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, 27 de abril 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, 27 de abril 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 (2.1), 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/

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

RQ2 - Quais frameworks ou bibliotecas disponíveis apresentam melhor desempenho, flexibilidade e facilidade de integração em soluções 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- 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, **soman2024observations** 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

3.2.2 RQ2 - Quais frameworks ou bibliotecas disponíveis apresentam melhor desempenho, flexibilidade e facilidade de integração em soluções RAG para suporte técnico?

TODO aqui antes de começar a falar das tecnolgias é importante referir que cada tecnologia foi consultada individualmente e a informação provem da documentação oficial encontrada nos websites oficiais

LangChain

https://www.ibm.com/think/topics/langchain

Fundada por Harrison Chase em 2022, LangChain é uma framework open-source de orquestração para o desenvolvimento de aplicações que fazem uso de LLMs. Está disponível em Python ou Javascript e oferece um ambiente centralizado para construir soluções que integram LLMs com diferentes fontes de dados externas e workflows de software.

É compatível como a maioria dos LLMs como GPT da OpenAI através da respetiva API Key e com modelos open-source como LLaMa e Google Flan-T5 através da plataforma Hugging Face. Oferece uma maneira comoda de manipular os Prompt Templates para gerar respostas consistentes e possibilita a organização e recuperação de dados contextuais externos.

Referencia de chains: https://python.langchain.com/v0.1/docs/modules/chains/

Chains são a sua funcionalidade core. Estas representam sequências de chamadas, tanto para um LLM como para ferramentas adicionais, que permitem compor fluxos de processamento complexos de forma modular. LangChain Expression Language (LCEL) é uma linguagem declarativa que facilita a criação de Chains reutilizávies. LCEL fornece diversos construtores de Chains 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.

==========

TODO falar dos agents? TODO referir que existe integração com ferramentas externas tipo google search TODO referir o suporte com bases de dados vetoriais TODO suporte/-comunidade

TODO ferramentas e ecossistema: TODO Falar Suporte a LangServe, LangSmith (monitoramento), LangGraph (para fluxos complexos).

TODO Comunidade mais ativa, integração com muitos serviços.

==========

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 serem usadas em casos de uso de pesquisa, como RAG, respostas a perguntas (question answering), classificação, extração de informação e pesquisa semântica em documentos, sendo a sua linguagem core o Python. A sua primeira versão surgiu em 2020, tendo recentemente evoluido para a sua versão 2.0, que levou à introdução de uma arquitetura completamente modular e orientada a componentes, com foco na flexibilidade, na reutilização e na fácil integração com serviços externos, como OpenAI, Hugging Face e outros.

A principal inovação do Haystack 2.0 reside na sua abordagem centrada em componentes. Cada componente representa uma unidade funcional independente, com uma responsabilidade bem definida dentro de uma pipeline. Esta arquitetura permite que os desenvolvedores construam soluções complexas de forma declarativa, combinando componentes reutilizáveis para formar pipelines personalizadas, robustas e facilmente escaláveis.

Por exemplo, os componentes de armazenamento de documentos, conhecidos como Document Stores, são responsáveis por armazenar e disponibilizar os documentos a serem consultados pelos restantes elementos da pipeline. Estes podem basear-se em tecnologias como Elasticsearch, Weaviate, Qdrant, entre outras, e suportam tanto índices tradicionais quanto vetoriais. Para alimentar estas bases de dados, o Haystack oferece Data Connectors e Indexers, que permitem extrair dados de múltiplas fontes (como diretórios locais, bases de dados ou APIs) e indexá-los com o formato e estrutura apropriados.

A recuperação da informação é realizada através dos Retrievers, que podem operar com base em métodos tradicionais (como BM25 TODO explicar isto na parte do RAG) ou com embeddings gerados por modelos de linguagem, permitindo uma recuperação densa e semântica. Após a recuperação, é possível utilizar Rankers para reordenar os documentos segundo critérios de relevância mais refinados. Em cenários em que se pretende gerar respostas completas com base na informação recolhida, os Generators entram em ação, utilizando modelos LLM para sintetizar respostas naturais e contextualizadas. Alternativamente, podem ser utilizados Prompt Nodes, que consistem em componentes configuráveis que enviam prompts para modelos de linguagem externos ou locais, oferecendo grande flexibilidade na forma como as instruções são formuladas.

