

Beyond Traditional Retrieval Systems: Leveraging AI with Documents, Knowledge Graphs and Databases

Antony Seabra, Claudio Cavalcante, and Sergio Lifschitz

PUC-Rio - Pontifical Catholic University of Rio de Janeiro, Brazil

Abstract. This study explores techniques for retrieving data from documents, knowledge graphs, and databases using Large Language Models (LLMs), specifically leveraging OpenAI's GPT models as foundational frameworks for embeddings and conversational models in question-answering (QA) systems. Our research focuses on the utilization of Prompt Engineering, Retrieval-Augmented Generation (RAG), and Text-to-SQL techniques to effectively extract information from these diverse data sources without the need for model retraining. A key aspect of our study is the emphasis on explainability, demonstrating how these techniques can reveal the rationale behind retrieved information and enhance the understanding of results. We highlight the challenges encountered in specific use cases during our tests and present effective strategies and solutions to overcome them. Our findings demonstrate the potential of LLMs to surpass traditional search and retrieval systems, paving the way for more efficient and comprehensible information systems.

Keywords: Information Retrieval, AI, Explainability, Documents, Knowledge Graphs, Databases, Recommendation System

1 Introduction

In the rapidly evolving landscape of Artificial Intelligence (AI), the integration of diverse data sources with Large Language Models (LLMs) represents a significant leap forward in the realm of information retrieval and question-answering systems.

Large Language Models, like the ones in GPT family, are the forefront of natural language processing, revealing their ability to understand, generate and even dialogue in a human-like way. Trained on high volumes of text data, these models can discern language patterns and nuances, enabling them to respond to queries, compose coherent passages and even mimic stylistic prose. The strength of LLMs lies in their capacity to process and generate language in a way that feels intuitive to users, bridging the gap between human communication and machine understanding.

The aim of this study is to explore and demonstrate how the integration of diverse data sources - such as documents, knowledge graphs and databases - can significantly augment the retrieval capabilities of LLMs to create a more efficient and comprehensive question-answering system. By leveraging techniques like Prompt Engineering, Retrieval-Augmented Generation (RAG) and Text-to-SQL, the study seeks to enhance the LLM's ability to access and synthesize information from various formats and structures of data, without the need for retraining the model on additional datasets. Specifically, the paper addresses three problems:

P1. How can we answer a question by synthesizing information from multiple documents of a uniform type contained within PDF files?

P2. How can a response to a query be formulated utilizing RDF (Resource Description Framework) triples derived from Knowledge Graphs?

P3. How can a response to a query be formulated through the retrieval of information from Relational Databases?

The paper is structured as follows: Section 2 presents a technical background on Large Language Models and the techniques used to integrate information from various data sources. Section 3 discusses related work in the field. Our methodology is outlined in Section 4. Section 5 investigates documents as a data source for enhancing the capabilities of LLMs, while Section 6 delves into the integration of Knowledge Graphs with LLMs. Furthermore, Section 7 explores databases as a potential data source. Finally, Section 8 derives some conclusions from our work and suggests future research directions in this field.

2 BACKGROUND

2.1 Large Language Models

Large Language Models (LLMs) have revolutionized the field of natural language processing (NLP) with their ability to understand and generate human-like text. At the heart of the most advanced LLMs is the Transformers architecture, a deep learning model introduced in the seminal paper *Attention Is All You Need* by [30]. Transformers leverage a mechanism called *attention*, which allows the model to weigh the influence of different parts of the input data at different times, effectively enabling it to focus on relevant parts of the text when making predictions.

Prior to Transformers, Recurrent Neural Networks (RNNs) and their variants like Long Short-Term Memory (LSTM) networks were the standard in NLP. These architectures processed input data sequentially, which naturally aligned with the sequential nature of language. However, they had limitations, particularly in dealing with long-range dependencies within text due to issues like vanishing gradients [23]. Transformers overcome these challenges by processing all parts of the input data in parallel, drastically improving the model's ability to handle long-distance relationships in text.

Chat models, a subset of LLMs, are specialized in generating conversational text that is coherent and contextually appropriate. This specialization is achieved through the training process, where the models are fed vast amounts of conversational data, enabling them to learn the nuances of dialogue. ChatGPT, for instance, is fine-tuned on a dataset of conversational exchanges and it was optimized for dialogue by using Reinforcement Learning with Human Feedback (RLHF) - a method that uses human demonstrations and preference comparisons to guide the model toward desired behavior [21].

The transformative impact of LLMs, and particularly those built on the Transformers architecture, has been profound. By moving away from the constraints of sequential data processing and embracing parallelization and attention mechanisms, these models have set new standards for what is possible in the realm of NLP. With the ability to augment generation with external data or specialize through fine-tuning, LLMs have become not just tools for language generation but platforms for building highly specialized, knowledge-rich applications that can retrieve information in a dialogue-like way, find useful information and generate insights for decision making.

2.2 Prompt Engineering

Prompt engineering is the art of designing and optimizing prompts to guide LLMs in generating desired outputs. The goal of prompt engineering is to maximize the potential of LLMs by providing them with instructions and context [22].

In the realm of prompt engineering, instructions are the crucial first steps. Through them, engineers can detail the roadmap to an answer, outlining the desired task, style and

format for the LLM's response [34]. For instance, To define the style of a conversation, a prompt could be phrased as "Use professional language and address the client respectfully" or "Use informal language and emojis to convey a friendly tone". To specify the format of dates in answers, a prompt instruction could be "Use the American format, MM/DD/YYYY, for all dates".

On the other hand, context refers to the information provided to LLMs alongside the core instructions. The most important aspect of a context is that it can provide information that supports the answer given by the LLM, and it is very useful when implementing question-answering systems. This supplemental context can include relevant background details, specific examples, data points or even previous dialogue exchanges, which collectively help the model to generate more accurate, detailed and contextually appropriate responses. According to [32], prompts provide guidance to ensure that ChatGPT generates responses aligned with the user's intent. As a result, well-engineered prompts greatly improve the efficacy and appropriateness of ChatGPT's responses.

2.3 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation represents a paradigm shift in the way LLMs process and generate information. The fundamental step behind RAG involves the use of a vector store to retrieve text chunks similar to the input query [6]. This process employs sophisticated algorithms to convert both the query and the database of potential sources into high-dimensional vectors, enabling the model to efficiently identify and select information that is contextually relevant to the input, which then informs and enriches the subsequent generative process.

One of the advantages of this technique is that we can provide additional data to the LLM without retraining the model [16]. By chunking the available text into manageable pieces and embedding these chunks into high-dimensional vector spaces, one can enable the rapid retrieval of contextually relevant information in response to a query, laying the foundational groundwork for the model's subsequent retrieval and generation tasks.

