U.PORTO
FEP FACULDADE DE ECONOMIA E GESTÃO UNIVERSIDADE DO PORTO

Development of a Low-Code Platform for RAG System Configuration: Integration with Corporate Data

Hugo Moreira Teixeira

Dissertation

Master in Modeling, Data Analysis and Decision Support Systems

Supervised by:

Flora Ferreira

Bruno Veloso

2026

Acknowledgments

I would first like to express my gratitude to my family and friends for the support they gave me along the way.

I am also grateful to everyone at B2F, both for their help throughout the project and for providing the requirements and the data on which this system was built and evaluated, without which this work would not have been possible.

Finally, I would like to deeply thank my supervisor, Professor Flora Ferreira, and my co-supervisor, Professor Bruno Veloso, for their assistance in the development of this dissertation

Abstract

Retrieval-Augmented Generation (RAG) systems have emerged as a powerful approach to ground Large Language Model responses in factual, domain-specific knowledge. However, building an effective RAG system over an organization's own enterprise data remains technically demanding. This dissertation presents a RAG system that gives non-technical users conversational access to an organization's data. The system answers natural language questions about both unstructured documents held in Microsoft SharePoint and structured data stored in SQL Server through a single interface. The document path uses retrieval-augmented generation, while the structured data path uses schema-aware text-to-SQL. Developed in collaboration with a partner company, B2F, and evaluated on real Portuguese-language enterprise data rather than public benchmarks, the work also compares the architectural choices that shape answer quality, such as query rewriting and reranking on the document path, assessed with the RAGAS framework, and the enrichment of the schema context on the structured data path, assessed using execution accuracy. The results show that enriching the schema with keys and sample rows improves text-to-SQL execution accuracy, while on the document side, the retrieval enhancements had a modest and uneven impact on the results, with the clearest benefit being a reduction in incorrect refusals.

Keywords: Enterprise Data; Large Language Models; Retrieval-Augmented Generation; Reranking; Text-to-SQL

Resumo

Os sistemas de Retrieval-Augmented Generation (RAG) surgiram como uma abordagem poderosa para fundamentar as respostas de Large Language Models em conhecimento factual e específico de um domínio. No entanto, a construção de um sistema RAG eficaz sobre os dados empresariais próprios de uma organização continua a ser tecnicamente exigente. Esta dissertação apresenta um sistema RAG que permite a utilizadores não técnicos acederem de forma conversacional aos dados de uma organização. O sistema responde a perguntas em linguagem natural sobre documentos não estruturados armazenados no Microsoft SharePoint e sobre dados estruturados armazenados num servidor SQL, através de uma única interface. O percurso para documentos utiliza Retrieval-Augmented Generation, enquanto o percurso para dados estruturados utiliza text-to-SQL com enriquecimento do esquema. Desenvolvido em colaboração com a empresa parceira B2F e avaliado com dados empresariais reais em língua portuguesa, em vez de benchmarks públicos, o trabalho compara também diferentes opções arquiteturais que influenciam a qualidade das respostas, nomeadamente a reformulação de consultas e o reranking no percurso dos documentos, avaliados através do framework RAGAS, e o enriquecimento do contexto do esquema no percurso dos dados estruturados, avaliado através da execução das consultas. Os resultados mostram que o enriquecimento do esquema com chaves e exemplos de linhas melhora a precisão das consultas text-to-SQL, enquanto, no percurso dos documentos, as melhorias na recuperação tiveram um impacto modesto e desigual nos resultados, sendo o benefício mais evidente a redução de recusas incorretas.

Palavras-chave: Dados Empresariais; Modelos de Linguagem de Grande Dimensão; Retrieval-Augmented Generation; Reranking; Text-to-SQL

Contents

Acknowledgements . . . iii
Abstract . . . v
Resumo . . . vii

List of Acronyms **xi**

1 Introduction **1**
1.1 Context and Motivation . . . 1
1.2 Problem Statement . . . 3
1.3 Objectives . . . 3
1.4 Dissertation Structure . . . 4

2 Literature Review **5**
2.1 Large Language Models and Gen-AI . . . 5
2.2 Adapting LLMs to Domain Knowledge: Fine-Tuning and RAG . . . 6
2.3 RAG Architecture . . . 7
2.3.1 Chunking . . . 7
2.3.2 Embeddings and Vector Databases . . . 8
2.3.3 Query Encoding . . . 9
2.3.4 Retrieval . . . 10
2.3.5 Query Rewriting . . . 10
2.3.6 Reranking . . . 11
2.4 Text-to-SQL . . . 12
2.5 Evaluation of RAG Systems and RAGAS . . . 13
2.6 Research Gap . . . 15

3 RAG System Design and Implementation **19**
3.1 System Overview and Architecture . . . 19
3.2 Technology Stack . . . 21
3.3 Enterprise Data Sources . . . 23
3.4 Document Retrieval System . . . 24
3.4.1 Ingestion . . . 24
3.4.2 Document Processing and Chunking . . . 25
3.4.3 Embedding and Indexing . . . 26
3.4.4 Retrieval and Generation . . . 27
3.4.5 Architecture Variants . . . 28
3.5 Structured-Data System . . . 31
3.6 Application Layer and Configuration . . . 32

4 Evaluation and Results **35**
4.1 Evaluation Process 35
4.2 Evaluation Datasets 36
4.3 Results 37

5 Conclusions and Future Work **41**

A Appendix **i**
A.1 Document Question and Answering Evaluation Set i
A.2 Structured Data (Text-to-SQL) Evaluation Set and Results iv
A.3 Prompts iv
A.3.1 Document Generation Prompt v
A.3.2 Query Rewriting Prompt v
A.3.3 Text-to-SQL Generation Prompt v
A.3.4 SQL-Answer Prompt vi
A.4 User Interface vii

Bibliography **ix**

List of Abbreviations and Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
BGE	BAAI General Embedding
BM25	Best Matching 25
DPR	Dense Passage Retrieval
GPT	Generative Pre-trained Transformer
HyDE	Hypothetical Document Embeddings
LLM	Large Language Model
LoRA	Low-Rank Adaptation
RAG	Retrieval-Augmented Generation
RAGAS	Retrieval-Augmented Generation Assessment
SQL	Structured Query Language

List of Figures

3.1 Overall architecture of the system. 20
3.2 Indexing pipeline. 25
3.3 The baseline query pipeline. 28
3.4 The query rewriting architecture. 29
3.5 The reranking architecture. 30
3.6 Comparison between Bi-encoder and Cross-encoder scoring 31
3.7 The text-to-SQL flow. 32

A.1 Minimum Schema Text-to-SQL Results. iv
A.2 Enriched Schema Text-to-SQL Results. iv
A.3 The conversational interface, showing a question, its answer and the supporting citations. vii
A.4 The configuration interface, for the SQL Server database viii

List of Tables

2.1 Summary of the primary studies underpinning this work. 16

3.1 Technologies used in the system and the rationale for each choice 22
3.2 SQL Server tables of the structured data source. 24

4.1 RAGAS metrics for the four document-retrieval configurations, averaged over all twenty-four questions. 37
4.2 RAGAS metrics for the four document-retrieval configurations, averaged over twenty-two answerable questions, with the two negative tests excluded. 37
4.3 Retrieval failures (answerable questions wrongly refused) by configuration, out of the 22 answerable questions. 38
4.4 Execution accuracy of the two schema configurations on the fifteen structured-data questions. 39

A.1 Document question-answering evaluation set (24 questions with ground-truth answers). i

Chapter 1

Introduction

This dissertation presents the implementation and evaluation of a Retrieval-Augmented Generation (RAG) system capable of answering natural language questions over heterogeneous enterprise data sources. This opening chapter situates the work within the broader context of enterprise knowledge access and chat-based solutions, defines the problem that it addresses, states the objectives and contributions, and outlines the structure of the remaining chapters.

1.1 Context and Motivation

During the past decade, organizations have accumulated large, and constantly growing volumes of information that are distributed across a range of heterogeneous systems. Internal documents, policies, reports, and meeting records are typically held in collaboration platforms such as Microsoft SharePoint, while structured operational data reside in relational databases such as Microsoft SQL Server. This accumulated knowledge is widely recognized as a strategic asset. However, its dispersion across formats and platforms makes it difficult to access. Employees often spend considerable time looking for information that already exists somewhere within the organization, and the value of that information is directly related to its availability when needed.

The emergence of Large Language Models (LLMs), built on the transformer architecture introduced by Vaswani et al. (2017), has changed how people interact with information. Models of the GPT family demonstrate a remarkable ability to interpret questions expressed in natural language and produce coherent responses. This capability has made conversational access to organizational knowledge not only feasible but also imminent, promising a change from the typical keyword search, towards systems that understand the user's intent and respond accordingly.

Despite these promises, LLMs face fundamental limitations when applied to enterprise environments, as they operate within the constraints of their training data, which has a fixed knowledge cutoff date and, in most cases, does not address the specific information of an industry or a business. In addition, LLMs are also susceptible to hallucinations, particularly when they are questioned about specialized information that is not addressed in the training data, which makes companies reluctant to use LLMs in tasks where accuracy and reliability are critical.

There are two broad strategies for adapting a general purpose LLM to organization spe-

cific knowledge. The first is fine-tuning, where the training of the model continues on proprietary data, so that the knowledge is encoded directly in its parameters. While it is effective when we want the model to specialize in a specific domain, or for narrower tasks, fine-tuning is computationally expensive and needs to be repeated every time the underlying information changes.

The second strategy, Retrieval-Augmented Generation (RAG), has then emerged as a compelling solution to address the limitations of standalone LLMs with the challenges of enterprise data integration. RAG combines the strengths of information retrieval systems with the generative capabilities of language models. The core principle of RAG is that, rather than relying only on the knowledge encoded in the language model weights, the system first retrieves relevant information from an external knowledge source, such as a document repository or database, and then conditions the language model on the retrieved context.

The RAG approach offers several advantages for enterprise applications. It grounds the model's responses in actual, verifiable source documents, reducing hallucinations (Lewis et al., 2020), which ultimately leads to an increase in accuracy. It also allows organizations to leverage their existing knowledge bases without the need for fine-tuning since external knowledge can be updated independently of the language model. In the end, RAG systems also provide a degree of transparency and explainability, as retrieved documents can be presented alongside the generated responses, which allows users to verify the information sources.

Despite the suitability of RAG systems to integrate with enterprise data, this knowledge spans between unstructured content, such as the documents stored in SharePoint, and structured content, such as the records held in SQL Server database. These two forms of data demand fundamentally different retrieval mechanisms, a semantic search over text passages on the one hand, and the translation of natural language questions into precise database queries on the other. So, a useful enterprise assistant must, therefore, be able to, at the same time, retrieve from both documents and from databases.

This difficulty is compounded by the problem of evaluation. Determining whether a given architectural choice genuinely improves system performance requires a reliable way of measuring answer quality. For document question answering, this can involve manually curated ground-truth answers, which are often expensive to produce and evaluate at scale. Automated evaluation frameworks can reduce this burden while preserving the open-ended nature of generated responses.

Reference free frameworks such as RAGAS (Es, James, Espinosa-Anke, & Schockaert, 2024) help address this for document-based question answering, but they are not directly applicable to the querying of structured data, which requires its own way of evaluation. Without a workable means of evaluation, the research gap cannot be adequately addressed because there is no reliable basis on which to decide which techniques are worth adopting.

The core problem for this dissertation therefore has two facets. The first is to build an RAG system that connects to the most common data sources in enterprise settings, documents in SharePoint, and structured data in SQL Server. The second, which speaks directly to the research gap, is the generation of evidence about which architectural choices measurably improve answer quality on real enterprise data, so that such decisions need not rest on trial and error alone.

1.2 Problem Statement

Although the effectiveness of RAG systems in enhancing large language models' performance is well documented, deploying them over real enterprise data remains a challenge, which prevents organizations from leveraging RAG capabilities effectively.

Organizations are faced with a practical dilemma that is rooted in the options currently available to them. Existing solutions tend to be at two extremes, neither of which is satisfactory for many organizations.

At one extreme are the commercial offerings, most prominently Microsoft Copilot and comparable integrated assistants. These provide ease of deployment and seamless integration with established ecosystems such as Microsoft 365, which are attractive to organizations with limited technical capacity. However, they operate largely as closed systems, the organization has limited ability to connect arbitrary or non-standard data sources, and no means of inspecting, adjusting, or experimenting with the underlying pipeline.

On the other hand, lies the development of a fully custom RAG system. This grants complete control over data sources, retrieval architecture, and generation, but requires substantial engineering resources and specialized expertise. Beyond the initial implementation effort, an organization must navigate a large and rapidly growing space of architectural options, including different chunking strategies, retrieval methods, query transformation techniques, and reranking approaches, and determine which of them actually improves answer quality for its particular data. This resource and expertise gap places an effective customized RAG beyond the reach of many organizations that would otherwise benefit from it.

1.3 Objectives

This dissertation is guided by a central research question, addressed through two subquestions aligned with the experiments reported in Chapter 4.

> **RQ:** How can a RAG system provide conversational access to heterogeneous enterprise data?
>
> **RQ1:** How do query rewriting and reranking affect document question answering?
>
> **RQ2:** How does schema enrichment affect text-to-SQL accuracy?

The primary objective of this work is to implement and evaluate a RAG system that answers natural language questions over two complementary enterprise data sources, documents that are held in Microsoft SharePoint and structured data that are present in Microsoft SQL Server, at the same time that it is determined, through the development of controlled experiments, which architectural choices most improve the quality of its responses.

This primary objective is pursued through the following more specific objectives:

1. **Document retrieval over SharePoint.** Implement a document RAG pipeline over the enterprise SharePoint corpus and evaluate, against a naive baseline, the effect of two architectural enhancements, query rewriting and reranking, using the RAGAS framework on a real enterprise document corpus.

2. **Structured data querying through text-to-SQL.** Implement a schema-aware text-to-SQL pipeline over SQL Server that generates and executes read only queries safely, and evaluate, by means of an execution accuracy protocol, the effect of enriching the schema context provided to the model.

3. **Integration of heterogeneous data sources.** Integrate unstructured SharePoint content and structured SQL Server data within a single conversational system, allowing users to query both data sources through a unified system.

In pursuit of these objectives, the dissertation makes the following contributions. It presents the implementation of a working enterprise RAG system that spans both unstructured and structured data sources through a single conversational interface. It contributes a schema-aware, text-to-SQL component and an evaluation of how the richness of the schema context affects the correctness of its answers. It also provides a controlled experimental comparison of document RAG architectures, a naive baseline against variants augmented with query rewriting and reranking, conducted not on a public benchmark but on a real, Portuguese-language enterprise corpus.

1.4 Dissertation Structure

The remainder of this dissertation is organized into four further chapters. Chapter 2 reviews the theoretical background and related work, beginning with the foundations of transformer-based language models and embeddings, comparing fine-tuning and RAG as adaptation strategies, and surveying the techniques most relevant to this work.

Chapter 3 describes the design and implementation of the system, detailing its architecture, document ingestion from the SharePoint connection, document retrieval and generation pipeline together with its query rewriting and reranking variants, and the schema-aware text-to-SQL component for structured data.

Chapter 4 sets out the experimental methodology, defining the evaluation questions, the datasets, the metrics used for each data type, and the experimental configurations compared, and then presents and discusses the results of these experiments, including an analysis of errors and the practical trade-offs involved. Chapter 5 concludes the dissertation, summarizing its contributions and limitations, and identifying directions for future work.

