Orchestrating Multi-Agent Systems for Multi-Source Information Retrieval and Question Answering with Large Language Models

Antony Seabra[1,2], Claudio Cavalcante[1,2], Joao Nepomuceno[1], Lucas Lago[1], Nicolaas Ruberg[1], and Sergio Lifschitz[2]

[1] BNDES - Área de Tecnologia da Informação, Rio de Janeiro, Brazil
[2] PUC-Rio - Departamento de Informática, Rio de Janeiro, Brazil

Abstract. We present a novel framework for developing robust multi-source question-answer systems by dynamically integrating Large Language Models with diverse data sources. This framework leverages a multi-agent architecture to coordinate the retrieval and synthesis of information from unstructured documents, like PDFs, and structured databases. Specialized agents, including SQL agents, Retrieval-Augmented Generation agents, and router agents, dynamically select and execute the most suitable retrieval strategies for each query. To enhance contextual relevance and accuracy, the framework employs adaptive prompt engineering, fine-tuned to the specific requirements of each interaction. We demonstrate the effectiveness of this approach in the domain of Contract Management, where answering complex queries often demands seamless collaboration between structured and unstructured data. The results highlight the framework's capability to deliver precise, context-aware responses, establishing a scalable solution for multi-domain question-answer applications.

Keywords: Information Retrieval, Question Answer, Large Language Models, Documents, Databases, Prompt Engineering, Retrieval Augmented Generation, Text-to-SQL.

1 Introduction

The rapid evolution of Large Language Models (LLMs) has transformed the fields of information retrieval and question-answering (Q&A) systems, enabling significant advancements in understanding and generating human-like text. These capabilities have unlocked new possibilities for retrieving precise and contextually relevant information from diverse sources. However, integrating data from heterogeneous sources - such as unstructured text documents, structured databases, and real-time APIs - into a unified system remains a complex challenge. Traditional systems often fall short in managing this complexity, struggling to retrieve and correlate information across varying formats, which can compromise the accuracy and relevance of responses. This challenge highlights the need for sophisticated frameworks that can dynamically orchestrate and retrieve information from multiple sources while leveraging the contextual understanding offered by LLMs.

Professionals across industries often face the daunting task of navigating vast amounts of unstructured text while simultaneously accessing structured data. This process is not only labor-intensive but also error-prone, as locating specific information and correlating it across disparate sources can be difficult. For example, in Contract Management, retrieving details from both lengthy contract documents and structured database records often requires extensive manual effort. Tasks such as identifying penalties, SLAs, or deadlines buried within hundreds of pages and linking them with structured metadata demand significant time and attention to detail.

To address these issues, we propose a dynamic multi-agent framework that leverages advanced techniques in orchestration and retrieval to enhance the capabilities of multi-source

Q&A systems using LLMs. By integrating Retrieval-Augmented Generation (RAG), text-to-SQL techniques, and adaptive prompt engineering, the framework enables the system to handle complex queries across heterogeneous data sources with precision, all without requiring retraining of the underlying language model. Central to this approach is an agent-based architecture that dynamically selects and executes the most suitable retrieval strategy for each query, optimizing data access from diverse sources.

We evaluate this framework in the context of Contract Management, a domain that exemplifies the challenges of multi-source information retrieval. Existing systems in this area often struggle to provide detailed, contextually relevant answers that require the integration of data from both unstructured documents and structured databases. By employing specialized agents—such as SQL agents, RAG agents, and router agents—our system dynamically routes queries to the appropriate sources, delivering more accurate and comprehensive responses.

A key innovation of our framework is dynamic prompt engineering, which adjusts prompt instructions in real-time based on the query context, the nature of the data being retrieved, and the user's input. This ensures that responses generated by the LLM are contextually optimized, whether the query involves extracting specific details from an unstructured document or querying structured database records.

The remainder of this paper is structured as follows: Section 2 reviews the technical background on agent-based orchestration and retrieval techniques, including RAG, text-to-SQL, and prompt engineering. Section 3 details our methodology and the application of these techniques, while Section 4 outlines the evaluation process and experimental results. Finally, Section 5 concludes the study and discusses potential directions for future research in this domain.

2 Background

Building an effective multi-source question-answer system requires leveraging advanced techniques that address the complexities of retrieving and processing information from diverse sources. These techniques must work cohesively under a dynamic, agent-based orchestration framework. This section explores the foundational technologies enabling our system: Large Language Models (LLMs) for advanced natural language understanding; Prompt Engineering, which optimizes LLMs for specific tasks; Retrieval-Augmented Generation (RAG), which incorporates external data into LLM contexts for accurate answers; Text-to-SQL, which translates natural language into database queries; and Agents, which dynamically manage workflows and select optimal strategies [Mialon et al., 2023]. Together, these technologies form the backbone of our proposed multi-agent methodology, enabling seamless integration across multiple data sources and enhancing Q&A system performance.

2.1 Large Language Models

Large Language Models (LLMs), based on the Transformer architecture [Vaswani et al., 2017], have revolutionized natural language processing (NLP), enabling machines to generate and interpret human-like text with exceptional accuracy. These models utilize self-attention mechanisms to evaluate the importance of various text segments, capturing intricate linguistic patterns and relationships. This versatility makes LLMs invaluable for tasks such as text generation, translation, and information retrieval [Seabra et al., 2024a].

LLMs like GPT [OpenAI, 2023a] have significantly advanced Q&A systems, offering a powerful interface for retrieving information from diverse data sources. However, while

LLMs can process vast text corpora and generate coherent responses, they are limited by static knowledge, potential factual inaccuracies, and challenges in domain-specific expertise [Chen et al., 2024]. To overcome these limitations, a set of techniques, such as Retrieval-Augmented Generation (RAG) and Text-to-SQL, have emerged as powerful approaches to integrate external data sources into Large Language Models (LLMs) without the need for retraining the language model.

These methods enable LLMs to access and utilize up-to-date, domain-specific, or structured information dynamically at query time. RAG enhances the generative capabilities of LLMs by retrieving relevant chunks of information from external repositories, such as document databases or knowledge graphs, and feeding them as additional context for response generation. This ensures that the model can provide accurate and contextually relevant answers, even when the required information is outside its static training dataset. Similarly, Text-to-SQL bridges the gap between natural language queries and structured databases by translating user input into executable SQL commands. This allows the system to fetch precise, structured data directly from relational databases, ensuring accuracy in scenarios requiring exact matches. Both techniques emphasize modularity and scalability, allowing the integration of LLMs with external data without modifying their core architecture. By combining these approaches, systems can dynamically adapt to evolving data sources and user needs while maintaining high performance and flexibility.

2.2 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) enhances LLMs by incorporating external data into the generation process, addressing the inherent limitations of static model knowledge. RAG retrieves relevant information from external repositories, such as document stores or databases, and integrates it into the LLM's context to generate informed responses [Gao et al., 2023b].

RAG operates by embedding user queries and data chunks into high-dimensional vector spaces, enabling semantic comparisons to retrieve the most relevant information. This retrieved data supplements the LLM, ensuring responses are accurate and current. Effective chunking strategies, which segment documents into manageable portions, are crucial for RAG's success [Gao et al., 2023b]. These strategies, whether token- or section-based, balance maintaining context with maximizing relevance, particularly in domains like Contract Management.

RAG methodologies have evolved into specialized types, such as Retrieve-and-Rerank and Graph RAG, to optimize performance for specific use cases. Retrieve-and-Rerank employs an initial retrieval step to gather a broad set of candidate documents, followed by a reranking process to identify the most relevant subset based on advanced scoring mechanisms. This approach is particularly effective in ensuring high precision in responses. Graph RAG, on the other hand, leverages structured relationships in knowledge graphs to guide retrieval and contextualize information, enabling the system to answer complex queries that require understanding entity relationships and dependencies. These variations highlight the versatility of RAG frameworks in addressing diverse information needs.

