



**Politécnico  
Castelo Branco**

Escola Superior  
de Tecnologia

# **AlbiBooks**

## Chatbot

Ana Margarida Pereira Duarte N°20200721

Gabriel de Santana Pereira N°20220165

### **Orientadores**

Professor Doutor Arlindo Ferreira da Silva

Professora Doutora Ana Paula Neves Ferreira da Silva

Trabalho de Projeto apresentado à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Licenciados em Informática e Multimédia, realizado sob a orientação científica do Professor Adjunto Doutor Arlindo Ferreira da Silva e coorientação, da Professora Adjunta Doutora Ana Paula Neves Ferreira da Silva, do Instituto Politécnico de Castelo Branco.

**julho,2025**



## **Composição do Júri**

### Presidente do Júri

Doutor, Alexandre José Pereira Duro da Fonte

Professor Adjunto da Escola Superior de Tecnologia de Castelo Branco

### Orientador

Doutor, Arlindo Ferreira da Silva

Professor Adjunto da Escola Superior de Tecnologia de Castelo Branco

### Arguente

Doutor, Fernando Sérgio Barbosa

Professor Adjunto da Escola Superior de Tecnologia de Castelo Branco



## **Agradecimentos**

Em primeiro lugar, expressamos a nossa sincera gratidão aos nossos orientadores, Professor Arlindo Silva e a Professora Ana Paula Silva pela orientação e pelo apoio ao longo de todo o desenvolvimento deste projeto. A sua disponibilidade foi essencial para o sucesso deste trabalho.

Agradecemos também à Escola Superior de Tecnologia de Castelo Branco que nos acolheu e forneceu os recursos necessários para a realização deste projeto, bem como a todos os colegas e professores que contribuíram direta ou indiretamente para o nosso crescimento académico e profissional. Um agradecimento especial a Professora Ângela Oliveira pelos conselhos fornecidos.

Não poderíamos deixar de manifestar o nosso profundo reconhecimento às nossas famílias e amigos, cujo encorajamento, compreensão e paciência nos acompanharam em cada etapa deste percurso.

Por fim, agradecemos a todos aqueles que, de forma direta ou indireta, contribuíram para a concretização deste projeto.



## Resumo

O AlbiBooks é um projeto desenvolvido em duas fases, que visa desenvolver um *chatbot* inteligente para interação com a base de dados de uma biblioteca. O objetivo principal é democratizar o acesso aos serviços bibliotecários e torná-lo mais eficiente, permitindo consultas e recomendações de livros de forma prática e intuitiva.

Na primeira fase (Projeto I), foi realizada uma revisão sistemática da literatura, com base na metodologia PRISMA, centrada na aplicação de *chatbots* em bibliotecas. A nível técnico, foi implementado um *pipeline* do tipo *Retrieval-Augmented Generation* (RAG), com testes realizados sobre três modelos de linguagem locais: DistilGPT2, Llama-2-13b-chat-hf e Zephyr-7b-beta.

Na segunda fase (Projeto II), o sistema foi reestruturado através do uso de modelos de linguagem acessíveis via API (como o Gemini), e um novo modelo de *embeddings* (*mistral-embed*). Foi também introduzido um *dataset* recolhido por *web scraping*, e a arquitetura do pipeline foi otimizada para melhorar o desempenho e a escalabilidade. O relatório descreve esta evolução, os testes realizados e a comparação entre as abordagens adotadas.

## Palavras-chave

*Chatbot*; Inteligência Artificial; *Retrieval-Augmented Generation* (RAG); *Large Language Model* (LLM); Biblioteca.

## **Abstract**

AlbiBooks is a two-phase project aimed at developing an intelligent chatbot for interacting with a library database. The main objective is to democratize access to library services and make them more efficient by enabling users to search and receive book recommendations in a practical and intuitive manner.

In the first phase (Project I), a systematic literature review was carried out using the PRISMA methodology, focusing on the application of chatbots in libraries. On a technical level, a Retrieval-Augmented Generation (RAG) pipeline was implemented and tested using three local language models: DistilGPT2, Llama-2-13b-chat-hf, and Zephyr-7b-beta.