Outro elemento importante da arquitetura são os Routers, que introduzem lógica condicional nas pipelines. Com eles, é possível definir fluxos alternativos com base nos resultados de componentes anteriores, facilitando a criação de pipelines mais inteligentes e adaptativas.

Todos estes componentes são integrados dentro de pipelines, que representam a sequência e lógica de execução entre os vários elementos da solução. As pipelines podem ser construidas de forma declarativa através de ficheiros YAML, ou programaticamente em Python, permitindo a configuração parâmetros, condições e integrações externas.

O Haystack disponibiliza pipelines template que são adequadas para uso em contexto RAG, como a *PredefinedPipeline.INDEXING* que importa ficheiros de texto, cria embeddings para cada um e armazena-os num *InMemoryDocumentStore*. Adicionalmente temos a *PredefinedPipeline.RAG* que usa os dados que foram previamente indexados para gerar respostas contextualizadas sobre essa informação.

==========

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

TODO referir o suporte com bases de dados vetoriais

TODO suporte/comunidade

Haystack:

Tem seu próprio servidor (Haystack REST API).

Ferramentas como Haystack UI para demonstrações.

Open source com foco corporativo.

LlamaIndex

TODO referir o suporte com bases de dados vetoriais

Spring AI

O projeto Spring AI ficou disponível a público no inicio de 2024. Surgiu com o objetivo de possibilitar o desenvolvimento de soluções orientadas à inteligencia artificial sem grandes complicações. Foi inspirado em alguns dos projetos que foram referidos anteriormente, como LangChain e LlamaIndex e criado com a crença de não restringir o desenvolvimento deste tipo de soluções apenas para o ecossistema Python, mas também para ecossistemas JVM com o Spring Framework.

Através da sua arquitetura modular, o Spring AI suporta a criação de fluxos RAG personalizados ou a utilização de fluxos template, isto através da Advisor API.

Dentro desta arquitetura, destaca-se o *QuestionAnswerAdvisor*, cujo objetivo principal é obter informação diretamente de uma base de dados vetorial, construindo uma resposta exclusivamente com os dados recuperados. Esta abordagem ignora o conhecimento prévio embutido no LLM utilizado, assegurando que a resposta é fundamentada apenas nas fontes de dados específicos.

Este advisor recebe por parametro a base de dados vetorial (vectorStore) e opcionalmente um SearchRequest que permite a configuração do grau de similariade de pesquisa. No snippet 3.1 é possível visualizar uma definição do grau de similaridade a 80%.

```
1 var qaAdvisor = new QuestionAnswerAdvisor(this.vectorStore,
2         SearchRequest.builder().similarityThreshold(0.8d).build());
```

Listing 3.1: Configuração do grau de similaridade do QuestionAnswerAdvisor

Além do *QuestionAnswerAdvisor*, o Spring AI disponibiliza também o *RetrievalAugmentationAdvisor*, que utiliza uma abordagem híbrida entre o conhecimento interno do LLM e a informação recuperada da base de dados vetorial. Neste caso, a resposta gerada pelo modelo pode incorporar tanto os dados recuperados quanto o conhecimento prévio do modelo base, permitindo um equilíbrio mais flexível entre a geração livre e a recuperação dirigida.

A utilização do *RetrievalAugmentationAdvisor* segue uma configuração semelhante, permitindo a parametrização da consulta à base de dados vetorial, mas com a diferença fundamental de que o LLM poderá usar o contexto adicional para enriquecer a resposta final, ao invés de se limitar exclusivamente aos dados recuperados.

TODO exemplo do RetrievalAugmentationAdvisor

Complementarmente, o Spring AI organiza as operações de RAG em componentes denominados *Modules*. Estes módulos são unidades configuráveis que encapsulam diferentes etapas de um fluxo de execução RAG, como a preparação de documentos (*Document Reader Modules*), a transformação de texto em embeddings (*Embedding Modules*), a pesquisa em bases vetoriais (*Retriever Modules*) e a geração de respostas (*LLM Modules*). Esta abordagem modular permite grande flexibilidade na definição de pipelines, podendo os desenvolvedores adaptar cada etapa do fluxo de acordo com os requisitos específicos da aplicação.

==========

TODO referir o suporte com bases de dados vetoriais

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: *arXiv preprint arXiv:2409.13707*. url: https://arxiv.org/abs/2409.13707.

Apêndice A

Appendix Title Here

Write your Appendix content here.