When it comes to the retrieval phase, the user's query is embedded into the same high-dimensional vector space as the pre-processed text chunks stored in the vector store; this enables a direct comparison between the embedded query and the vectors of stored chunks, allowing the system to retrieve the most similar chunks which are then seamlessly integrated into the context of the LLM for generating informed and relevant responses [25].

Furthermore, the integration of RAG with prompt engineering represents a sophisticated approach to AI-driven responses, where RAG's ability to fetch contextually relevant information is combined with the nuanced control of prompt engineering to direct LLMs towards generating highly context-aware answers, thereby enhancing the overall effectiveness and precision of conversational AI systems.

2.4 Text-to-SQL

Text-to-SQL is a technology that enables the conversion of natural language queries into SQL commands based exclusively on the database schema, eliminating the need for knowledge of the underlying data. This approach leverages the capabilities of large language models to understand and interpret human language, thus allowing users to retrieve, update or manipulate data in databases through simple text inputs, without the need for specialized knowledge of SQL syntax [7].

By translating natural language into SQL queries, Text-to-SQL bridges the gap between complex database structures and end-users, making data access more intuitive and efficient. This technique is particularly useful in environments where quick data insights are needed, enabling non-technical users to extract valuable information from databases by simply asking questions in plain language. It enhances data accessibility, reduces the learning curve associated with database querying, and significantly speeds up data analysis processes, thereby empowering a broader range of users to make data-driven decisions.

The main distinction between RAG and Text-to-SQL techniques lies in their underlying approach to fetching relevant information in response to user queries. RAG relies on retrieving text segments from a vector store that are similar to the user's question, using these segments to generate a coherent and contextually relevant answer. This method is effective for questions where the answer can be synthesized from existing text, but it may not always pinpoint the precise data needed. On the other hand, Text-to-SQL translates natural language queries into SQL commands, as shown in [24], which are then executed against a structured database to fetch exact data matches. This ensures that if the translation from text to SQL is accurate, the user receives a highly specific and accurate response directly from the relevant data fields in the database.

Thus, while RAG operates on the principle of textual similarity and generative capabilities, Text-to-SQL provides a more precise mechanism for data retrieval by executing queries that directly correspond to the user's intent, making it particularly effective for detailed and specific data inquiries.

3 RELATED WORK

The integration of LLMs with advanced data retrieval techniques represents a burgeoning area of research, aiming to enhance the efficiency and accuracy of information extraction and question-answering systems. This section reviews pertinent literature that explores the utilization of Prompt Engineering, RAG and Text-to-SQL in conjunction with LLMs.

Prompt engineering has been increasingly recognized for its potential to significantly improve the performance of LLMs by instructing them to behave differently from their default. [34] demonstrated how carefully designed prompts can enable more precise responses from LLMs across a range of tasks, underscoring the importance of prompt design in leveraging model capabilities. According to the authors, prompt patterns significantly enrich the capabilities that can be created in a conversational LLM. Indeed, this approach is essential for guiding LLMs to understand and respond to queries more effectively, by encapsulating the query within a context that the model is more likely to comprehend and respond to accurately. [8] states that, by employing prompt engineering techniques, academic writers and researchers can unlock the full potential of language models, harnessing their capabilities across various domains, and that this discipline opens up new avenues for improving AI systems and enhancing their performance in a range of applications, from text generation to image synthesis and beyond.

[8] presents the prompt components that can be manipulated by engineers to guide text generation of an LLM. These components include an instruction, a context, an input data and an output indicator. Instruction outlines what the LLM is expected to do, providing clear directions to guide the model's response. The context gives background information necessary for the model to generate relevant and informed responses. The input data refers to the actual data fed into the model for processing, like a question, an image or a set of data points. And the output indicator tells the model how to format its response and what type of output is expected. In fact, as we mentioned earlier in 2.3, the context component is

crucial in the realm of Retrieval-Augmented Generation (RAG) as it enriches the model's understanding, allowing it to fetch and integrate the most relevant external information to produce accurate and contextually appropriate responses.

In the context of enhancing retrieval capabilities of LLMs, Retrieval-Augmented Generation (RAG) emerges as a relevant technology. According to [4], LLMs face significant challenges such as factual hallucination, outdated knowledge and a lack of domain-specific expertise. One effective solution to these issues is the use of RAG, which incorporates external knowledge through information retrieval. Numerous works are investing in RAG to augment the retrieval domain of LLMs, emphasizing its importance in leveraging the extensive training and capabilities of these models. This approach not only mitigates the inherent limitations of LLMs by providing access to real-time, updated information but also maximizes the utility of their pre-trained knowledge, ensuring more accurate and contextually relevant outputs.

[6] introduced a novel framework for augmenting the generative capabilities of LLMs with information retrieved from a large corpus of documents. Their work illustrates how RAG can dynamically incorporate external information into the model's responses, thereby enriching the content and relevance of answers provided by the LLM. This technique not only enhances the quality of responses but also extends the knowledge base of LLMs beyond their initial training data. On the other hand, [5] presents a framework to generate relevant documents that simultaneously exploits parametric and nonparametric knowledge. The authors use large language models to comprehensively understand retrieved documents to generate new documents for answering questions.

The conversion of natural language queries into SQL commands for database querying represents a critical step forward in making databases more accessible to non-expert users. [18] presented a deep learning approach to Text-to-SQL, which significantly advances the field by enabling more accurate and complex queries to be constructed from natural language inputs. Their research highlights the potential for such techniques to bridge the gap between natural language processing and database management systems, facilitating direct and efficient access to stored information. This approach not only democratizes data access by simplifying user interaction with databases but also enhances the analytical capabilities of users by allowing them to formulate intricate queries without needing extensive SQL expertise. Consequently, this development could lead to broader adoption and more innovative uses of database technology across various sectors.

Recent studies have begun to explore the synergistic integration of these techniques with LLMs to create more sophisticated question-answering systems. For instance, [11] reinforce the importance of using Prompt Engineering with RAG to improve the retrieval of relevant documents, which are then used to generate responses that are both contextually relevant and information-rich. Similarly, [7] explored the integration of Text-to-SQL with Prompt Engineering to enhance the model's ability to interact with relational databases directly, thereby expanding the scope of queries that can be accurately answered.

In summary, the related work in this field underscores the potential of combining LLMs with advanced retrieval and querying techniques to create more powerful and versatile information extraction systems. These developments not only extend the capabilities of LLMs but also open up new avenues for research and application in the domain of artificial intelligence and natural language processing.

