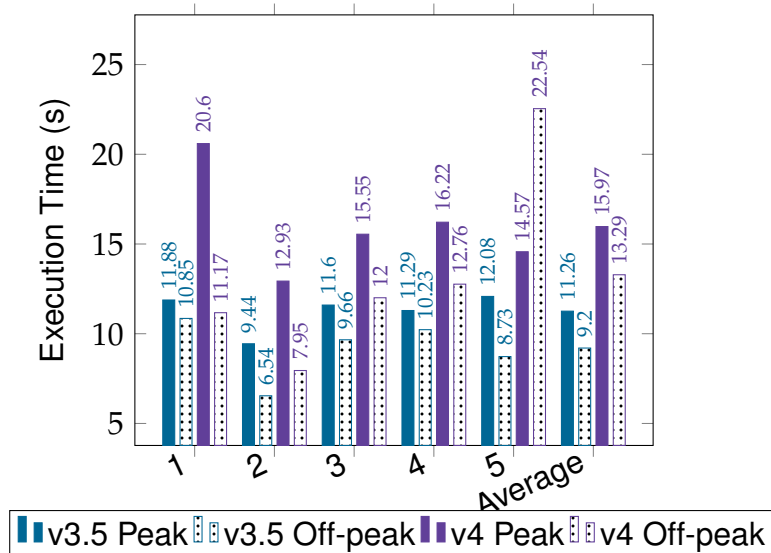


Figure A.7: Execution Time for Question 7 with GPT-3.5-Turbo and GPT-4-Turbo

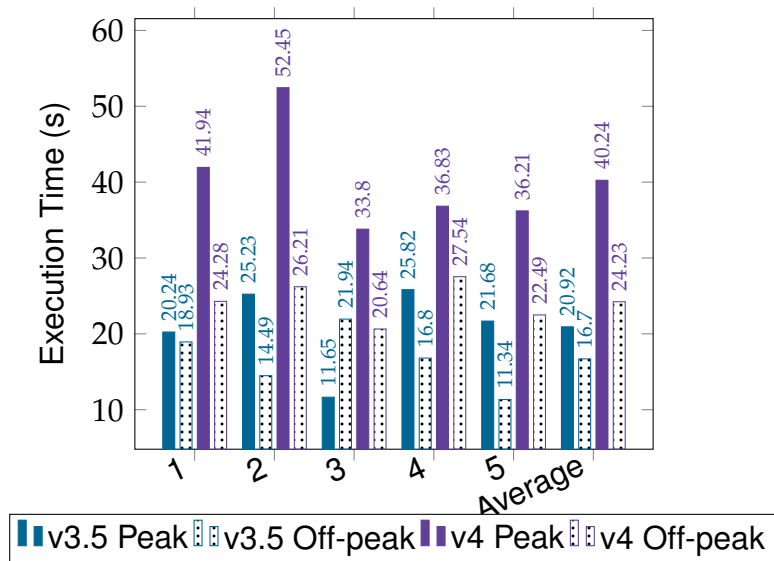


Figure A.8: Execution Time for Question 8 with GPT-3.5-Turbo and GPT-4-Turbo

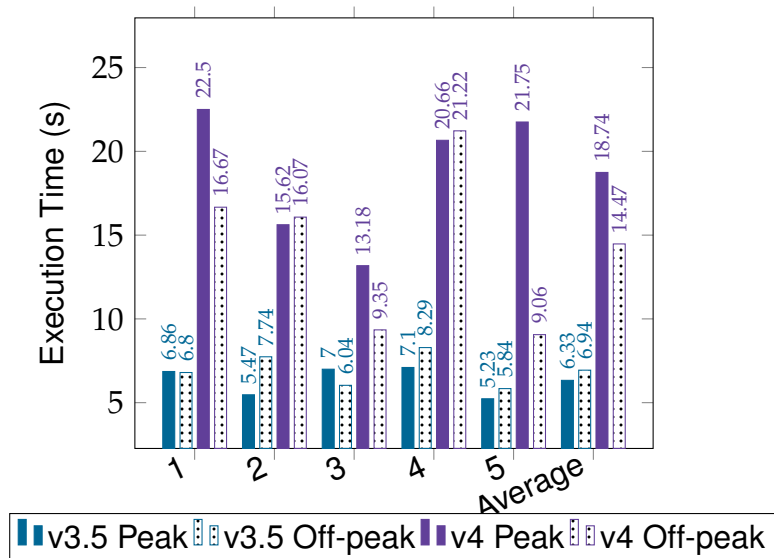


Figure A.9: Execution Time for Question 9 with GPT-3.5-Turbo and GPT-4-Turbo

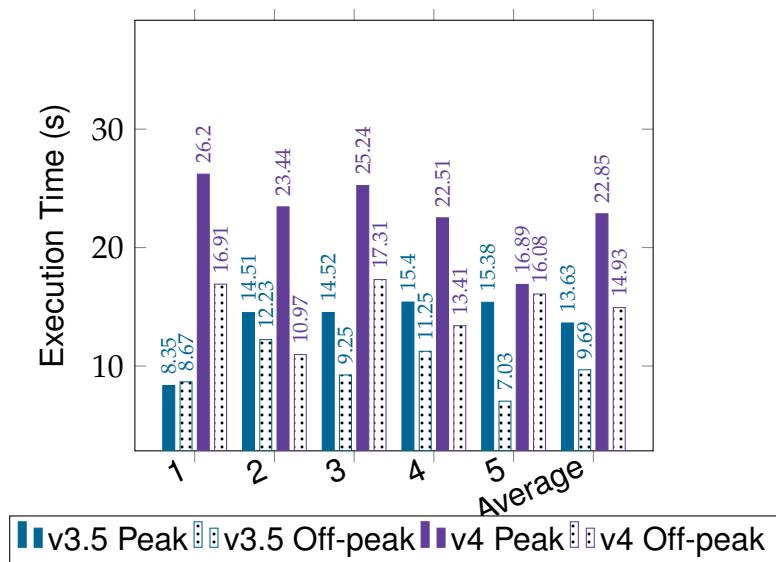


Figure A.10: Execution Time for Question 10 with GPT-3.5-Turbo and GPT-4-Turbo

A.5 SQL Rag

The result obtained from the experiments described in 4.2.2, where FS=2, is presented in this section.

Input 1: Is Manchester United in the database?

Expected Query: SELECT DISTINCT team.name FROM games JOIN teams team ON team.id = games.home_team_id OR team.id = games.away_team_id WHERE games.league_id = (SELECT id FROM leagues WHERE name = '<league_name>') AND games.season = '<season>'

Returned Queries: select distinct t.name as team_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id where p.name = '<players>' order by t.name, select distinct p.name as player_name, t.name as team_name, g.season, l.name as league_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id join games g on gl.game_id = g.id join leagues l on g.league_id = l.id where p.name = '<player>' order by p.name, t.name, g.season, l.name

Correct: No

Input 2: Give me the total home goals for Bayern M in the 2014-15 season

Expected Query: SELECT t.name AS TeamName, g.season, SUM(CASE WHEN g.home_team_id = t.id THEN g.goal_home ELSE 0 END + CASE WHEN g.away_team_id = t.id THEN g.goal_away ELSE 0 END) AS TotalGoals FROM games g JOIN teams t ON g.home_team_id = t.id OR g.away_team_id = t.id WHERE t.name = '<team>' AND g.season = '<season>' GROUP BY t.name, g.season;

Returned Queries: select teams.name, sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) as home_goals, sum(case when games.away_team_id = teams.id then games.goal_away else 0 end) as away_goals, count(*) as matches_played, (sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) + sum(case when games.away_team_id = teams.id then games.goal_away else 0 end)) * 1.0 / count(*) as avg_goals_per_match from games inner join teams on teams.id = games.home_team_id or teams.id = games.away_team_id where games.season = '2016-2017' group by teams.name order by (home_goals + away_goals) desc limit 3;); select t.name as teamname, g.season, sum(case when g.home_team_id = t.id then g.goal_home else 0 end + case when g.away_team_id = t.id then g.goal_away else 0 end) as totalgoals from games g join teams t on g.home_team_id = t.id or g.away_team_id = t.id where t.name = '<team>' and g.season = '<season>' group by t.name, g.season

Correct: Yes

Input 3: Calculate home advantage for Real Madrid in the 2015-16 season

Expected Query: SELECT teams.name, SUM(CASE WHEN games.home_team_id = teams.id THEN games.goal_home ELSE 0 END) AS home_goals, SUM(CASE WHEN games.away_team_id = teams.id THEN games.goal_away ELSE 0 END) AS away_goals, COUNT(*) AS matches_played, (SUM(CASE WHEN games.home_team_id = teams.id THEN games.goal_home ELSE 0 END) + SUM(CASE WHEN games.away_team_id = teams.id THEN games.goal_away ELSE 0 END)) * 1.0 / COUNT(*) AS avg_goals_per_match FROM games INNER JOIN teams ON teams.id = games.home_team_id OR teams.id = games.away_team_id WHERE games.season = '2016-2017' GROUP BY teams.name ORDER BY (home_goals + away_goals) DESC LIMIT 3;);

Returned Queries: select teams.name, sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) as home_goals, sum(case when games.away_team_id = teams.id then games.goal_away else 0 end) as away_goals, count(*) as matches_played, (sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) + sum(case when games.away_team_id = teams.id then games.goal_away else 0 end)) * 1.0 / count(*) as avg_goals_per_match from games inner join teams on teams.id = games.home_team_id or teams.id = games.away_team_id where games.season = '2016-2017' group by teams.name order by (home_goals + away_goals) desc limit 3;); with home_games as (select g.id, g.season, g.home_team_id as team_id, case when g.goal_home > g.goal_away then 1 else 0 end as won, case when g.goal_home = g.goal_away then 1 else 0 end as draw, case when g.goal_home < g.goal_away then 1 else 0 end as lost from games g join teams t on g.home_team_id = t.id where t.name = '<team>' and g.season = '<season>'), away_games as (select g.id, g.season, g.away_team_id as team_id, case when g.goal_away > g.goal_home then 1 else 0 end as won, case when g.goal_away = g.goal_home then 1 else 0 end as draw, case when g.goal_away < g.goal_home then 1 else 0 end as lost from games g join teams t on g.away_team_id = t.id where t.name = '<team>' and g.season = '<season>'), home_stats as (select count(*) as total_home_games, sum(won) as home_wins, sum(draw) as home_draws, sum(lost) as home_losses from home_games), away_stats as (select count(*) as total_away_games, sum(won) as away_wins, sum(draw) as away_draws, sum(lost) as away_losses from away_games) select hs.total_home_games, hs.home_wins, hs.home_draws, hs.home_losses, as_stats.total_away_games, as_stats.away_wins, as_stats.away_draws, as_stats.away_losses from home_stats hs, away_stats as_stats

Correct: Yes

Input 4: How many goals did Messi score in the 15-16 season?

Expected Query: SELECT COUNT(*) AS goal_count FROM player_events pe JOIN players p ON pe.player_id = p.hash JOIN games g ON pe.game_id = g.id JOIN player_event_labels pel ON pe.type = pel.id WHERE p.name = <player>AND g.season = <season>AND pel.label = 'Goal';

Returned Queries: select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and pel.label = 'goal', select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join leagues l on g.league_id = l.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and l.name = <league>and pel.label = 'goal'

Correct: Yes

Input 5: How many yellow-cards did Enzo Perez get in the 15-2016 season?

Expected Query: SELECT COUNT(*) AS goal_count FROM player_events pe JOIN players p ON pe.player_id = p.hash JOIN games g ON pe.game_id = g.id JOIN player_event_labels pel ON pe.type = pel.id WHERE p.name = <player>AND g.season = <season>AND pel.label = 'Goal';

Returned Queries: select count(*) as substitution_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id where p.hash = <player_hash>and g.season = <season>and (pe.type = 6 or pe.type = 7), select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and pel.label = 'goal'

Correct: Yes

Input 6: List all teams that played a game against Napoli in 2016-17 season in seriea? Do not limit the number of results.

Expected Query: SELECT DISTINCT CASE WHEN ht.name = '<team>' THEN at.name ELSE ht.name END AS OpponentTeam FROM games g JOIN teams ht ON g.home_team_id = ht.id JOIN teams at ON g.away_team_id = at.id JOIN leagues l ON g.league_id = l.id WHERE (ht.name = '<team>' OR at.name = '<team>') AND l.name = '<league>' AND g.season = '<season>' ORDER BY OpponentTeam;

Returned Queries: select distinct opponent.name as opponent_name from games join teams as opponent on (opponent.id = games.home_team_id or opponent.id = games.away_team_id) join teams as specified_team on (specified_team.id = games.home_team_id or specified_team.id = games.away_team_id) where (games.home_team_id = (select id from teams where name = '<team>') or games.away_team_id = (select id from teams where name = '<team>')) and games.season = '<season>' and opponent.name != '<team>', select distinct t.name as team_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id where p.name = '<player>' order by t.name

Correct: No

Input 7: Give all the teams in the league ucl in the 2015-2016 season?

Expected Query: SELECT DISTINCT team.name FROM games JOIN teams team ON team.id = games.home_team_id OR team.id = games.away_team_id WHERE games.league_id = (SELECT id FROM leagues WHERE name = '<league_name>') AND games.season = '<season>'

Returned Queries: select * from teams, select distinct t.name as team_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id where p.name = '<player>' order by t.name

Correct: No

Input 8: Give me all games in epl with yellow cards in the first half in the 2015-2016 season

Expected Query: SELECT p.name AS player_name, pel.label AS card_type, pe.time AS event_time FROM player_events pe JOIN players p ON pe.player_id = p.hash JOIN player_event_labels pel ON pe.type = pel.id WHERE pe.game_id = <game_id>AND (pel.label = 'Yellow card' OR pel.label = 'Red card') ORDER BY CAST(pe.time AS UNSIGNED) ASC;

Returned Queries: select p.name as player_name, pel.label as card_type, pe.time as event_time from player_events pe join players p on pe.player_id = p.hash join player_event_labels pel on pe.type = pel.id where pe.game_id = <game_id>and (pel.label = 'yellow card' or pel.label = 'red card') order by cast(pe.time as unsigned) asc, select ht.name as home_team, at.name as away_team, g.score, g.date from games g join leagues l on g.league_id = l.id join events e on g.id = e.game_id and g.home_team_id = e.team_id join teams ht on g.home_team_id = ht.id join teams at on g.away_team_id = at.id where l.name = '<leagues>' and g.season = '<season>' and e.period = 1 and e.label = '<event>' group by g.id

Correct: Yes

Input 9: What teams and leagues has Adnan Januzaj play in?

Expected Query: SELECT DISTINCT t.name AS team_name FROM game_lineup gl JOIN players p ON gl.player_id = p.hash JOIN teams t ON gl.team_id = t.id WHERE p.name = '<player>' ORDER BY t.name;

Returned Queries: select distinct p.name as player_name, t.name as team_name, g.season, l.name as league_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id join games g on gl.game_id = g.id join leagues l on g.league_id = l.id where p.name = '<player>' order by p.name, t.name, g.season, l.name, select distinct t.name as team_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id where p.name = '<player>' order by t.name

Correct: Yes

Input 10: List ALL players that started a game for Las Palmas in the 2016-2017 season? Do NOT limit the number of results.