In the second phase (Project II), the system was restructured through the use of API-based language models (such as Gemini) and a new embedding model (mistral-embed). A dataset was also introduced via web scraping, and the pipeline architecture was optimized to improve performance and scalability. This report presents the project's evolution, the tests carried out, and a comparison between the two approaches.

## **Keywords**

Chatbot; Artificial Intelligence; Retrieval-Augmented Generation; Large Language Model; Libraries.

# Índice Geral

1.Introdução .....	1
1.1. Enquadramento .....	1
1.2. Objetivos.....	2
1.2.1 Objetivos de Projeto I.....	3
1.2.2 Objetivos de Projeto II.....	3
1.3. Planeamento.....	4
1.3.1 Planeamento de Projeto I.....	4
1.3.2 Planeamento de Projeto II.....	4
1.4 Alterações entre Projeto I e Projeto II .....	4
2.Inteligência Artificial.....	6
2.1. Machine Learning .....	6
2.1.1. Deep Learning.....	7
2.2. Inteligência Artificial Generativa.....	8
2.3. Chatbot .....	9
2.4. Chatbots com IA Generativa.....	10
2.5. Large Language Model.....	11
2.6. Modelos de Embedding .....	11
2.7. Retrieval-Augmented Generation (RAG) .....	12
3. Revisão Sistemática.....	13
3.1. Metodologia e Processo .....	13
3.1.1. Questões de Pesquisa.....	13
3.1.2. Estratégia de Pesquisa .....	14
3.1.3. Critérios de Inclusão .....	14
3.1.4. Critérios de Exclusão.....	14
3.1.5. Extração e Análise dos dados .....	15
3.2. Análise dos artigos .....	16
3.3. Discussão dos Resultados.....	27
4.Tecnologias e Ferramentas.....	29

4.1. Python.....	29
4.2 Google Colab .....	29
4.3 Hugging face.....	30
4.4 Transformers.....	31
4.5. Pytorch.....	31
4.6. LangChain.....	32
4.7. Facebook AI Similarity Search .....	33
4.8. Visual Studio Code .....	34
4.9. GitHub.....	34
4.10. Web Scraping .....	35
4.10.1. Web Scrapers .....	35
4.11. Modelo mistral-embed.....	36
5. PipeLine RAG.....	37
5.1. Exemplo .....	37
5.2. Implementação de LLM.....	41
5.2.1. Modelo distilbert/distilgpt2 .....	41
5.2.2. Modelo meta-llama/Llama-2-13b-chat-hf.....	41
5.2.3. Modelo HuggingFaceH4/zephyr-7b-beta.....	42
5.3. Resultados e Análise de Métricas .....	42
6. Transição e Início do Projeto II.....	50
6.2 Conclusões Projeto I.....	50
6.3 Alterações Implementadas em Projeto II .....	50
7.DataSet .....	52
7.1. Obtenção dos dados .....	52
7.1.1. Web Scraping .....	52
7.1.2. Indexação.....	53
8. Implementação do AlbiBooks .....	54
8.1. Arquitetura Geral do Sistema.....	54
8.2. Inicialização e Configuração .....	55

8.3. Camada de Recuperação de Informação .....	59
8.3.1. Carregamento dos Recursos de Recuperação (initialize_retriever) .....	59
8.3.2. Geração de <i>Embeddings</i> para Consultas (get_embedding) .....	60
8.3.3. Pesquisa por Similaridades dos Livros (search) .....	61
8.4. Camada de Geração de Respostas.....	63
8.4.1. Formatação do Contexto (format_context).....	63
8.4.2. Formatação do Histórico de Conversa (format_history) .....	64
8.4.3. Geração da Resposta com o Gemini (generate_gemini_response) .....	64
8.5. Função Principal de Resposta (answer_question).....	66
8.6. Interfaces de Interação .....	67
8.6.1. Interface de Consola (perguntar_ao_assistente).....	67
8.6.2. Interface Gráfica com Gradio (chatbot_interface).....	69
8.7. Configuração do Ambiente e Execução.....	71
9. Comparação entre Fases do Projeto.....	72
9.1 Visão Geral da Evolução da Arquitetura .....	72
9.2. Estratégias de Recuperação e Embeddings.....	74
9.3. Análise do Desempenho.....	75
9.3.1 Análise Qualitativa do Desempenho .....	75
9.3.2 Análise Quantitativa do Desempenho .....	77
9.4 Alterações nos Dados.....	79
9.4.1 Fonte, Volume e Qualidade dos Dados .....	79
9.4.2 Estratégia de Chunking e Formatação para Embedding .....	80
9.5 Ajustes no Fluxo de Interação com o Utilizador.....	80
9.5.1 Interface e Modo de Interação .....	80
9.5.2 Gestão do Histórico de Conversa.....	81
9.5.3 Orientação do Comportamento do <i>Chatbot</i> .....	81
9.6 Ferramentas e Bibliotecas Utilizadas.....	81
9.7. Justificações Técnicas para as Principais Decisões.....	83
9.7.1. Mudança de Modelos de <i>Embedding</i> .....	84

