ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Uso de Retrieval-Augmented Generation combinada com LLMs para Auxílio em Atividades de Suporte Técnico

Rui Pedro Teles Ribeiro

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Engenharia de Software

Orientador: Piedade Carvalho
Supervisor: José Soares

Porto, 9 de junho de 2025

Declaração de Integridade

Declaro ter conduzido este trabalho académico com integridade.

Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Portanto, o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

ISEP, Porto, 9 de junho de 2025

Dedicatória

The dedicatory is optional. Below is an example of a humorous dedication.

"To my wife Marganit and my children Ella Rose and Daniel Adam without whom this book would have been completed two years earlier."in "An Introduction To Algebraic Topology"by Joseph J. Rotman.

Resumo

This document explains the main formatting rules to apply to a TMDEI Master Dissertation work for the MSc in Computer Engineering of the Computer Engineering Department (DEI) of the School of Engineering (ISEP) of the Polytechnic of Porto (IPP).

The rules here presented are a set of recommended good practices for formatting the disseration work. Please note that this document does not have definite hard rules, and the discussion of these and other aspects of the development of the work should be discussed with the respective supervisor(s).

This document is based on a previous document prepared by Dr. Fátima Rodrigues (DEI/ISEP).

The abstract should usually not exceed 200 words, or one page. When the work is written in Portuguese, it should have an abstract in English.

Please define up to 6 keywords that better describe your work, in the *THESIS INFORMATION* block of the `main.tex` file.

Palavras-chave: Keyword1, ..., Keyword6

Abstract

Trabalhos escritos em língua Inglesa devem incluir um resumo alargado com cerca de 1000 palavras, ou duas páginas.

Se o trabalho estiver escrito em Português, este resumo deveria ser em língua Inglesa, com cerca de 200 palavras, ou uma página.

Para alterar a língua basta ir às configurações do documento no ficheiro `main.tex` e alterar para a língua desejada ('english' ou 'portuguese')[1]. Isto fará com que os cabeçalhos incluídos no template sejam traduzidos para a respetiva língua.

[1] Alterar a língua requer apagar alguns ficheiros temporários; O target **clean** do **Makefile** incluído pode ser utilizado para este propósito.

Agradecimentos

The optional Acknowledgment goes here... Below is an example of a humorous acknowledgment.

"I'd also like to thank the Van Allen belts for protecting us from the harmful solar wind, and the earth for being just the right distance from the sun for being conducive to life, and for the ability for water atoms to clump so efficiently, for pretty much the same reason. Finally, I'd like to thank every single one of my forebears for surviving long enough in this hostile world to procreate. Without any one of you, this book would not have been possible."in "The Woman Who Died a Lot"by Jasper Fforde.

Conteúdo

Lista de Figuras **xv**

Lista de Tabelas **xvii**

Lista de Algorítmos **xvii**

Lista de Código **xvii**

Lista de Símbolos **xix**

1 Introduction **1**
1.1 Context . . . 1
1.2 Problem . . . 1
1.3 Objectives . . . 1
1.4 Metodologia . . . 2
1.5 Planeamento . . . 3
1.6 Considerações Éticas . . . 3
1.7 Estrutura do Documento . . . 4

2 Fundamentos Teóricos **5**
2.1 Inteligência Artificial . . . 5
2.2 Machine Learning & Deep Learning . . . 5
2.3 Large Language Models . . . 6
2.4 Fine-tuning LLMs . . . 7
2.5 RAG . . . 7
2.5.1 Arquitetura . . . 7
2.6 Aplicação ao Suporte Técnico . . . 8

3 State of the Art **9**
3.1 Metodologia . . . 9
3.1.1 Questões de Investigação . . . 9
3.1.2 Research Scope . . . 9
3.1.3 Critérios de Inclusão e Exclusão . . . 9
3.1.4 Processo de Seleção . . . 9
3.1.5 Coleção de Dados . . . 9
3.2 Melhores Práticas RAG associadas a Suporte Técnico (RQ1) . . . 9
3.2.1 Indexação . . . 9
3.2.2 Armazenamento . . . 10
3.2.3 Recuperação e Geração de Respostas . . . 10
3.2.4 Augmentation . . . 12
Prompt Engineering . . . 12

3.2.5 Aplicações Práticas 12
3.3 Desafios Técnicos na Implementação (RQ2) 12
3.3.1 Indexação, recuperação e relevância das respostas 12
3.3.2 Integração com Sistemas Existentes 13
3.4 Frameworks para Desenvolvimento de RAG (RQ3) 13
3.4.1 LangChain 13
3.4.2 Prompt Templates 14
3.4.3 Haystack 15
3.4.4 LlamaIndex 16
3.4.5 Spring AI 16
3.4.6 Comparação resumo entre tecnologias 18
3.5 Trabalhos Relacionados 20
3.5.1 Análise Individual 20
RAG Chatbot para a OptiMicro Technologies 20
Sistema RAG de Recomendação para Suporte Técnico 21
Fabula 22
3.5.2 Conclusões 23

Bibliografia **25**

Apêndice A Appendix Title Here **27**

Lista de Figuras

1.1 Metodologia DSR (**peffers2007design**) 2

2.1 Arquitetura simplificada do RAG - https://medium.com/@sahin.samia/what-is-retrieval-augmented-generation-rag-in-llm-and-how-it-works-a8c79e35a172 8

3.1 Processo de Indexação (Adaptado de Špeletić et al. 2024) 10
3.2 Processo de Recuperação e Geração exclusivamente com Embedding Model (Adaptado de Špeletić et al. 2024) 11
3.3 Processo de Recuperação e Geração com Recuperador Lexical e Embedding Model 11
3.4 Arquitetura do chatbot RAG da OptiMicro (Lee et al. 2024) 21
3.5 Arquitetura do sistema de recomendação de resolução de incidentes proposto por Isaza et al. 2024b 22

Lista de Tabelas

3.1 Comparação entre tecnologias RAG 19

Lista de Símbolos

a	distance	m
P	power	W ($\mathrm{J\,s^{-1}}$)
ω	angular frequency	rad

Capítulo 1

Introduction

1.1 Context

1.2 Problem

A Natixis é uma empresa do setor financeiro, parte do grupo bancário francês BPCE (Banque Populaire, Caisse d'Epargne). Ela atua principalmente em banca de investimentos, gestão de ativos, seguros e serviços financeiros especializados [1].

Atualmente a equipa "B2C"da Natixis efetua tarefas diárias de suporte técnico no sistema pelo qual é responsável. Este é um sistema maioritariamente responsável pelo cálculo de risco de crédito bancário. O sistema respeita um fluxo bem definido sendo que diariamente correm diversos processos. Primeiramente vem a fase de alimentação, onde o sistema injeta dados provenientes de sistemas externos e popula as tabelas brutas da base de dados. De seguida vem o processo de enriquecimento dos dados, onde os mesmos são analisados em termos de qualidade, sendo aplicadas regras de negócio para os alterar e armazenar. Todo esse processo é auditado em ficheiros e/ou tabelas específicas na base de dados. Por fim vem o processo de cálculo, que pode ser de vários tipos, como por exemplo RC B3 (Cálculo do Capital Ponderado pelo Risco - B3) que nos sistemas bancários se refere, normalmente, aos requisitos de capital regulamentares definidos pelo acordo de Basel III [2]. Todo este processo é controlado diariamente pelo Control-M. O Control-M é uma ferramenta de automação de workload e gestão de jobs [3]. Ele é amplamente utilizado para agendar, monitorizar e gerir processos batch. A equipa de suporte é responsável por gerir esta chain, sendo que cada processo (alimentação, enriquecimento e cálculo) é composto por diversos batchs que executam maioritariamente código Perl e Java.

Falhas na chain são comuns de acontecer e podem ter diversos motivos, como erros nos dados externos que violem as regras de negócio, problemas de código provenientes de desenvolvimentos recentes, entre outros. A equipa é também responsável por lançar processos "on demand", facilitar o esclarecimento de questões relacionadas com regras de negócio aos utilizadores/partes interessadas, entre outras atividades. Devido à dimensão do software e à quantidade de diferentes processos envolvidos, por vezes a atividade de suporte torna-se uma tarefa bastante complicada para a equipa responsável.

