ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

[Thesis Title]

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, 1 de maio 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, 1 de maio 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

Lista de Tabelas

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

1.3 Objectives

1.4 Methodology

1.5 Work Plan

1.6 Document Structure

Capítulo 2

Key Concepts

2.1 Artificial Intelligence

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 "boom"tecnológico, 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

Diferença entre aprendizado de máquina e aprendizado profundo. Como esses conceitos se relacionam com modelos de IA modernos.

2.3 Large Language Models

Os Large Language Models (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.

2.4 Retrieval-Augmented Generation

Retrieval-Augmented Generation (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 em base de dados ou documentos.

Esta técnica conta com dois principais componentes: o *Retriever* e o *Generator*.

- **Retriever**: Baseado na *query* de *input*, a função do *Retriever* é percorrer o conhecimento disponível (p.e. base de dados vetoriais, documentos, fontes *web*) 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.
- **Generator**: O *Generator* atua após o *Retriever* ter feito a recuperação de informação relevante, juntando-a com a *query* original para elaborar uma resposta contextualizada. O conhecimento pré-existente é tido em conta pelo o LLM permitindo que as respostas sejam mais inteligentes e informadas para o contexto em questão.

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.

TODO Incluir diagrama de arquitetura

2.4.1 Prompt Templates

TODO se calhar passar isto para o langchain

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
```

```
prompt_template = PromptTemplate.from_template("Tell me a joke about {
    topic}")

prompt_template.invoke({"topic": "cats"})
```

Listing 2.1: Using LangChain to create a prompt template

No contexto de uma aplicação para contar anedotas (**??**), 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/

2.4.2 Algoritmos Retriever

TODO tipo isto https://docs.llamaindex.ai/en/stable/examples/retrievers/bm25$_retriever$/

2.4.3 Processo de indexação

TODO entrar em mais detalhes do RAG pois isto é o core do projeto TODO falar de mais componentes do RAG

2.5 Bases de Dados Vetoriais

Conceito de embeddings e busca vetorial. Exemplos de ferramentas (FAISS, Weaviate, Pinecone).< Importância da base vetorial no contexto do RAG.

2.6 Aplicação ao Suporte Técnico

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

Capítulo 3

State of the Art

3.1 Methodology

3.1.1 Research Questions

RQ1 - Como as frameworks ou bibliotecas disponíveis para desenvolvimento RAG se comparam em termos de arquitetura, flexibilidade e facilidade de utilização?

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

RQ3 - Quais as melhores práticas para a indexação e recuperação de dados?

RQ4 - Qual a LLM mais adequada para uso de RAG para suporte técnico?

3.1.2 Research Scope

3.1.3 Eligibility Criteria

3.1.4 Selection Process

3.1.5 Data Collection

3.2 Results and Analysis

3.2.1 RQ1 - Como as frameworks ou bibliotecas disponíveis para desenvolvimento RAG se comparam em termos de arquitetura, flexibilidade e facilidade de utilização?

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.

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.

Arquitetura 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.

Flexibilidade 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.

Facilidade de utilização 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.

Integração com bases de dados vetoriais e document stores 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.

Suporte e comunidade 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/

==========

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.

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.

Flexibilidade 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.

Facilidade de utilização 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.

Integração com bases de dados vetoriais/document stores 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.

Suporte e comunidade 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

LlamaIndex

TODO

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.

Arquitetura 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.

Flexibilidade 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.

Facilidade de utilização 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.

Integração com bases de dados vetoriais e document stores 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.

Suporte e comunidade 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.

Comparação resumo entre tecnologias

Critério	LangChain	Haystack	Spring AI
Arquitetura	Baseada em "Chains"e componentes como Agents, Memory e LCEL; orquestração modular e declarativa.	Arquitetura orientada a componentes; pipelines declarativas compostas por módulos independentes com funções específicas.	Arquitetura modular baseada em "Advisors"e "Modules"; integração com o ecossistema Spring.
Flexibilidade	Elevada; permite composição livre de fluxos complexos com múltiplos modelos e fontes.	Elevada; pipelines altamente personalizáveis e componentes reutilizáveis.	Moderada a elevada; altamente configurável dentro do ecossistema Spring, mas ainda limitado em abstrações mais avançadas.
Facilidade de utilização	Intermédia; documentação extensa, mas curva de aprendizagem acentuada devido à diversidade de abstrações.	Intermédia a alta; pipelines declarativas em YAML ou Python facilitam a configuração, apesar da curva inicial.	Alta para programadores Java; abordagem familiar para utilizadores do Spring, com APIs bem documentadas.
Integração com bases vetoriais / document stores	Suporte nativo a diversos sistemas como FAISS, Pinecone, Chroma, Redis, Weaviate, Qdrant.	Suporte robusto com "Document Stores"como Elasticsearch, Qdrant, Weaviate, entre outros; suportando indexação vetorial e tradicional.	Suporte direto a FAISS, Qdrant, Pinecone, Weaviate; integração através de "VectorStore".
Suporte e comunidade	Comunidade ampla e ativa; suporte institucional (LangChain Inc.); atualizações regulares e ferramenta complementar (LangSmith).	Comunidade sólida e madura; mantido pela Deepset; documentação rica e projeto bem estabelecido.	Comunidade emergente; mantido pela VMware/Spring Team; documentação em crescimento e roadmap promissor.

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

Existem diversos desafios técnicos na implementação de uma solução RAG para suporte técnico, tais como para a indexação e recuperação de dados, a integração com sistemas existentes no sentido de alimentação do conhecimento e precisão e relevância das respostas.

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. 2024).

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.

Integração com Sistemas Existentes

https://arxiv.org/pdf/2409.13707

https://arxiv.org/pdf/2404.00657

3.3 Related Work

nao sei se fica bem aqui este topico

Bibliografia

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

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.

Apêndice A

Appendix Title Here

Write your Appendix content here.