Despite its strengths, RAG can face challenges when semantically similar yet contextually irrelevant chunks are retrieved. This highlights the importance of refinement techniques to align retrieved data with user intent. By addressing these issues, RAG bridges the gap between static LLM knowledge and real-time information needs. The chunking strategy employed in RAG is essential to its effectiveness, as it dictates how documents are divided into smaller segments for embedding and retrieval. By efficiently segmenting large

documents, RAG ensures that only the most pertinent sections are retrieved and incorporated into the LLM, reducing information overload and enhancing answer precision. The selection of similarity metrics, such as Cosine or Euclidean distance, significantly impacts which chunks are chosen for retrieval [Gao et al., 2023b]. In RAG, the chunking strategy is pivotal because it directly affects the quality of the retrieved content.

A well-crafted chunking approach ensures that the information is cohesive, semantically complete, and preserves its intended meaning. Various chunking methods can be applied depending on the nature and structure of the data. For example, one common technique involves dividing text into chunks based on a set number of tokens, often including an overlap parameter to maintain continuity between segments. This overlap is particularly useful in lengthy documents where important details may span multiple chunks. Another method, particularly effective for structured documents, involves chunking based on specific sections or headers, such as splitting contracts into clauses or legal sections. This ensures that each chunk is a self-contained, semantically meaningful unit. The choice of chunking technique plays a vital role in balancing the need to capture full context while ensuring relevance in the retrieved information.

2.3 Text-to-SQL

Text-to-SQL translates natural language queries into SQL commands, bridging the gap between plain-text inputs and relational databases [Seabra et al., 2024b]. This technique empowers users to access precise, structured data without requiring knowledge of SQL syntax [Liu et al., 2023]. By leveraging LLMs, Text-to-SQL systems interpret natural language, map it to database schemas, and generate accurate queries.

As noted in [Pinheiro et al., 2023], Text-to-SQL systems excel in complex database environments by linking entities to tables and generating SQL commands. This capability is particularly valuable in domains like Contract Management, where queries often span multiple tables with intricate relationships. The correlation between Text-to-SQL systems and the semantics embedded in the relational schema plays a crucial role in determining the accuracy of the generated SQL commands. Relational schemas inherently define the structure and relationships between tables, columns, and data types, providing a semantic framework that Text-to-SQL models rely on to map natural language queries to precise SQL statements.

When the schema is well-designed with clear, intuitive naming conventions and meaningful relationships, it enhances the model's ability to interpret user intent and generate accurate SQL commands. However, if the schema contains ambiguous or poorly named entities, lacks sufficient normalization, or features complex relationships, the Text-to-SQL system may struggle to correctly align the query's semantics with the database structure. This misalignment can lead to incomplete, incorrect, or overly broad SQL queries, reducing the accuracy of the retrieved data. Therefore, the interplay between the relational schema's semantics and the Text-to-SQL model's understanding is critical for achieving high-quality query translations. Improving schema clarity and incorporating semantic annotations can further enhance the system's performance by providing additional context for accurate SQL generation.

Text-to-SQL complements RAG by providing precise, structured data retrieval. While RAG focuses on retrieving semantically similar text for generative synthesis, Text-to-SQL delivers exact matches from structured databases [Seabra et al., 2024a]. This synergy enhances the flexibility of multi-source systems, enabling them to address a diverse range of queries effectively.

2.4 Prompt Engineering

Prompt Engineering is a powerful technique that customizes the behavior of Large Language Models (LLMs) by embedding carefully crafted instructions into the input prompts. These instructions serve to align the model's outputs with the specific needs and expectations of the user, providing a high degree of control over the generated responses. By defining parameters such as tone, format, and the required level of detail, Prompt Engineering enables developers to guide the model toward producing outputs that are not only accurate but also contextually appropriate and tailored to the task at hand [OpenAI, 2023b].

Carefully crafted prompts significantly enhances the accuracy and relevance of responses [White et al., 2023]. In the context of Contract Management, prompts can be tailored to explicitly specify tasks such as retrieving penalty clauses or summarizing contractual obligations, effectively directing the LLM to focus on the most pertinent sections of the text. For example, a prompt like "Identify and summarize penalties related to late delivery in this contract" provides clear and concise guidance, ensuring the model produces outputs aligned with user expectations. By embedding contextual details and precise instructions, well-designed prompts not only reduce ambiguity but also enhance the LLM's ability to deliver precise, task-specific information, making them invaluable in domains requiring high accuracy and contextual awareness [Giray, 2023].

Prompt Engineering also mitigates ambiguity and reducing factual hallucinations, which are common challenges when working with Large Language Models LLMs. By carefully designing prompts to restrict responses to specific, reliable data sources, this technique ensures that the LLM's outputs are both relevant and grounded in verifiable information [Wang et al., 2023]. For instance, prompts can be tailored to direct the model to retrieve data exclusively from trusted repositories or databases, explicitly instructing it to disregard unsupported prior knowledge. This level of control helps prevent the generation of plausible-sounding but inaccurate responses, a phenomenon often referred to as hallucination.

When integrated with advanced retrieval techniques like Retrieval-Augmented Generation (RAG) and Text-to-SQL, Prompt Engineering amplifies the capabilities of multi-source systems. For example, in the context of RAG, prompts can instruct the model to focus on the most relevant information retrieved from a vectorstore, ensuring that the contextual input aligns closely with the query's intent. Similarly, in Text-to-SQL systems, prompts can provide explicit instructions on how to interpret user queries, map them to database schemas, and prioritize certain fields or relationships for retrieval. Studies such as [Jeong, 2023] and [Gao et al., 2023a] demonstrate that the integration of Prompt Engineering with these techniques not only enhances the relevance and precision of responses but also streamlines the interaction between unstructured and structured data sources. Moreover, prompts can introduce dynamic contextualization, allowing systems to adapt instructions in real time based on the query's requirements, user intent, or the type of data being accessed. This synergy makes Prompt Engineering a cornerstone of modern multi-source question-answering frameworks, addressing limitations inherent in LLMs while improving reliability and user trust in their outputs.

2.5 Agents

Agents serve as the backbone of dynamic workflows in Q&A systems, enabling intelligent query routing and efficient resource utilization. By dynamically directing queries to the most appropriate retrieval paths, agents ensure that each request is processed using the method best suited to its nature and context. This adaptability is essential for multi-source

systems, where queries may require access to diverse data types, including unstructured text, structured databases, or even real-time APIs. In our framework, specialized agents such as Router Agents, RAG Agents, and SQL Agents work in tandem to manage these complexities and provide seamless query handling [Lewis et al., 2020].

Router Agents act as the central decision-makers within the system. When a query is received, the Router Agent analyzes its structure and intent to determine the optimal processing strategy. This analysis may involve applying predefined rules, natural language understanding techniques, or pattern recognition to identify key indicators that guide query routing. For instance, a query asking for specific numerical data might be routed to an SQL Agent, while one seeking a contextual explanation might be directed to a RAG Agent.

RAG Agents specialize in handling queries that involve retrieving unstructured information. They utilize Retrieval-Augmented Generation (RAG) to fetch relevant text fragments from vectorstores or document repositories, integrating this data into the context provided to the language model for response generation. This allows the system to deliver nuanced answers that incorporate insights from external text sources. SQL Agents, on the other hand, are designed for interacting with structured databases. By leveraging Text-to-SQL techniques, these agents translate natural language queries into SQL commands, enabling precise retrieval of structured data. This capability is particularly useful for fact-based queries requiring exact matches, such as financial metrics, inventory details, or contract deadlines.