1.3 Objectives

Explorar Retrieval-augmented generation (RAG) juntamente com Large Language Model (LLM) para construir um sistema que forneça auxílio à tomada de decisão nas atividades

de suporte, pela geração de informação contextualizada. Retrieval-Augmented Generation (RAG) combina a geração de texto por Large Language Models (LLMs) com recuperação de conhecimento externo, permitindo respostas mais informadas e específicas [4]. Os LLMs, como GPT, Llama e Mistral, são modelos que possuem um vasto conhecimento, mas não conseguem aceder a informações de domínios específicos. O RAG resolve essa limitação ao combinar uma base de dados vetorial para armazenar representações semânticas de documentos e consultas, permitindo que o modelo recupere a informação mais relevante antes de gerar uma resposta. Essas bases de dados vetoriais utilizam embeddings, que representam o significado semântico do texto em um espaço multidimensional, possibilitando pesquisas mais eficientes e contextuais [4,5]. A aplicação do RAG neste contexto permitirá otimizar a eficiência da equipa de suporte, reduzindo o tempo gasto na procura de informação e facilitando a resolução de incidentes.

Funcionalidades a explorar:

- Facilitar e agilizar a obtenção de informação relevante para resolução de determinado problema. O software deverá fornecer insights de passos a tomar com base no conhecimento que possui. - Automatizar a extração de conhecimento de fontes diversas, como manuais e logs de execução, integrando-se com as ferramentas utilizadas pela equipa (p.e. Control-M, Confluence, Outlook e Teams). - Auxiliar na resolução de problemas ao sugerir soluções com base em experiências anteriores e na análise de padrões de erro.

O projeto irá obedecer à seguinte ordem cronológica:

- Primeira fase: Análise de KPIs históricos na vertente de resolução de incidentes. - Segunda fase: Pesquisa / análise de trabalhos relacionados e boas práticas. - Terceira fase: Levantamento de requisitos. - Quarta fase: Implementação da solução. - Quinta fase: Testes.

1.4 Metodologia

A metodologia Design Science Research (DSR) (Figura 1.1) foi escolhida como quadro orientador para o desenvolvimento deste trabalho. Esta metodologia é particularmente adequada para investigações que visam resolver problemas do mundo real através da criação e avaliação de artefactos inovadores (**peffers2007design**).

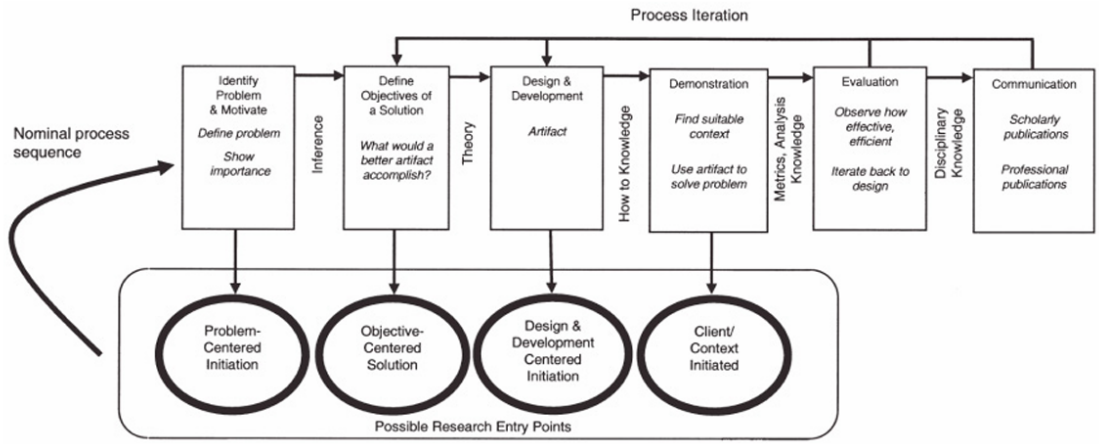

Figura 1.1: Metodologia DSR (**peffers2007design**)

A metodologia DSR é composta por seis etapas, que se relacionam com o presente projeto da seguinte forma:

TODO retificar esta lista no fim do trabalho

- **Identificação e Motivação do Problema:** Foi realizada uma análise aprofundada dos desafios enfrentados pelas equipas de suporte técnico, nomeadamente a dificuldade em aceder rapidamente a informação precisa, a sobrecarga de pedidos repetitivos e a falta de padronização nas respostas.
- **Definição de Objetivos para a Solução:** Com base nos problemas identificados, foram definidos objetivos claros, como a criação de uma solução baseada em RAG que melhore a eficiência do suporte técnico, assegure consistência nas respostas e possibilite a rastreabilidade das fontes de informação utilizadas.
- **Desenho e Desenvolvimento:** Foi desenvolvido um protótipo funcional que integra uma arquitetura baseada em *Retrieval-Augmented Generation* com uma LLM, utilizando Spring AI. Este protótipo inclui também uma interface simples para interação e teste por parte dos utilizadores de suporte.
- **Demonstração:** A solução foi aplicada a cenários simulados e reais de suporte técnico na área financeira, permitindo validar a sua aplicabilidade em tarefas como a resposta a incidentes, esclarecimento de dúvidas técnicas e recuperação de documentação normativa.
- **Avaliação:** O sistema foi avaliado com base em testes de desempenho, testes unitários e de integração, bem como critérios de utilidade, precisão e conformidade com os requisitos regulatórios aplicáveis, incluindo os definidos pelo Banco Central Europeu.
- **Comunicação:** Os resultados desta investigação foram documentados nesta dissertação e visam contribuir tanto para o avanço do conhecimento académico na área da IA aplicada ao suporte, como para a melhoria contínua de processos em contexto empresarial.

1.5 Planeamento

1.6 Considerações Éticas

Esta dissertação respeita o código de ética do Instituto Politécnico do Porto (**codigo_2020**). Em conformidade com o Artigo 2.º, o trabalho observa princípios fundamentais como a legalidade, transparência, responsabilidade, confidencialidade, integridade e honestidade. Adicionalmente, a investigação cumpre as orientações previstas no Artigo 10.º, assegurando os mais elevados padrões de integridade científica, originalidade e verificabilidade.

O plágio e qualquer forma de má conduta académica são rigorosamente evitados, conforme estipulado no Artigo 6.º, nomeadamente na alínea 2.8, que exige a correta citação de todas as fontes. Quaisquer contributos de terceiros ou propriedade intelectual são devidamente creditados, em conformidade com o Artigo 10.º, alínea *e*, que reforça a importância da citação precisa e da distinção relativamente a trabalhos anteriores.

Dado o carácter experimental desta investigação, apenas são utilizados conjuntos de dados e ferramentas com licenciamento explícito que permita a sua utilização académica. Além

disso, e de acordo com o Artigo 10.º, alínea *h*, eventuais conflitos de interesse e apoios externos são divulgados de forma transparente.

O conteúdo desta dissertação foi desenvolvido com diligência e rigor, garantindo a reprodutibilidade e a comunicação transparente dos resultados, conforme exigido pelo Artigo 10.º, alínea *g*. Todos os métodos, fontes de dados e ferramentas utilizados nesta dissertação estão totalmente documentados e são publicamente acessíveis, contribuindo assim para uma comunicação científica aberta e transparente.

Por fim, todas as atividades foram realizadas com o devido respeito pelas obrigações éticas e legais no que respeita à privacidade de dados e à propriedade intelectual, conforme estipulado no Artigo 10.º, alínea 2a, e no Artigo 9.º, assegurando a segurança, bem-estar e os direitos de todos os participantes envolvidos.

1.7 Estrutura do Documento

Capítulo 2

Fundamentos Teóricos

2.1 Inteligência Artificial

A inteligência artificial (IA) é um ramo cientifico da computação que se dedica ao desenvolvimento de sistemas capazes de executar tarefas que normalmente exigiriam inteligência humana. Estes sistemas têm a capacidade de executar funções avançadas e analisar dados de grande escala a fim de gerar respostas precisas. Baseado num conceito do filosofo do grego Aristoteles, a IA surgiu na década de 1950 por Allan Turing, onde o mesmo escreveu sobre a possibilidade de uma máquina pensar e imitar o comportamento humano inteligente. Atualmente a IA é aplicada em diversos setores, como na saúde através do diagnostico automatizado de doenças, no setor financeiro para análises de mercado e deteção de fraudes, entre outros. Recentemente a IA sofreu um rápido avanço, com a corrida da IA generativa, sendo o seu componente-chave a fundação da OpenAI em 2015 e surgimento do ChatGPT em 2022, sistema este capaz de processar linguagem natural (NLP) e gerar respostas precisas e corretas sobre variados assuntos (https://hai.stanford.edu/news/ai-spring-four-takeaways-major-releases-foundation-models).