Chapter 2

Literature Review

This chapter reviews the theoretical foundations and the body of research on which this dissertation is built. It covers Large Language Models, Retrieval-Augmented Generation, retrieval techniques, text-to-SQL, and the evaluation of RAG systems, before identifying the research gap addressed by this dissertation.

2.1 Large Language Models and Gen-AI

Generative artificial intelligence refers to a class of models that, rather than classifying or predicting a label for a given input, produce new content that resembles the data on which they were trained. Within this broad field, the models that are most relevant to this work are those that generate natural language, known as Large Language Models (LLMs). These models have become the basis of contemporary conversational systems and are what make it possible to pose a question in ordinary language and receive a fluent and coherent reply.

LLMs are an instance of what Bommasani et al. (2021) term foundation models, which are models trained on broad data at a very large scale that can subsequently be adapted to a wide range of downstream tasks. This paradigm marks a departure from earlier approaches, in which a separate model was trained for each task, as a single pre-trained model now serves as a general-purpose substrate that can be specialized through prompting or further training. The scale of these models, both in volume of training data and in the number of parameters, is closely associated with the accuracy of their capabilities, this relationship was surveyed in detail by Zhao et al. (2026).

The architectural advance that supports modern LLMs is the transformer architecture introduced by Vaswani et al. (2017). Earlier neural approaches to sequence modeling, based on recurrent and convolutional networks, processed text sequentially, struggling to capture dependencies between distant words. The transformer replaced recurrence with a self-attention mechanism in which each element of a sequence is related directly to every other relevant element, weighting their relevance to one another. This design captures long range context more effectively and lends itself to parallel computation, making it feasible to train models of unprecedented size, remaining the foundation of essentially all current LLMs.

Two variants of the transformer architecture are particularly relevant to the present work, and the distinction between them reappears throughout the chapters that follow. Encoder oriented models, exemplified by BERT (Devlin, Chang, Lee, & Toutanova, 2019), process an entire input bidirectionally and are designed to produce numerical representations of text,

also known as embeddings, such models are the basis for the embedding components used in retrieval. Decoder oriented models, exemplified by the GPT family (Brown et al., 2020), are autoregressive, as they generate one token at a time, each token conditioning the next one, therefore being suited to the generation of answers. A RAG system, as later sections show, uses both types of model, an encoder model to represent and retrieve information, and a decoder model to generate the final response.

Powerful as they are, LLMs exhibit limitations that are decisive for the development of the system addressed in this dissertation. Their knowledge is bounded by the data on which they were trained and is fixed at particular cutoff dates, leaving them unaware if private organizational information or information occurring after the training. Moreover, when asked about matters not represented in their training data, they tend to hallucinate, producing statements that are plausible but factually incorrect. These shortcomings cannot be remedied by scale alone and motivate the methods examined in the next section, by which a general model is supplied with the specific knowledge it lacks.

2.2 Adapting LLMs to Domain Knowledge: Fine-Tuning and RAG

The limitations in the previous section raise a central question for any organization that wishes to apply an LLM to its own information. How can a general-purpose model be supplied with the domain-specific, proprietary, or recent knowledge that it does not have? Two main strategies are found in the literature, Fine-Tuning and Retrieval-Augmented Generation.

Fine-Tuning adapts a model by continuing its training on a corpus of domain data, so that the desired knowledge and behavior become encoded in the model's own parameters. This technique is computationally demanding, since it updates all the model's weights, requiring substantial hardware and expertise. Parameter-efficient methods have considerably reduced this burden, for instance, in LoRA (Hu et al., 2021), the original weights are frozen, and further training is only performed in a small set of low-rank update matrices, reducing the number of trainable parameters by orders of magnitude while preserving performance. Techniques like this make fine-tuning more accessible, but they do not alter its fundamental characteristics as a means of knowledge injection.

Those characteristics impose real constraints in enterprise settings, as the knowledge is embedded in the weights, any change to the underlying information requires the model to be retrained, which is impractical where data changes frequently, as happens in organizations. The model also does not offer an inherent way to indicate the source of a particular statement, which means that its answers cannot be traced back to a document, making it a worse option when the user wants to know where the model got that response. Moreover, fine-tuning appears to be a weak mechanism for teaching a model new facts, as Ovadia, Brief, Mishaeli, and Elisha (2024) found that unsupervised fine-tuning struggled to give new factual knowledge, with retrieval being the more effective way to do so.

Retrieval-Augmented Generation, introduced by Lewis et al. (2020), takes the opposite approach. Rather than storing knowledge in the model's parameters, it leaves the model unchanged and instead retrieves relevant information from an external knowledge source at the moment a question is asked, supplying that information to the model as additional context from which to generate its answer. This separation of knowledge from the model yields

several advantages that directly address the constraints of fine-tuning. The knowledge base can be updated, extended, or corrected independently of the model, so that new information becomes available immediately without the need for any retraining. In addition, because the system generates answers from explicitly retrieved passages, those passages can be cited, giving the user a way to verify the source of each answer and reducing the incidence of hallucinations by grounding responses in real sources.

A growing body of comparative research supports the suitability of RAG for the type of application we propose to develop. Beyond the finding of Ovadia et al. (2024) already noted, Soudani, Kanoulas, and Hasibi (2024) compared Fine-Tuning and RAG on low-frequency, less popular knowledge, and found that although fine-tuning presented gains regardless of how common an entity was, RAG proved far more effective on the least popular facts, outperforming fine-tuning by a wide margin. This result is especially pertinent to enterprise data, which by its nature consists largely of niche proprietary information that is rare or, in most cases, absent from any general training corpus.

Taken together, this evidence motivates the decision, adopted throughout this dissertation, to build the system on Retrieval-Augmented Generation rather than fine-tuning. The requirements that characterize the enterprise context- frequently changing information, a need for traceable and verifiable answers, and a predominance of detailed organizational knowledge, these are precisely those for which RAG is particularly well suited. Therefore, Fine-Tuning is not pursued as a means of knowledge injection in this work. The remainder of this chapter examines, in turn, the components from which the RAG system is constructed.

2.3 RAG Architecture

Having established Retrieval-Augmented Generation as the approach in this work, this section examines its architecture in detail. An RAG system is best understood as a pipeline of distinct stages, conventionally divided into two phases. The first is an indexing phase, in which the documents that make up the knowledge base are ingested, divided into smaller units called chunks, converted into numerical representations, and stored in a database that is optimized for similarity search. The second is a query phase, where a user's question is itself converted into the same representation, in order to retrieve the most relevant stored units, and then combined with the question so that the language model can generate an answer. Each of these stages embodies design choices that materially affect the quality of the system's responses.

2.3.1 Chunking

The documents that make up an enterprise knowledge base are frequently too long to be treated as single units, as they cannot be embedded as a whole without losing the detail that distinguishes one passage from another, nor can an entire document be supplied to a language model whose context window is finite. The indexing phase therefore begins by dividing each document into smaller fragments, or chunks, which become units that are subsequently embedded, stored, and retrieved.

The foundation of chunking is based on a trade-off regarding its granularity. Chunks that are too small risk dividing a statement from the context required to interpret it, so that a retrieved fragment, though textually relevant, is no longer self-explanatory. Chunks that

are too large tend to mix several distinct topics, diluting the relevance signal and introducing irrelevant text that can affect the model during generation. This balance between the precision of the chunks and the preservation of the context depends on the nature of the documents and the types of questions they are expected to answer (Y. Gao et al., 2023). Empirical studies have tried to quantify this balance, and systematic comparisons of chunk sizes in retrieval-augmented generation report that segments of around 256 to 512 tokens tend to perform better, with substantially larger chunks degrading both faithfulness and the relevance of the generated answer (X. Wang et al., 2024).

Several strategies for partitioning text are described in the literature (Y. Gao et al., 2023). The simplest, fixed-size chunking, divides text into segments of predetermined length independently of its structure, which is straightforward but liable to cut through sentences or ideas. Structure-aware approaches instead split text at natural boundaries such as sentences or paragraphs, the widely used recursive character splitting, for example, attempts to divide text at a hierarchy of separators so as to respect its structure as far as possible. A common refinement is to allow consecutive chunks to overlap, repeating small amounts of text at their boundaries so that information spanning a boundary is not lost to either chunk. More elaborate semantic strategies group text according to meaning rather than surface structure. The system developed in this dissertation employs recursive character chunking with overlap, a choice whose details and rationale are presented in Chapter 3.

A recognized weakness of conventional chunking is that, because each chunk is embedded in isolation, the embedding cannot reflect context residing elsewhere in the document. Günther, Mohr, Williams, Wang, and Xiao (2024) addressed this with late chunking, a process that first encodes the entire document with a long-context embedding model and only afterwards divides the resulting token representations into chunks, so that each chunk embedding is informed by the surrounding text.

2.3.2 Embeddings and Vector Databases

Once a document has been divided into chunks, it is necessary to find a way to compare each of the chunks according to its meaning. This is the role of embeddings, which are dense numerical vectors that represent a piece of text as a point in a high-dimensional space, arranged so that texts with similar meaning are closer to each other, while texts with unrelated meaning are far apart from each other. Retrieval by meaning rather than by exact wording is possible precisely because, with embeddings, semantic similarity is reduced to geometric proximity.

The idea of representing language as vectors has a lineage that predates current models. Early methods such as word2vec (Mikolov, Chen, Corrado, & Dean, 2013) learned a single vector for each word from patterns of co-occurrence in large corpora, capturing meaningful relationships between words but assigning each word a fixed representation regardless of context. Then, the transformer architecture mentioned in Section 2.1 made it possible to produce contextual representations, in which the vector for a span of text reflects its unique context. Building on this, Sentence-Bert (Reimers & Gurevych, 2019) adapted such models to produce embeddings of whole sentences and passages that can be compared directly and efficiently, making them well suited to the retrieval setting.

Storing embeddings and searching them efficiently is the function of a vector database (Kukreja et al., 2023), with the central operation being nearest-neighbor search, as given the

vector of a query, finding the stored vectors closest to it under a similarity measure such as cosine similarity or inner product. Computing the exact distance to every vector becomes prohibitively expensive as the collection grows to millions of stored chunks, and vector databases therefore rely on approximate nearest-neighbor (ANN) methods that sacrifice a small amount of accuracy for large gains in speed.

Beyond the vectors themselves, a vector database typically stores metadata alongside each chunk, its source document, position or page, and similar attributes. This metadata enables two purposes that are explained in later chapters, it allows the retrieval to be filtered, and it enables each retrieved chunk to be traced back to its origin, which is the mechanism by which the system provides the source citations. The system in this work stores its chunk embeddings, together with such metadata, in the ChromaDB vector database, as detailed in Chapter 3.

2.3.3 Query Encoding

The indexing phase populates the vector database with embedded chunks, however, when a user writes a question, the query phase begins, and its first step is to bring the question into the same representational space as those chunks. The question is therefore passed through an embedding model to obtain a query vector, which can then be compared geometrically with the stored chunk vectors. For this comparison to be meaningful, the query must be encoded using the same model as the documents.

This arrangement, in which questions and documents are encoded independently by the same model or paired and then compared, is known as a bi-encoder or dual-encoder architecture. Dense Passage Retrieval (Karpukhin et al., 2020) is the canonical instance, as it trains encoders so that the vector of a question is drawn close to the vectors of passages that answer it, and demonstrated that such learned dense representations could outperform traditional term-based retrieval on open-domain question answering. The defining property of the bi-encoder is that documents can be embedded in advance, with only the query needed to be encoded at the time that it is made, which makes retrieval over large collections of documents computationally feasible. This is to be contrasted with the cross-encoder architecture examined in the context of reranking (Section 2.3.6), which jointly encodes a query and a passage together, in order to achieve greater precision, however, with a greater cost.

Encoding a question in this way is nonetheless subject to an intrinsic limitation. A question and the passage that answers it often differ markedly in form, as the question is typically short and can use different vocabulary, which may state the relevant fact only implicitly or in other words, in the end, this vocabulary and form mismatch means that the query vector may not fall as close to the relevant chunks as one would wish, so that pertinent information is ranked below less relevant text. Much of the effort to improve retrieval beyond the basic pipeline is directed at this problem, and it provides the motivation for the two techniques examined later in this section: query rewriting, which reformulates the question before it is encoded (Section 2.3.5), and reranking, which re-examines the retrieved candidates with another model (Section 2.3.6). In the system developed in this dissertation, the question is encoded with the same OpenAI embedding model used to index the documents, as described in Chapter 3.

2.3.4 Retrieval

Retrieval is the stage that gives Retrieval-Augmented Generation its name and, in many cases, determines the quality of everything that follows. Having encoded the user's question as a vector, the system searches the vector database for the stored chunks whose embeddings are most similar to it and returns a fixed number of them, the top-k, as the context to be passed to the language model. As the model can only reason over the chunks it is given, the retrieval imposes an upper bound on the system as a whole, as information that is not retrieved cannot inform the answer, however, if irrelevant information is retrieved, it may mislead the answer.

The number of chunks retrieved, k, is in itself a parameter that embodies the same tension that we observed regarding the length of the chunks. If the k value is too small, it risks omitting passages that are necessary to answer the question, harming recall, and a value that is too large will dilute the relevant content with extraneous text, which increases costs and also reduces the quality of the generated answer. An appropriate value of k depends on the granularity of the chunks and the nature of the questions, and is one of the settings that a RAG system must fix.

Retrieval methods are conventionally divided into two families with complementary strengths. Dense retrieval, of the kind described in the previous section, compares learned embeddings and so captures semantic similarity, matching a question to relevant text even when the two share little vocabulary. Sparse, or lexical, retrieval instead scores documents by term overlap, the BM25 function (Robertson & Zaragoza, 2009) remains a strong representative, weighting matches by term frequency and rarity. Lexical methods excel precisely where dense methods are weakest, in matching exact terms such as names, acronyms, and other rare tokens, but they fail to recognize paraphrase and synonymy, which is when Dense Retrieval has its strengths. Their complementarity has motivated hybrid approaches that combine the two, Blended RAG (Sawarkar, Mangal, & Solanki, 2024) for example, integrates lexical and dense retrieval in enterprise settings with domain-specific terminology. The system developed in this dissertation adopts dense retrieval as its baseline and returns the top-k most similar chunks.

2.3.5 Query Rewriting

Some of the limitations of the retrieval that were noted happen because of the user question. As observed in Section 2.3.3, a user's question is often short, ambiguous, or phrased differently from the documents that have the answer, and encoding it directly can place its vector far from the relevant chunks. Query rewriting addresses this at the source, transforming the question before it is encoded and retrieved so that the reformulated version matches the knowledge base better. It is the first of two enhancements to the basic pipeline evaluated in this dissertation.

Ma, Gong, He, Zhao, and Duan (2023) formalized this idea as a Rewrite-Retrieve-Read pipeline, inserting a rewriting step ahead of the conventional retrieve-then-read sequence, with the premise that the question written by the user is not necessarily the question that is most suitable for retrieval. They presented the rewriter in two ways, the first and simplest one, a language model is prompted, with a few-shot prompt, to reformulate the query, with no training being performed to fine-tune this LLM, the second and most elaborate one, a small dedicated model is trained, by reinforcement learning with the reader's feedback, to produce reformulations that lead to better answers. Jagerman, Zhuang, Qin, Wang, and Bendersky

(2023) studied this setting, prompting a language model to reformulate and expand queries, and found that even zero-shot prompting yields effective reformulations. Following this line, the system in this dissertation applies a single, example-free instruction that asks the language model to rewrite the question into one clear, self-contained search query before it is retrieved.