The orchestration of these specialized agents ensures that queries are routed and processed efficiently, maximizing the relevance and accuracy of the responses. Moreover, this architecture is inherently scalable and modular, allowing new agents to be integrated as needed to support additional data types or advanced processing techniques. By coordinating these agents within a unified framework, the system achieves a high degree of flexibility, adaptability, and performance, meeting the demands of complex, multi-source Q&A scenarios. This agent-based approach not only enhances query handling but also paves the way for future innovations in intelligent workflow management and data integration.

Agent frameworks, such as Langchain [Langchain, 2024] and Crew AI [cre, 2024], represent significant advancements in agent-based architectures, offering enhanced capabilities for orchestrating multi-agent workflows in dynamic environments. Crew AI provides tools for designing, managing, and monitoring specialized agents, ensuring efficient task routing and execution. By integrating cutting-edge frameworks like Crew AI, agent-based systems can achieve greater flexibility, scalability, and robustness, especially in complex multi-source environments. These innovations further underscore the potential of agents to dynamically adapt to evolving user requirements and domain-specific challenges.

Agents enable dynamic decision-making and modular scalability, improving the relevance and accuracy of responses. By integrating structured and unstructured data retrieval, they provide a robust foundation for multi-source Q&A systems [Jin et al., 2024]. The agent-based architecture also allows for adding new capabilities, enhancing adaptability across domains.

3 Our Methodology

In designing our multi-source question-answer methodology, we employ a combination of advanced techniques to seamlessly access diverse data sources and deliver accurate, contextually relevant responses tailored to the specific query and information source. By integrating Retrieval-Augmented Generation (RAG), Text-to-SQL, Dynamic Prompt Engineering, and Agent-based orchestration, the system effectively manages the complexities

inherent in interacting with both structured and unstructured data sources. Each component plays a critical role in addressing different aspects of the information retrieval process, ensuring the system's ability to dynamically adapt to the unique requirements of each query.

RAG is employed to handle unstructured data, such as text documents or knowledge repositories, by retrieving the most relevant segments of information and incorporating them into the model's context for generating precise and well-informed responses. Text-to-SQL complements this by enabling the system to interpret natural language queries and translate them into executable SQL commands, allowing precise access to structured data stored in relational databases. Together, these techniques bridge the gap between different data modalities, ensuring comprehensive coverage of query requirements.

Dynamic Prompt Engineering serves as the interface between the user's intent and the model's capabilities, guiding the system to focus on relevant aspects of the data and format responses in a way that aligns with the query's context. By embedding explicit instructions and contextual cues into the prompts, the system ensures relevance, accuracy, and clarity in the generated outputs.

Finally, Agent-based orchestration underpins the entire methodology, acting as the system's decision-making and coordination layer. Specialized agents, such as Router Agents, RAG Agents, and SQL Agents, dynamically analyze and route queries to the most suitable processing path based on their nature and complexity. This agent-based architecture not only streamlines the workflow but also allows the system to scale and evolve by integrating additional agents for new data types or advanced functionalities.

By harmonizing these components into a unified framework, our methodology effectively addresses the challenges of multi-source question answering, delivering robust performance and adaptability across diverse domains and data ecosystems. This approach ensures that the system can provide timely, accurate, and context-aware responses regardless of the complexity or heterogeneity of the underlying data sources.

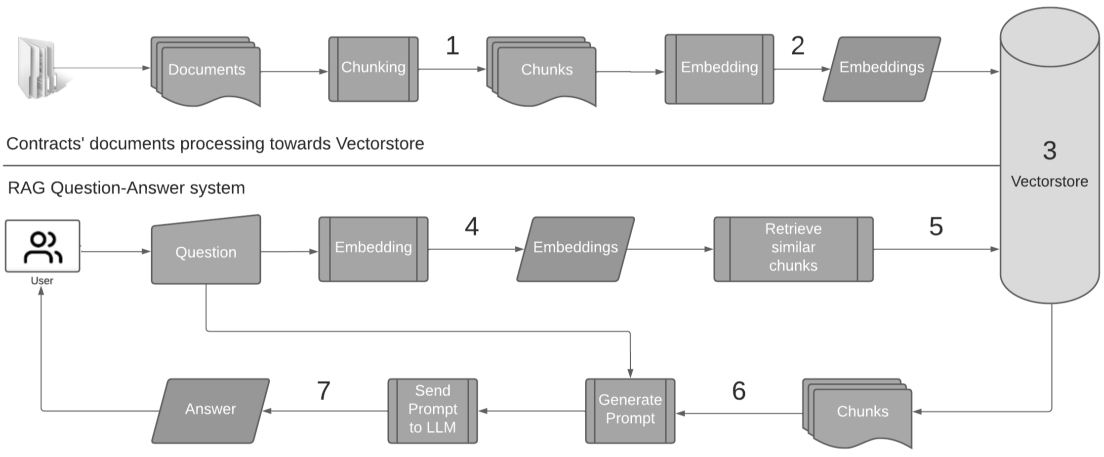

Fig. 1. Retrieval-Augmented Generation. Source: [Seabra et al., 2024a]

RAG enables the retrieval of relevant information from large volumes of unstructured text, while Text-to-SQL facilitates precise access to structured data within relational databases. Dynamic Prompt Engineering customizes the query context, ensuring that responses are tailored to user intent, and Agent-based orchestration coordinates these tech-

niques, directing queries to the appropriate modules and managing workflows seamlessly. In this section, we detail the approaches and challenges associated with implementing each of these techniques, along with the strategies we used to optimize their integration.

Our methodology was implemented and validated in a real-world project named Contrato360 [Seabra et al., 2024a], a question-answer system tailored to meet the specific demands of Contract Management. Contrato360 integrates a combination of advanced techniques, including Retrieval-Augmented Generation (RAG), Text-to-SQL, Dynamic Prompt Engineering, and Agent-based orchestration, to overcome the challenges of navigating and extracting information from intricate contract documents and structured databases. This system allows users to efficiently query critical contract-related data, such as penalty clauses, deadlines, service level agreements, and other contractual obligations, from diverse data sources. By leveraging these cutting-edge methods, Contrato360 ensures precise, contextually relevant, and timely responses, addressing the complexity and criticality of the contract management domain. This real-world deployment highlights the effectiveness and practicality of our methodology in a field where accuracy, relevance, and contextual comprehension are paramount for decision-making and operational efficiency.

3.1 Applying RAG

According to [Seabra et al., 2024a], the first step when applying RAG involves (1) reading the textual content of the PDF documents into manageable (*chunks*), which are then (2) transformed into high-dimensional vectors (*embedding*). The text in vector format captures the semantic properties of the text, a format that can have 1536 dimensions or more. These *embeddings* (vectors) are stored in a *vectorstore* (3), a database specialized in high-dimensional vectors. The vector store allows efficient querying of vectors through their similarities, using the distance for comparison (whether *Manhatan*, Euclidean or cosine). Once the similarity metric is established, the query is *embedded* in the same vector space (4); this allows a direct comparison between the vectorized query and the vectors of the stored chunks, retrieving the most similar chunks (5), which are then transparently integrated into the LLM context to generate a *prompt* (6). The *prompt* is then composed of the question, the texts retrieved from the *vectorstore*, the specific instructions and, optionally, the *chat* history, all sent to the LLM which generates the final response (7).

Chunking strategy One of the first decisions to be made when applying RAG is to choose the best strategy to segment the document, that is, how to perform the *chunking* of the PDF files. A common *chunking* strategy involves segmenting documents based on a specific number of *tokens* and an overlap (*overlap*). This is useful when dealing with sequential texts where it is important to maintain the continuity of the context between the *chunks*.