Dentro da IA existem diferentes sub-ramos cientificos, como:

- Machine Learning (ML): Ensina computadores a aprender padrões a partir de dados através de redes neronais ou arvores de decisão;
- Deep Learning (DL): Sub-ramo do ML que faz uso de redes neronais para modelar a intrepertar padrões complexos;
- Processamento de linguagem natural (NLP): Intrepertação de linguagem natural humana.
- Visão computacional: Intrepertação de imagens e vídeos

2.2 Machine Learning & Deep Learning

TODO rever secção (...)

O Machine Learning é um subcampo da inteligência artificial centrado no desenvolvimento de algoritmos capazes de identificar padrões em dados e realizar previsões ou decisões com base nesses padrões, sem que para isso sejam explicitamente programados. Esta abordagem baseia-se na experiência — os modelos são treinados com dados históricos e ajustam os seus parâmetros internos para generalizar para novos dados, muitas vezes em contextos altamente variáveis e complexos.

A aprendizagem automática pode ser agrupada em três principais paradigmas:

- **Aprendizagem supervisionada**, em que o modelo é treinado com exemplos rotulados (inputs associados a outputs desejados), aprendendo uma função que generaliza para novos dados;
- **Aprendizagem não supervisionada**, onde o objetivo é descobrir estruturas ou padrões ocultos em dados não rotulados, como agrupamentos ou relações estatísticas;
- **Aprendizagem por reforço**, na qual um agente interage com um ambiente, aprendendo uma política de acções com base num sistema de recompensas, com o objetivo de maximizar um retorno cumulativo.

Dentro do ML, destaca-se o Deep Learning, que se baseia no uso de redes neuronais artificiais profundas, compostas por múltiplas camadas de unidades de processamento. Estas redes têm a capacidade de modelar relações não lineares complexas, sendo particularmente eficazes em tarefas como a visão computacional, a tradução automática e o processamento de NLP.

Uma das principais vantagens da aprendizagem profunda reside na sua capacidade de extrair representações hierárquicas dos dados — as camadas iniciais aprendem características de baixo nível, enquanto as camadas superiores capturam abstrações mais elevadas, permitindo uma compreensão mais profunda do domínio em causa. Ao contrário dos métodos tradicionais, que requerem engenharia manual de atributos, o DL automatiza esse processo, o que se revela vantajoso em cenários com grandes volumes de dados.

Estas capacidades fizeram do Deep Learning a base das mais recentes inovações em inteligência artificial, nomeadamente os modelos de linguagem de grande escala (Large Language Models – LLMs), que se tornaram uma das áreas mais ativas e transformadoras da IA nos últimos anos.

2.3 Large Language Models

Os LLMs representam um avanço significativo na IA. Proposta pela Google em 2017, atualmente, Transformer é a arquitetura de DL mais explorada para esta componente. Os Transformers foram inicialmente desenvolvidos como melhoria das arquiteturas anteriores para a tradução automática, mas desde então têm encontrado muitas aplicações, como na visão computacional e NLP. Conduziram ao desenvolvimento de sistemas pré-treinados, tais como Generative Pre-trained Transformers (GPTs) and Bidirectional Encoder Representations from Transformers (BERT). Estes modelos são treinados através do paradigma Self-supervised learning (SSL), no qual aprendem representações úteis dos dados sem a necessidade de rótulos manuais. No SSL, o próprio modelo gera os seus rótulos a partir dos dados brutos, criando tarefas preditivas auxiliares chamadas pretext tasks. Masked Language Modeling é um exemplo de tarefa preditiva, utilizada pelo BERT, onde palavras altetórias são ocultadas em uma frase e o modelo aprende a prever as palavras corretas, isto no contexto de NLP. Em contraste o GPT faz uso do Casual Language Modeling onde o modelo prevê a próxima palavra numa sequência de texto, dado o contexto anterior.

TODO melhorar texto abaixo

O aumento da escala destes modelos — tanto em volume de dados como em parâmetros — contribuiu para ganhos significativos em capacidades linguísticas, raciocínio e compreensão contextual. Os LLMs modernos, como os da família GPT, constituem atualmente a espinha

dorsal de sistemas conversacionais, motores de busca inteligentes, e aplicações empresariais que requerem compreensão profunda da linguagem.

2.4 Fine-tuning LLMs

TODO e depois no RAG fazer uma comparação e mostrar vantagens do RAG

2.5 RAG

Apesar dos avanços dos LLMs, estes modelos enfrentam limitações em ambientes onde o acesso a conhecimento atualizado ao contexto é crucial. A técnica de fine-tuning abordada anteriormente é dispendiosa e não escala bem para conteúdos em constante mudança. Retrieval-Augmented Generation (RAG) surge como uma solução alternativa, permitindo enriquecer a geração com acesso em tempo real a fontes externas.

RAG é uma técnica que combina LLMs com um mecanismo de recuperação de informação externa. Enquanto que os LLMs apenas se baseiam em dados pré-treinados, o RAG recupera informação relevante de um contexto específico armazenado previamente.

Tendo em consideração que RAG é a peça fundamental de estudo do presente documento, nesta secção serão abordados todos os seus conceitos mais importantes.

2.5.1 Arquitetura

O fluxo típico do RAG encontra-se descrito na Figura 3.1. Em primeira instancia, o processo consiste na formulação de uma questão (*query*) a ser requisitada ao sistema. De seguida, os processos de Retrieval, Augmentation e Generation ocorrem de forma sequencial:

- **Retrieval**: Baseado na *query*, a função do *Retriever* é percorrer o conhecimento disponível e encontrar informação que vá de encontro a essa *query*. Funciona como uma espécie de motor de busca e é essencial pois determina a relavância e qualidade da informação que será usada para gerar a resposta final. O métodos de recuperação serão discutidos na secção 3.2.3.
- **Augmentation**: Através da *query* inicial e da informação recuperada no passo anterior, o processo de *augmentation* consiste na elaboração do *prompt* enriquecido com a informação contextual. Este *prompt* irá conter essa informação com objetivo aumentar o nível de precisão da resposta. Nesta fase, através de Prompt Engineering, é possível manipular o *prompt* de diversas formas, tal como por exemplo a definição da linguagem de resposta, nivel de formalidade ou algum detalhe mais específico. Este processo será analisado em detalhe na secção 3.2.4.
- **Generation**: Consiste na chamada ao LLM e geração da resposta através do input *prompt* formulado no passo anterior. O conhecimento pré-existente é tido em conta pelo LLM permitindo que as respostas sejam mais inteligentes e informadas para o contexto em questão.

https://medium.com/@sandyeep70/understanding-rag-evolution-components-implementation-and-applications-ecf72b778d15

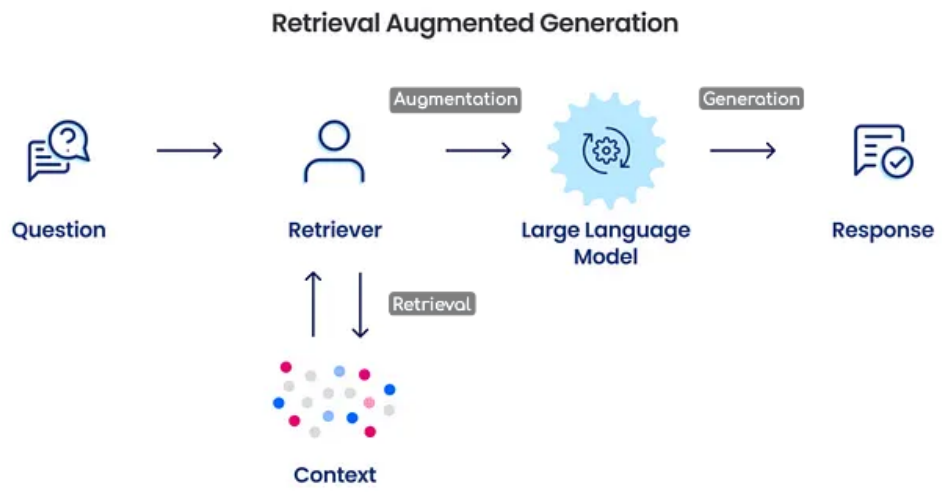

Figura 2.1: Arquitetura simplificada do RAG - https://medium.com/@sahin.samia/what-is-retrieval-augmented-generation-rag-in-llm-and-how-it-works-a8c79e35a172

2.6 Aplicação ao Suporte Técnico

TODO remover esta secção encaixar no estado da arte

Como esses conceitos são aplicáveis ao problema da dissertação. Benefícios esperados da implementação.