Despite significant advancements in Prompt Engineering, RAG, and Text-to-SQL for improving LLMs, there is still a considerable gap in the literature concerning the effective use of these techniques across varied information sources such as documents, knowledge graphs, and databases. Existing research tends to focus primarily on demonstrating these

techniques and their applications but often overlooks discussing diverse strategies for deploying them on real-world data sources. This gap poses substantial challenges in fully utilizing the available data spectrum, which is crucial for generating more accurate and contextually relevant responses. In fact, combining information from several documents, knowledge graphs and databases can really enrich the data context and information retrieval systems.

4 DOCUMENTS

The process of making data contained within PDF documents accessible to LLMs involves a series of steps designed to translate unstructured text into a format that can be efficiently processed and queried by these models. The initial step involves reading the textual content from PDF documents into manageable chunks, which are then embedded into high-dimensional vectors. This embedding process transforms the text into a numerical format that captures the semantic properties of the text, facilitating more effective information retrieval. Nowadays, embeddings often utilize a substantial number of dimensions, with 1536 being a common example.

Once the text chunks are embedded, these vectors are stored in a vector store - a specialized database designed to handle high-dimensional data. The vector store allows for efficient querying of vectors based on similarity metrics, such as cosine or euclidean distance. To answer a question, the query itself is first embedded into the same vector space as the document chunks. This embedded question vector is then compared against the vectors in the vector store to identify the chunks with the highest semantic similarity to the query. That's why the choice of chunking strategy is so important, as it directly influences the granularity of the information that will be retrieved. A well-devised chunking strategy ensures that each chunk is coherent and contextually complete, thereby allowing the embedded vectors to capture the essence of the document's content meaningfully.

A classical and widely adopted chunking strategy involves segmenting documents based on a specified number of tokens, often with an added overlap quota. This approach, known as overlapping token chunking, segments the document into blocks of text containing a fixed token count, ensuring that each block is not entirely discrete but shares some portion of its content with adjacent blocks. On the other hand, if the documents are neatly organized into distinct sections, like the one in figure 1, and the most frequent questions can be answered by the texts contained in these sections, then a viable chunking strategy would likely transform each section in a separate chunk [26]. This approach ensures that each chunk represents a complete thematic unit, preserving the context and facilitating more precise retrievals that align closely with the user's query.

One key aspect in RAG is the difference between similarity and relevance. Similar chunks may not contain the relevant information to answer a query, and this is a challenge when designing systems to accurately retrieve information, especially in cases where data comes from several documents of an uniform type. In such contexts, documents may share a high degree of structural and lexical similarity, making it difficult for retrieval algorithms to distinguish between content that is merely similar in form and content that is truly relevant to the specific inquiry. Consider documents provided by [35], like [36] and [37], describing key facts and characteristics about diseases, and the inquiry, "What are the key facts about Dengue?". In instances where the vector store contains multiple segments pertaining to "key facts," there is a possibility that a segment unrelated to Dengue might be retrieved. To address this potential issue, it is imperative to employ supplementary strategies, like the use of metadata or document hierarchy.

The figure shows two side-by-side screenshots of World Health Organization (WHO) pages. The left page is titled 'Coronavirus disease (COVID-19)' and is dated 9 August 2023. It features a 'Key facts' section with bullet points about the virus, symptoms, and vaccine administration. Below this is an 'Overview' section. The right page is titled 'Dengue and severe dengue' and is dated 17 March 2023. It also features a 'Key facts' section with bullet points about the virus, transmission, and symptoms, followed by an 'Overview' section. Both pages include language selection buttons for Arabic, Chinese, French, Russian, and Spanish.

Fig. 1. Temporal Contextualized Claims about Brazil. Source: Authors

The provided code snippet below outlines the basic structure of a Python class named `Document`, designed to encapsulate information about a document. The class constructor initializes a new instance of the `Document` class with two key attributes: `page_content` and `metadata`.

```
class Document:
    def __init__(self, page_content, metadata):
        self.page_content = page_content
        self.metadata = metadata

    def create_document(page_content, metadata):
        return Document(page_content, metadata)
```

Following the creation of the `Document` class, we iterate over the sections of each document, extract text from each section along with its hierarchical structure, and then create document objects with associated metadata for each section.

```
for section in doc.sections():
    section_text = section.to_text(
        include_children=True, recurse=True)
    metadata = {"chunk": ind, "source": file_name,
               "disease": doc_title, "section": section.title}
    doc = create_document(section_text, metadata)
    docs.append(doc)
    ind = ind + 1
```

In this manner, each segment is associated with three specific metadata elements in addition to its index: source, disease, and section. When querying the database for key facts about Covid-19, the vector store utilizes the disease and section metadata to selectively filter segments, thereby retrieving only those that are pertinent to the inquiry.

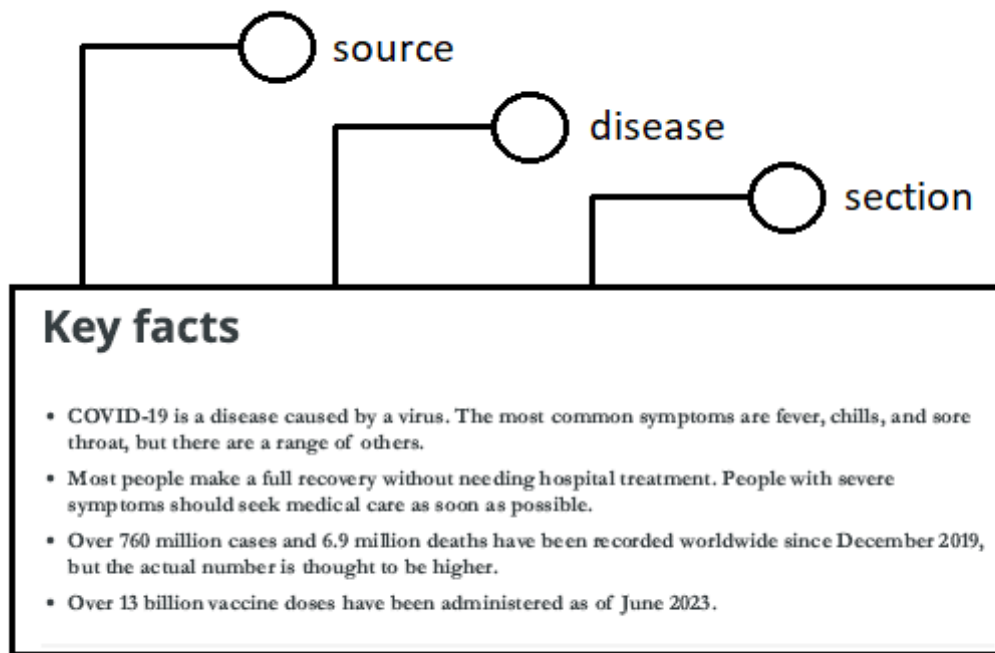


Fig. 2. Temporal Contextualized Claims about Brazil. Source: Authors

After that, we initialize a `vectordb` object using the Chroma library. This method is designed to create a vector database (or vector search index) from a collection of documents by leveraging embeddings generated by a model, in our case provided by the `OpenAIEmbeddings` object.

```
model_name = 'text-embedding-ada-002'
embeddings = OpenAIEmbeddings(model=model_name,
                               openai_api_key=OPENAI_API_KEY)
vectordb = Chroma.from_documents(docs,
                                 embeddings,
                                 persist_directory=persist_directory,
                                 collection_metadata={"hnsw:space": "cosine"})
```