Several other strategies appear in the literature. One family of strategies generates additional text from the query to enrich it, Hypothetical Document Embeddings, or HyDE (L. Gao, Ma, Lin, & Callan, 2023), prompts a language model to write a hypothetical passage that would answer the question and then retrieve using the embedding of that passage rather than the embedding of the question, while Query2doc (L. Wang, Yang, & Wei, 2023) generates a document and appends it to the query to expand it. A second family generates not one reformulation but several, defined as multi-query approaches, they issue a set of rewrites, retrieve for each one, and fuse the resulting ranked lists, normally with reciprocal rank fusion (Cormack, Clarke, & Buettcher, 2009), it was adapted for retrieval-augmented generation as RAG-Fusion (Rackauckas, 2024). These techniques can improve retrieval further, but they do so at the price of additional model calls, multiple retrieval phases and extra parameters to tune. The single zero-shot rewrite was preferred here for its simplicity and low cost.

2.3.6 Reranking

Reranking addresses a different vulnerability of the RAG pipeline. First-stage retrieval, as described in Section 2.3.3, is fast because it compares the question with each passage through independently computed embeddings, however, this independence limits the precision, which means that a relevant passage may score below a superficially similar one, therefore ranking lower in the returned list that is used. Ranking inserts a second, more discerning stage that re-examines the retrieved candidates and reorders them.

This gives rise to the two-stage retrieve-then-rerank architecture. The first stage retrieves a comparatively large pool of candidate chunks, then a reranker scores each candidate against the question directly, retaining only the best few for generation. The reranker is typically a cross-encoder, which contrasts with the bi-encoder present in the first-stage retrieval discussed in Section 2.3.3. Rather than encoding the question and the passage separately, a cross-encoder supplies the pair to the model, allowing it to range across both at once, producing a more accurate estimate of relevance. This accuracy comes at a considerable computational cost, since a fresh forward pass is required for every pair of question and passage, making it the reason why cross-encoder is applied only to a small candidate set defined by the first stage rather than the whole collection. Nogueira and Cho (2019) established the effectiveness of this design with a BERT-based passage reranker, substantially improving the accuracy of the retrieval over prior methods.

The benefit of reranking extends beyond the order of the retrieved chunks, to the quality of the RAG generation. Liu et al. (2024) showed that language models give more importance to information situated in the extremes of a long context, in comparison with information that is situated in the middle. Reranking mitigates this, as when discarding the weaker candidates, it reduces the amount of irrelevant text that is placed before the generator, and by ordering the best retrieved chunks, it gives the most relevant passages to the generator, serving not only for retrieval precision but also improving the generator's ability to exploit the context it is given.

An alternative to cross-encoder is to use a large language model as a reranker. RankGPT

(Sun et al., 2023) demonstrated that an LLM can act as a zero-shot reranker, prompted to permute a list of candidates into their order of relevance, achieving strong results without task-specific training. Such LLM-based reranking is powerful but expensive, as it requires a language model to call over every candidate in the list for every query.

The present work therefore favors a cross-encoder reranker, that runs locally and adds no further dependence on a generative model. More specifically, it employs a reranker from the multilingual BGE-M3 family (Chen, Xiao, et al., 2024), chosen for its coverage of many languages including Portuguese, applied to the candidates returned by dense retrieval and reducing them to the few most relevant for generation. The details are given in Chapter 3, and as with query rewriting, reranking adds computation to each query, and its effects on answer quality are assessed experimentally.

2.4 Text-to-SQL

The techniques discussed so far address unstructured content, documents whose meaning is retrieved by semantic similarity. A large part of an organization's knowledge, however, does not reside in documents at all but in relational databases, where it is held as structured records across tables of rows and columns. Answering a question such as how many orders a given customer placed last quarter cannot be accomplished by retrieving passages, because the answer is not written anywhere as text, so it needs to be computed from the data. Bridging natural language and structured data therefore calls into a formal database query that can be executed to obtain the answer. This task is known as text-to-SQL, and it is the structured-data counterpart to the document retrieval examined in the previous sections.

Text-to-SQL has a long history as a problem in semantic parsing, and its methods have evolved through successive paradigms. Early systems relied on hand-crafted rules and grammars, these laid the foundation for the sequence-to-sequence models trained to map a question, together with a representation of the database schema, to a SQL query. The introduction of Large Language Models has brought further development, as a sufficiently capable model can generate SQL through in-context learning, prompted with the schema and, optionally, a few worked examples, often without any task-specific training. This progression, and the current state of LLM-based approaches, are surveyed by Hong et al. (2025).

Among LLM-based methods, the manner in which the model is prompted is decisive. DIN-SQL (Pourreza & Rafiei, 2023) decomposes the task into a sequence of simpler subtasks, showing that this decomposition substantially improves the accuracy in comparison with generating a query in a single step. DAIL-SQL (D. Gao et al., 2023) examines how the database schema and the in-context examples should be represented and organized within the prompt, achieving leading execution accuracy on the Spider benchmark at the time of the study. Both illustrate a recurring theme, that the quality of generated SQL depends heavily on how effectively the model is supplied with and guided through the relevant information about the database.

This last point identifies the main difficulty of text-to-SQL. A model can only generate a correct query if it knows which tables and columns exist, what they contain, and how they relate to one another. The literature distinguishes two kinds of such knowledge. The first is structural, the keys and constraints that determine how tables may be joined, mainly the primary and foreign keys that link one table to another. Studies of schema representation as DAIL-SQL (D. Gao et al., 2023), find that making these relationships explicit in the prompt

allows a model to generate correct join conditions rather than guess from the superficial similarity of column names. The second kind is the content of the data itself, Rajkumar, Li, and Bahdanau (2022) show that including a small sample of a table's rows in the prompt supplies this information and measurably improves the accuracy of the queries a model generates.

A concern specific to structured data, and one with no counterpart on the document side, is the safety of execution. A generated SQL statement is not merely a read but runs against a database, and an unconstrained query could modify or delete data, and expose information that should remain private. Taking this into consideration, a safe text-to-SQL system must therefore constrain what may be executed, for example, restricting operations to read-only queries, limiting access to only an approved set of tables, and also setting a limit to the volume of data returned. These safeguards are integral to the design of the system presented in this dissertation and are described in Chapter 3.

2.5 Evaluation of RAG Systems and RAGAS

The techniques reviewed in the preceding sections offer many ways to construct and enhance a RAG system, but they raise an immediate practical question, "How do we know when a particular RAG design produces good answers?", evaluation is therefore an essential prerequisite in RAG system development. A RAG system has two distinct components whose quality must be assessed, the retriever and the generator, and a weaker answer can come from each of the components. Its outputs, moreover, are open-ended natural language, for which there is rarely a single correct wording. Traditional automatic metrics that compare a generated answer to a reference by surface overlap, such as exact match or the n-gram measures BLEU and ROUGE, are not suited to this setting (Zheng et al., 2023), since a factually correct answer expressed in different words can score poorly, while a fluent but wrong answer may get a good score. Constructing reference answers for every question is also slow and costly, which motivated the development of reference-free methods able to judge quality without a fixed gold answer.

Much recent progress has been made on using language models as evaluators. Zheng et al. (2023) examined this LLM-as-a-judge paradigm systematically and found that a sufficiently capable model, used to assess open-ended-responses, could agree with human judgments at a rate that exceeds eighty per cent, comparable to the agreement between human experts themselves, making this finding the foundation of automated, reference-free evaluation of generative systems. The same study was careful to document the weaknesses of this approach, identifying systematic biases to which LLM judges are prone, including a tendency to favor the first of two responses, position bias, a tendency to prefer longer answers, verbosity bias, together with limits to their reasoning. These points directly affect the interpretation of any evaluation built upon an LLM judge, including the one used in this work.

RAGAS (Es et al., 2024) applies this paradigm specifically to Retrieval-Augmented Generation, providing a framework that decomposes the quality of an RAG system into several metrics, with each one evaluating a different part of the RAG pipeline. One of the main goals of this framework is to lessen the dependence on manually written reference answers, and two of its metrics achieve this successfully. Faithfulness measures the extent to which the claims in an answer are supported by the retrieved context, being able to detect hallucination by comparing the answer directly to the retrieved passages. Answer relevancy measures how directly the answer addresses the question that was asked, penalizing responses that are

incomplete or that have irrelevant material, and so doesn't require a ground-truth answer. The remaining two metrics are related to the retrieval phase of the RAG system and, in the configuration adopted in this work, are computed against a reference answer (Ragas, 2025).

Formally, the four metrics adopted in this work are defined as follows:

Faithfulness is the proportion of the claims in the generated answer that are supported by the retrieved context (Equation 2.1).

$$\text{Faithfulness} = \frac{|\text{Number of claims in the response supported by the retrieved context}|}{|\text{Total number of claims in the response}|} \tag{2.1}$$

Answer relevancy is the mean cosine similarity between the original question and a set of N questions reverse-engineered from the answer by a language model, with N set to three by default (Equation 2.2). E with subscript gi denotes the embeddings of the i-th question reverse-engineered from the answer, and E with subscript o the embedding of the original question, so that answer relevancy is high when the questions reconstructed from the answer resemble the one actually asked.

$$\text{answer relevancy} = \frac{1}{N} \sum_{i=1}^{N} \cos(E_{g_i},\, E_o) \tag{2.2}$$

Context precision, evaluated at rank k over the retrieved chunks, rewards the placement of relevant chunks near the top by averaging the precision at each rank weighted by a binary relevance indicator (Equation 2.3). The precision @k is the precision computed over the retrieved top k chunks, and the v subscript k is a value between 0 and 1 that indicates whether the chunk at rank k is relevant, so the metric rewards the relevant chunks by ranking ahead of irrelevant ones.

$$\text{Context Precision@}K = \frac{\sum_{k=1}^{K} \big(\text{Precision@}k \times v_k\big)}{\text{Total number of relevant items in the top } K \text{ results}} \tag{2.3}$$

Context recall is the proportion of claims in the ground-truth answer that can be attributed to the retrieved context (Equation 2.4).

$$\text{context recall} = \frac{|\text{GT claims that can be attributed to context}|}{|\text{Number of claims in GT}|} \tag{2.4}$$

RAGAS is not the only framework to evaluate the quality of an RAG system. ARES (Saad-Falcon, Khattab, Potts, & Zaharia, 2024) evaluates along similar dimensions as RAGAS, but takes a different approach, it trains a lightweight language model to serve as a judge, with synthetically generated data that is calibrated with a small set of human annotations, so it

is possible to achieve a better reliability, the contrast with RAGAS, that operates in a zero-shot manner, illustrates a trade-off that is present in these types of frameworks, between annotation costs, and the robustness in automated evaluation. The RGB benchmark (Chen, Lin, Han, & Sun, 2024) evaluates four abilities in particular, the robustness to noisy retrieved passages, the integration of information across sources, resistance to misinformation, and negative rejection, which is the capacity to decline to answer when the retrieved context does not contain the answer. This last ability is of direct relevance to the present work, as its evaluation deliberately includes questions that cannot be answered from the corpus in order to test whether the system refuses to answer or hallucinates and creates a response.

For all their abilities, automated RAG evaluations carry limitations that must be acknowledged. They inherit the biases and the non-determinism of the LLM judge on which they rest, so that repeated evaluations of the same RAG system may lead to somewhat different results, they are also linked to the costs of all the calls that are made to the LLM that is evaluating, and individual metrics can behave awkwardly in some cases, for example when context recall is applied to a question where the correct answer is a refusal, for which the notion of recalling relevant information is not defined. Such considerations do not invalidate automated evaluation, although they require that its results be read with these limitations in mind.

On the basis of this review, the present work adopts RAGAS as the evaluation framework for document-based question answering, using its faithfulness, answer relevancy, context precision, and context recall metrics, using GPT-4o as the evaluation judge. As noted previously, these metrics presuppose retrieved textual context and are therefore not applicable to the structured-data path.

2.6 Research Gap

The literature reviewed in this chapter reveals several gaps that are particularly relevant to the enterprise setting addressed in this dissertation. Table 2.1 summarizes the main studies underpinning these observations, including their objectives, technologies, datasets, and evaluation approaches.

Table 2.1: Summary of the primary studies underpinning this work.

Study	Objective	Models & Technologies	Data / Benchmark	Evaluation
Lewis et al. (2020)	Introduce RAG, combining a retriever and a generator for knowledge-intensive tasks	DPR retriever and BART generator	Open-domain QA (NQ, TriviaQA, WebQuestions)	Exact match
Wang et al. (2024)	Identify best-practice RAG configurations, including effective chunk sizes	Comparison of embedders, rerankers and chunk-size settings	QA and information-retrieval benchmarks	Faithfulness, relevance, retrieval accuracy
Reimers & Gurevych (2019)	Produce sentence and passage embeddings comparable by cosine similarity for semantic search	Siamese/triplet BERT bi-encoder (Sentence-BERT)	STS benchmark, SentEval, NLI	Spearman correlation, accuracy
Karpukhin et al. (2020)	Learn dense passage embeddings for retrieval, surpassing term-based methods	Dual-encoder BERT (Dense Passage Retrieval)	Open-domain QA (NQ, TriviaQA, WebQuestions)	Top-k retrieval accuracy, exact match
Jagerman et al. (2023)	Reformulate and expand queries by prompting an LLM, including zero-shot	Prompted LLM (zero-shot, chain-of-thought, few-shot)	MS MARCO, BEIR	nDCG, recall
Nogueira & Cho (2019)	Re-rank first stage retrieval results with a cross-encoder for higher precision	BERT cross-encoder (joint query-passage scoring)	MS MARCO, TREC-CAR	MRR, MAP
Gao et al. (2023)	Compare prompt-engineering choices, for LLM text-to-SQL, and propose and integrated method (DAIL-SQL)	Multiple LLMs(GPT-4, GPT-3.5) with in-context learning; supervised fine-tuning of open-source LLMs	Spider, BIRD	Execution accuracy
Es et al. (2024)	Reference-reduced automated evaluation of RAG (RAGAS)	LLM-as-judge with component metrics	WikiEval	Correlation with human judgment

Three gaps emerge from this review. First, the techniques above are validated predominantly on public, English-language benchmarks. Whether their performance carries over to real enterprise corpora, which may be noisier and may use other languages, remains less well understood. Second, document retrieval and structured-data querying are studied as separate problems, each with its own literature, however, an enterprise assistant must serve both from a single system. Third, individual techniques are reported in isolation and typically on benchmark data, so there is little practical evidence as to which of them actually improve answer quality when applied, and combined, on a particular organization's real data, which is the guidance an organization would need in order to choose among them without resorting to costly trial and error. This dissertation addresses these gaps directly, as it builds a single system for both data types, applies it to a real Portuguese-language enterprise setting, and evaluates which architectural choices measurably improve the answers it produces.

Chapter 3

RAG System Design and Implementation

This chapter describes the design and implementation of the system developed in this dissertation. It presents the overall architecture, the technologies used, the enterprise data sources, and the two answering paths: document retrieval over unstructured content and text-to-SQL over structured data.

The chapter proceeds from the overall architecture and the technologies used, to the enterprise data sources and then to the two answering paths, the document retrieval over unstructured content and structured data querying through text-to-SQL, before closing with the application layer through which both are exposed.

3.1 System Overview and Architecture

The system follows a layered architecture that separates the tasks of request handling, orchestration, data access, and storage, allowing each to perform independently. A question enters the system through an application layer, implemented as a backend service that exposes an HTTP interface and routes each request to the appropriate processing logic. The core of the system is the retrieval-augmented generation engine, which orchestrates the retrieval of relevant information and then the generation of an answer. Within this core system, there is a data-integration layer that houses the connectors that reach the enterprise data sources, which in this case are the enterprise SharePoint and the SQL Server database, and a storage layer that holds only the vector index and the application's own metadata. In Figure 3.1, it is possible to see this overall system.