Expected Query: SELECT DISTINCT p.name AS player_name FROM game_lineup gl JOIN players p ON gl.player_id = p.hash JOIN teams t ON gl.team_id = t.id JOIN games g ON gl.game_id = g.id WHERE t.name = '<team>' AND g.season = '<season>' ORDER BY p.name;

Returned Queries: select distinct t.name as team_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id where p.name = '<player>' order by t.name, select distinct p.name as player_name, t.name as team_name, g.season, l.name as league_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id join games g on gl.game_id = g.id join leagues l on g.league_id = l.id where p.name = '<player>' order by p.name, t.name, g.season, l.name

Correct: No

Input 11: Did Ajax or Manchester United win the most games in the 2014-15 season?

Expected Query: WITH home_games AS (SELECT g.id, g.season, g.home_team_id AS team_id, CASE WHEN g.goal_home >g.goal_away THEN 1 ELSE 0 END AS won, CASE WHEN g.goal_home = g.goal_away THEN 1 ELSE 0 END AS draw, CASE WHEN g.goal_home <g.goal_away THEN 1 ELSE 0 END AS lost FROM games g JOIN teams t ON g.home_team_id = t.id WHERE t.name = '<team>' AND g.season = '<season>'), away_games AS (SELECT g.id, g.season, g.away_team_id AS team_id, CASE WHEN g.goal_away >g.goal_home THEN 1 ELSE 0 END AS won, CASE WHEN g.goal_away = g.goal_home THEN 1 ELSE 0 END AS draw, CASE WHEN g.goal_away <g.goal_home THEN 1 ELSE 0 END AS lost FROM games g JOIN teams t ON g.away_team_id = t.id WHERE t.name = '<team>' AND g.season = '<season>'), home_stats AS (SELECT COUNT(*) AS total_home_games, SUM(won) AS home_wins, SUM(draw) AS home_draws, SUM(lost) AS home_losses FROM home_games), away_stats AS (SELECT COUNT(*) AS total_away_games, SUM(won) AS away_wins, SUM(draw) AS away_draws, SUM(lost) AS away_losses FROM away_games) SELECT hs.total_home_games, hs.home_wins, hs.home_draws, hs.home_losses, as_stats.total_away_games, as_stats.away_wins, as_stats.away_draws, as_stats.away_losses FROM home_stats hs, away_stats as_stats;

Returned Queries: select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and pel.label = 'goal', select teams.name, sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) as home_goals, sum(case when games.away_team_id = teams.id then games.goal_away else 0 end) as away_goals, count(*) as matches_played, (sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) + sum(case when games.away_team_id = teams.id then games.goal_away else 0 end)) * 1.0 / count(*) as avg_goals_per_match from games inner join teams on teams.id = games.home_team_id or teams.id = games.away_team_id where games.season = '2016-2017' group by teams.name order by (home_goals + away_goals) desc limit 3;)

Correct: No

Input 12: How many yellow and red cards were given in UCL in the 2015-2016 season?

Expected Query: SELECT p.name AS player_name, pel.label AS card_type, pe.time AS event_time FROM player_events pe JOIN players p ON pe.player_id = p.hash JOIN player_event_labels pel ON pe.type = pel.id WHERE pe.game_id = <game_id>AND (pel.label = 'Yellow card' OR pel.label = 'Red card') ORDER BY CAST(pe.time AS UNSIGNED) ASC;

Returned Queries: select p.name as player_name, pel.label as card_type, pe.time as event_time from player_events pe join players p on pe.player_id = p.hash join player_event_labels pel on pe.type = pel.id where pe.game_id = <game_id>and (pel.label = 'yellow card' or pel.label = 'red card') order by cast(pe.time as unsigned) asc, select count(*) as substitution_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id where p.hash = <player_hash>and g.season = <season>and (pe.type = 6 or pe.type = 7)

Correct: Yes

Input 13: Is Messi and C Ronaldo in the database?

Expected Query: SELECT * FROM players WHERE name = 'name';

Returned Queries: select * from players where name = 'name', select distinct p.name as player_name, t.name as team_name, g.season, l.name as league_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id join games g on gl.game_id = g.id join leagues l on g.league_id = l.id where p.name = '<player>' order by p.name, t.name, g.season, l.name

Correct: Yes

Input 14: How many goals did E Hazard score in the game between Bournemouth and Chelsea in the 2015-2016 season?

Expected Query: SELECT COUNT(*) AS goal_count FROM player_events pe JOIN players p ON pe.player_id = p.hash JOIN games g ON pe.game_id = g.id JOIN player_event_labels pel ON pe.type = pel.id WHERE p.name = <player>AND g.season = <season>AND pel.label = 'Goal';

Returned Queries: select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and pel.label = 'goal', select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join leagues l on g.league_id = l.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and l.name = <league>and pel.label = 'goal'

Correct: Yes

Input 15: How many yellow cards were given in the game between Bayer Munich and S Donetsk in, and did anyone get a red card the 2014-15 ucl?

Expected Query: SELECT p.name AS player_name, pel.label AS card_type, pe.time AS event_time FROM player_events pe JOIN players p ON pe.player_id = p.hash JOIN player_event_labels pel ON pe.type = pel.id WHERE pe.game_id = <game_id>AND (pel.label = 'Yellow card' OR pel.label = 'Red card') ORDER BY CAST(pe.time AS UNSIGNED) ASC;

Returned Queries: select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and pel.label = 'goal', select distinct p.name as player_name, t.name as team_name, g.season, l.name as league_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id join games g on gl.game_id = g.id join leagues l on g.league_id = l.id where p.name = <player> order by p.name, t.name, g.season, l.name

Correct: No

Input 16: Make a list of when corners happened in the EPL 2015-2016 season. Aggregate by a period of 15 minutes.

Expected Query: SELECT ht.name AS home_team, at.name AS away_team, g.score, g.date, COUNT(e.id) AS event_count FROM games g JOIN leagues l ON g.league_id = l.id JOIN events e ON g.id = e.game_id AND g.home_team_id = e.team_id JOIN teams ht ON g.home_team_id = ht.id JOIN teams at ON g.away_team_id = at.id WHERE l.name = '<leagues>' AND g.season = '<season>' AND e.label = '<event>' GROUP BY g.id;

Returned Queries: select teams.name, sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) as home_goals, sum(case when games.away_team_id = teams.id then games.goal_away else 0 end) as away_goals, count(*) as matches_played, (sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) + sum(case when games.away_team_id = teams.id then games.goal_away else 0 end)) * 1.0 / count(*) as avg_goals_per_match from games inner join teams on teams.id = games.home_team_id or teams.id = games.away_team_id where games.season = '2016-2017' group by teams.name order by (home_goals + away_goals) desc limit 3;), select t.name as team_name, a.period, sum(case when a.label = 'shots on target' then 1 else 0 end) as shots_on_target, sum(case when a.label = 'goal' then 1 else 0 end) as goals from annotations a join teams t on a.team_id = t.id where a.game_id = x and (a.label = 'shots on target' or a.label = 'goal') group by t.name, a.period order by t.name, a.period

Correct: No

Input 17: What league is Manchester United, Arsenal, Bournemouth, Real Madrid, Chelsea and Liverpool in?

Expected Query: SELECT * FROM teams;

Returned Queries: select distinct p.name as player_name, t.name as team_name, g.season, l.name as league_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id join games g on gl.game_id = g.id join leagues l on g.league_id = l.id where p.name = '<player>' order by p.name, t.name, g.season, l.name, select p.name as player_name, pe.game_id, pe.time as event_time from player_events pe join players p on pe.player_id = p.hash join (select g.id from games g join leagues l on g.league_id = l.id where g.season = '<season>' and l.id = <league_id>order by g.id limit 1) as first_game on pe.game_id = first_game.id join player_event_labels pel on pe.type = pel.id where pel.label = <event>order by cast(pe.time as unsigned) asc limit 1

Correct: No

Input 18: How many players has Aleksandar as first name in the database, and how many goals have they scored in total?

Expected Query: SELECT * FROM players WHERE name = 'name';

Returned Queries: select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and pel.label = 'goal', select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join leagues l on g.league_id = l.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and l.name = <league>and pel.label = 'goal'

Correct: No

Input 19: What did the commentary say about the game between Arsenal and Southampton in the 2016-17 season?

Expected Query: SELECT description FROM captions WHERE game_id = <game_id>AND important = 1 AND LENGTH(description) >20 ORDER BY period, game_time;

Returned Queries: select description from captions where game_id = <game_id>and important = 1 and length(description) >20 order by period, game_time, select distinct p.name as player_name, t.name as team_name, g.season, l.name as league_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id join games g on gl.game_id = g.id join leagues l on g.league_id = l.id where p.name = '<player>' order by p.name, t.name, g.season, l.name

Correct: Yes

Input 20: Have Mesut Ozil, Pablo Insua or Alex Pike played for West Ham or Barcelona?

Expected Query: SELECT DISTINCT p.name AS player_name, t.name AS team_name, g.season, l.name AS league_name FROM game_lineup gl JOIN players p ON gl.player_id = p.hash JOIN teams t ON gl.team_id = t.id JOIN games g ON gl.game_id = g.id JOIN leagues l ON g.league_id = l.id WHERE p.name = '<player>' ORDER BY p.name, t.name, g.season, l.name;

Returned Queries: select distinct p.name as player_name, t.name as team_name, g.season, l.name as league_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id join games g on gl.game_id = g.id join leagues l on g.league_id = l.id where p.name = '<player>' order by p.name, t.name, g.season, l.name, select distinct t.name as team_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id where p.name = '<player>' order by t.name

Correct: Yes

Input 21: How many games has Real Madrid and Malaga played against each other? List each game with the date and score. Also say who was the home team. Output should be: home_team_name, away_team_name, date, home_team_score, away_team_score

Expected Query: SELECT DISTINCT opponent.name AS opponent_name FROM games JOIN teams AS opponent ON (opponent.id = games.home_team_id OR opponent.id = games.away_team_id) JOIN teams AS specified_team ON (specified_team.id = games.home_team_id OR specified_team.id = games.away_team_id) WHERE (games.home_team_id = (SELECT id FROM teams WHERE name = '<team>') OR games.away_team_id = (SELECT id FROM teams WHERE name = '<team>')) AND games.season = '<season>' AND opponent.name != '<team>'

Returned Queries: select teams.name, sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) as home_goals, sum(case when games.away_team_id = teams.id then games.goal_away else 0 end) as away_goals, count(*) as matches_played, (sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) + sum(case when games.away_team_id = teams.id then games.goal_away else 0 end)) * 1.0 / count(*) as avg_goals_per_match from games inner join teams on teams.id = games.home_team_id or teams.id = games.away_team_id where games.season = '2016-2017' group by teams.name order by (home_goals + away_goals) desc limit 3;), select t.name as team_name, p.name || case when l.captain then ' (c)' else '' end as player_name, l.shirt_number from games g join teams t on g.home_team_id = t.id join game_lineup l on t.id = l.team_id and l.game_id = g.id join players p on l.player_id = p.hash where g.id = x

Correct: No

Input 22: List all teams

Expected Query: SELECT * FROM teams;

Returned Queries: select * from teams, select distinct team.name from games join teams team on team.id = games.home_team_id or team.id = games.away_team_id where games.league_id = (select id from leagues where name = '<league_name>') and games.season = '<season>'

Correct: Yes

Input 23: Find player named Filipe Luis

Expected Query: SELECT * FROM players WHERE name = 'name';

Returned Queries: select * from players where name = 'name', select distinct t.name as team_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id where p.name = '<player>' order by t.name

Correct: Yes

Input 24: Select the names of teams and calculate their total home and away goals in the 2016-2017 season. Count the total matches played and calculate the average goals per match. Order the teams by their total goals scored and limit the results to the top three.

Expected Query: SELECT teams.name, SUM(CASE WHEN games.home_team_id = teams.id THEN games.goal_home ELSE 0 END) AS home_goals, SUM(CASE WHEN games.away_team_id = teams.id THEN games.goal_away ELSE 0 END) AS away_goals, COUNT(*) AS matches_played, (SUM(CASE WHEN games.home_team_id = teams.id THEN games.goal_home ELSE 0 END) + SUM(CASE WHEN games.away_team_id = teams.id THEN games.goal_away ELSE 0 END)) * 1.0 / COUNT(*) AS avg_goals_per_match FROM games INNER JOIN teams ON teams.id = games.home_team_id OR teams.id =

games.away_team_id WHERE games.season = '2016-2017' GROUP BY teams.name ORDER BY (home_goals + away_goals) DESC LIMIT 3;);

Returned Queries: select teams.name, sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) as home_goals, sum(case when games.away_team_id = teams.id then games.goal_away else 0 end) as away_goals, count(*) as matches_played, (sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) + sum(case when games.away_team_id = teams.id then games.goal_away else 0 end)) * 1.0 / count(*) as avg_goals_per_match from games inner join teams on teams.id = games.home_team_id or teams.id = games.away_team_id where games.season = '2016-2017' group by teams.name order by (home_goals + away_goals) desc limit 3;), select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join leagues l on g.league_id = l.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and l.name = <league>and pel.label = 'goal'

Correct: Yes

Input 25: Retrieve the name and country of a player identified by hash = Kty10JVG.

Expected Query: SELECT players.name, players.country FROM players WHERE players.hash = 'hash';

Returned Queries: select players.name, players.country from players where players.hash = 'hash', select * from players where name = 'name'

Correct: Yes

Input 26: Get information about what happened in game 5, between the time range of 10 and 20 minutes.

Expected Query: SELECT event_time_start, event_time_end, period, description FROM Commentary WHERE game_id = game_id AND period = period AND ABS(event_time_start - start_time) <= duration;

Returned Queries: select event_time_start, event_time_end, period, description from commentary where game_id = game_id and period = period and abs(event_time_start - start_time) <= duration, select a.game_id, a.label, a.position, a.period from annotations a join games g on a.game_id = g.id where a.game_id = x and a.label = 'offside' and a.team_id = g.away_team_id

Correct: Yes

Input 27: For the a game with ID 43, list the home team's name, players' names (with a captain indicator), and shirt numbers.

Expected Query: SELECT t.name AS team_name, p.name || CASE WHEN l.captain THEN ' (C)' ELSE '' END AS player_name, l.shirt_number FROM games g JOIN teams t ON g.home_team_id = t.id JOIN game_lineup l ON t.id = l.team_id AND l.game_id = g.id JOIN players p ON l.player_id = p.hash WHERE g.id = X;

Returned Queries: select t.name as team_name, p.name || case when l.captain then ' (c)' else '' end as player_name, l.shirt_number from games g join teams t on g.home_team_id = t.id join game_lineup l on t.id = l.team_id and l.game_id = g.id join players p on l.player_id = p.hash where g.id = x, select home_team.name as home_team, away_team.name as away_team from games join teams as home_team on games.home_team_id = home_team.id join teams as away_team on games.away_team_id = away_team.id where games.id = x

Correct: Yes

Input 28: Who was the home team, and away team in game 52?