Finally, we instantiate a `qa` object for question answering through the `RetrievalQA` module of the `langchain` library [13]. This module employs the RAG approach, wherein, upon receiving a query, retrieves the most similar chunks from the vector store and passes these chunks to the LLM to answer the question. The provided code snippet illustrates the execution of a query to retrieve key facts about COVID-19. In this context, the metadata associated with the specific textual segment plays a crucial role in facilitating the accurate retrieval of the pertinent answer.

```
qa = RetrievalQA.from_chain_type(llm=llm,
                                 chain_type="stuff",
                                 retriever=vectordb.as_retriever(),
                                 chain_type_kwargs=chain_type_kwargs)
```

The response generated aligns accurately with the 'Key facts' section of the designated COVID-19 document, notwithstanding the existence of analogous 'Key facts' sections pertinent to other subjects that might also correlate with the query.

```

query = 'What are the key facts about Covid-19?'
qa.run(query)s)
COVID-19 is a disease caused by a virus.
The most common symptoms include fever, chills, and sore throat.
Most people recover without needing hospital treatment.
There have been over 760 million recorded cases and 6.9 million deaths worldwide, but the
actual numbers may be higher.

```

In the next provided code snippet, a query is posed to a question answering system, specifically inquiring about the common symptoms shared between COVID-19 and Dengue. The query, "What most common symptoms do Covid-19 and Dengue share?", is processed by the `qa.run` function, which is likely part of a sophisticated question-answering framework designed to extract and synthesize information from a diverse range of documents. The output, "The most common symptoms shared by COVID-19 and dengue are fever and headache", indicates that the system successfully retrieved and consolidated information from at least two distinct documents—one pertaining to COVID-19 and the other to Dengue.

```

query = 'What most common symptoms do Covid-19 and Dengue share?'
qa.run(query)

The most common symptoms shared by COVID-19
and Dengue are fever and headache.

```

As a final example, the language model's response to the query below, about treatment and vaccines for COVID-19, Dengue and HIV [38], illustrates its ability to retrieve and synthesize information across several different chunks.

```

query = 'Summarize treatments and vaccines for
Covid-19, Dengue and HIV.'
qa.run(query)

Treatments for Covid-19 include antiviral medications, such as remdesivir, and supportive
care to manage symptoms. Vaccines, such as those developed by Pfizer-BioNTech, Moderna and
Johnson & Johnson, are available to prevent Covid-19.

There is no specific antiviral treatment for Dengue. Supportive care, such as rest, fluid
intake, and pain relievers, is provided to manage symptoms. Prevention is key, and vector
control measures, such as mosquito control, are important in preventing Dengue.

There is no cure for HIV, but antiretroviral therapy (ART) is used to manage the infection
and prevent progression to AIDS. ART involves taking a combination of antiretroviral drugs.
There is no vaccine available for HIV.

```

LLM's prompts often have a token limitation, which restricts the amount of text that can be processed at once, potentially hindering the analysis of lengthy documents in a single session [33]. If there is a high number of chunks to be inserted in the prompt's context, this may be a problem but, with the increasing token limits of large language models, users can now handle more comprehensive data sets and perform more detailed analyses without needing to overly segment the text.

5 KNOWLEDGE GRAPHS

Integrating data from Knowledge Graphs into LLMs represents a critical advancement in enhancing the LLM's ability to provide contextually rich and accurate responses. This section delineates the process from the acquisition and preparation of data, in the form

of Resource Description Framework (RDF) triplets, and explores how these triplets are effectively utilized within a prompt to augment the LLM's response capabilities.

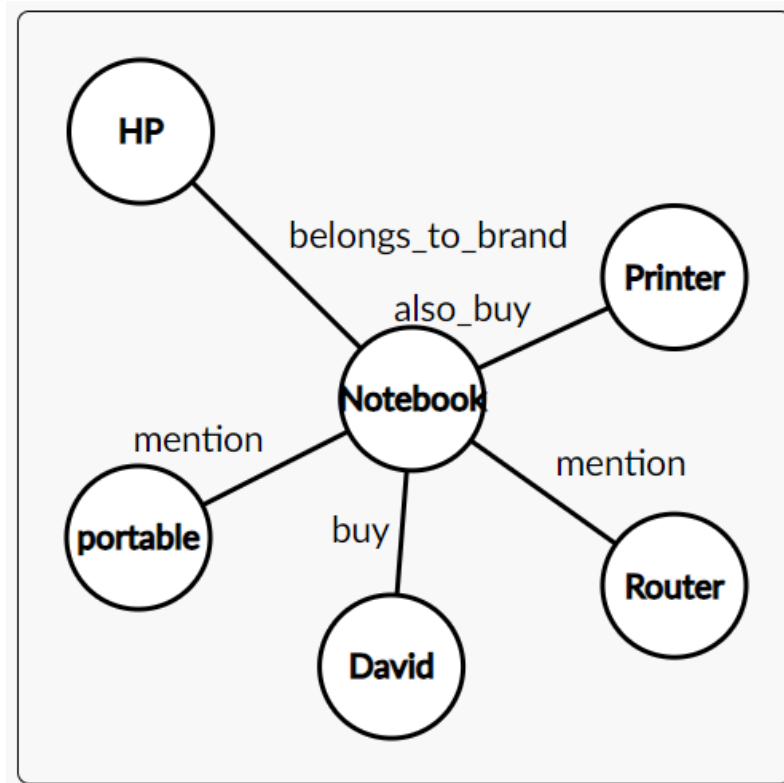


Fig. 3. Temporal Contextualized Claims about Brazil. Source: Authors

Once the data is prepared, the challenge is to format this information in a manner that can be seamlessly integrated into a prompt for the LLM. This involves creating a textual representation of the RDF triplets that is both comprehensible to the LLM and capable of providing the necessary context for answering a given question. The formatting process includes the conversion of RDF triplets into natural language sentences or structured statements that retain the semantic relationships encoded in the RDF format. This step ensures that the rich semantic information contained within the Knowledge Graph is preserved and made accessible to the LLM.