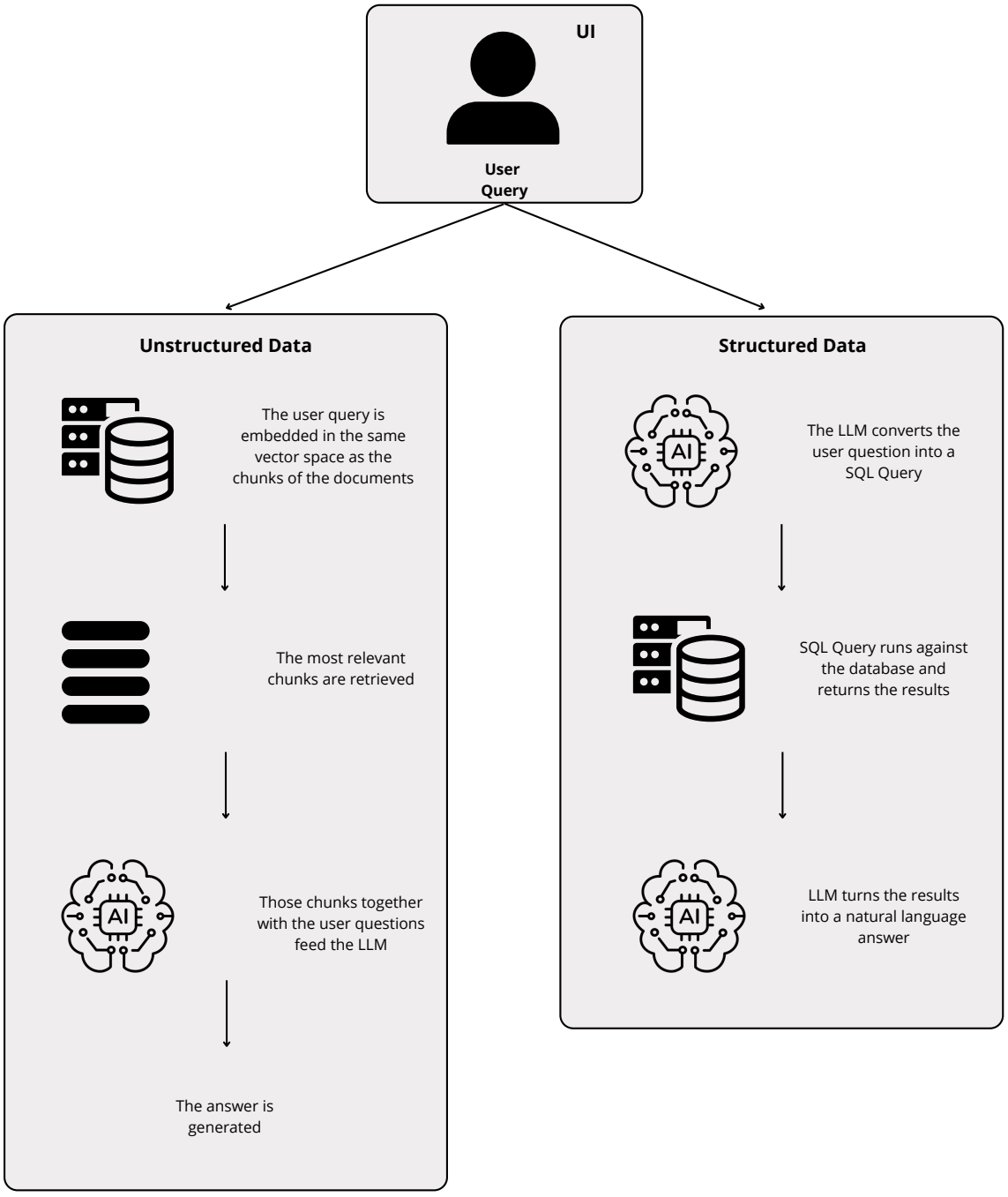


Figure 3.1: Overall architecture of the system.

A defining characteristic of the design is that a single system provides two distinct response paths, corresponding to the two types of enterprise data that it addresses. The first path answers questions over unstructured content, such as files stored in SharePoint, through semantic retrieval and generation, as reviewed in Section 2.3. The second path answers questions over the relational SQL Server database by translating the question into a database query, as reviewed in Section 2.4. The two paths share the same language model and application layer, but differ in how they obtain the information needed to answer a question. The document path retrieves relevant text passages, whereas the structured-data path executes SQL queries. The user explicitly selects the desired path through a toggle in the interface.

Data moves through the system in three main flows. The first is the ingestion and indexing flow. When the user connects the system to SharePoint, documents are retrieved, parsed, divided into chunks, converted into embeddings, and stored in the vector index. The second flow is triggered when a user submits a question about the documents. The question is embedded using the same model used for the document chunks, relevant chunks are retrieved, and these are included in a prompt sent to the language model, which generates the answer together with citations. The third flow handles structured-data questions. The language model converts the user's question into a read-only SQL query, which is validated and executed against the database. The returned rows are then passed to a second model call that generates the natural language answer shown to the user.

The storage layer reflects the same separation of concerns. Document embeddings, together with their metadata that is needed to cite their source, reside in a dedicated vector database, while the application's own operational data, which includes the documents that have been indexed together with a log of the questions and answers that it has served, are kept in a lightweight relational database, separate from any enterprise data. The enterprise structured data, by contrast, remains in the existing SQL Server database. The system only connects to this database to execute read-only queries and does not modify the stored data. This separation ensures that the application's own data remains separate from the organization's enterprise data.

Finally, the architecture is set up to be configuration-driven, with each component, the chunking strategy, the embedding model, the number of chunks retrieved, the prompts and the optional retrieval enhancements, being governed by centralized settings rather than being hard-wired into the logic that uses it. This enables the experimental comparison of pipeline configurations, reported in Chapter 4, making it a matter of changing settings rather than rewriting code.

3.2 Technology Stack

The system is implemented in Python and uses a set of technologies selected according to their suitability for the requirements of each component. Table 3.1 summarizes the main technologies and the rationale for their selection.

Table 3.1: Technologies used in the system and the rationale for each choice

Component	Technology	Rationale
Programming Language	Python	Dominant language of the machine-learning and RAG ecosystem, with libraries for embeddings, vector stores and LLM APIs
Backend / API	FastAPI	Modern framework with automatic request validation and lightweight support for API endpoints
Document Parsing	pypdf, python-docx	Well established libraries for extracting text from PDF and Word documents
Embedding model	OpenAI text-embedding-3-large	Strong multilingual retrieval quality, and a managed API removes the need to host the embedding model
Vector Store	ChromaDB	Lightweight vector store with local persistence, suitable for the scale and single-user nature of the system
Answer Generation and Query Rewriting	OpenAI gpt-4o-mini	Capable model for instruction-following, with low cost and low latency
Reranker	bge-reranker-v2-m3 (cross-encoder)	Multilingual cross-encoder supporting Portuguese; as a small model, it can run locally without the need for an additional API dependency
SQL Server access	SQLAlchemy	Standard, database-agnostic toolkit with connection pooling
SharePoint connector	MSAL + Microsoft Graph API	The official Microsoft authentication and access path for SharePoint content
Application Metadata	SQLite	Zero-configuration database, sufficient for the system's own records and query logs

The embedding model, which sets the ceiling on retrieval quality, was taken from OpenAI's text-embedding-3 family in its larger variant. Open-source multilingual embedders, such as E5 and BGE families, were viable candidates, but the managed model was preferred for its strong reported performance on multilingual retrieval (OpenAI, 2024c), and because it removes the operational cost of hosting an embedding model on dedicated hardware. The generative model, used both to produce answers and to rewrite queries, is gpt-4o-mini, the smaller tier of the GPT-4o family, that keeps cost per-query low while remaining capable enough for question answering.

A consideration follows from this reliance on managed, third-party models for both em-

bedding and generation, on one side, running open-source models of comparable capability would require substantial dedicated hardware to host and serve them, which would tie the system's quality to the compute available on the machine that is running it, a managed service avoids this, delivering strong models at a low operational cost and without the need to host them. This choice also has implications for enterprise data processing. Document chunks and user queries are sent to an external service during embedding, while prompts containing retrieved enterprise content are sent to the language model during answer generation. Therefore, using managed models introduces a dependency on an external provider and requires consideration of data confidentiality when the system is deployed with sensitive enterprise information. Local open-source models would provide greater control over data processing and reduce this external dependency, although they would require additional infrastructure and model-hosting capabilities.

ChromaDB was selected on similar grounds. Hosted alternatives such as Weaviate or a bare index built on a similarity-search library like FAISS could be justified for a large, multi-user deployment, but they introduce external service or additional infrastructure that a single-user system does not require. ChromaDB runs in-process, persists in a local directory, and performs exactly the operations that the system needs.

3.3 Enterprise Data Sources

As already mentioned, the system connects to two fundamentally different enterprise data sources, each one being handled by a different subsystem. The first is a collection of unstructured documents that live in a SharePoint library, the second is a structured relational database hosted on SQL Server. Both are drawn from real work carried out with the company B2F.

The unstructured source is a SharePoint document library containing the documentation for a real enterprise data platform and reporting projects. It has around 20 documents, in the common office formats that the ingestion supports, PDF and Word files, written in Portuguese. The content in these documents has two types, the first being technical documentation of the projects and documents that record their progress, such as transcriptions and meeting notes. For the present work, what is relevant is that these are genuine enterprise content rather than a curated public dataset, and that answering questions about its content is the task of the document retrieval subsystem described in Section 3.4.

The database follows a star schema, in which fact tables record business events and dimension tables provide descriptive information about those events. The system is granted access to seven of its tables, listed with their sizes in Table 3.2. Two of these tables are fact tables: one is the FACT_VENDAS, which records individual sales and is by far the largest table, with almost 1.4 million rows, and the FACT_STOCKS, which records the stock levels on a given day. The remaining five are dimension tables that describe the clients, products, dates, operators, and stores to which the facts refer. Answering natural language questions over this database by translating them into SQL is the task of the structured-data subsystem described in Section 3.5.

Table 3.2: SQL Server tables of the structured data source.

Table name	Description	Rows
FACT_VENDAS	Holds the retailer's sales, one row per sale, recording the product sold together with the store, operator, client, and date of the transaction, and the quantity and value involved.	1,388,785
FACT_STOCKS	Records stock levels, capturing the quantity of each product held in stock at each store.	812
LK_CLIENTE	Describes the retailer's clients, one row per client, with the attributes used to identify and group them.	7,214
LK_PRODUTO	Contains information on each existing product, including its identifying details and the category to which it belongs.	1,107
LK_DATA	The calendar dimension, with one row per date, providing the temporal attributes (day, month, year) by which the facts are aggregated over time.	5,844
LK_OPERADOR	Describes the operators who register the sales at the point of sale.	58
LK_LOJA	Contains the structural data of the retailer's stores, one row per store.	5

3.4 Document Retrieval System

This section describes the subsystem that answers questions over unstructured data stored in SharePoint. The pipeline consists of four stages: document ingestion (Section 3.4.1), document processing and chunking (Section 3.4.2), embedding and indexing (Section 3.4.3), and retrieval and generation (Section 3.4.4). The first three stages operate offline to prepare the knowledge base, while the fourth is executed when the user submits a question. Section 3.4.5 then presents the query rewriting and reranking variants evaluated in Chapter 4.

3.4.1 Ingestion

The system gets its documents from a single source, an enterprise SharePoint document library, to which it connects. Ingestion prepares the knowledge base before any question, accompanied by a status facility that reports the current phase, indexing, ready, or error, so that the process can be observed. The flow is shown in Figure 3.2.

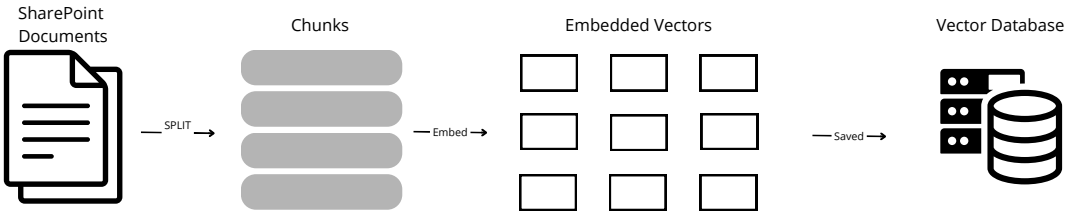


Figure 3.2: Indexing pipeline.

Access to SharePoint is obtained through the Microsoft identity platform. The system authenticates as an application in its own right, rather than on behalf of a signed-in user, using the client-credentials flow against the organization's Azure directory and acquiring a token for the Microsoft Graph API. With this token, it resolves the configured site, obtains the site's default document library, and walks that library recursively, going into each folder and sub-folder, being able to enumerate every file it holds.

Each file that was discovered is only considered if its extension belongs to a configured set of supported types, as files of other types are passed over and counted as unsupported. A supported file is then downloaded into memory and handed to the parsing and indexing stages described in the following subsections. It is also crucial to retain the file's location in SharePoint, which comes as a web address, so it is attached to every chunk that is later derived from the file, which will allow the answer to cite the document from which its information came.

Because a document library changes over time, the ingestion process is incremental, as the content from each file is reduced to a hash, and the system keeps a record of every document it has indexed, holding that hash together with the number of chunks the document has produced. On a later ingestion, a file whose hash is unchanged is skipped, a file whose hash changed has its previous vectors deleted and is indexed again, and a file that wasn't previously in SharePoint is indexed for the first time.

3.4.2 Document Processing and Chunking

Before a document can be indexed, its bytes must be turned into plain text, and that text divided into the units that will be retrieved. Parsing is dispatched according to file type, a PDF is read page by page, and each page's number is preserved so that it can later be mentioned in the citation. A page from which no text can be extracted is discarded. A Word document has both its paragraph text and the text of its table cells extracted and joined together. In this process, a document from which no text at all can be recovered is rejected rather than indexed empty, so that the knowledge base contains only genuine content.

The extracted text is then divided into chunks by the recursive, overlapping strategy discussed in Section 2.3.1, with this strategy implemented directly within the system, so that its behavior is fully understood and controlled. Sizes are measured in tokens rather than characters, using the same tokenizer family as the models the system relies on, a chunk is limited to four hundred tokens, and consecutive chunks overlap by fifty tokens, which is a size that falls within the range mentioned in the empirical studies reviewed in Section 2.3.1 that were found most effective, being neither so small as to fragment context nor so large that they dilute relevance.

The chunk splitter works through a hierarchy of separators, where a paragraph breaks first, then the line, then the sentence boundaries, then the word boundaries, and only as a last option the individual tokens. This way, a span of text that already fits within the token limit is kept whole, while one that exceeds it is split at the next separator in the hierarchy, keeping the granularity trade-off discussed in Section 2.3.1.

Two further steps refine the result. Adjacent pieces that are individually small are merged together until they reach the token limit, leading to a lower number of fragmented small chunks in the database. Each chunk after the first contains the final fifty tokens of its predecessor, so that the context from the last chunk is kept in the present one. In the end, each chunk contains the identifier of its source document, from the document's SharePoint address to its position on the sequence.

3.4.3 Embedding and Indexing

With the corpus of the documents turned into chunks, the final offline stage turns each chunk into its vector representation, on which the semantic search depends, and stores it in the vector index, completing the knowledge base that will be used by the model to generate the answers. This stage applies directly the notions reviewed in section 2.3.2, where a dense embedding places each piece of text at a point in a high-dimensional space where proximity reflects similarity of meaning, and the vector database organizes those points so that the nearest to a query can be found in a more efficient way.