There is a common type of document with well-defined sections; contracts are a prime example. The have a standardized textual structure, organized into contractual sections. Therefore, sections with the same numbering or in the same vicinity describe the same contractual aspect, that is, they have similar semantics. For example, in the first section of contract documents, we always find the object of the contract. In this scenario, we can assume that the best *chunking* strategy is to separate the *chunks* by section of the document. In this case, the *overlap* between the *chunks* occurs by section, since the questions will be answered by information contained in the section itself or in previous or subsequent sections. For the contract page in the example in Figure 2, we would have a *chunk* for the section on the object of the contract, another *chunk* for the section on the term of

the contract, that is, a *chunk* for each clause of the contract and its surroundings. This approach ensures that each snippet represents a semantic unit, making retrievals more accurate and aligned with queries.

Using predefined sections as the boundaries for *chunks* enhances the relevance of responses within a single contract. However, this approach presents two main challenges: (1) within a single document, when a term appears repeatedly, it can be difficult to identify the specific chunk that answers a question; and (2) as the number of documents increases, accurately selecting the correct document to address becomes more challenging for the system. In the Contract Management domain, consider a scenario where the user asks, "Who is the contract manager of contract number 123/2024?". This query is intended to retrieve the specific name of the contract manager for the given contract. However, the term "contract manager" can appear in various clauses of the contract document, often in sections that do not contain the name of the actual manager but refer to responsibilities or general rules related to contract management. For instance, multiple clauses across different sections of the contract might mention the term "contract manager" in contexts like assigning responsibilities, explaining the duties of a manager, or defining roles in contract supervision. Even though these clauses contain the term "contract manager," they do not answer the user's question, which is specifically asking for the name of the contract manager for contract 123/2024.

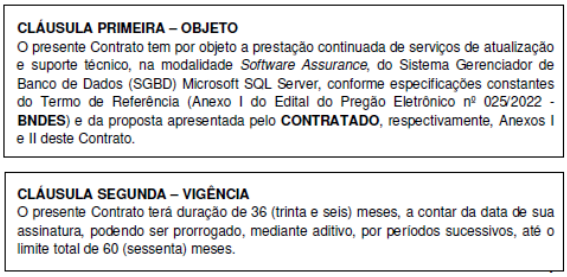

Fig. 2. Chunking based on Contract's clauses

Due to the similarity between the query and these irrelevant sections, the Retrieval-Augmented Generation (RAG) system may retrieve a chunk from one of these irrelevant clauses that does not actually contain the required name. For example, instead of retrieving the clause that explicitly names the contract manager, the system might retrieve a clause that discusses the general duties of a contract manager. This happens because the chunk embedding for a clause about the role or responsibilities of the manager may be semantically similar to the query, even though it lacks the specific information requested. In this case, the chunk retrieved is related to the term "contract manager" but does not include the answer the user expects. As a result, the system could return an incorrect response, such as a general description of the role of a contract manager, rather than identifying the actual manager for contract 123/2024. This illustrates the challenge of relying solely on textual similarity in chunk retrieval, as it can lead to the retrieval of information that is similar to the query in wording but not relevant to the specific context of the user's question. To mitigate this, additional filtering mechanisms, such as metadata checks or contract-specific identifiers, are required to ensure that the system retrieves the most contextually appropriate information from the correct contract section.

To overcome this issue, several strategies can be applied. One approach is to add metadata to the *chunks* and, when accessing the *vectorstore*, use this metadata to filter the information returned. This method improves the relevance of the retrieved texts by narrowing the search to only those chunks that match specific metadata criteria. Figure **??** displays the most relevant metadata attributes for the contracts: *source*, *contract*, and *clause*. Here, *source* represents the name of the contract's PDF file, *contract* refers to

the contract number, and *clause* indicates the section title. For instance, when querying, "Who is the contract manager of contract 123/2024?" the system first filters for chunks that belong to contract number 123/2024 and clauses related to the contract manager. Once these chunks are filtered, a similarity calculation is applied to identify the most relevant text segments, which are then sent to the LLM to generate the final response.

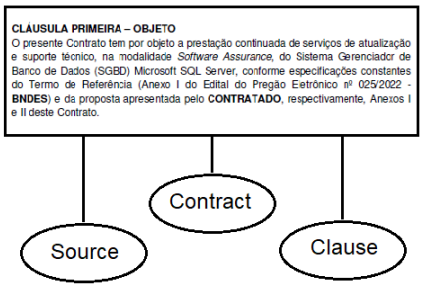

Fig. 3. Chunk's metadata

Embeddings models Embedding models are a cornerstone of modern NLP tasks and plays in importante role in our methodology. These models transform words, sentences, or even entire documents into high-dimensional vectors, or embeddings, and the key advantage of embeddings is that they enable more nuanced and semantically aware operations on text data, such as similarity comparisons and clustering. By embedding both the query and the text chunks in the same vector space, the system can measure how close they are to each other in meaning, ensuring that relevant information is retrieved even when it is not an exact keyword match.

Selecting the right embedding model depends on several factors related to the specific needs of a task, including the type of data, the complexity of the queries, and the computational resources available. Pretrained Models, such as BERT or GPT, are trained on vast amounts of general-purpose text data and are ideal for general tasks where the text spans multiple domains or where high-quality embeddings are required without the need for domain-specific customization. By contrast, custom models work better in specialized fields like legal or medical domains, as they can be beneficial to train an embedding model on a domain-specific corpus. This can help the model better capture the unique terminology and context of that field.

With respect to the vectors' dimensionality, embedding vectors can range in dimensionality depending on the model and the task. For instance, models like GloVe or Word2Vec often produce lower-dimensional embeddings (e.g., 300 dimensions), whereas modern transformer-based models like BERT and GPT can produce embeddings with 768 or more dimensions. Higher-dimensional embeddings typically capture more information and are better for complex tasks like Q&A systems or semantic search, but they also require more computational resources and storage. Lower-dimensional embeddings are computationally cheaper and faster but may not capture as much nuance, making them better suited for simpler tasks like keyword matching. If precision and detailed contextual understanding are important, high-dimensional embeddings are the better choice. For simpler or resource-constrained tasks, lower-dimensional embeddings may suffice.

In designing our multi-source Q&A methodology, we carefully evaluated various options for embedding models and vector dimensionality to optimize the system's performance. After considering several alternatives, we selected text-davinci-002, a model from OpenAI's GPT-3.5 family, along with embeddings with 1536 dimensions to strike a balance between accuracy, context understanding, and computational efficiency. One of the main advantages of text-davinci-002 is its ability to handle long sequences of text while maintaining a clear understanding of the context. This is essential when dealing with lengthy documents where information can be dispersed across various sections. The model can track the user's query context and dynamically retrieve or generate responses that are coherent and relevant

to the query. With 1536 dimensions, the embeddings can better represent the complex relationships between terms in the text, especially in documents where meaning often depends on subtle distinctions in wording. This is particularly useful in distinguishing between similar but contextually different terms, such as contract manager vs. contract supervisor, ensuring that the system retrieves the most relevant chunks.

Vectorstore The need to store and query high-dimensional vectors efficiently has led to the development of specialized vector databases, also known as vectorstores. These databases allow for the storage and retrieval of vector embeddings, making it possible to perform similarity searches - a key operation in tasks such as Retrieval-Augmented Generation (RAG) and semantic search. Unlike traditional databases that are optimized for structured, tabular data, vector databases are designed to handle embeddings generated by models like text-davinci-002, which represent semantic relationships in high-dimensional space.