As an example, consider an interactive network of consumer behavior, which maps purchases, mentions and product categories as shown in figure ???. The code snippet below demonstrates how to format data extracted from the RDF triplets associated with the graph, preparing it to be presented subsequently in a LLM's prompt.

```

triplets = g.find("(src)-[rel]->(dst)")
triplets_list = triplets.collect()
formatted_triplets = "\n".join(["{}-{}->{}"
    .format(row.src, row.rel, row.dst)
    for row in triplets_list])
  
```

The variable `formatted_triplets` contains a structured representation of relationships between different entities, specifically illustrating interactions such as purchases and associations between users and products. Each line in the content of `formatted_triplets` rep-

resents a distinct relationship, encoded as a triplet consisting of a source entity (src), a destination entity (dst), and the relationship (rel) that connects them.

```
Row(id='David')-Row(src='David', dst='Notebook', rel='buy')
  ->Row(id='Notebook')
Row(id='David')-Row(src='David', dst='portable', rel='mention')
  ->Row(id='portable')
Row(id='Printer')-Row(src='Printer', dst='portable', rel='mention')
  ->Row(id='portable')
Row(id='Notebook')-Row(src='Notebook', dst='portable', rel='mention')
  ->Row(id='portable')
```

When RDF triples are formatted for inclusion in a prompt, they can subsequently be appended to questions directed at the LLM. The prompt itself presents the triplets variable alongside instructions on interpreting the relationships within these triplets, a structured approach to guide the assistant's understanding and response generation. It offers a concrete dataset for analysis and instruction on how to interpret the relationships encoded like 'buy', 'also_buy' and 'also_view'. Each of these relationships carries distinct implications for consumer behavior.

```
messages = [
  {"role": "system",
   "content": "You are a helpful assistant."},
  {"role": "user",
   "content": f"Consider the following relationships represented as triplets:
   \n{formatted_triplets}"},
  {"role": "user",
   "content": "Consider that if a person has bought a product that participates
   in a relation 'also_buy' with other products, there is a high probability
   for this person to buy these other products."},
  ...
```

The table below outlines the relationships between products and their associated probabilities, which are crucial for analyzing customer purchasing behaviors and making informed product recommendations. Here is the table summarizing the relationships based on the instructions provided:

Table 1. Summary of Product Relationships and Associated Purchase Probabilities.

Relationship	Min Occ.	Purchase Prob.
also_buy	5	High
also_view	1	Medium
belongs_to_brand	-	Medium
belongs_to_category	-	Low

Based on these probabilities and the history of a particular user, we can generate personalized buying recommendations, as illustrated in the following user query. This approach leverages data on product relationships to tailor suggestions that enhance the shopping experience by predicting user preferences accurately.

```
{"role": "user",
 "content": "Based on products bought or
viewed together, What other products
David could buy?
Present the probabilities for each product.
```

```
    Order the recommendations from high to low."}]
]
```

The `openai.chat.completions.create` function is called with the specified model `gpt-4` and the prepared messages. Upon executing this function, GPT-4 processes the input messages and generates a completion, which is a response to the question 'What other products David could buy?'

```
response = openai.chat.completions.create(
    model="gpt-4",
    messages=messages
)
```

The model not only gives an answer to the presented question but also provides a *chain of thought* explanation, detailing the logical steps and reasoning it employed to construct the answer, offering insight into the model's thought process and the rationale behind its conclusions, as shown below.

Based on the information, David has bought a 'Notebook'.
The relations associated with this product are the following:

- The 'Notebook' has a 'also_buy' relation with 'Router', 'Mouse' and 'Printer'. This gives a high probability of David buying these products.
- The 'Notebook' has a 'also_view' relation with 'Router', 'Mouse'. This gives a medium probability of David buying these products.
- The 'Notebook' belongs to brand 'HP'. This gives a medium probability of David buying other products from 'HP', like 'Monitor' in this case.
- The 'Notebook' belongs to the 'Personal Computer' category which suggests a low probability on buying other items in the 'Personal Computer' category.

Recommendations based on high to low probability:

1. Router (also_buy = high probability)
2. Mouse (also_buy = high probability)
3. Printer (also_buy = high probability)
4. Router (also_view = medium probability, already involved in also_buy)
5. Mouse (also_view = medium probability, already involved in also_buy)
6. Monitor (same brand 'HP' = medium probability)
7. Any other product from the 'Personal Computer' category (belongs_to_category = low probability)

Chain of thought in LLMs exemplifies how these systems can utilize logical inference to process and elaborate on information, leading to a structured and reasoned conclusion. This capability mirrors human reasoning, wherein multiple steps of thought are aligned to navigate towards an answer or a decision [17]. In the context of product recommendations, for example, the LLM uses available data about past purchases and defined relationships between products to predict future buying behavior. In the presented scenario, the model considers various relational data points. The 'also_buy' relationship indicates that customers who bought a 'Notebook' also frequently purchased items like 'Router', 'Mouse', and 'Printer'. Given this relationship, there's a high probability that David might be interested in these products. The 'also_view' relationship shows that 'Router' and 'Mouse'

were often viewed by those who looked at 'Notebooks'. While this correlation is weaker than purchasing together, it still suggests a medium likelihood that these items could appeal to David. Brand loyalty or preference is captured under the relationship where the 'Notebook' is from 'HP'. This connection suggests that David might favor other 'HP' products like 'Monitor', assigning a medium probability to such purchases. And the category relationship, where the 'Notebook' falls under 'Personal Computer', suggests a lower probability of purchasing other items from the same category, likely due to redundancy or fulfillment of needs within the category.

These logical steps guide the model in prioritizing recommendations, ranking them from high to low probability based on the strength of their associated relationships. In conclusion, exposing a Knowledge Graph to an LLM through the careful acquisition, transformation, and formatting of RDF triplets into LLM prompts represents a sophisticated method for enhancing the model's question-answering capabilities. This process not only leverages the rich semantic structures of Knowledge Graphs but also aligns with the LLM's strengths in natural language understanding and generation, resulting in a synergistic enhancement of the LLM's functionality.

6 DATABASES

The integration of databases with LLMs via the Text-to-SQL approach marks a significant advancement in the field of natural language processing and data retrieval. This section elucidates the process by which questions posed in natural language are translated into SQL commands, under the precondition that the LLM is acquainted with the underlying database schema. The emphasis is on the LLM's capacity to understand the schema, rather than the data within the database, to accurately convert a question into an SQL command for effective data retrieval. Subsequently, it discusses how the data fetched from the database, in conjunction with the original question, is presented to the LLM to generate an appropriate response.