Capítulo 3

State of the Art

3.1 Metodologia

3.1.1 Questões de Investigação

RQ1 - Quais as melhores práticas RAG associadas a suporte técnico? RQ1.1 - Quais as melhores práticas de indexaçãoas e recuperação de conhecimento? RQ1.2 - Como se comparam os diferentes tipos de armazenamento de conhecimento (vector stores, document stores, etc.)? RQ1.3 - Quais as abordagens de Augmentation mais eficazes para melhorar a precisão das respostas?

RQ2 - Quais são os desafios técnicos na implementação de uma solução baseada em RAG para suporte técnico?

RQ3 - Como as frameworks ou bibliotecas disponíveis para desenvolvimento RAG se comparam em termos de arquitetura, flexibilidade e facilidade de utilização?

3.1.2 Research Scope

3.1.3 Critérios de Inclusão e Exclusão

3.1.4 Processo de Seleção

3.1.5 Coleção de Dados

3.2 Melhores Práticas RAG associadas a Suporte Técnico (RQ1)

TODO introduzir secção

3.2.1 Indexação

O processo de indexação ocorre antes do sistema RAG estar diponmível para consultas. Consiste na preparação dos dados e armazenamento dos mesmos de forma a que possam ser facilmente recuperados. Este processo encontra-se ilustrado na Figura 3.1. Primeiramente a documentação é dividida em *chunks*, que são representações legiveis de texto de tamanho reduzido. Estes fragmentos podem ser armazenados diretamente, mas atualmente a melhor prática é serem convertidos em *embeddings*, que são representações vetoriais, através dos embedding models. Estes armazenam informação semântica e dependendo da qualidade do modelo de geração, a eficácia de recuperação é diretamente afetada (Rau et al. 2024; Špeletić et al. 2024).

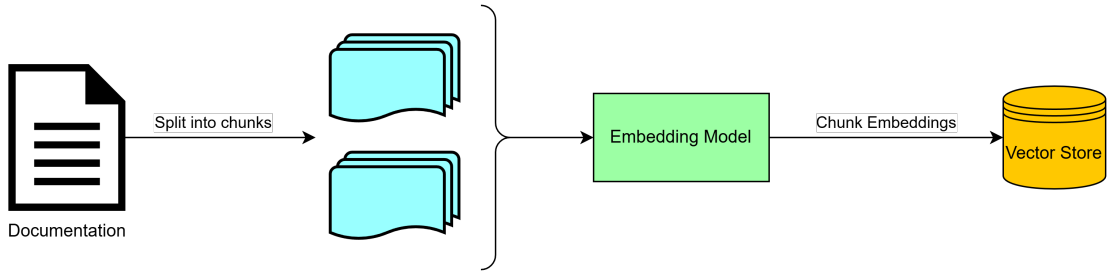

Figura 3.1: Processo de Indexação (Adaptado de Špeletić et al. 2024)

A efetividade de cada modelo depende muito do tipo de contexto e idioma que se pretende indexar. Modelos como o OpenAI ada-002 demonstram desempenho robusto em tarefas em inglês e em diversos domínios, enquanto modelos como Cohere multilingual e o Aleph-Alpha luminous destacam-se pela capacidade de lidar com multiplos idiomas e contextos de baixo recurso (Kamalloo et al. 2023).

3.2.2 Armazenamento

O armazenamento do conhecimento RAG pode ser feito de vários formas. Em casos de sistemas pequenos, onde a quantidade de documentação é reduzida, pode ser suficiente o uso de dados brutos presentes em ficheiros ou bases de dados relacionais (TODO referencia). Contudo, para sistemas onde a quantidade de documentação é extensa, a melhor prática é o uso de bases e dados vetoriais, que são otimizadas para armazenar e recuperar embeddings de forma eficiente e escalável (X. Wang et al. 2024).

Um estudo de X. Wang et al. 2024 comparou diferentes bases de dados vetorias open-source — incluindo Weaviate, Faiss, Chroma, Qdrant, e Milvus — de acordo com os seguintes critérios: Multiple Index Type — flexibilidade para otimizar pesquisas baseado em diferentes carateristicas de dados, Billion-scale — capacidade de lidar com grandes volumes de dados, Hybrid Search — combinação de pesquisa tradicional por keywords com pesquisa vetorial, Cloud-native — capacidade de gestão em ambientes cloud. Concluiu-se que Milvus é a solução mais completa por cumprir com o quatro critérios.

TODO remover este paragrafo? O FAISS é araterizado por ser uma biblioteca com excelente desempenho local. No entanto, não é cloud-native, nem oferece suporte a Hybrid Search, o que pode limitar seu uso em arquiteturas modernas baseadas em microserviços ou em cloud. TODO ====

Por outro lado, soluções como PGVector oferecem a vantagem de simplificar a infraestrutura ao integrar a busca vetorial diretamente em bases relacionais amplamente utilizadas, como o PostgreSQL. Embora não ofereça a mesma escalabilidade que sistemas como Milvus, pode ser uma escolha adequada para casos onde a complexidade e o volume de dados são moderados, ou quando há necessidade de integração com sistemas existentes (TODO referencia).

3.2.3 Recuperação e Geração de Respostas

Após a indexação e armazenamento do contexto, o sistema RAG está preparado para responder a consultas. A recuperação pode ser feita exclusivamente através de embedding model,

através de um comparação semântica entre a query e os chunks indexados, ou através de recuperadores lexicais quando se pretende velocidade e menor utilização de recursos.

O processo de recuperação semântica encontra-se representado na Figura 3.2.

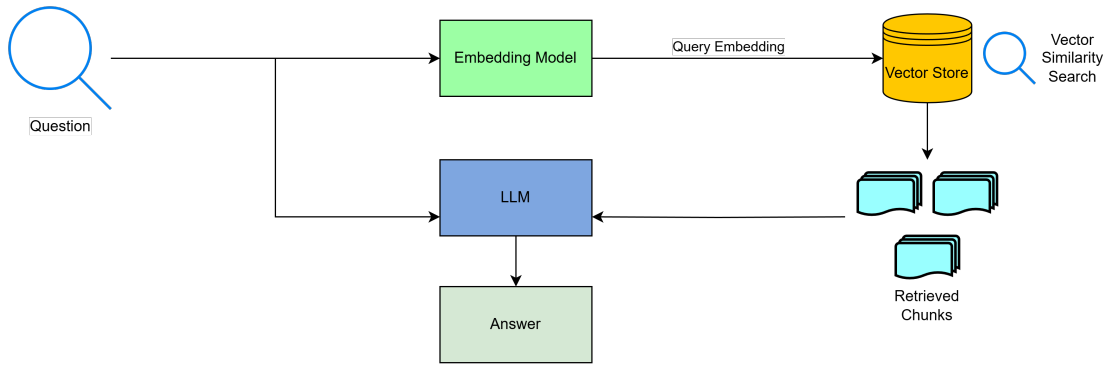

Figura 3.2: Processo de Recuperação e Geração exclusivamente com Embedding Model (Adaptado de Špeletić et al. 2024)

Primeiramente a query de consulta é vectorizada através do mesmo embedding model utilizado na indexação dos dados. É importante que o modelo seja o mesmo para garantir que os embeddings estejam no mesmo espaço semântico. A partir daí, é feita uma pesquisa por similaridade entre a query vectorizada e os chunks indexados na base de dados. Para finalizar, ocorre a geração da prompt query que resulta na junção da query inicial com os chunks recuperados. Esta prompt é então enviada para o LLM, que gera a resposta final.