When choosing the right vector database for a project, several factors come into play, including scalability, ease of use, latency, and integration with machine learning models. In our work, we evaluated three popular vector databases: Pinecone, Weaviate, and ChromaDB. Pinecone is a cloud-native vector database that excels in providing a fully managed service for high-performance similarity search. Weaviate is an open-source vector database that provides a highly flexible, schema-based approach to storing and querying vectors alongside structured metadata. ChromaDB is an open-source, lightweight vector database that focuses on simplicity and tight integration with machine learning workflows, making it ideal for embedding-based retrieval tasks in research and smaller projects. Our choice was the last one, specially because ChromaDB is easy to set up and integrate into a project without requiring extensive configuration or overhead. Given that our system is heavily Python-based, ChromaDB's Python-first design allowed us to quickly embed it into our machine learning pipelines. This streamlined our development process, enabling rapid iteration and testing, which was especially important in the early stages of system design. Also, by using ChromaDB, we can directly connect our text-davinci-002 embeddings with the vectorstore, enabling efficient similarity searches and accurate retrieval of contextually relevant information.

Similarity searches Similarity search is a fundamental operation in tasks that involve comparing vector embeddings to find data points that are semantically or contextually similar. This technique is widely used in fields such as information retrieval, recommendation systems, question-answering systems, and semantic search. The core of similarity search lies in the ability to measure how "close" two vectors are to each other in a high-dimensional space. Several distance metrics are commonly used to quantify this similarity, each with its own strengths and weaknesses depending on the nature of the data and the task. Three of the most popular algorithms for similarity searches include Cosine similarity, Euclidean distance, and Manhattan distance. Each method has a unique approach to measuring how similar two vectors are, and the choice of algorithm can significantly impact the performance and accuracy of a similarity-based system.

Cosine similarity measures the cosine of the angle between two vectors in a multi-dimensional space. It evaluates how "aligned" the two vectors are rather than how far apart they are. The cosine similarity value ranges from -1 to 1, where 1 indicates that the vectors are perfectly aligned (very similar), 0 means that the vectors are orthogonal (completely dissimilar), and -1 indicates that the vectors point in opposite directions. Cosine similarity is often used in text-based applications, where the magnitude of the vector is not as

important as the direction. Euclidean distance is the most common metric for measuring the straight-line distance between two points (or vectors) in a multi-dimensional space. It calculates the "as-the-crow-flies" distance between two vectors, treating each dimension as an axis in a Cartesian plane. Euclidean distance is widely used in geometric tasks or where the actual distance between points matters. Manhattan distance, also known as L1 distance or taxicab distance, measures the sum of the absolute differences between the corresponding coordinates of two vectors. Instead of measuring the direct straight-line distance (as in Euclidean), Manhattan distance measures how far one would have to travel along the axes of the space.

In our work, we chose cosine similarity for its ability to prioritize semantic alignment between query embeddings and document embeddings. Its strength in handling high-dimensional data, minimizing the influence of vector magnitude, and focusing on the directionality of vectors makes it the ideal choice for our Q&A system methodology. Cosine similarity is widely recognized as one of the best similarity measures for text-based applications, especially when using vector embeddings generated from NLP models like text-davinci-002. Since our system heavily relies on textual data, cosine similarity was the natural choice for ensuring that user queries are matched with the most relevant sections of the text, even if the exact phrasing differs. Whether we are retrieving specific sections in documents or providing general answers based on lenghty documents, cosine similarity ensures that the system is aligned with the semantic intent of the query.

3.2 Using structured data

In order to improve our question-answer system methodology, we explored two distinct approaches to integrate data from structured databases effectively. The first approach involved extracting data directly from the database, transforming it into text, and embedding this text into vector representations stored in the same vectorstore as our document-based embeddings. This method allowed us to convert structured data into a more flexible, text-based format, enabling semantic similarity searches alongside the unstructured text from contract documents. By embedding database information in this way, we created a unified search space where both structured and unstructured data could be queried with the same similarity-based techniques. This approach offered the advantage of simplicity, as it enabled direct integration of database information into our existing RAG framework, ensuring that queries could retrieve relevant data without needing to connect to the database during runtime.

The second approach we implemented involved a Text-to-SQL method, where natural language questions are dynamically translated into SQL queries. In this setup, the system interprets the user's query, converts it into a structured SQL command, and then submits it to the database for execution. The Text-to-SQL approach allows for precise data retrieval by directly querying the database, which is particularly beneficial for questions requiring exact, up-to-date values, such as specific dates, contract numbers, or quantitative information. Unlike the first approach, this method does not rely on pre-embedded representations; instead, it provides real-time access to structured data, ensuring that answers are accurate and reflect the current database state.

Each approach has its advantages. Embedding database data alongside unstructured text provides a unified search experience and reduces dependence on real-time database access. In contrast, the Text-to-SQL approach supports direct and precise querying, making it ideal for cases where exact values are necessary. Together, these approaches allow the system to leverage the strengths of both pre-embedded and dynamic querying, enhancing its versatility in handling a wide range of user queries.

3.3 Agents

Agents are central to the functionality and adaptability of our multi-source question-answer system, enabling it to handle diverse query types efficiently. By leveraging specialized agents, the system dynamically routes each query to the most suitable processing pathway, ensuring that user questions are handled with precision and contextual relevance. In our architecture, the Router Agent serves as the primary decision-maker, evaluating each incoming query and directing it to the appropriate agent based on predefined criteria.

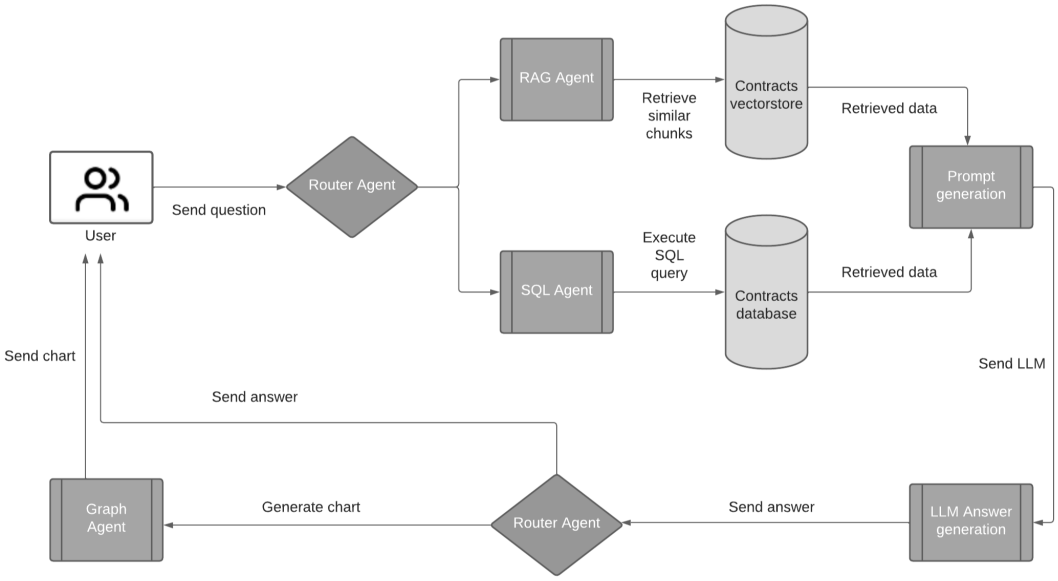

Fig. 4. Agents Architecture. Source: [Seabra et al., 2024a]

The Router Agent uses regular expressions to identify keywords, patterns, or structures within the query. If the query is specific to a clause within a contract, the Router Agent recognizes this pattern and assigns the query to the RAG Agent. The RAG Agent is optimized for handling unstructured text data, retrieving relevant text chunks from the vectorstore. By focusing on textual similarity, the RAG Agent retrieves semantically aligned information and generates responses that incorporate precise, contextually relevant excerpts from the documents, addressing the specifics of the the user's question.