9.7.2. Mudança de LLMs .....	84
9.7.3. Separação do Processo de Indexação .....	84
9.7.4. Alteração da Fonte e Estrutura dos Dados para <i>Embeddings</i> .....	84
9.7.5. Introdução de Interface Gráfica (Gradio) e Gestão de Histórico .....	85
9.7.6. Abandono do Langchain e Implementação Direta da Lógica RAG .....	85
9.8. Conclusões .....	85
10. Conclusão .....	87
10.1. Trabalho Futuro .....	87
11. Referências .....	89

## Índice de Figuras

Figura 1 - Fluxograma PRISMA .....	15
Figura 2- Website da biblioteca da Universidade de Zayed (fonte: [36]) .....	19
Figura 3- Interface de utilizador do chatbot Alsha (fonte [36]).....	20
Figura 4- Website da Biblioteca da Faculdade de Lehman (fonte: [38]).....	23
Figura 5-Interface de utilizador do chatbot Lehman Lightning (fonte: [38]) .....	24
Figura 6- Diagrama RAG (adaptado de: [49]) .....	37
Figura 7-Dataset da Hugging Face (fonte: [50]) .....	38
Figura 8- Definição do Modelo de Embedding .....	38
Figura 9-Código para Carregar o Dataset .....	39
Figura 10-Código Criação da Base de Conhecimentos .....	40
Figura 11-Código para Divisão do Texto.....	40
Figura 12- <i>Prompt</i> usado pelos modelos.....	43
Figura 13- Tempo de Execução [78] .....	44
Figura 14- Uso da memória [78].....	45
Figura 15-Tempo de Execução do Modelo [79] .....	46
Figura 16-Uso de Memória do Modelo [79] .....	46
Figura 17-Tempo de execução do Modelo [80].....	47
Figura 18-Uso de Memória do Modelo [80] .....	48
Figura 19--Comparação do Tempo de Execução dos 3 modelos .....	48
Figura 20- Comparação do Uso de Memória dos 3 modelos .....	49
Figura 21-Arquitetura Geral do Sistema.....	55
Figura 22- Instalação de Dependências.....	56
Figura 23-Importação de Bibliotecas.....	56
Figura 24-Variáveis Globais .....	58
Figura 25- Carregando Recursos .....	60
Figura 26- Geração de embeddings.....	61
Figura 27-Pesquisa por Similaridade .....	62

Figura 28- Estruturar informações .....	63
Figura 29-Estrutura o Histórico da Conversa .....	64
Figura 30-Geração de Respostas.....	65
Figura 31- Fluxo de execução .....	67
Figura 32-Interação com o Utilizador.....	68
Figura 33- Interface Gráfica (Gradio).....	70
Figura 34-Execução .....	71
Figura 35-Interface (gradio) do AlbiBooks .....	71
Figura 36-Pergunta ID 25 (Inconsistência).....	76
Figura 37-Pergunta ID 82 .....	77
Figura 38- Tempo de Execução Projeto II .....	78
Figura 39-Uso de Memória Projeto II.....	78

## **Lista de Tabelas**

Tabela 1-Análise Artigos .....	16
Tabela 2- Diferenças entre Projeto I e Projeto II .....	72
Tabela 3- Alterações nos Dados entre Projeto I e Projeto II .....	79
Tabela 4-Diferenças do Conjunto Tecnológico de Projeto I e Projeto II .....	82