Além da recuperação exclusiva com embeddings models, existem recuperadores lexicais que atuam diretamente sobre os chunks de texto legíveis. Estes recuperadores apenas atuam sobre uma base de dados textual e recuperam os chunks mais relevantes com base em palavras-chave. Um exemplo bastante utilizado é o BM25, que se destaca pela sua simplicidade e rapidez.

Recuperação de forma hibrida é também uma abordagem comum, onde o recuperador lexical é usado para recuperar os chunks mais relevantes e os embeddings são usados para calcular a similaridade semântica entre a query e os chunks recuperados para fins de reranking e filtragem final (Gupta, Ranjan e Singh 2024; Kamalloo et al. 2023). Esse processo encontra-se ilustrado na Figura 3.3.

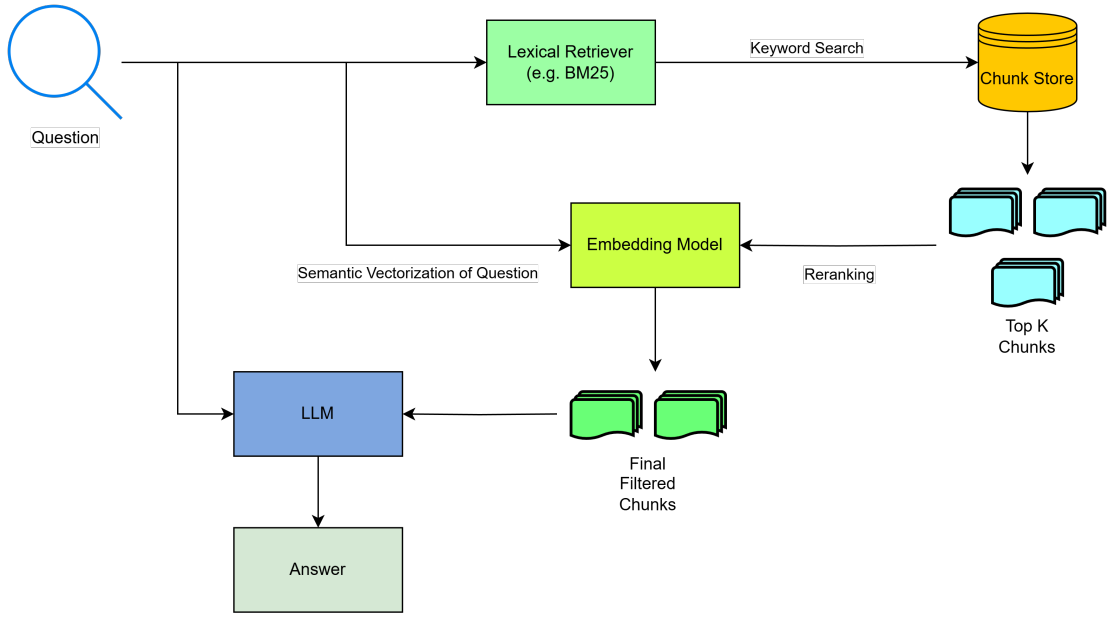

Figura 3.3: Processo de Recuperação e Geração com Recuperador Lexical e Embedding Model

A escolha entre recuperação semântica ou lexical depende do contexto e dos requisitos do sistema. Recuperação semântica é mais eficaz para contextos complexos e técnicos, onde a compreensão do significado é crucial, enquanto a recuperação lexical pode ser suficiente para contextos mais simples ou quando a velocidade é uma prioridade.

3.2.4 Augmentation

input: Query + documentos recuperados output: Prompt enriquecido

TODO

Prompt Engineering

TODO

3.2.5 Aplicações Práticas

TODO se calhar passar isto para os conceitos fundamentais e tambem introduzir o RAG lá

RAG é atualmente usado, por exemplo, para suporte ao consumidor através da criação de Chatbots capazes de recuperar FAQs e o conhecimento do negócio de forma fácil. Gestão do conhecimento empresarial é outro exemplo de caso de uso, pois permite que os funcionários recuperarem e acedam a informação do contexto de trabalho de forma mais rápida. Este último caso de uso vai de encontro aos objetivos do presente projeto.

3.3 Desafios Técnicos na Implementação (RQ2)

3.3.1 Indexação, recuperação e relevância das respostas

A eficiência do RAG depende da capacidade de recuperar documentação relevante. No contexto de suporte técnico, a recuperação necessita ser precisa para fornecer soluções corretas o que é desafiador devido à complexidade e especificidade da documentação (Isaza et al. 2024a).

No que toca recuperação por similaridades semanticas, Soman e Roychowdhury 2024 refere que o uso de embeddings para chunks de texto grandes (> 200 palavras) resulta em valores de similaridade artificialmente altos. Isso sugere que, mesmo quando as frases não são semanticamente parecidas, o modelo acha que são, apenas por serem longas. Contudo, documentação que usa grande número de abreviações e paragrafos para um tópico tornanam as obeservações mais relevantes. Além disso, concluiu-se palavras-chave mais próximas do começo de de uma frase são recuperadas com maior precisão.

Adicionalmente, estudos recentes revelam que sistemas RAG apresentam dificuldades significativas quando aplicados em ambientes empresariais. **RAGDoesNotWork2024** demonstram que, mesmo quando a resposta correta está presente no contexto, o sistema frequentemente falha em recuperá-la. Isso ocorre, em parte, devido ao desajuste entre a estrutura dos documentos técnicos (como FAQs, procedimentos, logs, etc.) e as estratégias tradicionais de segmentação em chunks.

Essa falha é confirmada por **SevenPoints2024**, que identificaram múltiplos pontos críticos no funcionamento de sistemas RAG, incluindo:

- Falta de conteúdo: Quando a informação necessária não está no contexto, o sistema pode responder com conteúdos enganosos, sugerindo que sabe a resposta mesmo sem dados de apoio.
- Fraca classificação da documentação: Mesmo com a informação presente no contexto, ela pode não ser corretamente classificada e consequentemente não recuperada.
- Informação não extraida: Caso haja informações contraditórias no contexto, o retriever pode apresentar falhas.
- Especificidade incorreta: O sistema pode gerar respostas muito vagas ou excessivamente específicas sem compreender especificamente o que foi solicitado.

Diversas técnicas de Finetunning podem ser utilizadas para contornar estas situações.

TODO melhorar daqui para baixo

Contextos maiores geram respostas mais precisas. A inclusão de metadados, como o nome do ficheiro e número do chunk, melhora a interpretação da informação recuperada. Modelos de embeddings open source também se mostram eficazes, especialmente em textos curtos. Para garantir resultados robustos, é essencial calibrar cuidadosamente o pipeline RAG — incluindo chunking, embeddings, recuperação e consolidação — além de manter uma monitorização contínua, dado que o sistema lida com entradas desconhecidas em tempo real.

3.3.2 Integração com Sistemas Existentes

https://arxiv.org/pdf/2409.13707

https://arxiv.org/pdf/2404.00657

3.4 Frameworks para Desenvolvimento de RAG (RQ3)

Antes de apresentar cada tecnologia, importa referir que as informações aqui descritas foram obtidas com base na respetiva documentação oficial disponível nos websites de cada projeto. Nesta análise, cada framework será avaliada segundo três critérios: (i) arquitetura, (ii) flexibilidade e (iii) facilidade de utilização. Adicionalmente, serão também consideradas as formas de integração com bases de dados externas (vector stores ou document stores), bem como o suporte e comunidade em torno de cada projeto.

3.4.1 LangChain

Fundada por Harrison Chase em 2022, LangChain é uma framework open-source para o desenvolvimento de aplicações que integram LLMs com fontes externas de dados e ferramentas de software. Está disponível em Python e JavaScript, oferecendo um ambiente centralizado e altamente extensível para construir soluções com LLMs de forma modular e reutilizável.

A arquitetura do LangChain é baseada no conceito de *Chains*, que representam sequências de chamadas a LLMs e a outras ferramentas auxiliares. Estas *Chains* podem ser compostas de forma modular, permitindo construir fluxos de execução complexos com facilidade.