Conversely, if the Router Agent detects that the question involves broader contract information, such as dates, financial details, or other exact values, it directs the query to the SQL Agent. The SQL Agent translates the natural language question into a structured SQL query, which is then executed against the database to retrieve exact data. This approach is particularly effective for queries requiring precise, structured responses, ensuring that the system provides accurate and up-to-date information directly from the database.

This dynamic agent-based architecture enables our system to handle both unstructured and structured data seamlessly. The Router Agent's decision-making process allows the system to optimize query processing based on the context and specific needs of each query. By directing contract-specific questions to the RAG Agent and structured data queries to the SQL Agent, the Router Agent ensures that user questions are handled efficiently, providing relevant answers whether they require interpretive text or exact data values.

This modular design not only improves response accuracy but also enhances the system's flexibility in adapting to a wide range of contract-related queries.

3.4 Dynamic Prompt Engineering

In our work, we employ Dynamic Prompt Engineering to enhance the precision and contextual relevance of generated responses, effectively guiding the behavior of the Large Language Model (LLM) to align with the user's specific needs [White et al., 2023,**?**]. This approach dynamically adapts prompts based on the specific agent handling the query, ensuring that the system delivers optimal responses regardless of whether the query involves unstructured text, structured data, or visual representations. By tailoring prompts to suit each agent, the system achieves a high degree of flexibility and accuracy, accommodating diverse data types and user intents.

For queries managed by the RAG Agent, dynamic prompts are constructed to include contextual instructions that direct the LLM to synthesize information from text chunks retrieved from the vectorstore. This enables the model to leverage semantically similar text embeddings while remaining aligned with the specific details of the user's question. For example, when asked about the responsibilities of a contract manager, the prompt explicitly instructs the LLM to extract and summarize relevant clauses, enhancing both the relevance and precision of the response.

For queries handled by the SQL Agent, dynamic prompts are designed to bridge the gap between natural language and structured queries, translating user input into precise SQL commands. This ensures accurate retrieval of structured data, such as contract details, dates, or financial figures. For instance, a query like "List all active contracts managed by John Doe" is dynamically transformed into a prompt that generates an SQL query, guiding the LLM to execute the task effectively and return the result in a tabular format. By aligning the prompt with the relational schema, the system ensures accurate interpretation of user intent and retrieval of the desired data.

In addition to text and SQL-focused prompts, we have introduced prompts for visual representation through a Graph Agent. This agent enhances the system's capability by translating query results into visual formats, such as bar charts or pie charts, when appropriate. For instance, if a user asks, "What are the monthly penalties across all contracts?", the prompt instructs the Graph Agent to interpret the retrieved data and generate a bar chart. This visualization complements textual explanations, providing users with clearer insights into trends, comparisons, or aggregated data. By integrating visual responses, the system improves accessibility and interpretability, particularly for data-heavy queries.

To illustrate this methodology, consider a scenario in Contract Management. If a user queries the RAG Agent with "What are the key deliverables in contract 123/2024?", the dynamic prompt might be constructed as: "Retrieve the sections from contract 123/2024 that outline the key deliverables, focusing on deadlines and specific tasks outlined for the supplier." This tailored instruction ensures the LLM focuses only on the relevant information, enhancing response accuracy. Alternatively, if a user queries the SQL Agent with "How many contracts are currently active with supplier ABC?", the prompt is dynamically crafted as: "Generate an SQL query to count all active contracts with supplier ABC and return the result." For queries requiring visual representation, such as "What is the total expenditure on contracts over the last year?", the system dynamically generates a prompt to present the data as a line graph, offering a clear visualization of expenditure trends.

By dynamically adapting prompts to the needs of each agent—whether extracting information, executing precise SQL commands, or generating visual insights—our system

ensures that responses are contextually accurate, actionable, and user-friendly. This comprehensive approach not only enhances the versatility of the question-answer system but also improves user experience by delivering tailored outputs that align with complex and varied query requirements.

4 Evaluation

The architecture depicted in the figure represents the implementation of our multi-source question-answer methodology, combining structured and unstructured data from contracts. The system is built using a modular approach, where each component plays a critical role in the data retrieval and response generation process. At the core of the architecture is the User Interface, built with Streamlit, as shown in figure 6, which allows users to input their queries and view responses in a user-friendly interface. Users can submit both broad questions or specific contract-related queries, which are then processed by the backend system.

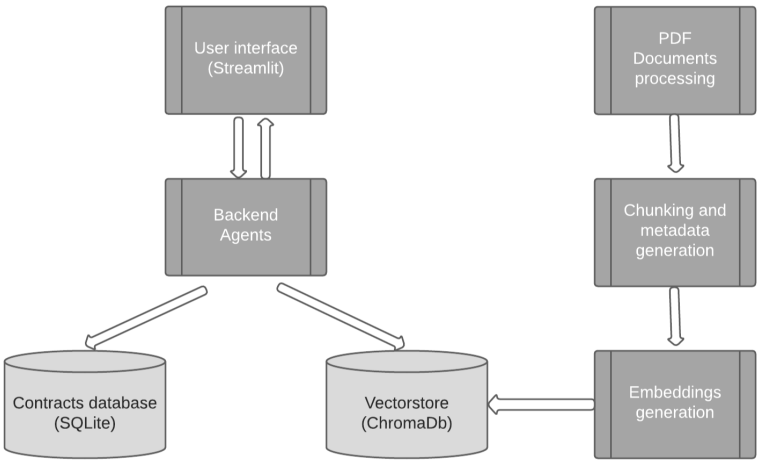

Fig. 5. Application architecture. Source: [Seabra et al., 2024a]

The Backend Agents act as the decision-making layer of the system, handling queries based on their type and content. These agents include the Router Agent, which determines whether to route the query to the RAG Agent (for unstructured text retrieval) or the SQL Agent (for structured data queries using Text-to-SQL). The agents communicate bidirectionally with the user interface, allowing for interactive feedback during the query resolution process.

For the unstructured data flow, contract documents in PDF format undergo processing in the PDF Documents Processing component. This involves extracting text and metadata from the documents, which is then passed to the Chunking and Metadata Generation module. This module divides the documents into manageable chunks, enriching them with metadata for easier retrieval. These chunks are further processed through the Embeddings Generation component, where each chunk is transformed into a high-dimensional vector representation using an embedding model. These embeddings are stored in the Vectorstore (implemented using ChromaDB) for efficient similarity search during retrieval.

On the structured data side, the Contracts Database (implemented using SQLite) stores relevant contract data such as specific terms, clauses, dates, and financial informa-

tion. When a query requires precise data retrieval, such as asking for contract values or deadlines, the SQL Agent retrieves the necessary information directly from this database.

By integrating both the vectorstore and structured database, the Backend Agents can provide comprehensive answers to user queries, dynamically choosing the most appropriate data source based on the type of question. This hybrid approach ensures that the system can handle both semantically complex queries and direct database queries, offering flexible and accurate responses.