Embeddings are produced by the OpenAI text-embedding-3-large model, whose selection was argued in Section 3.2 on the grounds of its multilingual retrieval quality and the convenience of a managed service that removes any need to host an embedding model locally. Each chunk's text is sent to the model and returned as a dense vector, and in order to keep the indexing of the whole collection of chunks more economical, those embeddings are done in batches rather than one request at a time. The same model is used later to embed each incoming question, so that user queries and stored chunks share the same vector space, making their similarity meaningful.

The resulting vectors are stored in Chroma DB, which the system runs in-process and saves in a local directory, for the reasons already explained in Section 3.2. Each vector is held in a collection along with the chunk's own text and its metadata, so that a search can also return the passage and the information needed to cite it. Because the system is a single-user prototype, one shared collection holds the entire corpus, however, the design can accommodate several collections with minimal changes above the storage layer. Every stored vector is matched by an identifier that combines the collection, the source document, and the chunk's position within it. The identifier also supports precise and efficient incremental re-indexing. When a document is added or modified, only the vectors associated with that document need to be created or removed, leaving the rest of the knowledge base unchanged.

Within a collection, ChromaDB maintains an approximate-nearest-neighbor index as discussed in section 2.3.2, which allows the chunks closest to a query vector to be found without an exhaustive comparison against every chunk. In that previous section, it was presented that the geometric proximity could be measured in more than one way, so this system adopts the vector store's default, the squared Euclidean distance between the query vector q and a stored chunk vector c (Chroma, 2025), defined in Equation 3.1.

$$d(q, c) = \|q - c\|^2 = \sum_i (q_i - c_i)^2 \tag{3.1}$$

Although the underlying computation is Euclidean, the ranking that is produced is equivalent to the one by cosine similarity, that is the measure of semantic closeness, represented by Equation 3.2, this happens because the vectors that are produced by the text-embedding-3-large are unit-normalized (OpenAI, 2024a), which means that for two vectors of unit length, the squared Euclidean distance and the cosine similarity can be related by Equation 3.3, which makes that a smaller distance corresponds exactly to a larger cosine (OpenAI, 2024a). Ordering the chunks by ascending distance is therefore identical to ordering them by descending cosine similarity, additionally, the choice of the squared Euclidean distance does not carry any loss relative to an explicit cosine metric.

$$\cos(q, c) = \frac{q \cdot c}{\|q\| \, \|c\|} \tag{3.2}$$

$$\|q - c\|^2 = 2\big(1 - \cos(q, c)\big) \qquad \text{for } \|q\| = \|c\| = 1 \tag{3.3}$$

At search time, the system converts this distance into a relevance score that sits in the interval between 0 and 1, defined by Equation 3.4. The score decreases as the distance grows, and the constant of one keeps it defined at zero distance, so an exact match scores one. Therefore, higher scores represent closer matches, giving the retrieval stage a scale on which to order the candidate chunks.

$$s(q, c) = \frac{1}{1 + d(q, c)} \tag{3.4}$$

3.4.4 Retrieval and Generation

When a user asks a question, the system enters its query phase, in which the knowledge base that was assembled is put to use. That flow is represented in Figure 3.3. The first step is to bring the question into the same representational space as the stored chunks, so the question is embedded with the same model that was used to index the chunks, the resulting query vector is then submitted to the vector index, performing the retrieval stage, which returns the top-k chunks whose embeddings are nearest to it, under the similarity measure defined in Section 3.4.3. In the baseline architecture of the system, k is set to five chunks, which are passed on directly to the model. Each retrieved chunk goes with the metadata attached to it at indexing, in particular the path of its source document together with its page number, so that later each statement in the answer can be traced to its origin.

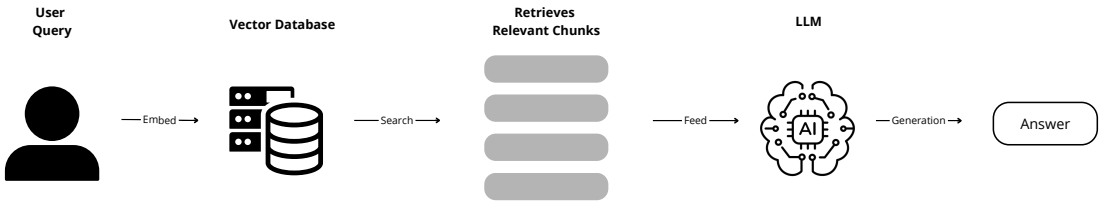


Figure 3.3: The baseline query pipeline.

The retrieved chunks are then assembled into the prompt from which the answer will be generated. They are concatenated into a single context block in which each chunk is numbered and prefixed with its source and page, so that the generator can refer to a specific passage when citing it. This block is combined with the user's question and a set of instructions into one prompt that has a deliberate structure, first, the context is presented as the basis of the answer, with the question being placed after it, following a common practice in prompt construction, where the task remains adjacent to the point at which generation begins, rather than being separated from it by a long block of retrieved text. The context is based on the five chunks that were retrieved, and as each one of the chunks has at most four hundred tokens, this assembled context together with the user's query amounts to no more than a few thousand tokens, which sits well within the 128,000 token context window of the gpt-4o-mini (OpenAI, 2024b), the model that generates the answer.

This model was chosen for the reasons given in Section 3.2 and is instructed to compose its answer only from the supplied passages together with further requirements that shape its behavior. It is directed to respond in Portuguese, a deliberate choice given that the enterprise setting is mostly in Portuguese. It also requires citing the passage that supports each affirmation in the answer, in the form of a bracketed number placed immediately after the assertion rather than gathered at the end, so that the sources of an answer can be verified claim by claim.

3.4.5 Architecture Variants

The baseline of the previous section can be improved at two independent points, each corresponding to a technique reviewed in Chapter 2. The first acts before retrieval, reformulating the question so it matches the knowledge base more closely, the second acts after retrieval, re-examining the chunks it returns and reordering them before generation. Either can be used together, or independently, therefore, defining four distinct configurations of the document retrieval subsystem, the baseline alone, the baseline with query rewriting, the baseline with reranking, and the baseline with both. These configurations are the object of the experimental comparison demonstrated in Chapter 4.

The first augmentation is query rewriting, introduced in Section 2.3.5, which addresses the mismatch between the language of a question and the language of the documents that answer it. This augmentation technique is placed at the start of the query phase, before the question is embedded, and rather than encoding the question that the user wrote, the system first passes it to a language model, in this case the gpt-4o-mini, the same one used for generation, with an instruction, issued at temperature zero for determinism, to rewrite it as a single, clear, and self-contained search query. The model is asked to make implicit intent explicit, to clarify vague phrasing, and to expand abbreviations, while being explicitly forbidden to answer the question or to introduce new content that was not already mentioned in the original question,

at the same time that it needs to keep the query in Portuguese. In keeping with the zero-shot, prompt-only form justified in Section 2.3.5, the instruction carries no worked examples, and no rewriter model is trained. The arrangement is shown in Figure 3.4.

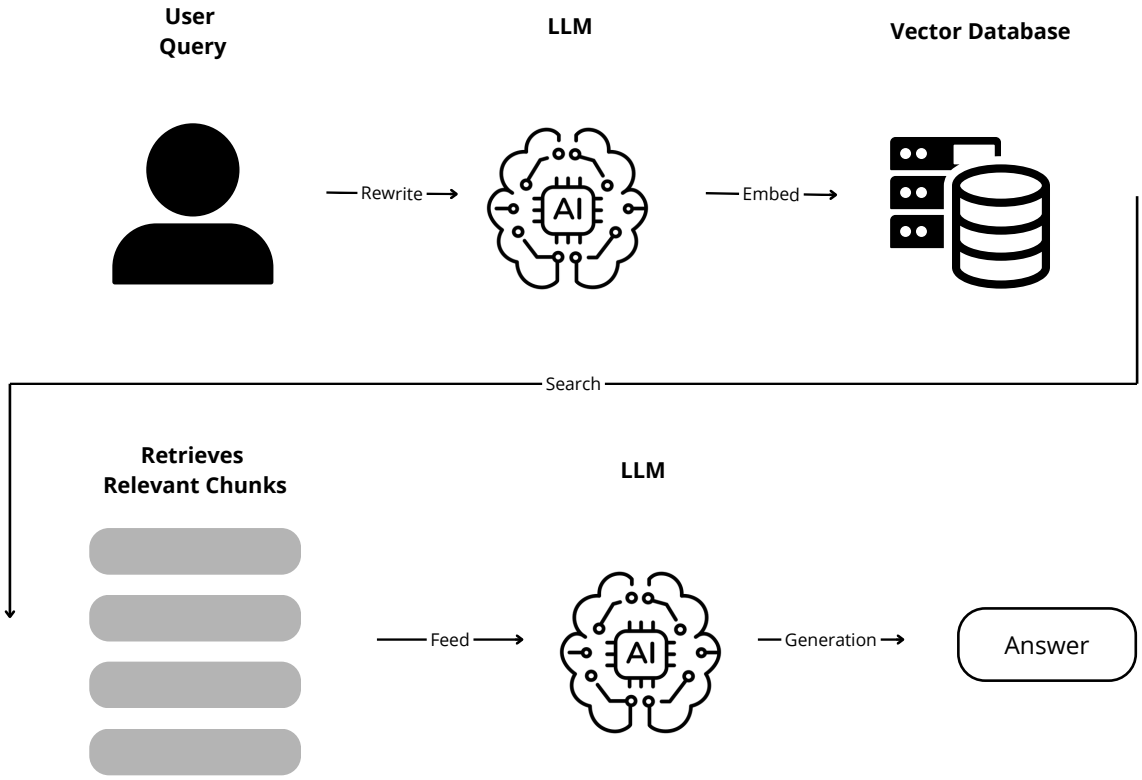


Figure 3.4: The query rewriting architecture.

A deliberate choice governs how the rewritten query is used. It replaces the original question only for the purpose of retrieval, as the answer is still generated from the question as the user actually wrote it. This separation guarantees that even though this reformulation is useful for matching the knowledge base, it does not change what the system is ultimately asked to answer. This step is also defensive, meaning that if the model fails or does not return anything, the system uses the original question, so that it still performs the retrieval even without any rewrite at all.

The second augmentation is reranking, mentioned in Section 2.3.6, which aims to improve the order and selection of the retrieved chunks. With this technique, the subsystem performs a two stage retrieve and then reranking process, the first stage keeps being a dense retrieval, however it is widened to return a larger pool of candidate chunks, retrieving now twenty chunks, instead of the five retrieved by the baseline architecture, then in the second phase, a more discerning model scores that pool, retaining only the best five for generation. The generation therefore still receives five chunks, but five that have been chosen and ordered with greater precision, as is possible to see in Figure 3.5.

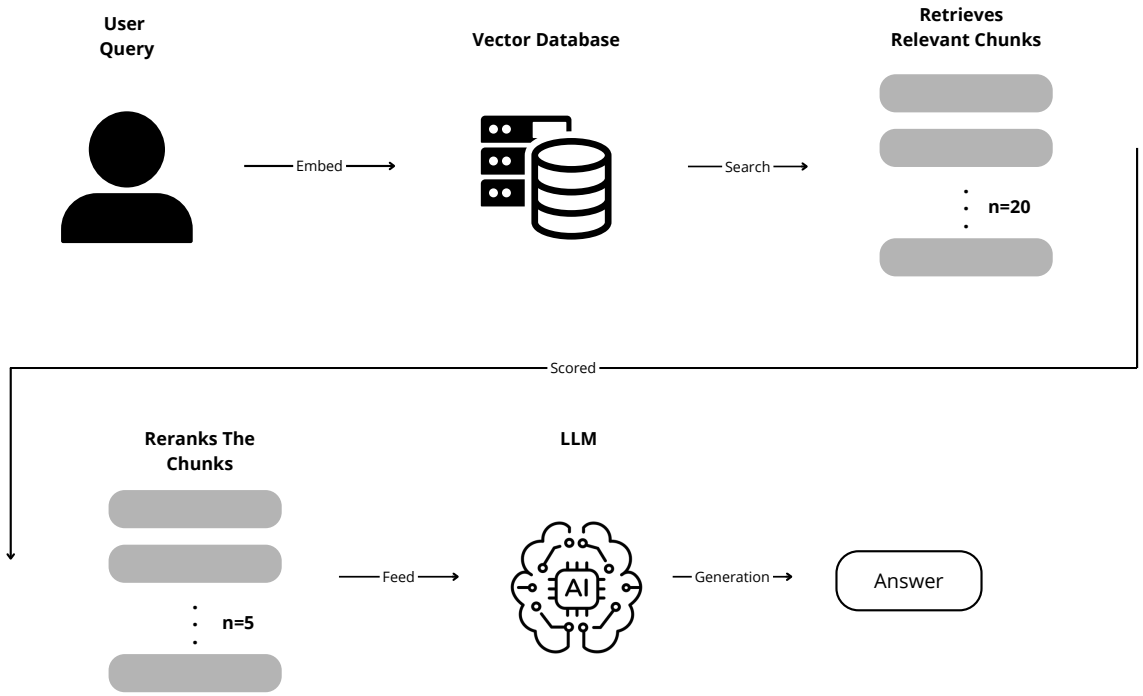


Figure 3.5: The reranking architecture.

The reranker is the multilingual bge-reranker-v2-m3 cross-encoder, which runs locally through the Sentence Transformers library. Its manner of operation differs from the retrieval baseline architecture, as the bi-encoder embeds the question and each chunk separately, and then compares the two vectors, the cross-encoder forms every question-chunk pair and then compares the pair through the model, allowing it to range across question and passage together (Figure 3.6), producing a more accurate estimate of relevance. Each candidate is scored this way, and the candidates are then reordered by score, with only the five most relevant chunks. As this is a relatively small model, it was set to run locally, meaning that an increase in precision that can be achieved does not add any further dependence on an external service, and, like rewriting, its effect on answer quality is left to be measured in Chapter 4.

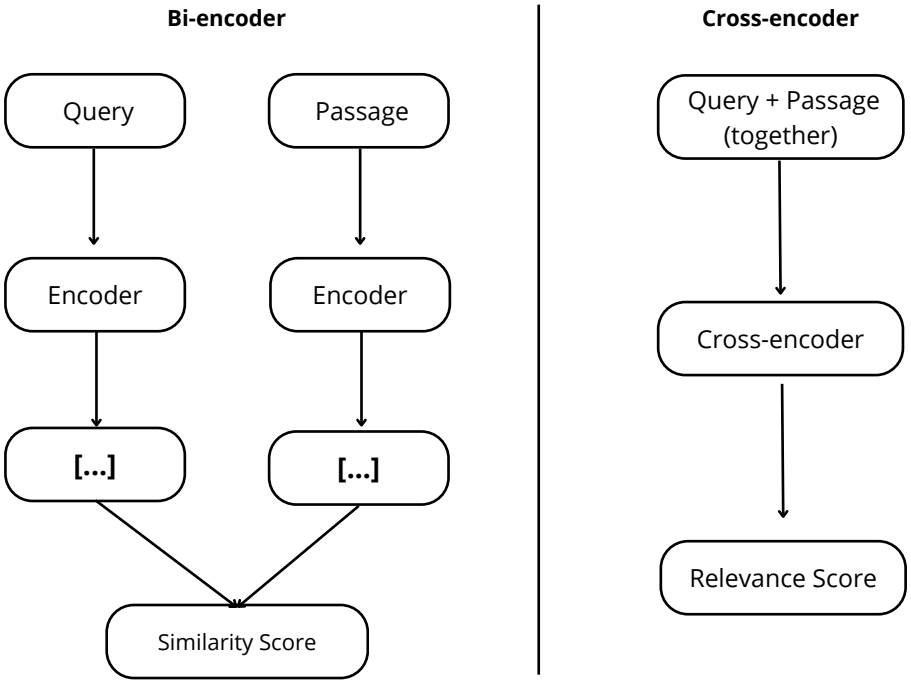


Figure 3.6: Comparison between Bi-encoder and Cross-encoder scoring

These two augmentation techniques are independent of each other, but they can be both combined in the same architecture, in that case, a question would first be rewritten, then that rewritten query retrieves a pool of twenty candidates that the cross-encoder reranker reduces to five, and the answer is generated from those five with reference to the original question. This architecture is also evaluated in Chapter 4 to see how it compares to the other ones.