Com a introdução do LangChain Expression Language (LCEL), passou a ser possível definir estas sequências de forma declarativa. O LCEL oferece diversos construtores prontos para uso, como:

- *create_stuff_documents_chain*: Formata uma lista de documentos em um prompt para o LLM.
- *create_sql_query_chain*: Gera consultas SQL a partir linguagem natural.
- *create_history_aware_retrieve*: Utiliza o histórico de conversas para gerar consultas de busca mais precisas.
- *create_retrieval_chain*: Integra recuperação de documentos relevantes com geração de respostas por LLM.

3.4.2 Prompt Templates

TODO retificar esta subsection

São usados para guiar as repostas do modelo reproduzindo um PromptValue. Este é o resultado final da intrução a ser transmitida ao LLM assim que o input do utilizador for executado em cima do template. São importantes pois direcionam a instrução para a obtenção de respostas que vão de em contra ao objetivo da aplicação.

```
1 from langchain_core.prompts import PromptTemplate
2
3 prompt_template = PromptTemplate.from_template("Tell me a joke about {
      topic}")
4
5 prompt_template.invoke({"topic": "cats"})
```

Listing 3.1: Using LangChain to create a prompt template

No contexto de uma aplicação para contar anedotas (3.1), apenas com a especificação do tema da anedota, neste caso gatos, o template cria o PromptValue "Tell me a joke about cats"a ser executado pelo LLM.

https://python.langchain.com/docs/concepts/prompt_templates/

LangChain é uma das frameworks mais flexíveis atualmente disponíveis. A sua estrutura modular permite criar soluções para diversos domínios — desde sistemas de chat com memória contextual até agentes com raciocínio multi-etapas e integração com APIs externas. Suporta LLMs de múltiplos fornecedores, incluindo:

- OpenAI (via API Key)
- Cohere
- Anthropic (Claude)
- Modelos open-source (como LLaMA, Flan-T5, Mistral, etc.) via Hugging Face

Os componentes podem ser combinados e substituídos livremente, o que torna o LangChain altamente adaptável a diferentes arquiteturas e necessidades de negócio.

A curva de aprendizagem do LangChain pode ser ligeiramente acentuada devido à grande variedade de conceitos e componentes disponíveis. No entanto, a documentação é abrangente e inclui exemplos práticos, tutoriais e guias para casos de uso comuns.

O LCEL veio simplificar bastante o processo de criação de pipelines, ao permitir a definição declarativa de fluxos sem necessidade de escrita de código imperativo detalhado. Ainda assim, o domínio completo da framework pode exigir tempo, especialmente na fase de composição de agentes ou integração de ferramentas externas.

LangChain disponibiliza integração nativa com diversos sistemas de armazenamento de documentos, tanto tradicionais como vetoriais, tais como Elasticsearch e Weaviate, duas tecnologias open-source que permitem pesquisa vetorial, armazenamento de embeddings e indexação de documentos.

Estes sistemas são acedidos através de abstrações como `VectorStoreRetriever` e `DocumentLoaders`, que permitem a configuração e adaptação do processo de indexação e recuperação de informação de acordo com os requisitos específicos da aplicação.

LangChain possui uma comunidade ativa e em crescimento, com elevado dinamismo no repositório GitHub, contanto com mais de cem mil estrelas. A empresa responsável pelo projeto, tem promovido o desenvolvimento contínuo da framework, bem como o lançamento de ferramentas complementares como o LangSmith, destinado à monitorização e depuração de aplicações baseadas em LLMs. Esta vitalidade comunitária traduz-se em documentação continuamente atualizada, partilha regular de boas práticas e integração frequente de contributos externos.

==========

https://www.ibm.com/think/topics/langchain https://python.langchain.com/v0.1/docs/modules/chains/

==========

3.4.3 Haystack

Haystack é uma framework open-source desenvolvida pela empresa alemã Deepset, com o objetivo de facilitar a construção de pipelines baseadas em LLMs, especialmente para casos de uso como RAG, question answering, classificação, extração de informação e pesquisa semântica em documentos. A linguagem principal utilizada é Python. A primeira versão surgiu em 2020, tendo recentemente evoluído para a versão 2.0, com uma reformulação completa da sua arquitetura.

A principal inovação do Haystack 2.0 é a sua arquitetura orientada a componentes. Cada componente representa uma unidade funcional independente, com responsabilidade bem definida dentro de uma pipeline. Estes componentes são combinados de forma declarativa para formar pipelines personalizadas, robustas e escaláveis. Entre os tipos de componentes disponíveis encontram-se:

- **Document Stores**: responsáveis por armazenar documentos e suportar tanto índices tradicionais quanto vetoriais.
- **Retrievers**: mecanismos de recuperação de informação, com suporte a métodos tradicionais como BM25 e a recuperação semântica com embeddings.
- **Rankers**: permitem reordenar os documentos recuperados com base em critérios de relevância mais refinados.
- **Generators**: utilizam LLMs para gerar respostas com base na informação recolhida.
- **Prompt Nodes**: enviam prompts configuráveis para modelos locais ou remotos.
- **Routers**: introduzem lógica condicional nas pipelines, facilitando a criação de fluxos adaptativos.

As pipelines podem ser definidas em ficheiros YAML ou diretamente em Python, promovendo flexibilidade tanto para utilizadores técnicos como não técnicos.

Haystack destaca-se pela sua grande flexibilidade. A arquitetura baseada em componentes independentes permite a substituição e reconfiguração de cada etapa do fluxo de processamento, facilitando a adaptação a diferentes domínios, fontes de dados e estratégias de interação com LLMs. Os desenvolvedores podem ainda combinar múltiplos retrievers, utilizar lógica condicional com routers e integrar diversos serviços externos com relativa facilidade.

A curva de aprendizagem do Haystack pode ser moderada, especialmente para utilizadores que não estejam familiarizados com conceitos como pipelines declarativas ou integração com serviços externos. No entanto, a documentação oficial é bastante completa e a existência de pipelines pré-configuradas — como a `PredefinedPipeline.INDEXING` e a `PredefinedPipeline.RAG` — facilita bastante o início do desenvolvimento. Estas permitem, respetivamente, indexar documentos e realizar tarefas de RAG com configuração mínima.

Haystack oferece suporte nativo a várias tecnologias de armazenamento de documentos, como Elasticsearch e Weaviate, referidos anteriormente. Em ambiente de desenvolvimento, o Haystack oferece o InMemoryDocumentStore, uma opção lightweight que armazena os documentos diretamente em memória, sendo ideal para testes.

Haystack possui uma comunidade ativa. O repositório oficial no GitHub conta com mais de vinte mil estrelas (TODO referencia github), issues frequentemente respondidas, e releases regulares. A documentação é extensa, com guias, tutoriais e exemplos práticos.

==========

https://medium.com/aimonks/haystack-an-alternative-to-langchain-carrying-llms-bf7c515c9a7e https://haystack.deepset.ai/overview/intro https://docs.haystack.deepset.ai/docs/pipeline-templates

3.4.4 LlamaIndex

TODO

3.4.5 Spring AI

Spring AI é uma framework recente, disponibilizada publicamente no início de 2024, com o objetivo de simplificar o desenvolvimento de aplicações baseadas em inteligência artificial, especialmente no ecossistema Java. Inspirado por projetos como LangChain e LlamaIndex, o Spring AI surge como uma extensão natural do ecossistema Spring, promovendo a integração de LLMs em aplicações corporativas de forma modular, acessível e escalável.

A arquitetura do Spring AI é orientada a componentes e organizada em torno do conceito de *Advisors*, responsáveis por encapsular diferentes estratégias de interação com LLMs e bases de dados vetoriais. A framework disponibiliza dois tipos principais de *advisors* para fluxos RAG:

- **QuestionAnswerAdvisor**: foca-se na recuperação exata de informação a partir de dados externos, construindo respostas exclusivamente com base nos documentos recuperados da base vetorial.
- **RetrievalAugmentationAdvisor**: combina a informação externa com o conhecimento prévio embutido no modelo base, permitindo uma resposta enriquecida e mais flexível.

A comunicação com estes componentes é configurada programaticamente através da API do Spring AI, possibilitando a definição de parâmetros como o limiar de similaridade na pesquisa.

Adicionalmente, a framework estrutura os fluxos de processamento em módulos reutilizáveis, entre os quais se destacam:

- *Document Reader Modules* - para leitura e preparação de documentos;
- *Embedding Modules* - para geração de vetores a partir de texto;
- *Retriever Modules* - para pesquisa em bases vetoriais;
- *LLM Modules* - para interação com o modelo de linguagem.