Expected Query: SELECT home_team.name AS home_team, away_team.name AS away_team FROM games JOIN teams AS home_team ON games.home_team_id = home_team.id JOIN teams AS away_team ON games.away_team_id = away_team.id WHERE games.id = X;

Returned Queries: select home_team.name as home_team, away_team.name as away_team from games join teams as home_team on games.home_team_id = home_team.id join teams as away_team on games.away_team_id = away_team.id where games.id = x, select t.name as team_name, p.name || case when l.captain then ' (c)' else '' end as player_name, l.shirt_number from games g join teams t on g.home_team_id = t.id join game_lineup l on t.id = l.team_id and l.game_id = g.id join players p on l.player_id = p.hash where g.id = x

Correct: Yes

Input 29: For game 132, list all Shots on targets and goals, for each team (using their name not id) for each period

Expected Query: SELECT t.name AS team_name, a.period, SUM(CASE WHEN a.label = 'Shots on target' THEN 1 ELSE 0 END) AS shots_on_target, SUM(CASE WHEN a.label = 'Goal' THEN 1 ELSE 0 END) AS goals FROM annotations a JOIN teams t ON a.team_id = t.id WHERE a.game_id = X AND (a.label = 'Shots on target' OR a.label = 'Goal') GROUP BY t.name, a.period ORDER BY t.name, a.period;

Returned Queries: select t.name as team_name, a.period, sum(case when a.label = 'shots on target' then 1 else 0 end) as shots_on_target, sum(case when a.label = 'goal' then 1 else 0 end) as goals from annotations a join teams t on a.team_id = t.id where a.game_id = x and (a.label = 'shots on target' or a.label = 'goal') group by t.name, a.period order by t.name, a.period, select teams.name, sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) as home_goals, sum(case when games.away_team_id = teams.id then games.goal_away else 0 end) as away_goals, count(*) as matches_played, (sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) + sum(case when

games.away_team_id = teams.id then games.goal_away else 0 end)) * 1.0 / count(*) as avg_goals_per_match from games inner join teams on teams.id = games.home_team_id or teams.id = games.away_team_id where games.season = '2016-2017' group by teams.name order by (home_goals + away_goals) desc limit 3;)

Correct: Yes

Input 30: How many offsides were caused by the away team in game 65, also get the time of the event

Expected Query: SELECT a.game_id, a.label, a.position, a.period FROM annotations a JOIN games g ON a.game_id = g.id WHERE a.game_id = X AND a.label = 'Offside' AND a.team_id = g.away_team_id;

Returned Queries: select a.game_id, a.label, a.position, a.period from annotations a join games g on a.game_id = g.id where a.game_id = x and a.label = 'offside' and a.team_id = g.away_team_id, select home_team.name as home_team, away_team.name as away_team from games join teams as home_team on games.home_team_id = home_team.id join teams as away_team on games.away_team_id = away_team.id where games.id = x

Correct: Yes

Input 31: all goals scored by Dortmund in 2014-2015

Expected Query: SELECT t.name AS TeamName, g.season, SUM(CASE WHEN g.home_team_id = t.id THEN g.goal_home ELSE 0 END + CASE WHEN g.away_team_id = t.id THEN g.goal_away ELSE 0 END) AS TotalGoals FROM games g JOIN teams t ON g.home_team_id = t.id OR g.away_team_id = t.id WHERE t.name = '<team>' AND g.season = '<season>' GROUP BY t.name, g.season;

Returned Queries: select t.name as teamname, g.season, sum(case when g.home_team_id = t.id then g.goal_home else 0 end + case when g.away_team_id = t.id then g.goal_away else 0 end) as totalgoals from games g join teams t on g.home_team_id = t.id or g.away_team_id = t.id where t.name = '<team>' and g.season = '<season>' group by t.name, g.season, select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and pel.label = 'goal'

Correct: Yes

Input 32: All games played by Benfica in 2015-2016 in cl

Expected Query: SELECT g.id, g.date, g.season, l.name AS LeagueName, ht.name AS HomeTeam, at.name AS AwayTeam, g.score FROM games g JOIN teams ht ON g.home_team_id = ht.id JOIN teams at ON g.away_team_id = at.id JOIN leagues l ON g.league_id = l.id WHERE (ht.name = '<team>' OR at.name = '<team>') AND l.name = '<league>' AND g.season = '<season>';

Returned Queries: select g.id, g.date, g.season, l.name as leaguename, ht.name as hometeam, at.name as awayteam, g.score from games g join teams ht on g.home_team_id = ht.id join teams at on g.away_team_id = at.id join leagues l on g.league_id = l.id where (ht.name = '<team>' or at.name = '<team>') and l.name = '<league>' and g.season = '<season>', select t.name as teamname, g.season, sum(case when g.home_team_id = t.id then g.goal_home else 0 end + case when g.away_team_id = t.id then g.goal_away else 0 end) as totalgoals from games g join teams t on g.home_team_id = t.id or g.away_team_id = t.id where t.name = '<team>' and g.season = '<season>' group by t.name, g.season

Correct: Yes

Input 33: List all teams that played against ManU in season 2015-2016 and league epl

Expected Query: SELECT DISTINCT CASE WHEN ht.name = '<team>' THEN at.name ELSE ht.name END AS OpponentTeam FROM games g JOIN teams ht ON g.home_team_id = ht.id JOIN teams at ON g.away_team_id = at.id JOIN leagues l ON g.league_id = l.id WHERE (ht.name = '<team>' OR at.name = '<team>') AND l.name = '<league>' AND g.season = '<season>' ORDER BY OpponentTeam;

Returned Queries: select distinct opponent.name as opponent_name from games join teams as opponent on (opponent.id = games.home_team_id or opponent.id = games.away_team_id) join teams as specified_team on (specified_team.id = games.home_team_id or specified_team.id = games.away_team_id) where (games.home_team_id = (select id from teams where name = '<team>') or games.away_team_id = (select id from teams where name = '<team>')) and games.season = '<season>' and opponent.name != '<team>', select distinct case when ht.name = '<team>' then at.name else ht.name end as opponentteam from games g join teams ht on g.home_team_id = ht.id join teams at on g.away_team_id = at.id join leagues l on g.league_id = l.id where (ht.name = '<team>' or at.name = '<team>') and l.name = '<league>' and g.season = '<season>' order by opponentteam

Correct: Yes

Input 34: Get home and away stats for Real Madrid in 2015-2016

Expected Query: WITH home_games AS (SELECT g.id, g.season, g.home_team_id AS team_id, CASE WHEN g.goal_home >g.goal_away THEN 1 ELSE 0 END AS won, CASE WHEN g.goal_home = g.goal_away THEN 1 ELSE 0 END AS draw, CASE WHEN g.goal_home <g.goal_away THEN 1 ELSE 0 END AS lost FROM games g JOIN teams t ON g.home_team_id = t.id WHERE t.name = '<team>' AND g.season = '<season>'), away_games AS (SELECT g.id, g.season, g.away_team_id AS team_id, CASE WHEN g.goal_away >g.goal_home THEN 1 ELSE 0 END AS won, CASE WHEN g.goal_away = g.goal_home

THEN 1 ELSE 0 END AS draw, CASE WHEN g.goal_away <g.goal_home THEN 1 ELSE 0 END AS lost FROM games g JOIN teams t ON g.away_team_id = t.id WHERE t.name = '<team>' AND g.season = '<season>'), home_stats AS (SELECT COUNT(*) AS total_home_games, SUM(won) AS home_wins, SUM(draw) AS home_draws, SUM(lost) AS home_losses FROM home_games), away_stats AS (SELECT COUNT(*) AS total_away_games, SUM(won) AS away_wins, SUM(draw) AS away_draws, SUM(lost) AS away_losses FROM away_games) SELECT hs.total_home_games, hs.home_wins, hs.home_draws, hs.home_losses, as_stats.total_away_games, as_stats.away_wins, as_stats.away_draws, as_stats.away_losses FROM home_stats hs, away_stats as_stats;

Returned Queries: with home_games as (select g.id, g.season, g.home_team_id as team_id, case when g.goal_home >g.goal_away then 1 else 0 end as won, case when g.goal_home = g.goal_away then 1 else 0 end as draw, case when g.goal_home <g.goal_away then 1 else 0 end as lost from games g join teams t on g.home_team_id = t.id where t.name = '<team>' and g.season = '<season>'), away_games as (select g.id, g.season, g.away_team_id as team_id, case when g.goal_away >g.goal_home then 1 else 0 end as won, case when g.goal_away = g.goal_home then 1 else 0 end as draw, case when g.goal_away <g.goal_home then 1 else 0 end as lost from games g join teams t on g.away_team_id = t.id where t.name = '<team>' and g.season = '<season>'), home_stats as (select count(*) as total_home_games, sum(won) as home_wins, sum(draw) as home_draws, sum(lost) as home_losses from home_games), away_stats as (select count(*) as total_away_games, sum(won) as away_wins, sum(draw) as away_draws, sum(lost) as away_losses from away_games) select hs.total_home_games, hs.home_wins, hs.home_draws, hs.home_losses, as_stats.total_away_games, as_stats.away_wins, as_stats.away_draws, as_stats.away_losses from home_stats hs, away_stats as_stats, select teams.name, sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) as home_goals, sum(case when games.away_team_id = teams.id then games.goal_away else 0 end) as away_goals, count(*) as matches_played, (sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) + sum(case when games.away_team_id = teams.id then games.goal_away else 0 end)) * 1.0 / count(*) as avg_goals_per_match from games inner join teams on teams.id = games.home_team_id or teams.id = games.away_team_id where games.season = '2016-2017' group by teams.name order by (home_goals + away_goals) desc limit 3;)

Correct: Yes

Input 35: How many goals did Messi score in 2014-2015 in la liga?

Expected Query: SELECT COUNT(*) AS goal_count FROM player_events pe JOIN players p ON pe.player_id = p.hash JOIN games g ON pe.game_id = g.id JOIN leagues l ON g.league_id = l.id JOIN player_event_labels pel ON pe.type = pel.id WHERE p.name = <player>AND g.season = <season>AND l.name = <league>AND pel.label = 'Goal';

Returned Queries: select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join leagues l on g.league_id = l.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and l.name = <league>and pel.label = 'goal', select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and pel.label = 'goal'

Correct: Yes

Input 36: How many goals did John Obi Mikel score in 2015-2016?

Expected Query: SELECT COUNT(*) AS goal_count FROM player_events pe JOIN players p ON pe.player_id = p.hash JOIN games g ON pe.game_id = g.id JOIN player_event_labels pel ON pe.type = pel.id WHERE p.name = <player>AND g.season = <season>AND pel.label = 'Goal';

Returned Queries: select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and pel.label = 'goal', select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join leagues l on g.league_id = l.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and l.name = <league>and pel.label = 'goal'

Correct: Yes

Input 37: List all teams that played against Chelsea in season 2014-2015

Expected Query: SELECT DISTINCT opponent.name AS opponent_name FROM games JOIN teams AS opponent ON (opponent.id = games.home_team_id OR opponent.id = games.away_team_id) JOIN teams AS specified_team ON (specified_team.id = games.home_team_id OR specified_team.id = games.away_team_id) WHERE (games.home_team_id = (SELECT id FROM teams WHERE name = '<team>') OR games.away_team_id = (SELECT id FROM teams WHERE name = '<team>')) AND games.season = '<season>' AND opponent.name != '<team>'

Returned Queries: select distinct opponent.name as opponent_name from games join teams as opponent on (opponent.id = games.home_team_id or opponent.id = games.away_team_id) join teams as specified_team on (specified_team.id = games.home_team_id or specified_team.id = games.away_team_id) where (games.home_team_id = (select id from teams where name = '<team>') or games.away_team_id = (select id from teams where name = '<team>')) and games.season = '<season>' and opponent.name != '<team>', select distinct case when ht.name = '<teams>' then at.name else ht.name end as opponentteam from games g join teams ht on g.home_team_id = ht.id join teams at on g.away_team_id = at.id join leagues l on

g.league_id = l.id where (ht.name = '<team>' or at.name = '<team>') and l.name = '<league>' and g.season = '<season>' order by opponentteam

Correct: Yes

Input 38: List all teams in cl in 15-16

Expected Query: SELECT DISTINCT team.name FROM games JOIN teams team ON team.id = games.home_team_id OR team.id = games.away_team_id WHERE games.league_id = (SELECT id FROM leagues WHERE name = '<league_name>') AND games.season = '<season>'

Returned Queries: select * from teams, select distinct team.name from games join teams team on team.id = games.home_team_id or team.id = games.away_team_id where games.league_id = (select id from leagues where name = '<league_name>') and games.season = '<season>'

Correct: Yes

Input 39: List all games in seria in 14-15 with Goal in first half

Expected Query: SELECT ht.name AS home_team, at.name AS away_team, g.score, g.date FROM games g JOIN leagues l ON g.league_id = l.id JOIN events e ON g.id = e.game_id AND g.home_team_id = e.team_id JOIN teams ht ON g.home_team_id = ht.id JOIN teams at ON g.away_team_id = at.id WHERE l.name = '<league>' AND g.season = '<season>' AND e.period = 1 AND e.label = '<event>' GROUP BY g.id;

Returned Queries: select t.name as teamname, g.season, sum(case when g.home_team_id = t.id then g.goal_home else 0 end + case when g.away_team_id = t.id then g.goal_away else 0 end) as totalgoals from games g join teams t on g.home_team_id = t.id or g.away_team_id = t.id where t.name = '<team>' and g.season = '<season>' group by t.name, g.season, select t.name as team_name, a.period, sum(case when a.label = 'shots on target' then 1 else 0 end) as shots_on_target, sum(case when a.label = 'goal' then 1 else 0 end) as goals from annotations a join teams t on a.team_id = t.id where a.game_id = x and (a.label = 'shots on target' or a.label = 'goal') group by t.name, a.period order by t.name, a.period

Correct: No

Input 40: List all games in ucl in 2016-17 with offside, and include the number of times the event occurred

Expected Query: SELECT ht.name AS home_team, at.name AS away_team, g.score, g.date, COUNT(e.id) AS event_count FROM games g JOIN leagues l ON g.league_id = l.id JOIN events e ON g.id = e.game_id AND g.home_team_id = e.team_id JOIN teams ht ON g.home_team_id = ht.id JOIN teams at ON g.away_team_id = at.id WHERE l.name = '<leagues>' AND g.season = '<season>' AND e.label = '<event>' GROUP BY g.id;

Returned Queries: select a.game_id, a.label, a.position, a.period from annotations a join games g on a.game_id = g.id where a.game_id = x and a.label = 'offside' and a.team_id = g.away_team_id, select ht.name as home_team, at.name as away_team, g.score, g.date, count(e.id) as event_count from games g join leagues l on g.league_id = l.id join events e on g.id = e.game_id and g.home_team_id = e.team_id join teams ht on g.home_team_id = ht.id join teams at on g.away_team_id = at.id where l.name = '<league>' and g.season = '<season>' and e.label = '<event>' group by g.id

Correct: Yes

Input 41: What teams and in what season did John Obi Mikel play in?