3.5 Structured-Data System

Alongside the documents, the system can also answer questions about structured enterprise data, in this case, the SQL Server data warehouse described in Section 3.3. Because the answer to such questions isn't written anywhere as text, it needs to be extracted from the records in the database, meaning that this subsystem cannot rely on the retrieval of specific chunks, so it translates the questions into a database query. It follows a two stage process, as it has two calls to a language model. First, it converts the user's query into a SQL query, and then it turns the query's results into a natural language answer.

For the model to generate a correct query, it needs to be told what the database contains, and the manner in which the schema is presented to it is the central design decision of this subsystem. The minimum is the database schema, the accessible tables, their columns, and their respective types. The system goes beyond this minimum, enriching the schema with two types of information identified in Section 2.4. The structure of the tables and the content they hold. Whether this enrichment actually improves the queries the model produces, compared to the bare schema, is subject to evaluation in Chapter 4.

As the database has a particularly small schema, with only seven tables, for each table, a CREATE TABLE definition is used, listing its columns and types, and also the primary and foreign keys that reference another accessible table. These keys make the relationships between tables explicit, so that the model can develop the correct join conditions rather than

guessing them from the similarity of column names. Beneath each definition is appended a small sample of the table's content, three example rows, which reveal the actual form of the values a column holds, information that a column's declared type alone does not give. The whole assembled context is built once and then saved, as it does not change from one question to the next. The resulting flow and the context on which it draws are shown in Figure 3.7.

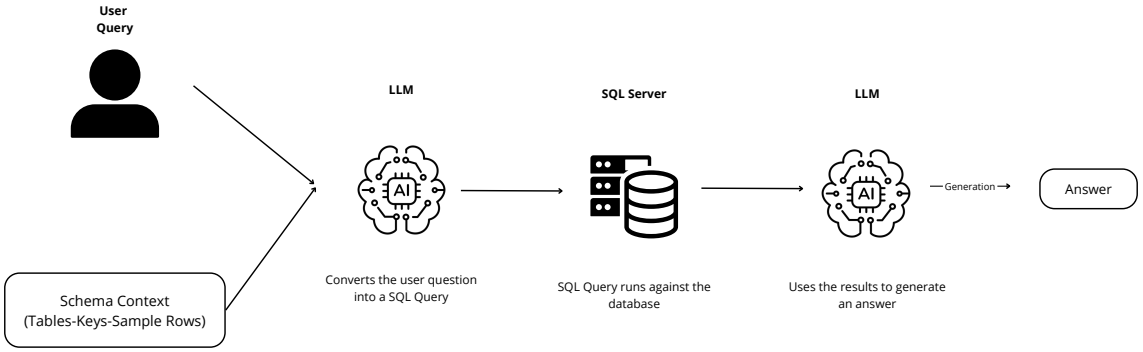


Figure 3.7: The text-to-SQL flow.

One limitation of the design in the way that is present can be clearly identified, as describing the full schema is only possible because it is small, in a database with a greater number of tables, the full schema would be too large to supply at once, and the most relevant tables would first need to be selected, and that approach is identified as a direction for future work in Chapter 5.

With the schema context prepared, the first model call turns the user question into a SQL query. The model, the same gpt-4o-mini used in the unstructured data subsystem, receives the question and the schema context, under the instructions to produce one SQL query that can be a single SELECT or a common table expression, as raw SQL.

Because the generated query runs against the production enterprise database, its execution needs to be constrained. Before execution, the query is validated to reduce the risk of data or schema modification. The query must begin with a SELECT statement or a common table expression, and statements containing operations such as INSERT, UPDATE, DELETE, DROP, or ALTER are rejected. The number of returned rows is also capped at one hundred.

The rows returned by the query are not shown directly to the user, a second model call turns those rows into a natural language answer. The question, the query, and its results are given to the model with an instruction to respond in Portuguese, with complete sentences, and using only the values present in the returned rows. As in the document subsystem, a grounded refusal is provided for when the returned rows do not support an answer, the model returns a fixed statement to the user.

3.6 Application Layer and Configuration

The two paths described in the previous sections are exposed through the application layer, a backend service built with FastAPI, that presents an HTTP interface over which the client submits a question and receives an answer. Each request is routed according to the path it is meant for, with one endpoint answering questions over the documents, invoking the retrieval-and-generation pipeline, and a separate endpoint that answers the question over

the database, invoking the text-to-SQL pipeline. On its own, the application layer has no answering logic, it only receives each request and calls the appropriate subsystem, returning the result in the end.

The system keeps a small amount of operational data in a lightweight SQLite database, which is entirely separated from the enterprise sources, where two types of records are saved. The first is a registration of the documents that have been indexed, storing for each one its source, a content hash, and the number of chunks that it produced, it is the content hash that enables the index to be incremental, since a document can be recognized and skipped if it did not have any changes. The second is a log of the questions that the system answered, with each entry recording the question, the answer returned, the sources, and also the time that it took to answer.

The configurations of the system are governed by a single central configuration, so rather than scattering choices such as the models to use, the number of chunks to retrieve, the size of the reranking pool, the number of sample rows to include, and the row limit of the query, these are collected in one settings module and read from it when they are needed.

Finally, a thin web client provides the interface through which a user interacts with the system, submitting a question and reading the answer together with the sources cited in support of the answer. The interface also includes a configuration page, where the user can connect the system to the enterprise SQL Server database, allowing a non-technical user to connect the system to an organization's own data without touching the code. As this self-service configuration currently only covers the structured data source, extending it to the SharePoint document source is identified as future work.

Chapter 4

Evaluation and Results

This chapter evaluates the system designed in Chapter 3. Its purpose is to establish empirically whether the design choices that were left open to evaluation actually improve the quality of the answers the system produces, with both of the system's answering paths being evaluated. For the document retrieval path, the four configurations defined in Section 3.4.5 are compared: the baseline, the addition of query rewriting, the addition of reranking, and the two combined. For the structured data path, the two schema representations defined in Section 3.5 are compared: the bare minimum schema and the schema enriched with keys and sample rows. Because the two paths differ in the nature of their outputs, they are evaluated by different means that are set in Section 4.14.1, over the datasets described in Section 4.2. The results are presented in Section 4.3.

4.1 Evaluation Process

The two answer paths call for different methods of evaluation, as their way of giving the user the answer to their query is different, meaning that the output is also different. The document retrieval path returns open-ended natural language answers, for which, as seen in Section 2.5, metrics based on surface overlap with a reference are unsuitable, therefore, it will be evaluated with the RAGAS framework. The structured data path returns answers computed from database records, for which the notion of a single correct value applies, so it can be evaluated by direct comparison against that value.

The document retrieval path is assessed with four RAGAS metrics defined in Section 2.5, faithfulness, answer relevancy, context precision, and context recall. GPT-4o is used as the evaluation judge. Each metric is computed by the judge from a different combination of the question, the generated answer, the retrieval passages, and the ground-truth answer, and because the interpretation of the results depends on how each score is computed, the four are described below.

Faithfulness measures whether the answer is grounded in the retrieved passages. The judge first decomposes the generated answer into its individual claims and then, for each claim, determines whether it can be deduced from those passages, with the score being the proportion of claims that are supported by those passages, as we saw in Equation 2.1. An answer in which every statement follows from the context scores one, while an answer that mentions something that the context does not support is penalized in proportion.

Answer relevancy measures whether the answer actually addresses the question asked.

From the generated answer, the judge reverse-engineers several questions, for which the answer would be a fitting response, with each one of those being embedded and compared by cosine similarity with the embedding of the original question, with the mean being the score, as given in Equation 2.2. An answer that speaks directly to the questions yields reconstructed questions close to the original and scores highly, and an evasive and incomplete answer drifts away from it. A consequence of this method, important to read the results, is the refusal, as an answer that declines to respond does not give any genuine question to reconstruct, therefore, it is assigned a relevancy score of essentially zero.

Context precision assesses the retrieval rather than the generation. From the passages that are retrieved, how many of those are relevant to the question and whether the relevant ones are ranked ahead of the rest. The judge marks each retrieved passage as relevant or not with respect to the ground-truth answer, and the metric rewards the relevant passages that appear near the top of the list, as was possible to observe in Equation 2.3.

Finally, context recall measures whether the retrieved passages contain the information required to answer the question. It compares the information present in the ground-truth answer with the information that can be found in the retrieved passages (Equation 2.4).

The structured data path is evaluated differently, for the reason already given in Section 2.5, with evaluation frameworks such as RAGAS assuming a retrieved textual context from which an answer is generated, the text-to-SQL system has none of it, as its answer is computed from database rows not drawn from any passages. Each evaluation question has a known correct value, established by querying the database by hand, against which the system's final answer is compared. An answer is counted as correct when it conveys the same result as the ground-truth value. The measure reported is therefore an execution accuracy, the proportion of questions for which the final answer presents the correct value, computed for each of the two schema configurations.

4.2 Evaluation Datasets

Document retrieval is evaluated over a set of twenty-four Portuguese questions, each one paired with a ground-truth answer, drawn from the domain of the SharePoint corpus described in Section 3.3. The questions were written with the goal of exercising the corpus across its range, addressing factual details present in the content, from specific platform integration steps to detailed matters recorded in the meeting notes. Because the questions were written directly from the content of the documents, each ground-truth answer was taken from the passage on which its question was based, and checked against that passage. The set is deliberately small, as each question is evaluated four times, one per configuration, with each evaluation needing several judge calls to gather the results for all of the metrics. Two of the twenty-four questions are negative tests, which means that the answer for those questions is not present in the documents, so the correct response is a refusal, which allows us to check if the system correctly declines to answer when it does not have information to generate a response, rather than inventing one, which would be a hallucination case.

The structured data path is evaluated on fifteen questions about the retail database of Section 3.3, ranging from simple aggregate counts to questions that require filters, groupings, and joins across several tables. Each question is paired with its correct answer, which was determined by querying the database directly so we could use them as ground-truth answers. Together, the fifteen questions aim to exercise the kind of requests the system is intended to

serve, with specific cases such as filtering by the name of a store rather than by its code, that confirm whether the model generates an answer based on the actual content of the database.

4.3 Results

The results of the two paths are presented one after the other, beginning with the document retrieval path. Table 4.1 reports the four RAGAS metrics for each one of the four architecture configurations, averaged over all twenty-four questions.

Table 4.1: RAGAS metrics for the four document-retrieval configurations, averaged over all twenty-four questions.

Configuration	Faithfulness	Answer relevancy	Context precision	Context recall
Baseline	0.733	0.534	0.533	0.802
Query rewriting	0.672	0.632	0.570	0.786
Reranking	0.739	0.579	0.556	0.740
Rewriting + Reranking	0.714	0.593	0.561	0.740

These averages must be read with care because the presence of two negative test questions in the batch of questions can distort them. On those two negative tests, the system did exactly what was intended, as it refused to answer in every configuration, however, as RAGAS is not designed to score a refusal, the metrics faithfulness, answer relevancy, and context precision all fall to zero. In a batch of twenty-four questions, having two that score zero is enough to lower the averages. Therefore, Table 4.2 repeats the metrics over the twenty-two answerable questions, with the two negative tests being reported on their own terms, since every configuration correctly refused both of them.

Table 4.2: RAGAS metrics for the four document-retrieval configurations, averaged over twenty-two answerable questions, with the two negative tests excluded.

Configuration	Faithfulness	Answer relevancy	Context precision	Context recall
Baseline	0.799	0.582	0.581	0.784
Query rewriting	0.733	0.689	0.622	0.767
Reranking	0.806	0.631	0.607	0.716
Rewriting + Reranking	0.779	0.646	0.612	0.716

Starting with the baseline architecture, as it is the one that the others are compared to, it presented a faithfulness score of 0.80, indicating that the generated answers were generally well supported by the retrieved passages. Its context recall, at 0.78, indicates that the retrieved passages usually contain most of the information required to answer the questions. The

remaining two scores are lower, as context precision was 0.58, indicating that relevant passages arrive mixed with a considerable number of less useful ones and are not always well ordered, and answer relevancy also had 0.58, showing that, despite the answers being grounded in the passages that were retrieved, they do not always address the question asked by the user.

The differences between the other configurations are modest and not linear. Every augmentation raises answer relevancy and context precision above the baseline, while all three augmented configurations reduce context recall. Query rewriting produced the largest single improvement, lifting answer relevancy from 0.58 to 0.69, and also the context precision from 0.58 to 0.62, but it does so at the expense of faithfulness that decreases from 0.80 to 0.73, suggesting that the reformulated queries retrieve passages that better match the question. This suggests a trade-off between improving the relevance of the retrieved information and maintaining faithfulness to the retrieved context. Reranking improves the same three metrics by a smaller margin. Unlike query rewriting, reranking does not reduce faithfulness relative to the baseline, while improving answer relevancy and context precision. The results of the combination of the two augmentation techniques sit between their individual scores, with slightly reduced faithfulness in comparison with the baseline, and answer relevancy and context precision that have values between the scores for the two augmentation techniques alone. Therefore, the results show that no single configuration dominates across all four metrics.

Beyond the two negative tests, the system sometimes refused to answer a question that the corpus had the information to answer, which is a retrieval failure rather than a proper refusal. As Table 4.3 shows, the number of such failures varied with the configuration, falling from three under the baseline to only one when rewriting and reranking were combined, while query rewriting alone raised it slightly. The lower number of these failures in both architectures with reranking suggests that reranking may help the system retrieve more relevant passages.

Table 4.3: Retrieval failures (answerable questions wrongly refused) by configuration, out of the 22 answerable questions.

Configuration	**Retrieval failures**	**Percentage**
Baseline	3	13.6%
Query rewriting	4	18.2%
Reranking	2	9.1%
Rewriting + Reranking	1	4.5%

The structured data path shows a clearer effect. Table 4.4 reports the execution accuracy of the two schema configurations over the fifteen questions.

Table 4.4: Execution accuracy of the two schema configurations on the fifteen structured-data questions.

Schema configuration	Correct Answers	Execution Accuracy
Bare schema	11 / 15	73.3%
Enriched (keys and sample rows)	14 / 15	93.3%

Enriching the schema with keys and sample rows increased the number of correctly answered questions from 11 out of 15 to 14 out of 15. Two of the three questions corrected by the enriched schema concerned a specific store identified by name. With the bare schema, the model generated queries that filtered on the column containing the store code rather than on the column containing the store name, resulting in empty query results. When sample rows were included in the schema, the model could observe that the store name was stored in the description column and generated the appropriate filter. This provides a concrete example of how exposing information about the actual database contents, rather than only its structure, can affect the SQL query generated by the model and, consequently, the final answer.