Esta abordagem modular facilita a composição e manutenção de pipelines RAG personalizadas.

Apesar de ser uma framework emergente, o Spring AI apresenta uma estrutura suficientemente flexível para acomodar diversos casos de uso. A separação clara entre módulos permite que cada fase do pipeline seja configurada ou substituída conforme os requisitos específicos da aplicação.

A framework suporta múltiplos provedores de LLMs, incluindo OpenAI, Hugging Face, Mistral e Cohere, entre outros, com mecanismos de autenticação e configuração standard através do ecossistema Spring.

Por estar profundamente integrado com o ecossistema Spring, a framework beneficia de recursos como injeção de dependências, configuração centralizada, gestão de contexto e integração com outras soluções do universo Spring Boot, o que a torna especialmente atrativa para ambientes corporativos baseados em Java.

Um dos principais objetivos do Spring AI é tornar a utilização de LLMs mais acessível a programadores do ecossistema Java. A framework herda a familiaridade e consistência do paradigma Spring, oferecendo uma experiência previsível e bem documentada.

Os *advisors* e módulos podem ser facilmente instanciados e configurados, sendo suportados por exemplos concisos e tutoriais disponíveis na documentação oficial. A integração com ferramentas padrão do Spring (como o Spring Boot Actuator ou o Spring Configuration) facilita a observabilidade e a gestão de parâmetros em ambientes de produção.

Contudo, como se trata de uma framework ainda em evolução, podem existir limitações ao nível de abstrações mais avançadas quando comparada com alternativas mais consolidadas no ecossistema Python.

O Spring AI oferece integração nativa com várias bases de dados vetoriais, através da abstração `VectorStore`. Atualmente, estão disponíveis conectores para:

- **FAISS**
- **Qdrant**
- **Pinecone**
- **Weaviate**

Estes armazenamentos vetoriais podem ser utilizados diretamente nos módulos de recuperação, com suporte para parâmetros como o número de documentos a recuperar e o grau de similaridade exigido.

A framework prevê ainda a evolução para incluir mecanismos mais elaborados de pré-processamento de documentos, nomeadamente para segmentação, limpeza e enriquecimento semântico dos conteúdos.

Dado o seu lançamento recente, a comunidade do Spring AI ainda se encontra em crescimento. No entanto, beneficia do forte ecossistema da Spring Framework e da extensa base de utilizadores da comunidade Java. O projeto é mantido pela equipa oficial da Spring, o que garante qualidade no design, documentação consistente e ciclos de lançamento regulares.

A documentação oficial cobre os principais casos de uso, e já existem exemplos práticos disponíveis em repositórios públicos que demonstram a aplicação da framework em pipelines RAG.

3.4.6 Comparação resumo entre tecnologias

Para resumir, na Tabela 3.1 estão descritos os principais pontos de comparação entre as diferentes tecnologias analisadas.

Tabela 3.1: Comparação entre tecnologias RAG

Ferramenta	Arquitetura	Flexibilidade	Facilidade de utilização	Integração com bases vetoriais / document stores	Suporte, comunidade e documentação
LangChain	Orquestração modular e declarativa	Elevada	Intermédia	FAISS, Pinecone, Chroma, Redis, Weaviate, Qdrant.	Comunidade / Suporte ++ ; Documentação extensa
Haystack	Arquitetura orientada a componentes	Elevada	Intermédia a alta	Suporte robusto com "Document Stores"como Elasticsearch, Qdrant, Weaviate; suportando indexação vetorial e tradicional.	Comunidade / Suporte +++ ; Documentação extensa
Spring AI	Arquitetura modular; integração com o ecossistema Spring.	Moderada a elevada	Alta para programadores Java	FAISS, Qdrant, Pinecone, Weaviate; integração através de "VectorStore".	Comunidade / Suporte + ; Documentação em crescimento

Em termos de arquitetura as tecnologias são semelhantes em alguns principios gerais, mas diferem na forma como organizam os componentes. A LangChain usa o conceito de chains e agents que permite encadear multiplos passos de forma declarativa. Já a Haystack possui um arquitetura orientada a componentes reutilizáveis. Na Spring AI a arquitetura é modular e integrada no ecossistema Sprint Boot com foco para desenvolvedores Java, usando abstrações típicas do Spring.

No que diz respeito à flexibilidade, LangChain e Haystack oferecem maior capacidade de personalização, com suporte a múltiplas configurações e integrações. O Spring AI, apesar de mais recente, apresenta uma modularidade sólida, embora mais orientada a padrões corporativos Java.

Em termos de facilidade de utilização, o Spring AI proporciona uma curva de aprendizagem

mais suave para programadores familiarizados com Spring Boot. A Haystack combina flexibilidade com pipelines pré-configuradas, ao passo que a LangChain, apesar de poderosa, exige maior domínio técnico para compor soluções complexas.

Todas as ferramentas analisadas oferecem integração com sistemas de armazenamento vetorial da atualidade, como FAISS, Qdrant e Weaviate. O suporte é robusto e adaptável em todas, com destaque para a abstração VectorStore no Spring AI e a diversidade de conectores nativos no Haystack.

Por fim, no que respeita a suporte e comunidade, LangChain e Haystack apresentam ecossistemas consolidados, com documentação extensa, comunidades ativas e evolução contínua — refletido na Tabela 3.1 com os níveis ++ e +++, respetivamente. O Spring AI, embora ainda em fase de maturação, conta com documentação estruturada e o respaldo da comunidade Spring, o que lhe confere um nível + na escala adotada, indicando um bom ponto de partida com potencial de crescimento acelerado.

3.5 Trabalhos Relacionados

TODO incluir isto?: Embora o foco desta dissertação seja a aplicação de RAG em contextos de suporte técnico, são também analisados trabalhos em domínios adjacentes quando estes oferecem contributos relevantes em termos de arquitetura, avaliação ou boas práticas.

3.5.1 Análise Individual

RAG Chatbot para a OptiMicro Technologies

A OptiMicro, empresa de software Canadiana, decidiu apostar no desenvolvimento de um chatbot utilizando RAG para melhorar o seu serviço de apoio técnico da empresa, mais especificamente no seu serviço DentalWare.

O software foi desenvolvido através de uma plataforma lowcode denominada Flowise AI e a sua arquitetura consistiu em três fases principais: indexação, recuperação e geração. Na fase de indexação, foram utilizados registos de suporte técnico e manuais do software DentalWare, segmentados em chunks de texto. Os dados foram convertidos em vetores através do modelo all-MiniLM-L6-v2, acedido via API da HuggingFace, e armazenados numa base de dados vetorial Pinecone, escolhida pela sua elevada performance. Na etapa de recuperação, a query do utilizador é embedded com o mesmo modelo e comparada com os vetores indexados, sendo selecionados os segmentos mais relevantes. Por fim, a resposta é gerada com o modelo LLaMA 2 (7B), executado localmente via Ollama.

Esta arquitetura encontra-se descrita no diagrama da Figura 3.4, onde é possível verificar de que forma os componentes se relacionam. É de notar que foram utilizados dois documentos como fonte de informação externa definidos de forma estática, tornando o software um pouco restrito.