Expected Query: SELECT DISTINCT p.name AS player_name, t.name AS team_name, g.season, l.name AS league_name FROM game_lineup gl JOIN players p ON gl.player_id = p.hash JOIN teams t ON gl.team_id = t.id JOIN games g ON gl.game_id = g.id JOIN leagues l ON g.league_id = l.id WHERE p.name = '<player>' ORDER BY p.name, t.name, g.season, l.name;

Returned Queries: select distinct p.name as player_name, t.name as team_name, g.season, l.name as league_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id join games g on gl.game_id = g.id join leagues l on g.league_id = l.id where p.name = '<player>' order by p.name, t.name, g.season, l.name, select distinct t.name as team_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id where p.name = '<player>' order by t.name

Correct: Yes

Input 42: List all players in Aston Villa in 14-15

Expected Query: SELECT DISTINCT p.name AS player_name FROM game_lineup gl JOIN players p ON gl.player_id = p.hash JOIN teams t ON gl.team_id = t.id JOIN games g ON gl.game_id = g.id WHERE t.name = '<team>' AND g.season = '<season>' ORDER BY p.name;

Returned Queries: select distinct t.name as team_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id where p.name = '<player>' order by t.name, select distinct p.name as player_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id join games g on gl.game_id = g.id where t.name = '<team>' and g.season = '<season>' order by p.name

Correct: Yes

Input 43: List all teams Steven Reid has played for

Expected Query: SELECT DISTINCT t.name AS team_name FROM game_lineup gl JOIN players p ON gl.player_id = p.hash JOIN teams t ON gl.team_id = t.id WHERE p.name = '<player>' ORDER BY t.name;

Returned Queries: select distinct t.name as team_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id where p.name = '<player>' order by t.name, select * from teams

Correct: Yes

Input 44: List all yellow and red cards for game 43, sorted by time

Expected Query: SELECT p.name AS player_name, pel.label AS card_type, pe.time AS event_time FROM player_events pe JOIN players p ON pe.player_id = p.hash JOIN player_event_labels pel ON pe.type = pel.id WHERE pe.game_id = <game_id>AND (pel.label = 'Yellow card' OR pel.label = 'Red card') ORDER BY CAST(pe.time AS UNSIGNED) ASC;

Returned Queries: select p.name as player_name, pel.label as card_type, pe.time as event_time from player_events pe join players p on pe.player_id = p.hash join player_event_labels pel on pe.type = pel.id where pe.game_id = <game_id>and (pel.label = 'yellow card' or pel.label = 'red card') order by cast(pe.time as unsigned) asc, select a.game_id, a.label, a.position, a.period from annotations a join games g on a.game_id = g.id where a.game_id = x and a.label = 'offside' and a.team_id = g.away_team_id

Correct: Yes

Input 45: What player had the first yellow card in league epl in season 14-15?

Expected Query: SELECT p.name AS player_name, pe.game_id, pe.time AS event_time FROM player_events pe JOIN players p ON pe.player_id = p.hash JOIN (SELECT g.id FROM games g JOIN leagues l ON g.league_id = l.id WHERE g.season = '<season>' AND l.id = <leauge_id>ORDER BY g.id LIMIT 1) AS first_game ON pe.game_id = first_game.id JOIN player_event_labels pel ON pe.type = pel.id WHERE pel.label = <event>ORDER BY CAST(pe.time AS UNSIGNED) ASC LIMIT 1;

Returned Queries: select p.name as player_name, pe.game_id, pe.time as event_time from player_events pe join players p on pe.player_id = p.hash join (select g.id from games g join leagues l on g.league_id = l.id where g.season = '<season>' and l.id = <leauge_id>order by g.id limit 1) as first_game on pe.game_id = first_game.id join player_event_labels pel on pe.type = pel.id where pel.label = <event>order by cast(pe.time as unsigned) asc limit 1, select count(*) as substitution_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id where p.hash = <player_hash>and g.season = <season>and (pe.type = 6 or pe.type = 7)

Correct: Yes

Input 46: How many times did Sean Dyche get substituted in 15-16?

Expected Query: SELECT COUNT(*) AS substitution_count FROM player_events pe JOIN players p ON pe.player_id = p.hash JOIN games g ON pe.game_id = g.id WHERE p.hash = <player_hash>AND g.season = <season>AND (pe.type = 6 or pe.type = 7)

Returned Queries: select count(*) as substitution_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id where p.hash = <player_hash>and g.season = <season>and (pe.type = 6 or pe.type = 7), select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and pel.label = 'goal'

Correct: Yes

Input 47: List the goals scored by Lukas Podolski in game 153. Include the event time.

Expected Query: SELECT COUNT(*) AS goal_count FROM player_events pe JOIN players p ON pe.player_id = p.hash JOIN games g ON pe.game_id = g.id JOIN player_event_labels pel ON pe.type = pel.id WHERE p.name = <player>AND g.season = <season>AND pel.label = 'Goal';

Returned Queries: select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join leagues l on g.league_id = l.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and l.name = <league>and pel.label = 'goal', select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and pel.label = 'goal'

Correct: Yes

Input 48: List all games ManU played against ManC and the final score. List the games in chronological order, based on the game date.

Expected Query: SELECT teams.name, SUM(CASE WHEN games.home_team_id = teams.id THEN games.goal_home ELSE 0 END) AS home_goals, SUM(CASE WHEN games.away_team_id = teams.id THEN games.goal_away ELSE 0 END) AS away_goals, COUNT(*) AS matches_played, (SUM(CASE WHEN games.home_team_id = teams.id THEN games.goal_home ELSE 0 END) + SUM(CASE WHEN games.away_team_id = teams.id THEN games.goal_away ELSE 0 END)) * 1.0 / COUNT(*) AS avg_goals_per_match FROM games INNER JOIN teams ON teams.id = games.home_team_id OR teams.id =

games.away_team_id WHERE games.season = '2016-2017' GROUP BY teams.name ORDER BY (home_goals + away_goals) DESC LIMIT 3;');

Returned Queries: select teams.name, sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) as home_goals, sum(case when games.away_team_id = teams.id then games.goal_away else 0 end) as away_goals, count(*) as matches_played, (sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) + sum(case when games.away_team_id = teams.id then games.goal_away else 0 end)) * 1.0 / count(*) as avg_goals_per_match from games inner join teams on teams.id = games.home_team_id or teams.id = games.away_team_id where games.season = '2016-2017' group by teams.name order by (home_goals + away_goals) desc limit 3;); select g.id, g.date, g.season, l.name as leaguename, ht.name as hometeam, at.name as awayteam, g.score from games g join teams ht on g.home_team_id = ht.id join teams at on g.away_team_id = at.id join leagues l on g.league_id = l.id where (ht.name = '<team>' or at.name = '<team>') and l.name = '<league>' and g.season = '<season>'

Correct: Yes

Input 49: What is the current roster of teams?

Expected Query: SELECT * FROM teams;

Returned Queries: select * from teams, select distinct p.name as player_name, t.name as team_name, g.season, l.name as league_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id join games g on gl.game_id = g.id join leagues l on g.league_id = l.id where p.name = '<player>' order by p.name, t.name, g.season, l.name

Correct: Yes

Input 50: I need to pull up a player's details using just their first or last name. Lukas Podolski

Expected Query: SELECT * FROM players WHERE name = 'name';

Returned Queries: select * from players where name = 'name', select distinct t.name as team_name from game_lineup gl join players p on gl.player_id = p.hash join teams t on gl.team_id = t.id where p.name = '<player>' order by t.name

Correct: Yes

Input 51: Which teams scored the most in the 2016-2017 season, including their average goals per game?

Expected Query: SELECT teams.name, SUM(CASE WHEN games.home_team_id = teams.id THEN games.goal_home ELSE 0 END) AS home_goals, SUM(CASE WHEN games.away_team_id = teams.id THEN games.goal_away ELSE 0 END) AS away_goals, COUNT(*) AS matches_played, (SUM(CASE WHEN games.home_team_id = teams.id THEN games.goal_home ELSE 0 END) + SUM(CASE WHEN games.away_team_id = teams.id THEN games.goal_away ELSE 0 END)) * 1.0 / COUNT(*) AS avg_goals_per_match FROM games INNER JOIN teams ON teams.id = games.home_team_id OR teams.id = games.away_team_id WHERE games.season = '2016-2017' GROUP BY teams.name ORDER BY (home_goals + away_goals) DESC LIMIT 3;);

Returned Queries: select teams.name, sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) as home_goals, sum(case when games.away_team_id = teams.id then games.goal_away else 0 end) as away_goals, count(*) as matches_played, (sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) + sum(case when games.away_team_id = teams.id then games.goal_away else 0 end)) * 1.0 / count(*) as avg_goals_per_match from games inner join teams on teams.id = games.home_team_id or teams.id = games.away_team_id where games.season = '2016-2017' group by teams.name order by (home_goals + away_goals) desc limit 3;); select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and pel.label = 'goal'

Correct: Yes

Input 52: How do I find the personal and national information of a player with hash lpabyhMe?

Expected Query: SELECT players.name, players.country FROM players WHERE players.hash = 'hash';

Returned Queries: select * from players where name = 'name', select players.name, players.country from players where players.hash = 'hash'

Correct: Yes

Input 53: What were the line-ups for the home team in game 21, specifically noting the team captain?

Expected Query: SELECT t.name AS team_name, p.name || CASE WHEN l.captain THEN ' (C)' ELSE '' END AS player_name, l.shirt_number FROM games g JOIN teams t ON g.home_team_id = t.id JOIN game_lineup l ON t.id = l.team_id AND l.game_id = g.id JOIN players p ON l.player_id = p.hash WHERE g.id = X;

Returned Queries: select home_team.name as home_team, away_team.name as away_team from games join teams as home_team on games.home_team_id = home_team.id join teams as away_team on games.away_team_id = away_team.id where games.id = x, select t.name as team_name, p.name || case when l.captain then ' (c)' else '' end as player_name, l.shirt_number from games g join teams t on g.home_team_id = t.id join game_lineup l on t.id = l.team_id and l.game_id = g.id join players p on l.player_id = p.hash where g.id = x

Correct: Yes

Input 54: Could you tell me who were the home and away teams for the specific match identified by 52?

Expected Query: SELECT home_team.name AS home_team, away_team.name AS away_team FROM games JOIN teams AS home_team ON games.home_team_id = home_team.id JOIN teams AS away_team ON games.away_team_id = away_team.id WHERE games.id = X;

Returned Queries: select home_team.name as home_team, away_team.name as away_team from games join teams as home_team on games.home_team_id = home_team.id join teams as away_team on games.away_team_id = away_team.id where games.id = x, select t.name as team_name, p.name || case when l.captain then ' (c)' else '' end as player_name, l.shirt_number from games g join teams t on g.home_team_id = t.id join game_lineup l on t.id = l.team_id and l.game_id = g.id join players p on l.player_id = p.hash where g.id = x

Correct: Yes

Input 55: In game 34, I want to see a breakdown of goals and shots on target for each team per period.

Expected Query: SELECT t.name AS team_name, a.period, SUM(CASE WHEN a.label = 'Shots on target' THEN 1 ELSE 0 END) AS shots_on_target, SUM(CASE WHEN a.label = 'Goal' THEN 1 ELSE 0 END) AS goals FROM annotations a JOIN teams t ON a.team_id = t.id WHERE a.game_id = X AND (a.label = 'Shots on target' OR a.label = 'Goal') GROUP BY t.name, a.period ORDER BY t.name, a.period;

Returned Queries: select t.name as team_name, a.period, sum(case when a.label = 'shots on target' then 1 else 0 end) as shots_on_target, sum(case when a.label = 'goal' then 1 else 0 end) as goals from annotations a join teams t on a.team_id = t.id where a.game_id = x and (a.label = 'shots on target' or a.label = 'goal') group by t.name, a.period order by t.name, a.period, select teams.name, sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) as home_goals, sum(case when games.away_team_id = teams.id then games.goal_away else 0 end) as away_goals, count(*) as matches_played, (sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) + sum(case when games.away_team_id = teams.id then games.goal_away else 0 end)) * 1.0 / count(*) as avg_goals_per_match from games inner join teams on teams.id = games.home_team_id or teams.id = games.away_team_id where games.season = '2016-2017' group by teams.name order by (home_goals + away_goals) desc limit 3;)

Correct: Yes

Input 56: Need the count and timing of offsides by the away team in game 2.

Expected Query: SELECT a.game_id, a.label, a.position, a.period FROM annotations a JOIN games g ON a.game_id = g.id WHERE a.game_id = X AND a.label = 'Offside' AND a.team_id = g.away_team_id;

Returned Queries: select a.game_id, a.label, a.position, a.period from annotations a join games g on a.game_id = g.id where a.game_id = x and a.label = 'offside' and a.team_id = g.away_team_id, select home_team.name as home_team, away_team.name as away_team from games join teams as home_team on games.home_team_id = home_team.id join teams as away_team on games.away_team_id = away_team.id where games.id = x

Correct: Yes

Input 57: Show me how many goals a Arsenal scored in a 2014-15 season.

Expected Query: SELECT t.name AS TeamName, g.season, SUM(CASE WHEN g.home_team_id = t.id THEN g.goal_home ELSE 0 END + CASE WHEN g.away_team_id = t.id THEN g.goal_away ELSE 0 END) AS TotalGoals FROM games g JOIN teams t ON g.home_team_id = t.id OR g.away_team_id = t.id WHERE t.name = '<team>' AND g.season = '<season>' GROUP BY t.name, g.season;

Returned Queries: select count(*) as goal_count from player_events pe join players p on pe.player_id = p.hash join games g on pe.game_id = g.id join player_event_labels pel on pe.type = pel.id where p.name = <player>and g.season = <season>and pel.label = 'goal', select teams.name, sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) as home_goals, sum(case when games.away_team_id = teams.id then games.goal_away else 0 end) as away_goals, count(*) as matches_played, (sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) + sum(case when games.away_team_id = teams.id then games.goal_away else 0 end)) * 1.0 / count(*) as avg_goals_per_match from games inner join teams on teams.id = games.home_team_id or teams.id = games.away_team_id where games.season = '2016-2017' group by teams.name order by (home_goals + away_goals) desc limit 3;)

Correct: No

Input 58: In game 5, who was the home team?

Expected Query: SELECT home_team.name AS home_team, away_team.name AS away_team FROM games JOIN teams AS home_team ON games.home_team_id = home_team.id JOIN teams AS away_team ON games.away_team_id = away_team.id WHERE games.id = X;

Returned Queries: select home_team.name as home_team, away_team.name as away_team from games join teams as home_team on games.home_team_id = home_team.id join teams as away_team on games.away_team_id = away_team.id where games.id = x, select t.name as team_name, p.name || case when l.captain then ' (c)' else '' end as player_name, l.shirt_number from games g join teams t on g.home_team_id = t.id join game_lineup l on t.id = l.team_id and l.game_id = g.id join players p on l.player_id = p.hash where g.id = x

Correct: Yes

Input 59: List all goals scored by Dortmund in 2014-2015

Expected Query: SELECT t.name AS TeamName, g.season, SUM(CASE WHEN g.home_team_id = t.id THEN g.goal_home ELSE 0 END + CASE WHEN g.away_team_id = t.id THEN g.goal_away ELSE 0 END) AS TotalGoals FROM games g JOIN teams t ON g.home_team_id = t.id OR g.away_team_id = t.id WHERE t.name = '<team>' AND g.season = '<season>' GROUP BY t.name, g.season;

Returned Queries: select t.name as teamname, g.season, sum(case when g.home_team_id = t.id then g.goal_home else 0 end + case when g.away_team_id = t.id then g.goal_away else 0 end) as totalgoals from games g join teams t on g.home_team_id = t.id or g.away_team_id = t.id where t.name = '<team>' and g.season = '<season>' group by t.name, g.season, select teams.name, sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) as home_goals, sum(case when games.away_team_id = teams.id then games.goal_away else 0 end) as away_goals, count(*) as matches_played, (sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) + sum(case when games.away_team_id = teams.id then games.goal_away else 0 end)) * 1.0 / count(*) as avg_goals_per_match from games inner join teams on teams.id = games.home_team_id or teams.id = games.away_team_id where games.season = '2016-2017' group by teams.name order by (home_goals + away_goals) desc limit 3;)

Correct: Yes

Input 60: For game 56, give the number of shots on target and goals for each team in each period.