The foundational step in this process involves the LLM understanding the structure or schema of the database. This knowledge includes an awareness of the tables, fields, and relationships within the database but does not extend to the actual data contained therein. The schema provides a map that guides the LLM in formulating SQL queries based on natural language questions. It is crucial for the LLM to comprehend this structure to accurately interpret the intent of the query and identify which parts of the database to access for relevant information.

As an example, consider the entity-relationship diagram shown in figure 3 for a simple database involving car sales. The diagram includes four entities: sales, car_models, customers and car_brands. The following code snippet establishes a connection to a PostgreSQL database and sets up a query engine for natural language SQL querying in Python using SQLAlchemy.

```
from sqlalchemy import create_engine, MetaData
from llama_index import SQLDatabase
from llama_index.indices.struct_store import
(
    NLSQLTableQueryEngine,
    SQLTableRetrieverQueryEngine,
)

engine = create_engine("postgresql://
    postgres:postgres@localhost:5432/postgres")
metadata_obj = MetaData()
```

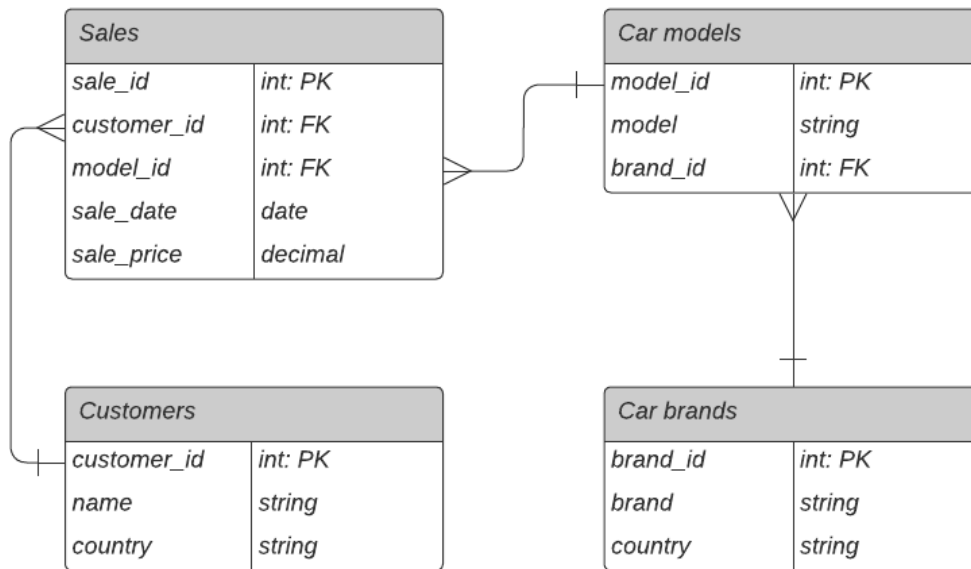


Fig. 4. Car Sales System Relational Model. Source: Authors.

```

sql_database = SQLiteDatabase(engine)
query_engine = NLSQLTableQueryEngine(
    sql_database)
  
```

The `query_engine` object is created from the `NLSQLTableQueryEngine` class, which is a `llama_index` class [14] designed to take the `sql_database` object and enable the processing of natural language queries that are translated into SQL queries to be run against the connected PostgreSQL database. Upon receiving a query in natural language, the LLM employs the Text-to-SQL approach to transform this query into a corresponding SQL command. This conversion process is intricate, requiring the LLM to not only parse the natural language question for its semantic content but also to translate this understanding into the formal language of SQL [12].

```

response = query_engine.query("Which car
brands are from France?")
str(response)
  
```

The car brands from France are Citroen and Renault.

We can examine the metadata of the response variable to access its contents, which include a unique transaction identifier, an SQL query selecting the brand from `car_brands` where the country is France, and the resulting data listing brands such as Citroen and Renault, with `brand` being the key for the column data.

```

{'d7c05388-78f4-49fd-8534-b537db3e8ea4': {},
'sql_query': "SELECT brand FROM car_brands WHERE country = 'France';",
'result': [(('Citroen',),), (('Renault',),)],
'col_keys': ['brand']}
  
```

The accuracy of this step is paramount, as the resulting SQL command must precisely reflect the query's intent to retrieve the correct data from the database. The LLM's abil-

ity to perform this transformation correctly hinges on its understanding of the database schema and its capacity to map the query's intent to specific database operations [15]. The next example is more intricate. The question is complex due to the necessary transformation of this nuanced natural language inquiry into an SQL query that accurately reflects the relationships between the tables in the database. The underlying code captures the essence of this intricacy.

```
response = query_engine.query("Who bought cars made in the country where they were born? \
Show me their names and the cars they bought.")
str(response)
```

The customers who bought cars made in the country where they were born are Darcy, Kim, and Lee. Darcy bought a Sandero, Kim bought a Santa Fe, and Lee bought a Tucson.

As we can see below, the translated query includes all the four tables in the database to achieve a correct response.

```
{'bf1d27d8-e287-4969-90a3-8e6b5d9cecf5': {}},
'sql_query': 'SELECT c.name, cm.model
FROM customers c
JOIN sales s ON c.customer_id=s.customer_id
JOIN car_models cm ON s.model_id=cm.model_id
JOIN car_brands cb ON cm.brand_id=cb.brand_id
WHERE c.country=cb.country
ORDER BY c.name;',
'result': [('Darcy', 'Sandero'),
('Kim', 'Santa Fé'),
('Lee', 'Tucson')],
'col_keys': ['name', 'model']}
```

Once the SQL command is executed and the relevant data is retrieved from the database, this data is then fed back to the LLM along with the original question. This step is critical for providing context to the LLM, enabling it to understand the information retrieved and how it relates to the query. The LLM then synthesizes the retrieved data with the context of the question to formulate a coherent and accurate response. This response not only addresses the query but does so with a specificity and depth of understanding that is grounded in the actual data retrieved from the database. In the next given example, a query is executed to identify car models sold after July 31, 2023.

```
response = query_engine.query("What were the car models sold after 07/31/2023?")
str(response)
```

The car models sold after 07/31/2023 were the Renault Sandero, Jeep Compass, Hyundai Santa Fé and Hyundai HB20. These models were sold on various dates in August 2023.