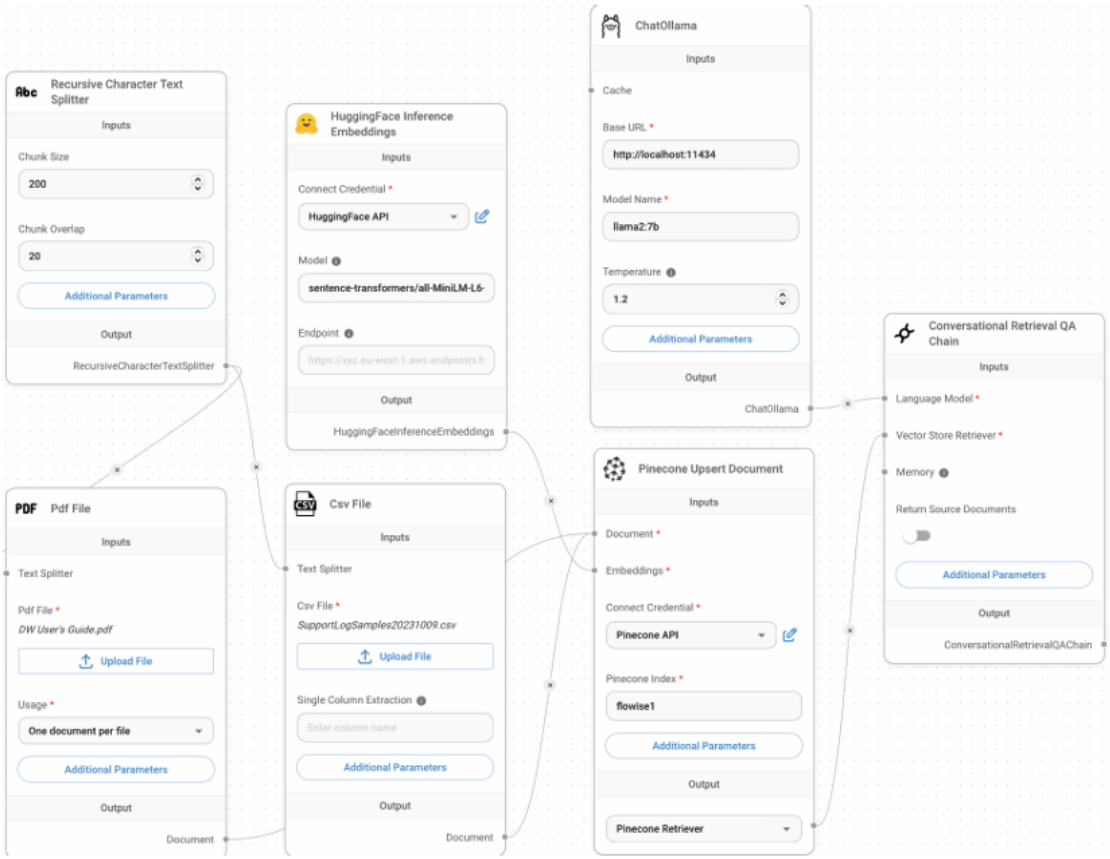

Figura 3.4: Arquitetura do chatbot RAG da OptiMicro (Lee et al. 2024)

Para avaliar o desempenho do chatbot, foram utilizadas 75 questões reais relacionadas com suporte técnico ao software DentalWare. As respostas geradas por este chatbot foram comparadas com as de um chatbot LLM genérico, tendo ambas sido avaliadas com base nos critérios de ROUGE (ROUGE-1, ROUGE-2 e ROUGE-L) e por humanos da OptiMicro. Os resultados demonstraram que o chatbot RAG obteve melhores pontuações em todas as métricas ROUGE, destacando-se com aumentos de 38%, 18% e 40%, respetivamente, face ao modelo genérico, o que evidencia uma maior fidelidade aos conteúdos técnicos. Na avaliação humana, o chatbot RAG obteve 20 respostas classificadas como boas, 18 como razoáveis e apenas 2 como fracas, enquanto o LLM genérico apresentou apenas 3 boas respostas e 2 razoáveis, totalizando 70 respostas consideradas fracas. Estes resultados confirmam a superioridade do modelo RAG no contexto de apoio técnico, pela sua capacidade de gerar respostas mais relevantes e alinhadas com os dados específicos da empresa.

TODO ponto negativa deste software: não foi implementado mecanismo para atualizar automaticamente a base de dados vetorial consoante updates na documentação

Sistema RAG de Recomendação para Suporte Técnico

Isaza et al. 2024b propuseram um sistema de recomendação para resolução de incidentes em suporte técnico baseado em RAG, com o objetivo de sugerir soluções fundamentadas em documentação.

A arquitetura do sistema encontra-se representada na Figura 3.5, sendo composta por cinco etapas principais. A primeira é o pré-processamento, onde certas características do incidente são normalizadas e comparadas com siglas ou nomes alternativos, de forma a melhorar a query de recuperação. Segue-se a classificação de tickets, através do classificador transformer IBM Slate 125m, onde o assunto e a descrição do incidente são analisados para determinar se a sua resolução é viável apenas com a informação fornecida — casos denominados single-turn. Caso o ticket não seja classificado como single-turn, a pipeline termina imediatamente. As etapas seguintes consistem na geração de uma query concisa a partir do conteúdo do ticket, seguida da recuperação com reranking, onde são selecionados os três documentos mais relevantes a partir de uma base de dados vetorial Milvus. A resposta final é gerada com base nesses documentos e é acompanhada de links para as fontes consultadas.

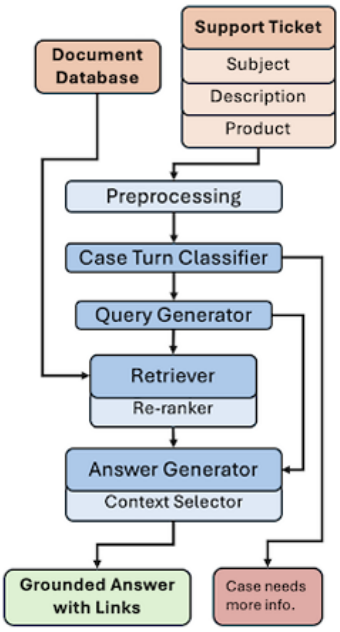

Figura 3.5: Arquitetura do sistema de recomendação de resolução de incidentes proposto por Isaza et al. 2024b

O sistema foi testado em diversos aspetos, quer na geração de queries ou quer na geração da respostas, tendo sido utlizados diferentes LLMs para cada vertente. O modelo Mixtral 8x7B Instruct, que é um LLM fine-tuned para interações dialogadas como chatbots, obteve um BertScore F1 de 0,91 na tarefa de normalizar incidentes em queries concisas, semelhante a LLMs maiores como Falcon-40B, porém com menor custo computacional. Na geração de respostas, este mesmo modelo, combinado com RAG, superou o GPT-4 em métricas como ROUGE-L F1 (0,41 vs. 0,34) e BertScore (0,87 vs. 0,86), destacando a eficienciencia de modelos menores com contexto recuperado. Os autores destacaram também que modelos fine-tuned com infusão de conhecimento técnico, como o modelo InstructLab-IT (7 mil milhões parâmetros), podem superar modelos de grande porte para casos gerais.

Fabula

ficheiro fabula.pdf nas transferencias não é relacionado com IT support mas se calhar é interessante incluir para demonstrar outro caso de uso do RAG e pode dar algum insight util

3.5.2 Conclusões

Bibliografia

Gupta, Shailja, Rajesh Ranjan e Surya Narayan Singh (2024). «A comprehensive survey of retrieval-augmented generation (rag): Evolution, current landscape and future directions». Em: *arXiv preprint arXiv:2410.12837*.

Isaza, Paulina Toro et al. (2024a). «Retrieval Augmented Generation-Based Incident Resolution Recommendation System for IT Support». Em: url: `https://arxiv.org/abs/2409.13707`.

– (2024b). «Retrieval Augmented Generation-Based Incident Resolution Recommendation System for IT Support». Em: *arXiv preprint arXiv:2409.13707*.

Kamalloo, Ehsan et al. (2023). «Evaluating embedding APIs for information retrieval». Em: *arXiv preprint arXiv:2305.06300*.

Lee, Ho-Chit et al. (2024). «Development of an RAG-Based LLM Chatbot for Enhancing Technical Support Service». Em: *TENCON 2024-2024 IEEE Region 10 Conference (TENCON)*. IEEE, pp. 1080–1083.

Rau, David et al. (2024). «Context embeddings for efficient answer generation in rag». Em: *arXiv preprint arXiv:2407.09252*.

Soman, Sumit e Sujoy Roychowdhury (2024). «Observations on Building RAG Systems for Technical Documents». Em: Published as a Tiny Paper at ICLR 2024. url: `https://arxiv.org/pdf/2404.00657`.

Špeletić, Matija et al. (2024). «EXPLORING RAG IN MEDICAL QUESTION ANSWERING: INTEGRATING LLMS AND VECTOR DATABASES». Em.

Wang, Xiaohua et al. (2024). «Searching for best practices in retrieval-augmented generation». Em: *arXiv preprint arXiv:2407.01219*.

Apêndice A

Appendix Title Here

Write your Appendix content here.