Expected Query: SELECT t.name AS team_name, a.period, SUM(CASE WHEN a.label = 'Shots on target' THEN 1 ELSE 0 END) AS shots_on_target, SUM(CASE WHEN a.label = 'Goal' THEN 1 ELSE 0 END) AS goals FROM annotations a JOIN teams t ON a.team_id = t.id WHERE a.game_id = X AND (a.label = 'Shots on target' OR a.label = 'Goal') GROUP BY t.name, a.period ORDER BY t.name, a.period;

Returned Queries: select t.name as team_name, a.period, sum(case when a.label = 'shots on target' then 1 else 0 end) as shots_on_target, sum(case when a.label = 'goal' then 1 else 0 end) as goals from annotations a join teams t on a.team_id = t.id where a.game_id = x and (a.label = 'shots on target' or a.label = 'goal') group by t.name, a.period order by t.name, a.period, select teams.name, sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) as home_goals, sum(case when games.away_team_id = teams.id then games.goal_away else 0 end) as away_goals, count(*) as matches_played, (sum(case when games.home_team_id = teams.id then games.goal_home else 0 end) + sum(case when games.away_team_id = teams.id then games.goal_away else 0 end)) * 1.0 / count(*) as avg_goals_per_match from games inner join teams on teams.id = games.home_team_id or teams.id = games.away_team_id where games.season = '2016-2017' group by teams.name order by (home_goals + away_goals) desc limit 3;)

Correct: Yes

Appendix B

Publications

During the work with this thesis, we submitted two papers for publication to CBMI 2024 21st International Conference on Content-Based Multimedia Indexing¹

¹<https://cbmi2024.org/>

B.1 SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries

Authors Strand, Aleksander Theo; Gautam, Sushant; Midoglu, Cise; Halvorsen, Pål

Venue

CBMI 2024 21st International Conference on Content-Based Multimedia Indexing
September 18-20, 2024 | Reykjavik, Iceland

Status Under review

Abstract The rapid evolution of digital sports media necessitates sophisticated information retrieval systems that can efficiently parse extensive multimodal datasets. This paper introduces SoccerRAG, an innovative framework designed to harness the power of Retrieval Augmented Generation (RAG) and Large Language Models (LLMs) to extract soccer-related information through natural language queries. By leveraging a multimodal dataset, SoccerRAG supports dynamic querying and automatic data validation, enhancing user interaction and accessibility to sports archives. Our evaluations indicate that SoccerRAG effectively handles complex queries, offering significant improvements over traditional retrieval systems in terms of accuracy and user engagement. The results underscore the potential of using RAG and LLMs in sports analytics, paving the way for future advancements in the accessibility and real-time processing of sports data.

SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries

Aleksander Theo Strand
OsloMet
Oslo, Norway
0009-0008-2749-2347

Sushant Gautam
OsloMet, SimulaMet
Oslo, Norway
0000-0001-9232-2661

Cise Midoglu
SimulaMet, Forzasys
Oslo, Norway
0000-0003-0991-4418

Pål Halvorsen
OsloMet, SimulaMet, Forzasys
Oslo, Norway
0000-0003-2073-7029

Abstract—The rapid evolution of digital sports media necessitates sophisticated information retrieval systems that can efficiently parse extensive multimodal datasets. This paper introduces SoccerRAG, an innovative framework designed to harness the power of Retrieval Augmented Generation (RAG) and Large Language Models (LLMs) to extract soccer-related information through natural language queries. By leveraging a multimodal dataset, SoccerRAG supports dynamic querying and automatic data validation, enhancing user interaction and accessibility to sports archives. Our evaluations indicate that SoccerRAG effectively handles complex queries, offering significant improvements over traditional retrieval systems in terms of accuracy and user engagement. The results underscore the potential of using RAG and LLMs in sports analytics, paving the way for future advancements in the accessibility and real-time processing of sports data.

Index Terms—association football, information retrieval, large language models, multimodal data fusion, natural language processing, sports

I. INTRODUCTION

The rapid growth of digital sports content has created a demand for efficient retrieval systems that can understand and process natural language queries [1], [2]. The motivation behind our research is to address the challenges in retrieving specific content from extensive sports libraries using intuitive, natural language requests. This approach simplifies user interaction and enhances the accessibility of sports archives. Soccer, as one of the most popular sports worldwide, serves as a prominent use case and context for our proof-of-concept retrieval application.

We propose SoccerRAG, a framework for retrieving multimodal soccer information using natural language queries, from an augmented soccer dataset based on SoccerNet [3], [4], which includes game videos with image frames and audio, timestamped captions (transcribed audio), annotations for game events, and player information. In short:

- We highlight the opportunities presented by the rapid advancements in Large Language Models (LLMs) and Retrieval Augmented Generation (RAG), and motivate the use of these technologies in the context of sports and more specifically for soccer analytics (Section II).
- We propose a framework concept and design for the retrieval of multimodal soccer information through natural queries, called SoccerRAG, which integrates components for data representation, feature extraction and validation,

and database querying. Our conceptualization includes a proposed database schema for an augmented version of the SoccerNet dataset, as well as a novel extractor-validator chain (Section III).

- We provide an open source implementation for SoccerRAG which is accessible under [5] and fully reproducible through the instructions provided therein.
- We present a preliminary evaluation of the SoccerRAG framework through the analysis of the extractor-validator chain, query complexity, individual component contributions, and execution time, for different LLMs (Section IV), and discuss our insights (Section V).

II. BACKGROUND AND RELATED WORK

LLMs have revolutionized the field of Natural Language Processing (NLP) and generative AI [6]. The evolution of LLMs, from early rule-based models to contemporary GPT iterations, exemplifies significant technological advancements [7]. The capabilities of these models extend beyond text generation to include reasoning, decision-making, and multimodal data fusion, addressing both the static nature of early models and the computational demands of training and fine-tuning [8].

A. Multimodal Soccer Understanding

Soccer, a globally beloved sport, serves as a rich domain for analytical exploration. The multifaceted aspects of soccer analytics are explored through the SoccerNet dataset and challenges [3], [4], a comprehensive repository of soccer broadcast recordings and various annotations, facilitating research in areas such as action spotting, video captioning, and game state reconstruction [9]. Midoglu et al. [10] focused on automatic event clipping, thumbnail selection, and game summarization through AI, to streamline the production of engaging soccer game highlights and summaries. There are interesting lines of research in soccer game understanding around event detection [11], [12], automatic game summarization [13], highlight generation [14], caption generation [15], [16], and similar. Most of this research demonstrates that the use of multiple modalities (including but not limited to videos, images, text, audio, and structured metadata such as commentary text, event information, and team statistics) can enhance game understanding [1], [2].

B. Information Retrieval (IR)

Information Retrieval (IR) is focused on the extraction of relevant information from vast datasets based on user queries [17]. It encompasses a range of techniques including indexing, search algorithms, and natural language processing, to organize and retrieve unstructured data efficiently [18]. Recent advancements have leveraged machine learning and AI to enhance the precision of search outcomes through a better understanding of user intent and document semantics [19]. Despite its progress, IR continues to face challenges such as data volume management and privacy concerns, highlighting the need for innovative solutions in data retrieval [20].

C. Retrieval Augmented Generation (RAG)

RAG systems represent a paradigm shift in open-domain question answering, combining document retrieval with generative modeling to produce contextually relevant answers [21]. The utility of vector databases in RAG systems, particularly in efficiently managing and retrieving high-dimensional vector representations of text, underscores their significance in enhancing the accuracy and relevance of generated responses [22]. RAG has found application in different domains including enterprise contexts where LLMs are integrated with chatbots, enabling them to automatically derive more accurate answers from company documents and knowledge bases [23].

D. Terminology

LLMs are advanced deep learning models capable of understanding and generating human-like text [24]. Our research predominantly focuses on OpenAI's closed-source GPT-3.5-Turbo [25] and GPT-4.0-Turbo [26], although we acknowledge the broader landscape of available LLMs. **Tokens** are the atomic elements used by LLMs for text generation and prediction. Tokens can represent varying levels of linguistic units, from single characters to multiple words, crucial for modeling and API cost calculations [27]. Other concepts that are integral to LLM frameworks include **chains** which are sequences of tasks executed in a predefined order [28], and **agents** which are dynamic entities that use LLMs for decision-making and executing actions based on reasoning [29]. **Vector databases** handle high-dimensional data, such as text and image embeddings [30], enabling quick similarity searches [31] through Approximate Nearest Neighbor (ANN) algorithms. They are crucial for integrating with LLMs across various industries, allowing for complex queries over extensive datasets, offering a faster alternative to traditional database search methods [32]. **Extractors** are specialized mechanisms that harness the capabilities of LLMs for efficient information extraction, and are traditionally manual, rule-based processes [33]. They adapt to tasks via instructions or emulate JSON for structured output [34]. They offer tool calling for schema structuring, JSON mode for structured output, and prompting-based extraction for versatile retrieval, streamlining the costly traditional systems [35].

E. Novelty

The burgeoning interest in RAG, fueled by the rapid advancements in LLMs, has paved new avenues for exploring innovative use cases across various domains, notably within multimodal information retrieval [36]. Despite the wide applicability of RAG frameworks in enhancing the capabilities of generative AI for open-domain question answering and beyond, its potential in the sports domain, particularly within soccer analytics, remains largely untapped. Our research endeavors to bridge this gap by harnessing the multimodal datasets inherent to the soccer broadcast pipeline. By integrating RAG with multiple data modalities, we aim to develop a sophisticated system capable of understanding complex game scenarios and responding to user queries in a manner that resonates with the intricate language of soccer. The potential of our proposed system for video retrieval offers a promising avenue for users and broadcasters alike, enabling the recollection of game moments through natural queries.

III. PROPOSED FRAMEWORK

Figure 1 presents an overview of our proposed SoccerRAG framework, which includes 4 main components: **database**, **feature extractor**, **feature validator**, and **SQL agent**. The application flow is as follows: (1) The user provides a natural language query related to the contents of the database. (2) The user input is sent to an LLM along with the properties schema and a system prompt describing the properties the LLM should extract from the question. The LLM then returns a list of extracted properties relevant to the query. (3) Each extracted feature is checked against the appropriate table in the database using string matching algorithms. This step aims to correct spelling mistakes and abbreviations. Once a value is found, both the value and its primary key are added to the extracted value. (4) The cleaned user prompt is combined with system-specific prompts to guide the LLM in generating SQL queries that will answer the user's question. The constructed query is then passed to the SQL chain, which designs and executes the SQL queries against the underlying database. The SQL chain handles communication between the system and the database, retrieves the requested data, and prepares the results for

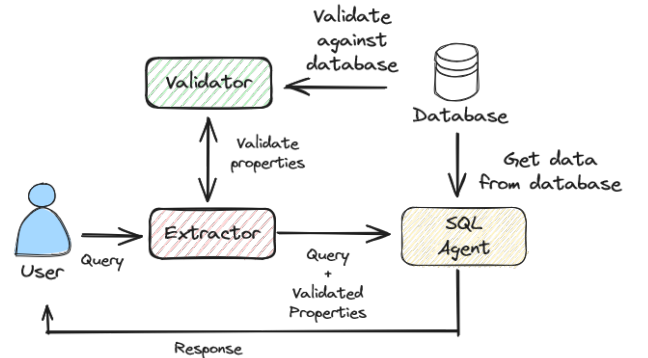


Fig. 1. Overview of the proposed SoccerRAG framework.

presentation to the user. The complete SoccerRAG framework can be replicated in an end-to-end manner by running our open source software, which is accessible under [5], through the instructions provided.

A. Database

The SoccerNet dataset consists of game broadcast videos and metadata information for 550 soccer games, from a number of top European leagues across multiple seasons. We have run Automatic Speech Recognition (ASR) on all games with available audio using Open AI’s Whisper [37] to transcribe the audio, converting spoken commentary into textual data. Overall, for each game, we have files containing transcribed commentary (*<1/2>_half-ASR.json*), annotations describing key events during the game such as goals, cards, fouls, etc. (*Labels-v2.json*), general game information such as home and away teams, lineups, score, referee, and other details (*Labels-caption.json*), as shown in Figure 2. In the context of this work, an **event** refers to in-game events, **commentary** refers to the transcribed commentary from the game broadcast, and **caption** refers to a short description of an event.

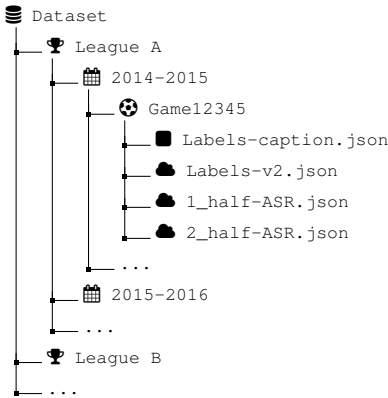


Fig. 2. Dataset structure for the enhanced SoccerNet dataset used.

To make this diverse dataset usable for an SQL agent, we converted the JSON files into a structured SQLite database (detailed in Figure 3) with a script utilizing SQLAlchemy [5]. This script creates tables and columns based on a predefined schema, follows a systematic process of extracting league/season details, game details, lineups, events, commentary, and creates links between related entities. Clear and descriptive naming conventions for tables/columns were iteratively refined to improve the LLM’s understanding of the data structure.

B. Feature Extractor

In the extractor chain, instructions are provided to the LLM to return a JSON structure, where pre-defined properties (defined in the *schema.json* file under [5]) are extracted from the given prompt. We use the Langchain framework for its easy adjustability to new tasks. The schema also includes a list of the tables and columns a property exists in in the database, as a preparation for the validation function.

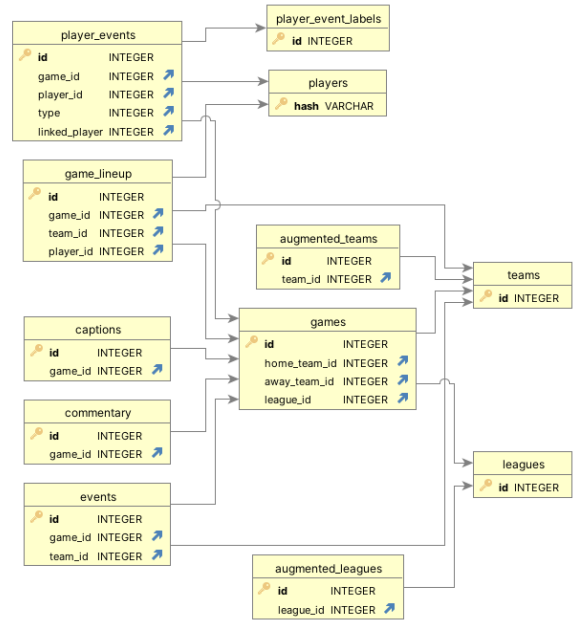


Fig. 3. Database schema of the structured SQLite database.