Across the two paths, the evaluations produced results of different magnitudes. In the structured-data path, schema enrichment had a pronounced effect, increasing execution accuracy from 73.3% to 93.3% and correcting three of the four previously incorrect questions. In the document retrieval path, the effects of query rewriting and reranking were more mixed: each configuration improved some metrics while reducing others, and no configuration consistently performed best across all four metrics. The clearest practical effect was the reduction in answerable questions that were incorrectly refused, particularly when reranking was used.

Chapter 5

Conclusions and Future Work

This dissertation investigated how a RAG-based system can provide conversational access to heterogeneous enterprise data, and how different architectural choices affect the quality of the resulting answers. To address this problem, a system was designed and implemented to answer natural language questions over two complementary enterprise data sources through a single conversational interface: unstructured documents held in SharePoint, addressed through retrieval-augmented generation, and structured data stored in a SQL Server database, addressed through schema-aware text-to-SQL. The system was evaluated using real Portuguese-language enterprise data provided by the partner company.

Regarding which architectural choices help, the evaluation produced results of different strength for the two paths. For the document retrieval subsystem, the two augmentations examined, query rewriting and reranking, produced small and uneven changes relative to the baseline. Each configuration improved some of the RAGAS metrics while reducing others, with no single configuration consistently performing best across all four metrics. The clearest practical benefit was not observed in the aggregate metrics, but in the reduction of cases where the system wrongly refused an answerable question, particularly when reranking was used. For the structured data subsystem, by contrast, the effect of the design choice was more pronounced: enriching the schema context supplied to the model with primary and foreign keys and a few sample rows increased execution accuracy from eleven to fourteen of the fifteen evaluation questions.

The contributions of this dissertation can be summarized in three main points. First, it presents the design and implementation of a conversational system that integrates unstructured SharePoint documents and structured SQL Server data within a single interface. Second, it provides a schema-aware text-to-SQL component and evaluates the effect of enriching schema information with keys and sample rows. Third, it provides a controlled experimental comparison of document retrieval configurations, examining the effects of query rewriting and reranking on a real Portuguese-language enterprise corpus.

These conclusions are qualified by several limitations. The main limitation concerns the size of the evaluation datasets. The document retrieval evaluation comprised 24 questions, while the structured data evaluation comprised 15 questions. Although these questions were designed to cover different types of queries over the available enterprise data, the relatively small sample sizes limit the extent to which the findings can be generalized beyond this corpus and these query types. The evaluation also relies on automated assessment through RAGAS, whose scores are produced by a language model acting as judge. This introduces potential

variability in the evaluation and creates a limitation for questions where the correct behavior is to refuse an answer, since the metrics do not adequately capture the quality of a correct refusal.

A few directions remain open for future work. The most immediate directions concern the transition from a single-user prototype to a system deployed within an enterprise. Such a deployment would have to accommodate a far larger and continually growing body of documents and data, and to support multiple users simultaneously, raising questions of multi-user and multi-tenant operation that are beyond the present scope. The structured data path would need to scale, as the schema examined was small enough to be present to the model in full, but a production database with more tables could make this approach impractical. An approach such as schema linking would be necessary, where for each question, the tables and columns relevant to it are selected.

The document retrieval could also benefit from some adjustments, as one further test could be to make the retrieval hybrid, combining the dense semantic search used in the current system with lexical methods such as BM25. This could be promising in enterprise settings, where questions often turn on acronyms or other rare exact terms that lexical matching handles well and dense retrieval does not. The query-rewriting step, implemented here as a single zero-shot reformulation, could be extended to the richer strategies surveyed in Section 2.3.5, such as the generation of a hypothetical answer to retrieve against (HyDE) or issuing several reformulations and fusing their results (RAG-Fusion). These approaches could improve retrieval, although they would require additional model calls.

A further direction would be to give users greater control over the generated answers. A configurable instruction mechanism could allow users to provide their own instructions, which would be incorporated into the generation prompt alongside the retrieved context. This would extend the system's configurability beyond the current architectural parameters and give non-technical users greater control over how answers are presented.

The configuration interface could also be extended. Currently, it only allows the user to connect the system to a SQL Server database. A natural extension would be to provide equivalent configuration for the document source, allowing users to connect their own SharePoint environment directly through the interface.

Overall, the results show that the value of architectural choices depends on the type of enterprise data being addressed. While schema enrichment produced a substantial improvement in the structured-data path, retrieval enhancements produced more mixed effects on document question answering. These findings highlight the importance of evaluating RAG architectures on the data and tasks for which they are intended, rather than assuming that techniques that perform well on benchmark datasets will produce equivalent benefits in an enterprise setting.

Appendix A

Appendix

A.1 Document Question and Answering Evaluation Set

As described in Section 4.2, twenty-four Portuguese questions were used to evaluate the document retrieval subsystem, each one paired with a ground-truth answer and listed in the table below. These questions were drawn from the SharePoint corpus presented in Section 3.3 and written to evaluate it across the range of its content, covering both the technical documentation and the meeting records of two enterprise projects. The ground-truth answers were established manually from the source documents and serve as a reference against which the generated answers are assessed with the RAGAS framework. The client names have been changed to Client A and Client B.

Two questions, 23 and 24 are negative tests, asking about information that is deliberately absent from the corpus, so that the correct response is a refusal rather than an answer. Their ground-truth is the system's fixed refusal phrase, and they are included to verify that the system declines to answer when the corpus does not support a response, instead of fabricating one.

Table A.1: Document question-answering evaluation set (24 questions with ground-truth answers).

Nº	**Pergunta**	**Resposta de referência**
1	Quantas tabelas de dimensão existem no modelo semântico do Cliente A especificado na v5?	13 tabelas de dimensão (Tempo, Cliente, Geografia, Mercado, Programa, Tipo Capa, Colaborador, Gestor, Vendedor, SV, Status, Tipologia, Tipo Negócio) mais Departamento, Plano de Contas, Empresa e Linha Mapa Financeiro.
2	Quais são as tabelas de factos definidas no modelo semântico v5 do Cliente A?	4 tabelas de factos: gold_Mov_Contabilidade, gold_Orcamento, gold_Mov_Contabilidade_Rubricas e gold_Orcamento_Rubricas.

Nº	Pergunta	Resposta de referência
3	Qual é o schema e a source database de onde vêm os dados do Gold Layer no modelo semântico do Cliente A?	Schema 'gold', base de dados PC-HOUSE_DEV (Gold Layer).
4	Que tipo de relações existem entre as tabelas de factos e dimensões no modelo semântico do Cliente A e qual a direção do filtro?	Relações Many-to-One (M:1) com Single filter direction (filtro unidirecional das dimensões para os factos).
5	Quais são os passos principais para integrar dados do Business Central com o Microsoft Fabric?	1) Criar Mirrored Database no Fabric, 2) Instalar extensão bc2adls no BC, 3) Escolher ambiente BC, 4) Pesquisar 'Export to Azure Data Lake Storage', 5) Preencher credenciais e escolher tabelas, 6) Obter link da Landing Zone, 7) Exportar Schema, 8) Exportar dados.
6	Porque é que a primeira extração de dados do Business Central para o Fabric deve ser feita fora do horário de trabalho?	A primeira extração tem impacto significativo no desempenho do Business Central, podendo afetar a utilização normal do sistema.
7	Quais são os 4 mapas financeiros (Account Schedules) mencionados na reunião de migração de reporting do Cliente A?	MENREP (Mapa de Exploração/Mensal de Reporting), OPEX (Despesas Operacionais), DR (Demonstração de Resultados) e AID.
8	Quantas empresas existem no grupo do Cliente A que precisam de ser incluídas no modelo de dados?	19 empresas. Inicialmente só a entidade global do Cliente A estava a ser extraída.
9	Qual foi o problema identificado com o pipeline de CI/CD no Fabric durante o projeto do Cliente A?	Bug no CI/CD que exigia recriação do workspace, e problemas nas regras de deployment de parâmetros (apontamento DB de Dev para Prod).
10	Qual é a hierarquia departamental utilizada no projeto do Cliente A e como é gerida ao longo do tempo?	Departamento > Sub-Divisão > Divisão, gerida com SCD (Slowly Changing Dimension) por ano.
11	Que problema foi identificado com linhas fantasma com sufixo 'W' nos dados do Cliente A?	Linhas com sufixo 'W' que apareciam ao filtrar por dimensões, causando desaparecimento de dimensões e falhas na filtragem contextual.
12	Como é que os utilizadores do Cliente A se podem ligar ao Modelo Semântico a partir do Excel?	Via Excel: Get Data, depois From Power Platform seguido de From Power BI. Também via Analysis Services ou OneLake Catalog.

Nº	Pergunta	Resposta de referência
13	Que capacidade de Fabric (SKU) está a ser utilizada no projeto do Cliente A e que limitações foram encontradas?	Capacidade F4. Limitações de desempenho durante refreshes e acessos concorrentes.
14	Como era feita a inversão de sinal nos valores das rubricas financeiras do Cliente A?	Multiplicação por -1 em certas rubricas, porque no BC receitas são registadas com sinal negativo e no reporting devem ser positivas.
15	Qual é a arquitetura proposta pela B2F para o projeto do Cliente B?	BigQuery com Mirroring (CDC) gravado no OneLake transformado em Semantic Model (Direct Lake) visualizado no Power BI.
16	Qual é o requisito mínimo de licenciamento Fabric para incluir Power BI Pro sem custo adicional no projeto do Cliente B?	F64 ou superior. Abaixo de F64, são necessárias licenças Power BI Pro separadas.
17	Quantas tabelas foram inicialmente propostas no scope do projeto do Cliente B e quantas foram realmente encontradas?	6-7 tabelas no scope inicial. Encontradas ∼45 tabelas DWH + ∼60 data marts (100+ no total).
18	Porque é que o mirroring do BigQuery para o Fabric falhou no projeto do Cliente B e qual foi a solução?	Views do BigQuery não são suportadas pelo mirroring. Solução foi copy data flow com Warehouse.
19	Qual é a estratégia de extração incremental definida para o projeto do Cliente B?	Tabela de parametrização baseada em datas, com campo de data configurado por tabela para extrair apenas dados novos/alterados.
20	Qual é a limitação do Direct Lake que afetou o desenho do modelo semântico no projeto do Cliente B?	Direct Lake não suporta views. Transformações têm de ser materializadas em tabelas físicas.
21	Qual foi a estratégia de licenciamento de trial discutida no kickoff do projeto do Cliente B?	Uso de security groups para gerir o trial de licenças Fabric e controlar o acesso à capacidade trial.
22	Como é que o product_number é utilizado no modelo de dados do BigQuery do Cliente B?	product_number funciona como natural key (chave natural) para identificar produtos nas relações entre tabelas.
23	Que modelo de machine learning é utilizado para previsão de receitas no projeto do Cliente B?	Esta informação não consta nos documentos disponíveis. (teste negativo)
24	Qual é a política de retenção de dados definida para o Lakehouse do Cliente A?	Esta informação não consta nos documentos disponíveis. (teste negativo)

A.2 Structured Data (Text-to-SQL) Evaluation Set and Results

The structured data subsystem was evaluated on fifteen Portuguese questions, over the data on the retail SQL Server database presented in Section 3.3. Each question was run under the two schema context configurations compared in the evaluation, the minimum schema configuration, in which the model is only given the table and column names, and the enriched configuration, which additionally supplies the primary and foreign keys and also a few sample rows. Figure A.1 and Figure A.2 report, for each question, the expected value (Actual Response), the answer produced by the system (RAG Response) and whether if the answer was correct.

Nº	Question	Actual Response	RAG Response	Correct Answer
1	Qual o número de linhas da tabela FACT_VENDAS	1.388.785	1.388.785	Yes
2	Qual foi o valor total de vendas para a Loja Porto, para o ano de 2016	1984406.57	1.984.406,57	Yes
3	Total de Quantidade Vendida do Produto 6UN LEITE UHT GRESSO M/G LT	1012556	1012556	Yes
4	Produto mais vendido por quantidade no ano de 2016	6UN LEITE UHT GRESSO M/G LT	6UN LEITE UHT GRESSO M/G LT	Yes
5	Qual o número de clientes únicos na loja 'Porto'	2595	0	No
6	Qual o total de vendas processadas pelo operador Matilde Barros Pinto	9089.11	9089.11	Yes
7	Qual o número de Lojas	5	5	Yes
8	Qual o nº de operadores	58	58	Yes
9	Qual o produto mais vendido?	6UN LEITE UHT GRESSO M/G LT	6UN LEITE UHT GRESSO M/G LT	Yes
10	Qual o produto mais vendido na loja Porto?	6UN LEITE UHT GRESSO M/G LT	Não tem informação suficiente	No
11	Qual o valor total de descontos aplicados no ano de 2016?	1025300.00	1025300.00	Yes
12	Qual a loja com maior valor total de vendas?	Valongo (10248254.50)	Valongo	Yes
13	Qual o distrito com mais clientes?	Porto (4472)	Porto	Yes
14	Qual o valor médio por ticket na loja Porto?	33.64	Não conseguiu calcular	No
15	Quantos produtos estão em rutura de stock?	50	Não há produtos em rotura	No

Figure A.1: Minimum Schema Text-to-SQL Results.

Nº	Question	Actual Response	RAG Response	Correct Answer
1	Qual o número de linhas da tabela FACT_VENDAS	1.388.785	1.388.785	Yes
2	Qual foi o valor total de vendas para a Loja Porto, para o ano de 2016	1984406.57	1.984.406,57	Yes
3	Total de Quantidade Vendida do Produto 6UN LEITE UHT GRESSO M/G LT	1012556	1012556	Yes
4	Produto mais vendido por quantidade no ano de 2016	6UN LEITE UHT GRESSO M/G LT	6UN LEITE UHT GRESSO M/G LT	Yes
5	Qual o número de clientes únicos na loja 'Porto'	2595	2595	Yes
6	Qual o total de vendas processadas pelo operador Matilde Barros Pinto	9089.11	9089.11	Yes
7	Qual o número de Lojas	5	5	Yes
8	Qual o nº de operadores	58	58	Yes
9	Qual o produto mais vendido?	6UN LEITE UHT GRESSO M/G LT	6UN LEITE UHT GRESSO M/G LT	Yes
10	Qual o produto mais vendido na loja Porto?	6UN LEITE UHT GRESSO M/G LT	6UN LEITE UHT GRESSO M/G LT	Yes
11	Qual o valor total de descontos aplicados no ano de 2016?	1025300.00	1025300.00	Yes
12	Qual a loja com maior valor total de vendas?	Valongo (10248254.50)	Valongo	Yes
13	Qual o distrito com mais clientes?	Porto (4472)	Porto	Yes
14	Qual o valor médio por ticket na loja Porto?	33.64	Não conseguiu calcular	No
15	Quantos produtos estão em rutura de stock?	50	50	Yes

Figure A.2: Enriched Schema Text-to-SQL Results.

A.3 Prompts

The prompts used by the system are reproduced below exactly as they are sent to the language model. The variable parts filled in at run time are shown in braces, for example context, question and schema. The document generation, query rewriting and SQL-answer prompts are written in European Portuguese, so the model returns an answer in that language by default, while the text-to-SQL generation prompt, whose output is SQL is written in English.