The SQL query integrates data from multiple tables: car_models, car_brands and sales. It joins these tables based on the relationships between brand and model identifiers and selects records where the sale date is later than the specified date.

```
{'58924689-a631-40e7-b94f-ca2690edf364': {}},
'sql_query': "SELECT cm.model, cb.brand,
s.sale_date
FROM car_models cm
JOIN car_brands cb ON cm.brand_id = cb.brand_id
JOIN sales s ON cm.model_id = s.model_id
WHERE s.sale_date > '2023-07-31'
ORDER BY s.sale_date;",
```

```
'result': [('Sanderо', 'Renault',
datetime.date(2023, 8, 1)),
('Compass', 'Jeep',
datetime.date(2023, 8, 1)),
('Santa Fé', 'Hyundai',
datetime.date(2023, 8, 4)),
('HB20', 'Hyundai',
datetime.date(2023, 8, 9))],
'col_keys': ['model', 'brand', 'sale_date']}]
```

Our final example summarizes the sales made in 2023 by querying the database for customer names, car prices, and sale dates. The SQL query fetched data indicating that car sales involved customers named Mark, Kim, Karl, Darcy, Manon, Antony, and Lee. These sales occurred throughout 2023, with prices ranging from 1,000 to 3,000. Notably, Mark made the most significant purchase of 3,000 on August 9th. This summary effectively highlights both the range of customer engagement and the financial scope of transactions within the given year, illustrating a diverse and dynamic market activity captured efficiently through the structured query of a well-maintained database.

```
response = query_engine.query("Summarize the
car sales made in 2023 showing customers names,
price
and sale date.")
str(response)
```

In 2023, car sales were made to customers named Mark, Kim, Karl, Darcy, Manon, Antony, and Lee. The prices of the cars sold ranged from \$1000 to \$3000, with Mark making the highest purchase on August 9th.

The SQL command generated by the LLM in this example is a testament to the model's ability to accurately translate natural language queries into functional SQL syntax. The command effectively combines multiple elements: it joins the sales and customers tables to correlate sales data with customer information, selects the customer's name, the sale price, and the date of the sale, and filters the results to include only transactions from the year 2023. This is accomplished using the `EXTRACT(YEAR FROM s.sale_date) = 2023` condition, ensuring that only relevant sales are considered. The results are then ordered by sale price in descending order, prioritizing the highest sale values at the top of the list. This SQL command demonstrates how LLMs can handle complex query requirements, such as joining tables, applying filters, and sorting data, thus enabling efficient data retrieval tailored to specific user inquiries.

```
{'ab1d616d-a97f-4817-ad79-b4aac963cd17': {}},
'sql_query': 'SELECT c.name, s.sale_price,
s.sale_dateFROM sales s
JOIN customers c
ON s.customer_id = c.customer_id
WHERE EXTRACT(YEAR FROM s.sale_date) = 2023
ORDER BY s.sale_price DESC;',
'result': [('Mark', Decimal('3000.00'),
datetime.date(2023, 8, 9)),
('Kim', Decimal('2500.00'),
datetime.date(2023, 8, 4)),
('Karl', Decimal('2000.00'),
datetime.date(2023, 7, 13)),
('Darcy', Decimal('1500.00'),
datetime.date(2023, 8, 1)),
('Manon', Decimal('1200.00'),
```

```
datetime.date(2023, 8, 1)),  
('Antony', Decimal('1000.00')),  
datetime.date(2023, 7, 1)),  
('Lee', Decimal('1000.00')),  
datetime.date(2023, 7, 25))],  
'col_keys': ['name', 'sale_price', 'sale_date']}
```

The efficacy of this entire process relies heavily on the LLM's ability to generate accurate SQL commands from natural language questions. A misinterpretation of the question or an incorrect SQL query can lead to irrelevant data retrieval, thereby compromising the quality of the LLM's response. Therefore, the precision of the Text-to-SQL conversion process is of utmost importance, emphasizing the need for LLMs to have a robust understanding of the database schema and a high capability for natural language comprehension. On the other hand, the importance of a semantically understandable database schema in the context of employing text-to-SQL techniques with LLMs cannot be overstated. When a database schema is designed with clear, descriptive naming conventions and logical relationships, it greatly enhances the ability of an LLM to accurately interpret user queries and generate corresponding SQL commands. This clarity in schema design helps the model to effectively map natural language input to the structured format required for SQL queries.

7 CONCLUSIONS

This article has elucidated the critical role of techniques such as Prompt Engineering, Retrieval-Augmented Generation (RAG) and text-to-SQL in enhancing the functionality and applicability of LLMs in accessing and integrating external data sources. The Python scripts utilized in our analyses are openly accessible at [9]. These methodologies are fundamental in interacting with and leverage vast, dynamic external knowledge repositories, without the need of retraining a model.

Notably, these approaches have been applied with notable success to various data-intensive environments, including documents, knowledge graphs, and databases. By enabling LLMs to dynamically query and retrieve relevant information from these structured and unstructured data sources, the techniques enhance the model's ability to generate informed and contextually accurate outputs. This synergy not only maximizes the utility of existing data but also expands the potential applications of LLMs across different sectors, including business intelligence, legal advisement, and academic research.

The promising results obtained from these techniques underscore the potential for data interaction and retrieval. However, to fully ascertain their effectiveness and scalability, future work should focus on testing these methodologies across more voluminous and diverse data sets, encompassing extensive documents, knowledge graphs (KGs), and expansive databases. Such rigorous testing is essential to validate the robustness and adaptability of the strategies employed, ensuring that they maintain high levels of accuracy and efficiency when scaled.

Moreover, exploring these techniques in larger, more complex data environments will also shed light on their limitations and the potential need for refinement or adaptation. As mentioned in the paper, the continuous expansion of token limits in LLMs marks a significant trend in the evolution of artificial intelligence technologies. As these limits grow, developers are empowered to work with increasingly larger blocks of text in a single submission, enabling a deeper and more comprehensive analysis of data. This future exploration will not only bolster the confidence in deploying these techniques in real-world scenarios but also pave the way for their optimization and potential customization to spe-

cific domains or data types, ultimately enhancing the utility and impact of LLMs across various sectors.