C. Feature Validator

Extracted features can have spelling errors or include abbreviations, rendering them disparate from the representations stored in the database. This discrepancy necessitates a validation procedure. To address the challenge of abbreviations (e.g., the natural query referring to "ManU" in reference to Manchester United F.C.), we manually constructed auxiliary tables within the database and populated them with abbreviations pertaining to common soccer team and player nomenclature (*augmented_<teams/leagues>.csv* files under [5]). These supplementary tables are also delineated in the extractor schema for properties that warrant their employment.

The validation process commences by selecting the initial property key, which comprises a list of the extracted items for that property. If the property is associated with an auxiliary table, a search for a matching entry is performed within that table. In the event that no match is found, the system retrieves all items in the table linked to the property and computes the Levenshtein Distance [38] to ascertain the closest match for each item. Utilizing a predefined threshold for closeness, the system attempts to automatically select the optimal match, but may solicit user input for clarification in cases of ambiguity. Upon the validation of all items, the original prompt is returned with updated information containing the correct item designation, and where feasible, the corresponding foreign key in the database.

D. SQL Agent

The final component of the SoccerRAG framework leverages the SQL agent from the LangChain framework [39]. To enhance the capabilities of the agent, we implemented a few-shot SQL RAG solution. By employing this approach, the

system prompt for the agent is consistently accompanied by N number of queries that are pertinent to the user’s question, where N is externally configurable. To identify the matching queries, we employ vector search facilitated by a FIASS database. This database is populated with data originating from a JSON file (*sqli.json* under [5]), which contains a collection of human crafted SQL queries. The vector search technique entails embedding both the user’s question and the SQL queries into high-dimensional vector spaces, thereby enabling the identification of semantically similar queries through the calculation of vector similarities.

By providing the agent with a few-shot context consisting of relevant SQL queries, we provide the user of the system a way of telling the system what data is expected to be retrieved for different questions. This approach leverages the strengths of both retrieval and generation components, harnessing the domain knowledge encoded within the SQL query repository while enabling the agent to adapt and construct tailored queries based on the specific requirements of the user’s input.

E. Choosing an LLM

LLMs are used in the feature extractor and SQL agent components of the SoccerRAG framework. In the LLM field, new open-source models such as LLaMA 2 [40], [41] and Mistral-7B [42] are rapidly emerging, but lack advanced function calling capabilities required by LangChain. While fine-tuning a model for feature extraction is viable, it demands substantial data and is time-consuming. Therefore, we decided on using OpenAI’s GPT-4.0-Turbo [26] and GPT-3.5-Turbo [25] models.

IV. EVALUATION

We use the following set of example questions throughout our experiments to evaluate SoccerRAG and its components:

- **Question 1:** Is Manchester United in the database?
- **Question 2:** Give me the total home goals for Bayern M in the 2014-15 season
- **Question 3:** Calculate home advantage for Real Madrid in the 2015-16 season
- **Question 4:** How many goals did Messi score in the 15-16 season?
- **Question 5:** How many yellow-cards did Enzo Perez get in the 15-2016 season?
- **Question 6:** List all teams that played a game against Napoli in 2016-17 season in seriea? Do not limit the number of results
- **Question 7:** Give all the teams in the league ucl in the 2015-2016 season?
- **Question 8:** Give me all games in epl with yellow cards in the first half in the 2015-2016 season
- **Question 9:** What teams and leagues has Adnan Januzaj play in?
- **Question 10:** List ALL players that started a game for Las Palmas in the 2016-2017 season? Do NOT limit the number of results
- **Question 11:** Did Ajax or Manchester United win the most games in the 2014-15 season?
- **Question 12:** How many yellow and red cards were given in the UEFA Champions League in the 2015-2016 season?
- **Question 13:** Are Messi and C. Ronaldo in the database?
- **Question 14:** How many goals did E. Hazard score in the game between Bournemouth and Chelsea in the 2015-2016 season?
- **Question 15:** How many yellow cards were given in the game between Bayern Munich and Shakhtar Donetsk in the 2014-15 UEFA Champions League, and did anyone receive a red card?
- **Question 16:** Make a list of when corners happened in the English Premier League (EPL) 2015-2016 season. Aggregate by a period of 15 minutes.
- **Question 17:** What league is Manchester United, Arsenal, Bournemouth, Real Madrid, Chelsea, and Liverpool in?
- **Question 18:** How many players have "Aleksandar" as their first name in the database, and how many goals have they scored in total?
- **Question 19:** What did the commentary say about the game between Arsenal and Southampton in the 2016-17 season?
- **Question 20:** Have Mesut Ozil, Pablo Insuia, or Alex Pike played for West Ham or Barcelona?

A. Extractor-Validator Evaluation

The extractor-validator chain was evaluated on questions 1-20 subjectively, with the following evaluation criteria: (1) Were all properties extracted? (2) Were extra properties extracted? (3) Could extracted properties be validated? (4) Did the chain need human input? A perfect score (indicated with ● in Table I) was given if all properties were extracted and validated. If any properties were missed or validation failed, the result was marked as failed (▲). Extracting extra information or needing human feedback resulted in a 50% score (●).

GPT-3.5-Turbo was used for all questions except 15 and 18, which were re-run with GPT-4.0-Turbo. The results in Table I show optimal performance for questions 2-10 and most of 11-20. In question 1, an extra item was extracted, without disrupting the rest of the chain. GPT-3.5-Turbo failed to extract one property in question 15, which GPT-4.0-Turbo handled correctly, this is demonstrated by presenting two results for Question 15. In Table I Question 18, which was a question designed to confuse the extractor by asking about players named "Aleksandar". The extractor extracts the name, and then tries to validate it against a *single* player. This shows that the chain does not understand context, as opposed to the SQL agent which is more content aware.

B. Query Complexity Analysis

To investigate the difficulty of composing database queries to represent natural language queries, query complexity scores were calculated for questions 1-10 using the Halstead metrics [43]. For each of the 10 questions, a database query was manually composed, and the number of unique operators (e.g. "SELECT", "FROM", "=") and operands (e.g. table and column names) were counted, as well as the total occurrences of operators and operands across the query. Halstead metrics were then calculated based on these counts. The results (presented in the middle part of Table I) show that question 3 had the highest complexity score. Further analysis revealed that this was due to the query for question 3 performing several *SUM* operations.

C. Ablation Study

By testing the framework components under different configurations, it is possible to quantify the contribution of each component to the final SoccerRAG framework. This information is crucial for understanding the strengths and weaknesses of the various parts of the system, and identifying areas for potential improvement. We performed an extensive ablation study using the following pipeline configurations, where (6) corresponds to the full SoccerRAG framework.

- 1) **SQL agent only:** Sending the prompt directly to the SQL agent without RAG
- 2) **SQL agent with RAG:** Sending the prompt directly to the SQL agent with RAG
- 3) **Extractor only:** Sending the prompt with extracted values to the SQL agent
- 4) **Extractor and SQL RAG:** Sending the prompt with extracted values to the SQL agent, the SQL agent uses RAG to get examples
- 5) **Extractor and validator:** Sending the prompt with extracted and validated values to the SQL agent
- 6) **Extractor, validator, and SQL RAG:** Sending the prompt with extracted and validated values to the SQL agent, the SQL agent uses RAG to get examples

TABLE I

FRAMEWORK EVALUATION: EXTRACTOR-VALIDATOR EVALUATION (TOP), QUERY COMPLEXITY ANALYSIS (MIDDLE), ABLATION STUDY (BOTTOM).

Module Evaluation		Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Extractor		●	●	●	●	●	●	●	●	●	●
Validator		●	●	●	●	●	●	●	●	●	●
Overall		●	●	●	●	●	●	●	●	●	●
		Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
Extractor		●	●	●	●	▲ / ●	●	●	■	●	●
Validator		●	●	●	●	● / ●	●	●	●	●	●
Overall		●	●	●	●	■ / ●	●	●	●	●	●
Query Complexity Metric		Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Number of distinct operators, n_1		5	6	8	8	8	10	10	9	8	9
Number of distinct operands, n_2		5	14	31	26	26	31	20	36	28	25
Total number of operators, N_1		5	11	33	18	18	22	16	27	19	19
Total number of operands, N_2		5	16	110	26	26	38	22	41	31	26
Vocabulary, n		10	20	39	34	34	41	30	45	36	34
Length, N		10	27	143	44	44	60	38	68	50	45
Volume, V		33.22	116.69	755.81	223.85	223.85	321.45	186.46	373.45	258.50	228.94
Difficulty, D		2.50	3.43	14.19	4.00	4.00	6.13	5.50	5.12	4.43	4.68
Effort, E		83.05	400.09	10727.66	895.39	895.39	1970.20	1025.54	1913.91	1144.77	1071.42
Time to understand (s), T		4.61	22.23	595.98	49.74	49.74	109.46	56.97	106.33	63.60	59.52
LLM	Pipeline	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
GPT-3.5-Turbo	1	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	2	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗
	3	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	4	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
	5	✗	✓	✓	✗	✗	✓	✗	✗	✗	✗
	6	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
GPT-4.0-Turbo	1	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗
	2	✓	✗	✗	✗	✗	✗	✓	✗	✓	✓
	3	✓	✓	✗	✗	✗	✗	✓	✗	✗	✓
	4	✓	✗	✗	✗	✗	✗	✓	✗	✗	✓
	5	✓	✗	✗	✗	✓	✗	✓	✗	✗	✓
	6	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓

For the ablation study, questions 1-10 were used. A correct answer resulted in a pass score (indicated with ✓ in Table I), while an incorrect or incomplete answer resulted in a fail score (✗). The answers generated by the prompts created during the query complexity analysis (Section IV-B) were used to determine the success rate. The experimental setup involved using both GPT-3.5-Turbo and GPT-4.0-Turbo models to gain insights into their respective performance and to investigate the difference in quality, speed and cost between the two models. For the SQL-RAG, the value of K was set to 2.

As shown in Table I, the standalone SQL agent struggled with both models. The results indicate that the extractor-validator chain is crucial for increasing the hit rate. Interestingly, both models exhibited similar performance when the full pipeline was applied. GPT-4.0-Turbo failed on question 7 due to model laziness, where it provided only a partial answer. For question 8, the query returned a substantial amount of text, causing the language model to refuse to provide a correct answer.

D. Execution Time

We investigated the execution time (total time from query until model output) for GPT-3.5-Turbo and GPT-4.0-Turbo. As the performance of cloud-based services might change based on global usage, we explored "peak" and "off-peak" periods separately (peak defined as 8AM-5PM EST and off-peak defined as 5PM-8AM EST). To observe and quantify the difference in execution time and response quality, we ran questions 1-10 consecutively, five times per batch, during peak and off-peak periods, with both models. Figure 4 presents our

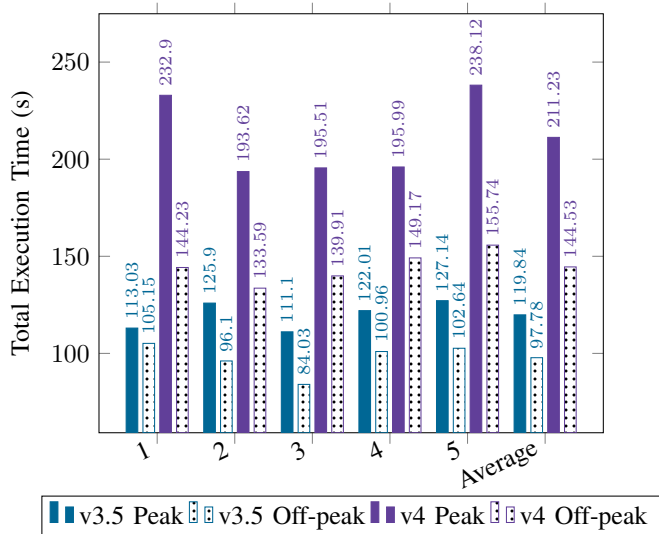


Fig. 4. Total execution time for questions 1-10.

findings, which indicate that peak usage is related to working hours in the US, and that performance degrades during peak usage periods for both models, although GPT-3.5-Turbo is significantly faster than GPT-4.0-Turbo overall.

V. DISCUSSION

A. Handling Input Errors

The extractor-validator chain has proven invaluable in mitigating spelling mistakes and other input errors and enhancing

the overall accuracy of the system. Its functions can range from simple substitutions (e.g., heterogeneous season indicators in questions 5 and 6 into the uniform yyyy-yyyy format, or "seriea" in question 6 to "Serie A", as expected by the database) to resolving more complex issues. In the examples below, we see that when the LLM doesn't manage to resolve keywords, it will in many cases produce a wrong output.

Prompt: Did Real or Man U win the most games in the 2015 season?
Output: Swansea won the most games in the 2014-2015 season with a total of 37 wins.
Cleaned prompt: Did Real or Man U win the most games in the 2015 season?
 Updated Information:

- Real (now referred to as Real Madrid) has a primary key: 30.
- Man U (now referred to as Manchester United) has a primary key: 7.
- 2015 (now referred to as the 2015-2016 season).

Output: In the 2015-2016 season, Real Madrid won 26 games, while Manchester United won 5 games. Therefore, Real Madrid won the most games in that season.

B. Retrieving Statistics

It is possible to extract detailed statistics using natural language queries with SoccerRAG, as demonstrated below.

Prompt: Calculate home advantage for Real Madrid in the 2015-16 season
Output: For Real Madrid in the 2015-2016 season, the home advantage can be calculated as follows:

- Total home goals scored: 55
- Total away goals scored: 39
- Number of home games played: 18
- Number of away games played: 17

This indicates that Real Madrid scored significantly more goals at home compared to away games, suggesting a strong home advantage during the 2015-2016 season.

However, LLMs make mistakes with complex mathematics or large lists. We found that asking for a list often led to only a subset being returned (e.g., question 8, Prompt A), whereas asking for a count of events provided the correct answer (Prompt B=modified version of question 8).

Prompt A: Give all the teams in the league ucl in the 2015-2016 season
Output: The teams in the UEFA Champions League (UCL) during the 2015-2016 season, limited to five examples, are:

- 1) Galatasaray
- 2) Atletico Madrid
- 3) Manchester City
- 4) Juventus
- 5) Paris Saint-Germain (Paris SG)

Prompt B: Count all games in epl with yellow cards in the first half in the 2015-2016 season
Output: There were 70 games in the English Premier League (england_epl) with yellow cards in the first half in the 2015-2016 season.

C. Influence of LLM

Execution time: We discovered that the OpenAI API works slower during peak periods, which follow the US timezone, with the execution time 22.56% slower for GPT-3.5-Turbo, and 46.5% slower for GPT-4.0-Turbo.