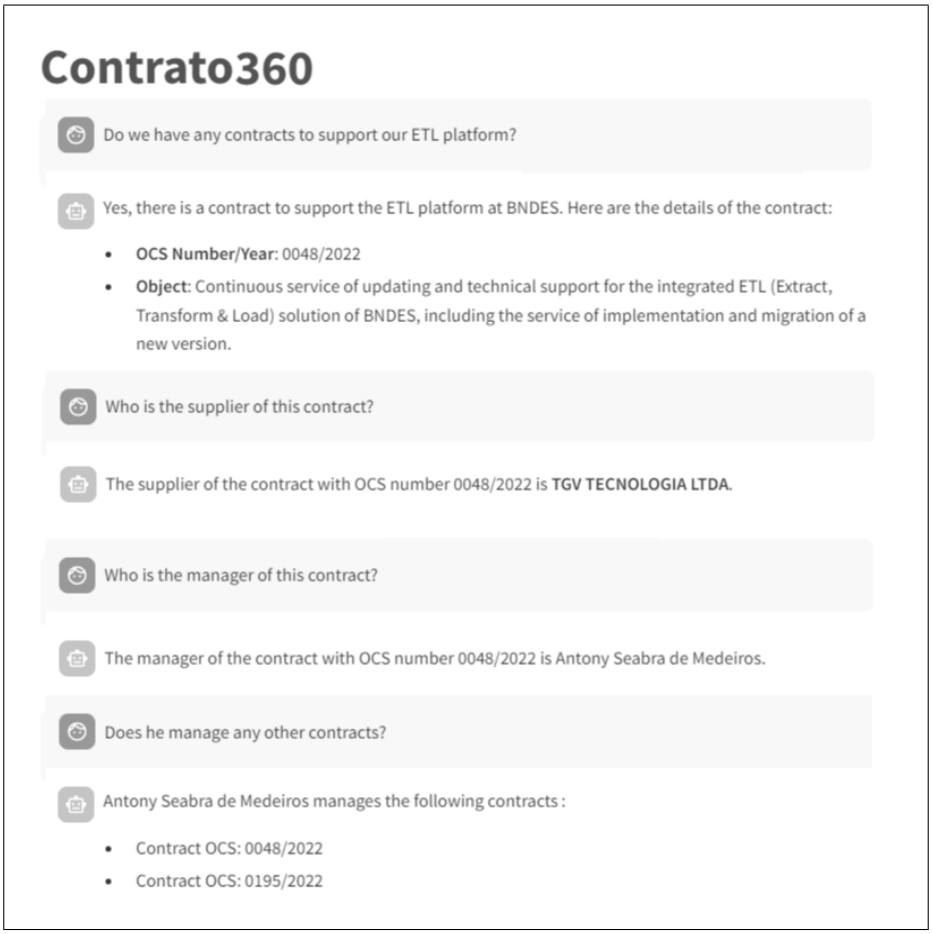

Fig. 6. Contracts Q&A Streamlit application

The system was evaluated through experiments conducted by specialists from BNDES (Social and Economic Development Bank of Brazil), who validated its performance using a set of 75 contracts. These contracts, including both PDFs and associated metadata, were processed to assess the system's ability to retrieve relevant information from both unstructured documents and structured data. To evaluate the system's effectiveness in answering various query types, a set of benchmark questions was developed, divided into two categories: direct and indirect questions.

Direct questions refer to those that could be answered using information directly available in the contract PDFs and their metadata. Examples include questions about contract subjects, suppliers, managers, and contract terms. The results demonstrated that for these direct questions, the system consistently provided complete and relevant responses, meeting the users' expectations for accuracy and comprehensiveness.

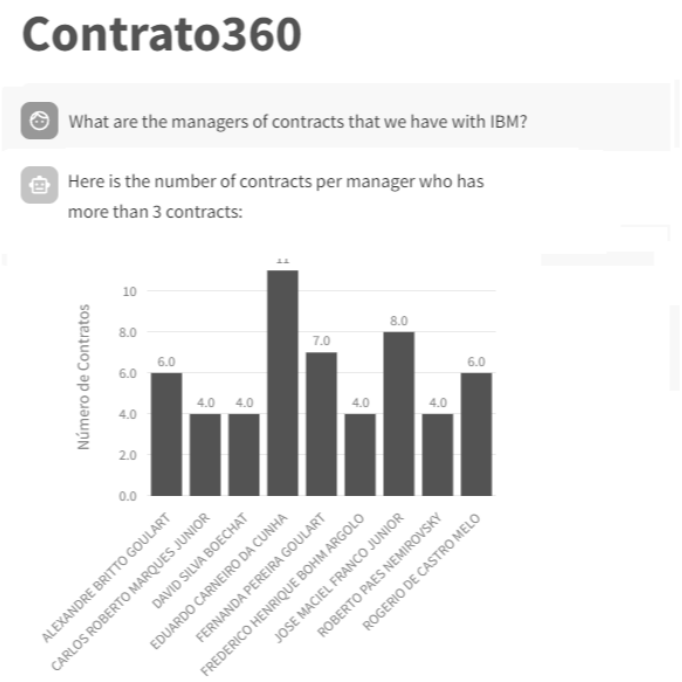

Fig. 7. Plotly Agent

Indirect questions, however, required information that would yield better relevance when retrieved from the database. Examples include questions about the number of active contracts, upcoming contract expirations, and specific details regarding exemptions from tender processes. The results for these indirect questions were generally satisfactory, although in certain cases, such as questions about contract inflexibility and exemptions, the answers provided were marked as incomplete. This is likely due to the more complex semantics of the terms involved. For example, the term "Waiver of Bidding" proved challenging for the system, as its meaning was not fully captured in the retrieval process. Adjustments to the prompts or query structure are expected to improve the system's ability to interpret and respond accurately to these nuanced questions.

User feedback highlighted that one of the system's most valuable features is its ability to seamlessly integrate information from both the structured data store and the unstructured text in contracts. This feature significantly reduces the time users spend locating and accessing relevant contract data, as they would typically need to identify the contracts, open the PDFs, and manually search for information. For instance, the system efficiently retrieves answers regarding contract managers and outlines any penalties related to contractual non-compliance, eliminating the need for users to sift through lengthy documents. By directly addressing questions with specific details, the system enhances the user experience, providing critical information quickly and effectively.

Additionally, users appreciated the system's capacity to automatically generate visual summaries through its Plotly agent when a table of values was included in the response. This feature was positively received, as it not only provides immediate visual insights but also supports users in preparing professional presentations. By integrating dynamic graph generation directly into the response process, the system offers users a more comprehensive analytical experience, enabling clearer communication and a deeper understanding of contract-related data.

5 Conclusions and Future Work

In this work, we presented a comprehensive multi-source question-answer system that integrates unstructured text from contract documents with structured data from relational databases. By employing a combination of Retrieval-Augmented Generation (RAG), Text-to-SQL techniques, and dynamic prompt engineering, we demonstrated how our system

efficiently retrieves relevant information from diverse data sources to provide precise and contextually accurate responses. The use of backend agents, particularly the Router Agent, allowed for a flexible and adaptive workflow where queries are dynamically routed to the appropriate processing module—whether that be the RAG agent for text-based retrieval or the SQL agent for direct database queries.

The 6 demonstrates the ability of Contrato360 in retrieving and summarizing contract information related to Oracle through a question-and-answer interface. Our implementation, which includes the use of ChromaDB as the vectorstore for storing document embeddings and SQLite for managing contract data, ensures that the system can handle complex legal documents while maintaining real-time performance in answering user queries. The combination of these technologies enables the system to provide a seamless experience where both structured and unstructured data are processed cohesively, offering a unified approach to contract management and information retrieval.