References

1. Asudani, D.S., Nagwani, N.K., Singh, P.: Impact of word embedding models on text analytics in deep learning environment: a review. *Artificial Intelligence Review*, **56**(9), 10345–10425 (2023).
2. Barnett, S., Kurniawan, S., Thudumu, S., Brannelly, Z., Abdelrazek, M.: Seven Failure Points When Engineering a Retrieval Augmented Generation System. *arXiv preprint arXiv:2401.05856* (2024).
3. Cai, D., Wang, Y., Bi, W., Tu, Z., Liu, X., Shi, S.: Retrieval-guided dialogue response generation via a matching-to-generation framework. In: *Proc. EMNLP-IJCNLP*, pp. 1866–1875 (2019).
4. Chen, J., Lin, H., Han, X., Sun, L.: Benchmarking large language models in retrieval-augmented generation. In: *Proc. AAAI Conference on Artificial Intelligence*, **38**(16), 17754–17762 (2024).
5. Feng, Z., Feng, X., Zhao, D., Yang, M., Qin, B.: Retrieval-generation synergy augmented large language models. In: *ICASSP 2024 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 11661–11665. IEEE (2024).
6. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, H.: Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).
7. Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., Zhou, J.: Text-to-SQL empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363* (2023).
8. Giray, L.: Prompt engineering with ChatGPT: a guide for academic writers. *Annals of Biomedical Engineering*, **51**(12), 2629–2633 (2023).
9. Seabra, A.: Github Repository. Available at: <https://github.com/antonyseabramedeiros/qasystems>. Accessed: 2024-04-01 (2024).
10. Heston, T.F., Khun, C.: Prompt Engineering in Medical Education. *International Medical Education*, **2**(3), 198–205 (2023). <https://www.mdpi.com/2813-141X/2/3/19>.
11. Jeong, C.: A Study on the Implementation of Generative AI Services Using an Enterprise Data-Based LLM Application Architecture. *arXiv preprint arXiv:2309.01105* (2023).
12. Katsogiannis-Meimarakis, G., Koutrika, G.: A survey on deep learning approaches for text-to-SQL. *The VLDB Journal*, **32**(4), 905–936 (2023).
13. Langchain: Langchain RetrievalQA documentation. Available at: https://api.python.langchain.com/en/latest/chains/langchain.chains.retrieval_qa.base.RetrievalQA.html. Accessed: 2024-03-01 (2024).
14. LlamaIndex: Llamaindex documentation. Available at: <https://www.llamaindex.ai/>. Accessed: 2024-03-01 (2024).
15. Li, H., Zhang, J., Li, C., Chen, H.: Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In: *Proc. AAAI Conference on Artificial Intelligence*, **37**(11), 13067–13075 (2023).
16. Li, H., Su, Y., Cai, D., Wang, Y., Liu, L.: A survey on retrieval-augmented text generation. *arXiv preprint arXiv:2202.01110* (2022).
17. Ling, Z., Fang, Y., Li, X., Huang, Z., Lee, M., Memisevic, R., Su, H.: Deductive verification of chain-of-thought reasoning. *Advances in Neural Information Processing Systems*, **36** (2024).
18. Liu, A., Hu, X., Wen, L., Yu, P.S.: A comprehensive evaluation of ChatGPT’s zero-shot Text-to-SQL capability. *arXiv preprint arXiv:2303.13547* (2023).
19. Manathunga, S.S., Illangasekara, Y.A.: Retrieval Augmented Generation and Representative Vector Summarization for large unstructured textual data in Medical Education. *arXiv preprint arXiv:2308.00479* (2023).
20. Moore, R., Lopes, J.: Paper templates. In: *Proc. TEMPLATE’06, 1st International Conference on Template Production*, pp. –. SCITEPRESS (1999).
21. OpenAI: ChatGPT Fine-tune Description. Available at: <https://help.openai.com/en/articles/6783457-what-is-chatgpt>. Accessed: 2024-03-01 (2023).
22. OpenAI: ChatGPT Prompt Engineering. Available at: <https://platform.openai.com/docs/guides/prompt-engineering>. Accessed: 2024-04-01 (2023).
23. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: *International Conference on Machine Learning*, pp. 1310–1318 (2013).
24. Pinheiro, J., Victorio, W., Nascimento, E., Seabra, A., Izquierdo, Y., García, G., Coelho, G., Lemos, M., Leme, L.A.P.P., Furtado, A., et al.: On the Construction of Database Interfaces Based on Large Language Models (2023).
25. Sahoo, P., Singh, A.K., Saha, S., Jain, V., Mondal, S., Chadha, A.: A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. *arXiv preprint arXiv:2402.07927* (2024).

26. Seabra, A., Cavalcante, C., Nepomuceno, J., Lago, L., Ruberg, N., & Lifschitz, S. (2024). Contrato360 2.0: A Document and Database-Driven Question-Answer System Using Large Language Models and Agents. *International Conference on Knowledge Discovery and Information Retrieval*.
27. Smith, J.: *The Book*. 2nd ed., The Publishing Company, London (1998).
28. Sun, R., Arik, S.O., Nakhost, H., Dai, H., Sinha, R., Yin, P., Pfister, T.: SQL-PaLM: Improved Large Language Model Adaptation for Text-to-SQL. *arXiv preprint arXiv:2306.00739* (2023).
29. Tian, Y., Li, T.J.-J., Kummerfeld, J.K., Zhang, T.: Interactive Text-to-SQL Generation via Editable Step-by-Step Explanations. *arXiv preprint arXiv:2305.07372* (2023).
30. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., Polosukhin, I.: Attention is all you need. *Advances in Neural Information Processing Systems*, **30** (2017).
31. Wang, P., Shi, T., Reddy, C.K.: Text-to-SQL Generation for Question Answering on Electronic Medical Records. *ACM Transactions*, DOI: 10.1145/3366423.3380120 (2020).
32. Wang, M., Wang, M., Xu, X., Yang, L., Cai, D., Yin, M.: Unleashing ChatGPT's Power: A Case Study on Optimizing Information Retrieval in Flipped Classrooms via Prompt Engineering. *IEEE Transactions on Learning Technologies* (2023).
33. Watkins, R.: Guidance for researchers and peer-reviewers on the ethical use of Large Language Models (LLMs) in scientific research workflows. *AI and Ethics*, pp. 1–6 (2023).
34. White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., Schmidt, D.C.: A prompt pattern catalog to enhance prompt engineering with ChatGPT. *arXiv preprint arXiv:2302.11382* (2023).
35. World Health Organization: *World Health Organization*. Available at: <https://www.who.int/>. Accessed: 2024-03-01 (2024).
36. World Health Organization: Covid-19 Factsheet. Available at: [https://www.who.int/news-room/fact-sheets/detail/coronavirus-disease-\(covid-19\)](https://www.who.int/news-room/fact-sheets/detail/coronavirus-disease-(covid-19)). Accessed: 2024-03-01 (2024).
37. World Health Organization: Dengue and Severe Dengue Factsheet. Available at: <https://www.who.int/news-room/fact-sheets/detail/dengue-and-severe-dengue>. Accessed: 2024-03-01 (2024).
38. World Health Organization: HIV/AIDS Factsheet. Available at: <https://www.who.int/news-room/fact-sheets/detail/hiv-aids>. Accessed: 2024-03-01 (2024).

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.

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.