Robustness: Robustness refers to providing the same correct answer when presented with the same question multiple times. We did not observe significant degradation in the robustness of responses during peak periods. However, responses can exhibit variations across different runs, even for the same input. While strict prompting techniques can mitigate this inconsistency to some extent, the inherent stochastic nature of LLMs poses challenges in achieving complete determinism. To address this issue and gain greater control over the output,

a solution would be to explore the usage of self-hosted LLMs for the framework. Self-hosted LLMs also provide users with control over computational load, request prioritization, etc. that might influence the quality of the response.

Model laziness: The phenomenon where an LLM prematurely stops a task without fully completing it [44] can be manifested in two ways. *Model stopping before querying:* While attempting to answer a question, the framework makes multiple requests to the LLM. We observed instances where the framework prematurely halted during this reasoning phase, responding to the user with "I now have the information I need, and should query the database for the answer." This was addressed by including "I will not stop until I query the database and return the answer." in the system prompt for the GPT-3.5-Turbo model, which seemed to fix most issues. The GPT-4.0-Turbo model did not exhibit this problem. (2) *Model returning a subset:* Both models faced this issue when asked to return a list. Despite fetching the correct data, the LLMs would often limit their response to 5-10 items. Stricter prompting did not resolve this issue, which is acknowledged by OpenAI [45].

D. Future Work

SoccerRAG supports multimodal data integration by connecting metadata about game events with video timestamps, enabling several future applications. These include searching and retrieving specific video clips associated with particular game events using metadata, performing tasks such as object detection, action recognition, and scene understanding on video clips, and ultimately, automatically generating video highlights centered around game events of interest, or entire game summaries. SoccerRAG also faces several limitations that need addressing. First, the system struggles with complex queries and large data volumes. LLM model laziness might cause tasks to conclude prematurely or provide partial data, especially with extensive outputs. There is also a need to explore emerging LLMs, including open-source alternatives, for potential performance improvements with respect to function calling, language to query translation, and extraction abilities. Multimodal data synchronization and real-time processing capabilities require significant improvement to be viable for live sports analytics applications. Finally, ensuring privacy and ethical data handling is paramount, necessitating robust security measures and ethical guidelines in future developments.

VI. CONCLUSION

We introduce SoccerRAG, a framework that leverages RAG and LLMs to efficiently retrieve multimodal soccer information via natural language queries, enhancing the accessibility of vast sports datasets by allowing for intuitive user interactions with complex data archives. Ultimately, SoccerRAG exemplifies the transformative potential of AI in sports analytics, promising a future where technology enriches the global fan experience and advances the accessibility of sports information.

REFERENCES

- [1] S. Gautam *et al.*, “Assisting soccer game summarization via audio intensity analysis of game highlights,” in *Proc. 12th IOE Graduate Conference*. Inst. of Eng. Tribhuvan University, Nepal, Oct. 2022, pp. 25–32. [Online]. Available: <http://conference.ioe.edu.np/publications/ioegc12/IOEGC-12-004-12009.pdf>
- [2] S. Gautam, “Bridging Multimedia Modalities: Enhanced Multimodal AI Understanding and Intelligent Agents,” in *ICMI '23: Proceedings of the 25th International Conference on Multimodal Interaction*. New York, NY, USA: Association for Computing Machinery, Oct. 2023, pp. 695–699.
- [3] S. Giancola *et al.*, “SoccerNet: A Scalable Dataset for Action Spotting in Soccer Videos,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2018, pp. 18–22.
- [4] A. Delière *et al.*, “SoccerNet-v2: A Dataset and Benchmarks for Holistic Understanding of Broadcast Soccer Videos,” in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2021, pp. 19–25.
- [5] “SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries,” Apr. 2024, [Online; accessed 12. Apr. 2024]. [Online]. Available: <https://github.com/simula/soccer-rag>
- [6] J. Yang *et al.*, “Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond,” *ACM Trans. Knowl. Discovery Data*, Jun. 2023.
- [7] D. Myers *et al.*, “Foundation and large language models: fundamentals, challenges, opportunities, and social impacts,” *Cluster Comput.*, vol. 27, no. 1, pp. 1–26, Feb. 2024.
- [8] Z. Xi *et al.*, “The Rise and Potential of Large Language Model Based Agents: A Survey,” *arXiv*, Sep. 2023.
- [9] S. Akan and S. Varli, “Use of deep learning in soccer videos analysis: survey,” *Multimedia Systems*, vol. 29, no. 3, pp. 897–915, Jun. 2023.
- [10] C. Midoglu *et al.*, “MMSys’22 Grand Challenge on AI-based Video Production for Soccer,” *arXiv*, Feb. 2022.
- [11] L. Morra *et al.*, “Slicing and Dicing Soccer: Automatic Detection of Complex Events from Spatio-Temporal Data,” in *Image Analysis and Recognition*. Cham, Switzerland: Springer, Jun. 2020, pp. 107–121.
- [12] O. A. N. Rongved *et al.*, “Real-Time Detection of Events in Soccer Videos using 3D Convolutional Neural Networks,” in *IEEE International Symposium on Multimedia (ISM)*. IEEE, 2020, pp. 02–04.
- [13] S. Gautam *et al.*, “Soccer Game Summarization using Audio Commentary, Metadata, and Captions,” in *NarSUM '22: Proceedings of the 1st Workshop on User-centric Narrative Summarization of Long Videos*. New York, NY, USA: Association for Computing Machinery, Oct. 2022, pp. 13–22.
- [14] C. Midoglu *et al.*, “AI-Based Sports Highlight Generation for Social Media,” in *MHV '24: Proceedings of the 3rd Mile-High Video Conference*. New York, NY, USA: Association for Computing Machinery, Feb. 2024, pp. 7–13.
- [15] H. Mkhallati *et al.*, “SoccerNet-Caption: Dense Video Captioning for Soccer Broadcasts Commentaries,” in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2023, pp. 17–24.
- [16] J. Qi *et al.*, “GOAL: A Challenging Knowledge-grounded Video Captioning Benchmark for Real-time Soccer Commentary Generation,” in *CIKM '23: Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. New York, NY, USA: Association for Computing Machinery, Oct. 2023, pp. 5391–5395.
- [17] G. G. Chowdhury, *Introduction to modern information retrieval*. Facet publishing, 2010.
- [18] K. Boukhari and M. N. Omri, “DL-VSM based document indexing approach for information retrieval,” *J. Ambient Intell. Hum. Comput.*, vol. 14, no. 5, pp. 5383–5394, May 2023.
- [19] J. Guo *et al.*, “A Deep Look into neural ranking models for information retrieval,” *Information Processing & Management*, vol. 57, no. 6, p. 102067, Nov. 2020.
- [20] A. Olteanu *et al.*, “FACTS-IR: fairness, accountability, confidentiality, transparency, and safety in information retrieval,” *SIGIR Forum*, vol. 53, no. 2, pp. 20–43, Mar. 2021.
- [21] L. Liu *et al.*, “Challenges in Generalization in Open Domain Question Answering,” *ACL Anthology*, pp. 2014–2029, Jul. 2022.
- [22] W. Yu *et al.*, “A Survey of Knowledge-enhanced Text Generation,” *ACM Comput. Surv.*, vol. 54, no. 11s, pp. 1–38, Nov. 2022.
- [23] C. Jeong, “A Study on the Implementation of Generative AI Services Using an Enterprise Data-Based LLM Application Architecture,” *arXiv*, Sep. 2023.
- [24] W. X. Zhao *et al.*, “A Survey of Large Language Models,” *arXiv*, Mar. 2023.
- [25] “GPT-3.5-Turbo,” Apr. 2024, [Online; accessed 12. Apr. 2024]. [Online]. Available: <https://platform.openai.com/docs/models/gpt-3-5-turbo>
- [26] “GPT-4,” Apr. 2024, [Online; accessed 12. Apr. 2024]. [Online]. Available: <https://openai.com/gpt-4>
- [27] A. Vaswani *et al.*, “Attention is all you need,” in *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., Dec. 2017, pp. 6000–6010.
- [28] T. Wu, M. Terry, and C. J. Cai, “AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts,” in *CHI '22: Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, Apr. 2022, pp. 1–22.
- [29] X. Liu *et al.*, “AgentBench: Evaluating LLMs as Agents,” *arXiv*, Aug. 2023.
- [30] S. Ayesha, M. K. Hanif, and R. Talib, “Overview and comparative study of dimensionality reduction techniques for high dimensional data,” *Information Fusion*, vol. 59, pp. 44–58, Jul. 2020.
- [31] K. Echihibi, K. Zoumpatianos, and T. Palpanas, “New trends in high-D vector similarity search: al-driven, progressive, and distributed,” *Proc. VLDB Endow.*, vol. 14, no. 12, pp. 3198–3201, Jul. 2021.
- [32] L. Friedman *et al.*, “Leveraging Large Language Models in Conversational Recommender Systems,” *arXiv*, May 2023.
- [33] D. Xu *et al.*, “Large Language Models for Generative Information Extraction: A Survey,” *arXiv*, Dec. 2023.
- [34] K. Zhang, B. J. Gutiérrez, and Y. Su, “Aligning Instruction Tasks Unlocks Large Language Models as Zero-Shot Relation Extractors,” *arXiv*, May 2023.
- [35] “Extracting structured output | LangChain,” Apr. 2024, [Online; accessed 8. Apr. 2024]. [Online]. Available: https://python.langchain.com/docs/use_cases/extraction
- [36] Y. Cheng *et al.*, “Exploring Large Language Model based Intelligent Agents: Definitions, Methods, and Prospects,” *arXiv*, Jan. 2024.
- [37] A. Radford *et al.*, “Robust speech recognition via large-scale weak supervision,” 2022.
- [38] V. I. Levenshtein *et al.*, “Binary codes capable of correcting deletions, insertions, and reversals,” in *Soviet physics doklady*, vol. 10, no. 8. Soviet Union, 1966, pp. 707–710.
- [39] “Agents | LangChain,” Apr. 2024, [Online; accessed 10. Apr. 2024]. [Online]. Available: https://python.langchain.com/docs/use_cases/sql/agents
- [40] H. Touvron *et al.*, “Llama 2: Open foundation and fine-tuned chat models,” 2023.
- [41] “Llama 2,” Apr. 2024, [Online; accessed 12. Apr. 2024]. [Online]. Available: <https://llama.meta.com/llama2/>
- [42] A. Q. Jiang *et al.*, “Mistral 7b,” 2023.
- [43] M. H. Halstead, *Elements of Software Science (Operating and programming systems series) | Guide books*. Oxford, England, UK: Elsevier Science Inc., 1977. [Online]. Available: <https://dl.acm.org/doi/abs/10.5555/540137>
- [44] J. Wei *et al.*, “Measuring and Reducing LLM Hallucination without Gold-Standard Answers via Expertise-Weighting,” *arXiv*, Feb. 2024.
- [45] D. Coldewey, “OpenAI drops prices and fixes ‘lazy’ GPT-4 that refused to work | TechCrunch,” *TechCrunch*, Jan. 2024. [Online]. Available: <https://techcrunch.com/2024/01/25/openai-drops-prices-and-fixes-lazy-gpt-4-that-refused-to-work/>

B.2 Demo: Soccer Information Retrieval via Natural Queries using SoccerRAG

Authors Strand, Aleksander Theo; Gautam, Sushant; Midoglu, Cise; Halvorsen, Pål

Venue

CBMI 2024 21st International Conference on Content-Based Multimedia Indexing
September 18-20, 2024 | Reykjavik, Iceland

Status Under review

Abstract The rapid evolution of digital sports media necessitates sophisticated information retrieval systems that can efficiently parse extensive multimodal datasets. This paper demonstrates SoccerRAG, an innovative framework designed to harness the power of Retrieval Augmented Generation (RAG) and Large Language Models (LLMs) to extract soccer-related information through natural language queries. By leveraging a multimodal dataset, SoccerRAG supports dynamic querying and automatic data validation, enhancing user interaction and accessibility to sports archives. We present a novel interactive user interface (UI) based on the Chainlit framework which wraps around the core functionality, and enable users to interact with the SoccerRAG framework in a chatbot-like visual manner.

Demo: Soccer Information Retrieval via Natural Queries using SoccerRAG

Aleksander Theo Strand
OsloMet
Oslo, Norway
0009-0008-2749-2347

Sushant Gautam
OsloMet, SimulaMet
Oslo, Norway
0000-0001-9232-2661

Cise Midoglu
SimulaMet, Forzasys
Oslo, Norway
0000-0003-0991-4418

Pål Halvorsen
OsloMet, SimulaMet, Forzasys
Oslo, Norway
0000-0003-2073-7029

Abstract—The rapid evolution of digital sports media necessitates sophisticated information retrieval systems that can efficiently parse extensive multimodal datasets. This paper demonstrates SoccerRAG, an innovative framework designed to harness the power of Retrieval Augmented Generation (RAG) and Large Language Models (LLMs) to extract soccer-related information through natural language queries. By leveraging a multimodal dataset, SoccerRAG supports dynamic querying and automatic data validation, enhancing user interaction and accessibility to sports archives. We present a novel interactive user interface (UI) based on the Chainlit framework which wraps around the core functionality, and enable users to interact with the SoccerRAG framework in a chatbot-like visual manner.

Index Terms—association football, information retrieval, large language models, natural language processing, sports, UI

I. INTRODUCTION

The burgeoning interest in Retrieval Augmented Generation (RAG), fueled by the rapid advancements in Large Language Models (LLMs), has paved new avenues for exploring innovative use cases across various domains, notably within multimodal information retrieval [1]. Despite the wide applicability of RAG frameworks in enhancing the capabilities of generative AI for open-domain question answering and beyond, its potential in the sports domain, particularly within soccer analytics, remains largely untapped. Our recent research has endeavored to bridge this gap by harnessing multimodal datasets inherent to the soccer broadcast pipeline and integrating RAG with different data modalities. We have presented SoccerRAG, a framework for retrieving multimodal soccer information using natural language queries, from an augmented soccer dataset based on SoccerNet [2], [3], which includes game videos with image frames and audio, timestamped captions (transcribed audio), annotations for game events, and player information.

In this paper, we demonstrate the use of the SoccerRAG framework through the command-line interface (CLI) and a novel user interface (UI). Figure 1 presents an overview of the SoccerRAG framework, with the core components (database, feature extractor, feature validator, and SQL agent) as presented earlier, wrapped by a novel interactive UI which serves as a user-friendly tool for visual interactions. The UI was created using the Chainlit framework. Chainlit is an open-source Python package that enables developers to build production-ready conversational AI applications, providing features for quick integration. It offers integrations

with popular libraries and frameworks, such as OpenAI and LangChain, and allows for custom front-ends with React-based user interfaces [4]. The SoccerRAG integrates functionality for data representation, feature extraction and validation, database querying (based on our proposed database schema for the augmented version of the SoccerNet dataset), and a novel extractor-validator chain. The open source implementation for SoccerRAG is accessible under [5] and fully reproducible through the instructions provided therein, which are detailed below.

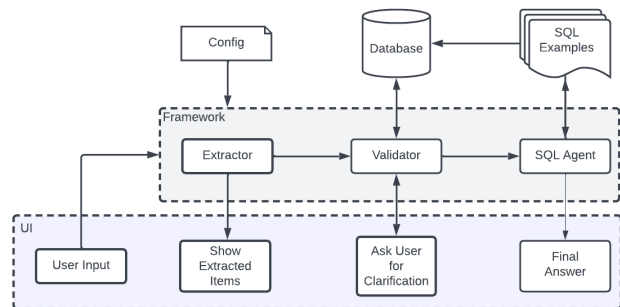


Fig. 1: Overview of the SoccerRAG framework, including the novel interactive UI.