A.3.1 Document Generation Prompt

Used by the document retrieval subsystem to generate the answer from the retrieved passages

```
És um assistente que responde a perguntas usando apenas os trechos de contexto fornecidos.

Regras:
- Responda em português (PT-PT), mesmo que a pergunta esteja noutra língua.
- Cite o trecho no formato [1], [2], etc. Não cites apenas no final.
- Se nenhuma informação relevante estiver nos trechos, responda exatamente:
  "Esta informação não consta nos documentos disponíveis."

Trechos de contexto:
{context}

Pergunta:
{question}
```

A.3.2 Query Rewriting Prompt

System prompt of the zero-shot query rewriter.

```
És um reescritor de consultas para um sistema de recuperação de documentos.
A tua tarefa é reescrever a pergunta do utilizador como uma única consulta de pesquisa clara e
↪ autocontida para uma base de conhecimento.
Instruções:
- Torna explícita a intenção implícita e desambigua formulações vagas.
- Expande abreviações e acrescenta sinónimos ou terminologia provável quando útil.
- NÃO respondas à pergunta.
- NÃO introduzas factos, nomes ou entidades que não estejam implícitos na pergunta original.
- Mantém a consulta em português (o mesmo idioma da pergunta).
- Devolve APENAS a consulta reescrita, sem explicação e sem aspas.
```

A.3.3 Text-to-SQL Generation Prompt

Used to translate the question into a single read-only SQL query. The system message sets the rules, and the user message supplies the schema context and the question

```
System:
You generate exactly one Microsoft SQL Server query. Rules: SELECT or WITH ... SELECT
↪ only; no DDL/DML; no comments; no semicolon unless required; output raw SQL only
↪ (no markdown).

User:
Allowed schema:
{schema}

User question: {question}
```

A.3.4 SQL-Answer Prompt

Used in the second step of the structured data path to turn the rows returned by the query into a natural language answer.

```
System:
You answer using only the rows provided in the user message. Always reply in natural language,
↪ never as JSON or code.

User:
És um assistente que responde a perguntas sobre dados de negócio.

Regras:
- Responda em português (PT-PT), mesmo que a pergunta esteja noutra língua.
- Escreva a resposta em frases completas. NUNCA devolva JSON, blocos de código, tabelas em
↪ markdown, nem os nomes técnicos das colunas.
- Usa APENAS os valores presentes nas linhas devolvidas. Não estimes, não arredondes para
↪ valores aproximados e não completes informação em falta.
- Apresente os números de forma legível, com separador de milhares e no máximo duas casas
↪ decimais, indicando sempre a que se referem.
- Se as linhas não tiverem informação suficiente para responder, responde exatamente:
  "Os dados disponíveis não permitem responder a esta pergunta."

Consulta SQL executada:
{sql}

Linhas devolvidas (JSON):
{rows}

Pergunta:
{question}

Resposta:
```

A.4 User Interface

The system is operated through a web interface, implemented as a thin client over the backend described in Chapter 3. Figure A.3 shows the conversational interface, where a user can ask a question in natural language and receive an answer together with the citations that support it. Figure A.4 shows the configuration page, where it is possible to connect the system to one of the organization's data sources, the SQL Server, by supplying the corresponding credentials.

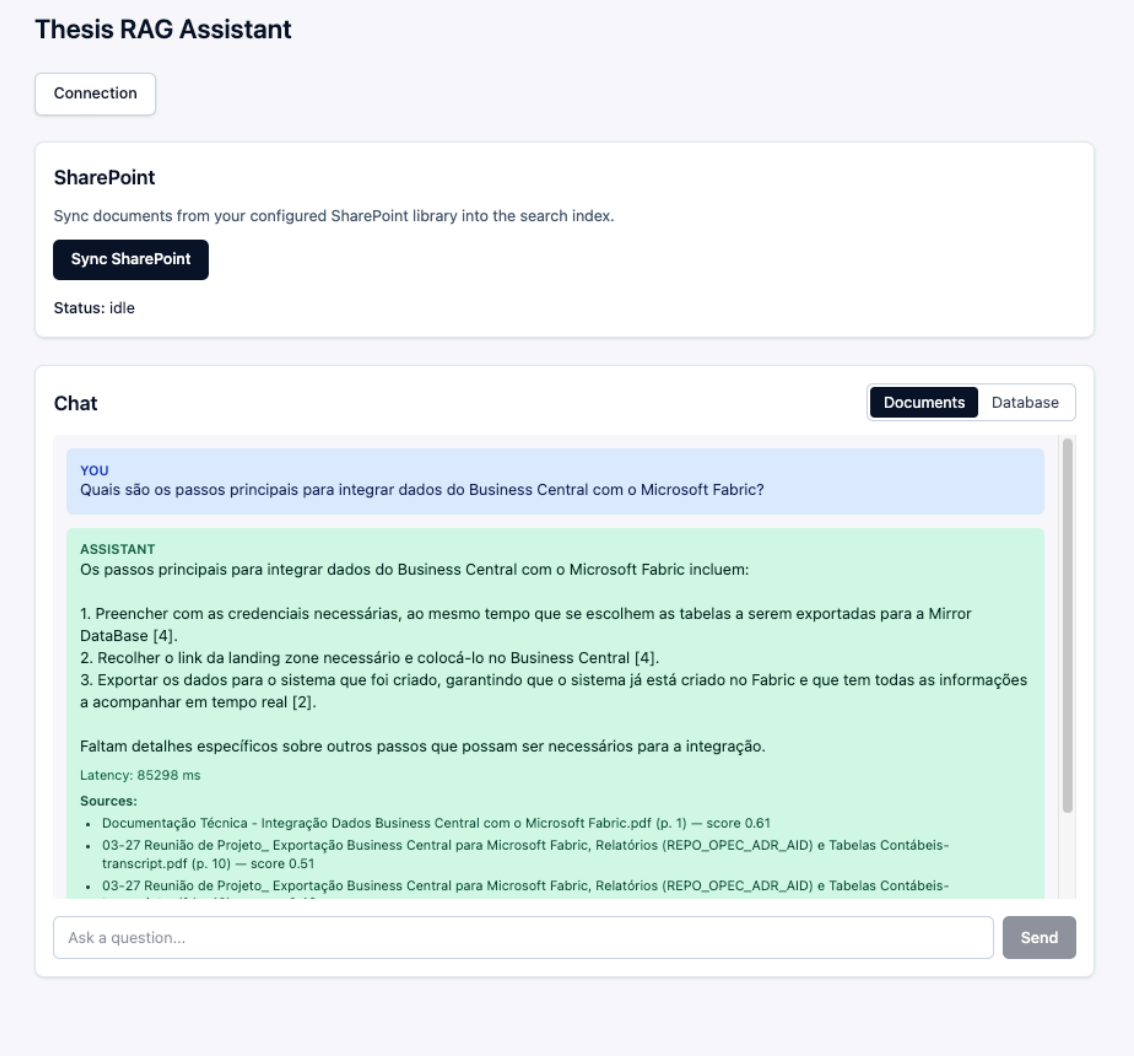


Figure A.3: The conversational interface, showing a question, its answer and the supporting citations.

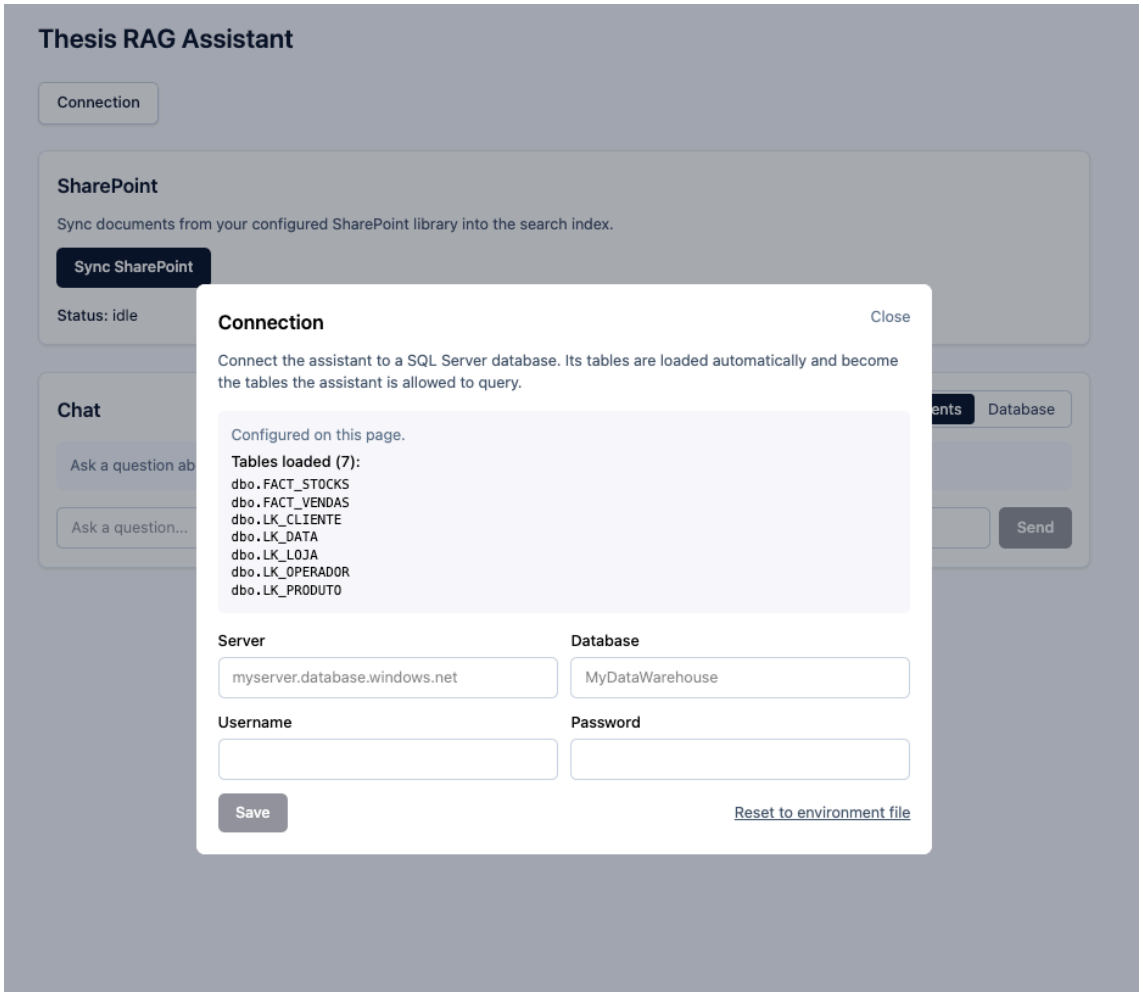


Figure A.4: The configuration interface, for the SQL Server database

Bibliography

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., … Liang, P. (2021). On the opportunities and risks of foundation models. *ArXiv*, *abs/2108.07258*. Retrieved from https://api.semanticscholar.org/CorpusID:237091588

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., … Amodei, D. (2020). Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (Vol. 33, pp. 1877–1901). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

Chen, J., Lin, H., Han, X., & Sun, L. (2024). Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 38, pp. 17754–17762).

Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D., & Liu, Z. (2024). Bge m3-embedding: Multilingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*.

Chroma. (2025). *Configure collections.* https://docs.trychroma.com/docs/collections/configure.

Cormack, G. V., Clarke, C. L., & Buettcher, S. (2009). Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international acm sigir conference on research and development in information retrieval* (pp. 758–759).

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)* (pp. 4171–4186).

Es, S., James, J., Espinosa-Anke, L., & Schockaert, S. (2024). RAGAs: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th conference of the european chapter of the association for computational linguistics: system demonstrations* (pp. 150–158).

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

Gao, L., Ma, X., Lin, J., & Callan, J. (2023). Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 1762–1777).

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., … Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Günther, M., Mohr, I., Williams, D. J., Wang, B., & Xiao, H. (2024). Late chunking: Contextual chunk embeddings using long-context embedding models. *arXiv preprint arXiv:2409.04701*.

Hong, Z., Yuan, Z., Zhang, Q., Chen, H., Dong, J., Huang, F., & Huang, X. (2025). Next-generation database interfaces: A survey of llm-based text-to-sql. *IEEE Transactions on Knowledge and Data Engineering*.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., … Chen, W. (2021). LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Jagerman, R., Zhuang, H., Qin, Z., Wang, X., & Bendersky, M. (2023). Query expansion by prompting large language models. *arXiv preprint arXiv:2305.03653*.

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., … Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (emnlp)* (pp. 6769–6781).

Kukreja, S., Kumar, T., Bharate, V., Purohit, A., Dasgupta, A., & Guha, D. (2023). Vector databases and vector embeddings-review. In *2023 international workshop on artificial intelligence and image processing (iwaiip)* (pp. 231–236).

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., … Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, *33*, 9459–9474.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2024). Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, *12*, 157–173.

Ma, X., Gong, Y., He, P., Zhao, H., & Duan, N. (2023). Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing* (pp. 5303–5315).

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Nogueira, R., & Cho, K. (2019). Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*.

OpenAI. (2024a). *Embeddings*. https://platform.openai.com/docs/guides/embeddings.

OpenAI. (2024b). *GPT-4o mini model*. https://platform.openai.com/docs/models/gpt-4o-mini.

OpenAI. (2024c). *New embedding models and API updates*. https://openai.com/index/new-embedding-models-and-api-updates/.

Ovadia, O., Brief, M., Mishaeli, M., & Elisha, O. (2024). Fine-tuning or retrieval? comparing knowledge injection in LLMs. In *Proceedings of the 2024 conference on empirical methods in natural language processing* (pp. 237–250).

Pourreza, M., & Rafiei, D. (2023). Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in neural information processing systems*, *36*.

Rackauckas, Z. (2024). Rag-fusion: a new take on retrieval-augmented generation. *arXiv preprint arXiv:2402.03367*.

Ragas. (2025). *List of available metrics*. https://docs.ragas.io/en/latest/concepts/metrics/available_metrics/.

Rajkumar, N., Li, R., & Bahdanau, D. (2022). Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498*.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)* (pp. 3982–3992).

Robertson, S., & Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and

beyond. *Foundations and trends® in information retrieval*, *3*(4), 333–389.

Saad-Falcon, J., Khattab, O., Potts, C., & Zaharia, M. (2024). Ares: An automated evaluation framework for retrieval-augmented generation systems. In *Proceedings of the 2024 conference of the north american chapter of the association for computational linguistics: Human language technologies (volume 1: Long papers)* (pp. 338–354).

Sawarkar, K., Mangal, A., & Solanki, S. R. (2024). Blended rag: Improving rag (retriever-augmented generation) accuracy with semantic search and hybrid query-based retrievers. In *2024 ieee 7th international conference on multimedia information processing and retrieval (mipr)* (pp. 155–161).

Soudani, H., Kanoulas, E., & Hasibi, F. (2024). Fine tuning vs. retrieval augmented generation for less popular knowledge. In *Proceedings of the 2024 annual international acm sigir conference on research and development in information retrieval in the asia pacific region* (pp. 12–22).

Sun, W., Yan, L., Ma, X., Wang, S., Ren, P., Chen, Z., … Ren, Z. (2023). Is chatgpt good at search? investigating large language models as re-ranking agents. In *Proceedings of the 2023 conference on empirical methods in natural language processing* (pp. 14918–14937).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Wang, L., Yang, N., & Wei, F. (2023). Query2doc: Query expansion with large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing* (pp. 9414–9423).

Wang, X., Wang, Z., Gao, X., Zhang, F., Wu, Y., Xu, Z., … Huang, X. (2024). Searching for best practices in retrieval-augmented generation. In *Proceedings of the 2024 conference on empirical methods in natural language processing* (pp. 17716–17736).

Zhao, W. X., Zhou, K., Li, J., Tang, T., Dong, Z., Hou, Y., … Wen, J.-R. (2026). A survey of large language models. *Frontiers of Computer Science*, *20*(12), 2012627.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., … Stoica, I. (2023). Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th international conference on neural information processing systems*. Curran Associates Inc.