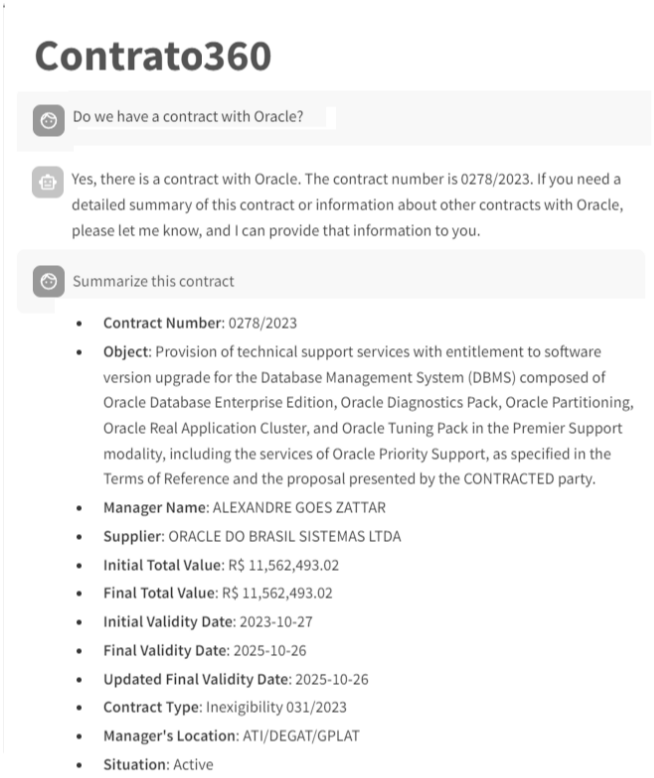

Fig. 8. Contract Summarization

Despite the success of our approach, there remain several areas for future development. One significant avenue for improvement is the further refinement of the Router Agent. Currently, it relies on predefined regular expressions to route queries, but integrating machine learning models to dynamically adapt and learn from query patterns could increase the precision and flexibility of the system. Additionally, expanding the system's capability to handle a wider variety of legal documents and domains, beyond contract management, would provide greater scalability and versatility.

Another important direction for future work involves improving the system's interaction with graph-based data. We have already implemented a Graph Agent to visualize data using bar graphs, but incorporating more advanced data visualizations, such as time-series analysis or multi-dimensional comparisons, would provide users with deeper insights into the retrieved data. Moreover, enhancing the chunking strategy for document segmentation and metadata generation could mitigate the issue of misalignment between query intent and retrieved text, especially for more complex and ambiguous legal queries.

Finally, while our current system integrates effectively with contract documents and databases, there is potential to expand its multi-source retrieval capabilities by incorporating external data sources such as APIs, web services, or even real-time data streams. This would provide users with even more comprehensive and up-to-date information.

In conclusion, while our system already demonstrates significant advancements in combining text-based and structured data retrieval for question-answer tasks, the ongoing development of more sophisticated routing, visualization, and data integration techniques will further enhance its capabilities and application across different domains.

References

[cre, 2024] (2024). Crewai agents framework. `https://www.crewai.com/`. Accessed: 2024-12-30.

[Chen et al., 2024] Chen, J., Lin, H., Han, X., and Sun, L. (2024). Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17754–17762.

[Gao et al., 2023a] Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., and Zhou, J. (2023a). Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

[Gao et al., 2023b] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., and Wang, H. (2023b). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

[Giray, 2023] Giray, L. (2023). Prompt engineering with chatgpt: a guide for academic writers. *Annals of biomedical engineering*, 51(12):2629–2633.

[Jeong, 2023] 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*.

[Jin et al., 2024] Jin, H., Huang, L., Cai, H., Yan, J., Li, B., and Chen, H. (2024). From llms to llm-based agents for software engineering: A survey of current, challenges and future. *arXiv preprint arXiv:2408.02479*.

[Langchain, 2024] Langchain (2024). Langchain retrievalqa documentation. `https://api.python.langchain.com/en/latest/chains/langchain.chains.retrieval\_qa.base.RetrievalQA.html`. Accessed: 2024-03-01.

[Lewis et al., 2020] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

[Liu et al., 2023] Liu, A., Hu, X., Wen, L., and Yu, P. S. (2023). A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *arXiv preprint arXiv:2303.13547*.

[Mialon et al., 2023] Mialon, G., Dessì, R., Lomeli, M., Nalmpantis, C., Pasunuru, R., Raileanu, R., Rozière, B., Schick, T., Dwivedi-Yu, J., Celikyilmaz, A., et al. (2023). Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*.

[OpenAI, 2023a] OpenAI (2023a). Chatgpt fine-tune description. `https://help.openai.com/en/articles/6783457-what-is-chatgpt`. Accessed: 2024-03-01.

[OpenAI, 2023b] OpenAI (2023b). Chatgpt prompt engineering. `https://platform.openai.com/docs/guides/prompt-engineering`. Accessed: 2024-04-01.

[Pinheiro et al., 2023] Pinheiro, J., Victorio, W., Nascimento, E., Seabra, A., Izquierdo, Y., Garcıa, G., Coelho, G., Lemos, M., Leme, L. A. P. P., Furtado, A., et al. (2023). On the construction of database interfaces based on large language models. In *Proceedings of the 19th International Conference on Web Information Systems and Technologies - Volume 1: WEBIST*, pages 373–380. INSTICC, SciTePress.

[Seabra et al., 2024a] Seabra, A., Cavalcante, C., Nepomuceno, J., Lago, L., Ruberg, N., and Lifschitz, S. (2024a). Contrato360 2.0: A document and database-driven question-answer system using large language models and agents. In Coenen, F., Fred, A., and Bernardino, J., editors, *Proceedings of the 16th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, IC3K 2024, Volume 1: KDIR, Porto, Portugal, November 17-19, 2024*, pages 167–178. SCITEPRESS.

[Seabra et al., 2024b] Seabra, A., Cavalcante, C., Nepomuceno, J., Lago, L., Ruberg, N., and Lifschitz, S. (2024b). Dynamic multi-agent orchestration and retrieval for multi-source question-answer systems using large language models. *arXiv preprint arXiv:2412.17964*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[Wang et al., 2023] Wang, M., Wang, M., Xu, X., Yang, L., Cai, D., and Yin, M. (2023). Unleashing chatgpt's power: A case study on optimizing information retrieval in flipped classrooms via prompt engineering. *IEEE Transactions on Learning Technologies*.

[White et al., 2023] White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., and Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*.

Authors

Antony Seabra is an IT executive at BNDES, the Development Bank of Brazil, where he leads the Data Engineering team. He received his Master's Degree in Computer Science (Databases) from PUC-Rio, Brazil, in 2017, and he is currently pursuing his PhD in Computer Science at PUC-Rio under the guidance of Prof. Sérgio Lifschitz. His research interests focus on Databases and their integration with Artificial Intelligence and Natural Language Processing.

Claudio Cavalcante is a Data Engineer at BNDES with a solid academic background. He is currently pursuing his Master's Degree in Computing at PUC-Rio, Brazil, under the guidance of Prof. Sérgio Lifschitz. His research interests lie in Artificial Intelligence and Natural Language Processing.

João Nepomuceno received his Bachelor's Degree in Physics from Universidade Federal Fluminense, Brazil, and he is currently pursuing his Bachelor's Degree in Computer Science at Universidade Federal Fluminense, Brazil. His research interests include Data Engineering, Artificial intelligence and Natural Language Processing.

Lucas Lago is currently pursuing his Bachelor's Degree in Computer Science at Universidade do Estado do Rio de Janeiro, Brazil. His research interests include Artificial Intelligence and Natural Language Processing.

Nicolaas Ruberg is a Data Engineer at BNDES with a solid academic background. He holds a Bachelor's in Computer Science from the Universidade Federal da Paraíba in Brazil. He further enhanced his expertise by earning a Master's Degree in Distributed Databases from the Universidade Federal do Rio de Janeiro and later a Master's in Artificial Intelligence from the University of Bologna in Italy. His research interests include Databases, Artificial Intelligence, and Natural Language Processing.

Sérgio Lifschitz is an Associate Professor at PUC-Rio with a research emphasis in Databases. He received his Bachelor's Degree in Electrical Engineering (1986) and Master's Degree in the same field (1987) from PUC-Rio and completed his PhD in Computer Science at the École Nationale Supérieure des Télécommunications (ENST Paris) in 1994.