The potential of our proposed system for information retrieval offers a promising avenue for fans and broadcasters alike, enabling the recollection of game specific information and highlights, as well as broader statistics and insights (within the scope of the entire database) through natural queries. Pursuing this avenue of research, we aspire not only to contribute to the body of knowledge surrounding RAG and multimodal information retrieval, but also to pioneer a smart system that encapsulates the dynamic essence of soccer, fostering a deeper connection between the sport and its global audience.

II. DEMONSTRATION

A. Application Flow

The core functionality of the SoccerRAG framework can be represented with the following application flow:

- The user provides a natural language query related to the contents of the database.

- The user input is sent to an LLM along with the properties schema and a system prompt describing the properties the LLM should extract from the query. The LLM then returns a list of extracted properties relevant to the query.
- Each extracted feature is checked against the appropriate table in the database using string matching algorithms. This step aims to correct spelling mistakes and abbreviations. Once a value is found, both the value and its primary key are added to the extracted value.
- The cleaned user prompt is combined with system-specific prompts to guide the LLM in generating SQL queries that will answer the user’s natural language query. The constructed query is then passed to the SQL chain, which designs and executes the SQL queries against the underlying database. The SQL chain handles communication between the system and the database, retrieves the requested data, and prepares the results for presentation to the user.

B. Artifacts

Figure 2 presents the structure of the SoccerRAG codebase, publicly accessible under [5]. The `data` folder is used by the core pipeline to read in the source dataset (including the original SoccerNet data, as well as our augmented league and team tables), and storing the database files. The `src` folder contains the configuration and executable files. The root directory includes the main executable (`main.py`), environment variables (`.env`), and a list of dependencies (`requirements.txt`). A proof-of-concept deployment of the SoccerRAG UI can be found under: <https://simulamet-host-soccerrag.hf.space/>

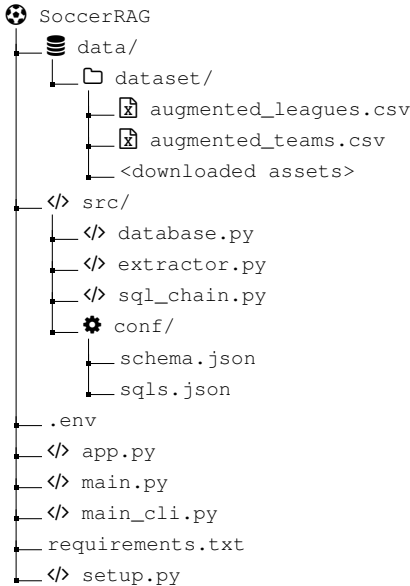


Fig. 2: Structure of the SoccerRAG codebase.

C. Requirements

The SoccerRAG codebase requires Python 3.12 or above to be run. The `requirements.txt` file in the codebase

specifies the required packages, which can be installed through `pip`. SoccerRAG does not require a GPU, and can be run on any machine with a CPU. For the experiments presented in the research paper, a machine with a Windows 11 operating system, Intel(R) Core(TM) i5-9300H CPU @2.40GHz, and 16GB memory has been used. SoccerRAG requires an OpenAI API key, which should be specified in the `.env` file. In order to log all queries to the OpenAI endpoints, it is possible to set a LangSmith API key [6]. This makes it possible to monitor the cost of each call, and keep history.

D. Running the Framework

Below are the steps to run the SoccerRAG framework (core functionality described in Section II-A).

1) *Configuration*: The SoccerRAG framework is set up using environment variables. Table I presents an overview of the configuration parameters. The parameters marked as mandatory do not have default values, and need to be configured before runtime. There should be an entry in `.env` for each configuration parameter. The SoccerRAG framework has been tested with GPT-3.5-Turbo [7] and GPT-4.0-Turbo [8], [9], but can support any model available from OpenAI [10]. The model version can be specified through the `OPENAI_MODEL` configuration parameter.

Parameter	Description	M	Default
<code>OPENAI_API_KEY</code>	OpenAI API key	✓	NA
<code>OPENAI_MODEL</code>	GPT version	✗	<code>gpt-3.5-turbo-0125</code>
<code>DATABASE_URL</code>	Location of DB	✗	<code>"/data/games."</code>
<code>LANGSMITH</code>	Trace api calls	✗	False
<code>LANGSMITH_API_KEY</code>	API key for LangSmith	✓ ¹	NA
<code>LANGSMITH_PROJECT</code>	Project tag for LangSmith	✗	SoccerRag
<code>FEW_SHOT</code>	Num. options in user validation	✗	3

TABLE I: SoccerRAG configuration parameters (M: mandatory).

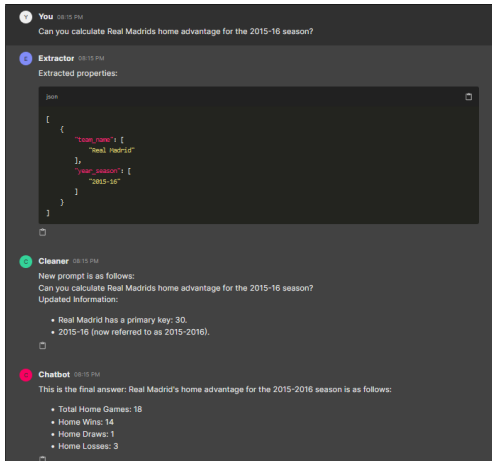
2) *Running via UI*: Run `python setup.py` to download the dataset and set up the DB, and `chainlit run app.py` for the UI.

Figure 3 presents an example screenshot, depicting alternative background colors² for the UI, as well as the intermediate feedback to users that the SoccerRAG provides for each operational step, which are retrieved from the feature extractor and feature validator components in the core pipeline. The meaningful response returned to the user prompt "Can you calculate Real Madrids [*sic*] home advantage for the 2015/16 season?" demonstrates that the framework is able to grasp soccer-related concepts such as a home advantage (which refers to the difference in scores achieved by a team in home vs. away games).

Figures 4 and 5 present further example screenshots, demonstrating the capabilities of SoccerRAG which range from database-wide statistics (example 2: "How many yellow cards did messi [*sic*] get in the 2015-16 season on home turf?" is returned with the correct aggregation across all games of

¹If `LANGSMITH` is set to True.

²SoccerRAG UI background color is configurable: click the user icon in the top right corner, select settings, then toggle the dark mode switch.



(a) Black background



(b) White background

Fig. 3: SoccerRAG UI example 1. The user prompts: "Can you calculate Real Madrids [sic] home advantage for the 2015/16 season?"

the desired season), to supporting multi-format information presentation (example 3: "List all games played by ManU [sic] in the 16-17 season. / Give the result as a markdown table with following format / HomeTeam AwayTeam Score Venue Attendance Date").

One of the challenges faced by systems which are prompted via natural language queries is the potential lack of clarification with respect to the entities in the prompt. In the context of soccer, these can be player, team, or league names, places, season dates, etc. SoccerRAG requests clarification from the user in cases of inadequate information, as shown in Figure 6. Here (example 4: "Create a list of all games Lionel got a yellow card / Make the list in markdown with following coulms [sic] / GameId, HomeTeam, AwayTeam, Score, Date"), as opposed to directly returning a response ("one-step") as in examples 1-3, the framework needs to ask the user to select the correct value for "Lionel", as there are multiple entities in the database with this name ("two-step"). In the UI, the user can



Fig. 4: SoccerRAG UI example 2. The user prompts: "How many yellow cards did messi [sic] get in the 2015-16 season on home turf?"

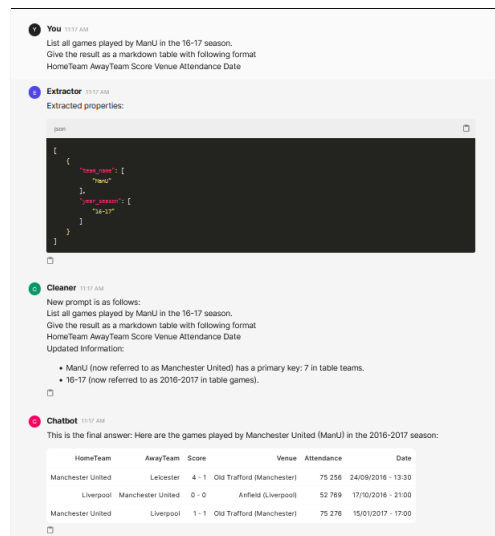
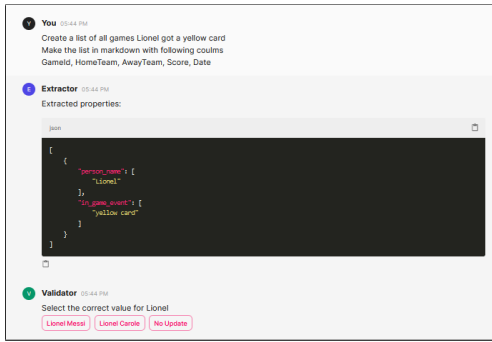


Fig. 5: SoccerRAG UI example 3. The user prompts: "List all games played by ManU [sic] in the 16-17 season. / Give the result as a markdown table with following format / HomeTeam AwayTeam Score Venue Attendance Date"

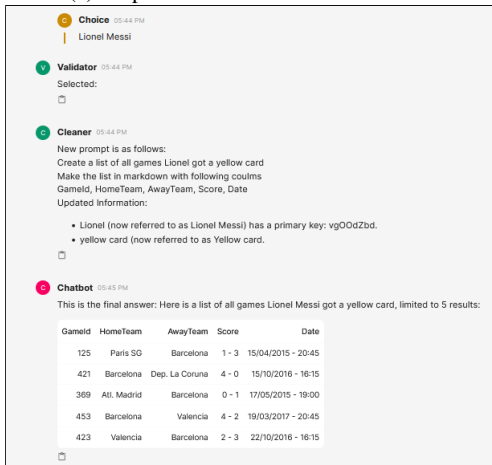
click a button to communicate their choice and validate one of the presented options, or pass the original string untranslated to the next component.

3) *Running via CLI*: Run `python setup.py` to download the dataset set up the DB, and `python main_cli.py -q <query>` to use the CLI. The `<query>` should be encapsulated by quotation marks (e.g., "How many goals did Arsenal score in the 2015-16 season?"). If linebreaks are intended, "\n" can be used.

Figure 7 presents an example screenshot, depicting the



(a) Step 1: user asked for clarification



(b) Step 2: response returned based on additional user input

Fig. 6: SoccerRAG UI example 4. The user prompts: "Create a list of all games Lionel got a yellow card / Make the list in markdown with following columns [sic] / GameId, HomeTeam, AwayTeam, Score, Date".



Fig. 7: SoccerRAG CLI example 1. The user prompts: "In the game between Chelsea and Burnley in the 2014-15 season, did anyone get a yellow card? If yes, who."

console outputs from the CLI for the query "In the game between Chelsea and Burnley in the 2014-15 season, did anyone get a yellow card? If yes, who.", which receives a response directly (one-step).

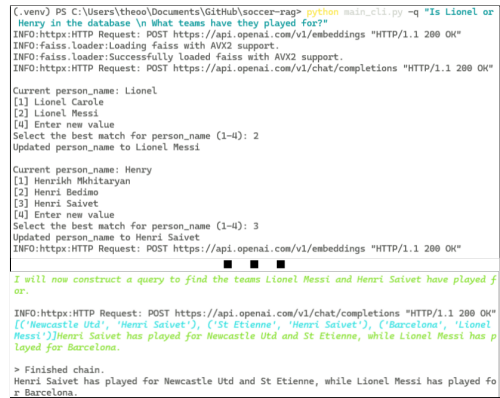


Fig. 8: SoccerRAG CLI example 2. The user prompts: "Is Lionel or Henry in the database \n What teams have they played for?"

Figure 8 presents an example screenshot, depicting the console outputs for the query "Is Lionel or Henry in the database \n What teams have they played for?", which requires clarification from the user before returning the response (two-step). Differently than the UI, the user can also provide a custom string as a new value (option).

III. CONCLUSION

We demonstrate SoccerRAG, a framework that leverages RAG and LLMs to efficiently retrieve multimodal soccer information via natural language queries, enhancing the accessibility of sports datasets by allowing for intuitive user interactions with complex data archives. In addition to the core pipeline presented in our research paper, we have introduced a novel UI to facilitate visual user interactions with the framework in a chatbot-like manner. The UI serves as a wrapper around the core pipeline, whereby it passes the user query to the feature extractor component, after which the query is validated by the feature validator against the database, the query with validated properties are passed to SQL agent, which retrieves data from the database and returns a response, and the UI presents the response to the user.

We present the entire codebase for the SoccerRAG as open source software, along with instructions to run the framework either via CLI or via UI. Our experiments up to now demonstrate SoccerRAG's capability to accurately interpret complex queries and facilitate dynamic user engagements with soccer content. We believe that SoccerRAG exemplifies the transformative potential of AI in sports analytics, promising enriched global fan experiences and advances in the accessibility of sports information.

ACKNOWLEDGMENT

This research was partly funded by the Research Council of Norway, project number 346671 (AI-storyteller).

REFERENCES

- [1] Y. Cheng *et al.*, “Exploring Large Language Model based Intelligent Agents: Definitions, Methods, and Prospects,” *arXiv*, Jan. 2024.
- [2] S. Giancola *et al.*, “SoccerNet: A Scalable Dataset for Action Spotting in Soccer Videos,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2018, pp. 18–22.
- [3] A. Delière *et al.*, “SoccerNet-v2: A Dataset and Benchmarks for Holistic Understanding of Broadcast Soccer Videos,” in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2021, pp. 19–25.
- [4] Chainlit, “Chainlit,” <https://docs.chainlit.io/get-started/overview>, 2023, accessed: 2024-05-03. [Online]. Available: <https://docs.chainlit.io/get-started/overview>
- [5] “SoccerRAG: Multimodal Soccer Information Retrieval via Natural Queries,” Apr. 2024, [Online; accessed 12. Apr. 2024]. [Online]. Available: <https://github.com/simula/soccer-rag>
- [6] LangSmith, “Get started with langsmith,” <https://docs.smith.langchain.com/>, 2024, accessed: 2024-05-07. [Online]. Available: <https://docs.smith.langchain.com/>
- [7] “GPT-3.5-Turbo,” Apr. 2024, [Online; accessed 12. Apr. 2024]. [Online]. Available: <https://platform.openai.com/docs/models/gpt-3-5-turbo>
- [8] “GPT-4,” Apr. 2024, [Online; accessed 12. Apr. 2024]. [Online]. Available: <https://openai.com/gpt-4>
- [9] OpenAI, “Gpt-4 and gpt-4 turbo documentation,” <https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo>, 2023, accessed: 2024-03-20. [Online]. Available: <https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo>
- [10] —, “Models,” <https://platform.openai.com/docs/models/>, 2024, accessed: 2024-05-07. [Online]. Available: <https://platform.openai.com/docs/models/>