## **Lista de abreviaturas, siglas e acrónimos**

**ADN** – *Ácido Desoxirribonucleico*

**ANNs** – *Artificial Neural Networks*

**API** – *Application Programming Interface*

**BERT** – *Bidirectional Encoder Representations from Transformers*

**CNNs** – *Convolutional neural network*

**CPU** – *Central Processing Unit*

**DL** – *Deep Learning*

**DPO** – *Direct Preference Optimization*

**FAISS** – *Facebook AI Similarity Search*

**GPT** – *Generative Pre-trained Transformer*

**GPU** – *Graphics Processing Unit*

**IA** – *Inteligência Artificial*

**IBM** – *International Business Machines Corporation*

**JSON** – *JavaScript Object Notation*

**LLM** – *Large Language Model*

**ML** – *Machine Learning*

**MLPs** – *Multi-layer Perceptrons*

**NLP** – *Natural Language Processing*

**NLU** – *Natural Language Understanding*

**PE** – *Perceived Enjoyment*

**PEOU** – *Perceived Ease of Use*

**PU** – *Perceived Usefulness*

**PRISMA** – *Preferred Reporting Items for Systematic Reviews and Meta-Analyses*

**RAG** – *Retrieval-Augmented Generation*

**RAM** – *Random Access Memory*

**RL** – *Reinforcement Learning*

**RNNs** – *Recurrent Neural Networks*

**RPA** – *Automação Robótica de Processos*

**TAM** – *Technology Acceptance Model*

**TPU** – *Tensor Processing Units*

**URL** – *Uniform Resource Locator*

**VR** – *Virtual reality*

**VS Code** – *Visual Studio Code*

**WCAG** – *Web Content Accessibility Guidelines*

# 1.Introdução

As bibliotecas são centros de conhecimento fundamentais, mas a forma como os utilizadores interagem com os seus acervos continua, em muitos casos, pouco intuitiva e limitada. A necessidade de métodos mais eficientes de pesquisa e recomendação de livros torna-se cada vez mais evidente num mundo onde a informação está disponível instantaneamente em diversas plataformas digitais. Neste cenário, a Inteligência Artificial (IA) surge como um elemento transformador, possibilitando a criação de sistemas inteligentes que melhoram significativamente a experiência dos utilizadores.

O AlbiBooks é um projeto desenvolvido em duas fases, que procura explorar essa transformação, através de um *chatbot* baseado em IA capaz de interagir com bases de dados bibliográficas. O objetivo é proporcionar uma forma simples e natural de pesquisar livros, obter recomendações personalizadas e aceder a informações relevantes de uma biblioteca.

No Projeto I, foi realizada uma revisão sistemática da literatura e desenvolvida uma primeira versão funcional do sistema, utilizando um pipeline RAG com modelos de linguagem locais (*DistilGPT2*, *Llama-2* e *Zephyr*). Esta fase permitiu comprovar a viabilidade do conceito, mas revelou limitações importantes, como tempos de resposta elevados e um consumo elevado de recursos.

Em Projeto II, procedeu-se à reestruturação do sistema com recurso a modelos via API, uma nova estratégia de *embeddings* e a introdução de um *dataset* recolhido por *web scraping*. A nova arquitetura visa melhorar o desempenho, a escalabilidade e a qualidade das respostas.

Este relatório descreve o percurso desde a primeira fase até à versão otimizada do sistema, incluindo a recolha de dados, a implementação técnica, os testes realizados e a comparação entre as duas fases do projeto.

## 1.1. Enquadramento

O projeto proposto consiste no desenvolvimento de um *chatbot* (um *software* baseado em inteligência artificial capaz de manter uma conversa em tempo real por texto) com capacidade para interagir com a base de dados de uma biblioteca. Através de uma interface conversacional, o *chatbot* permitirá que os utilizadores façam consultas sobre o catálogo da biblioteca. Além disso, o *chatbot* poderá sugerir títulos relacionados, com base nas preferências indicadas pelo utilizador.

A motivação principal é oferecer um serviço prático e de fácil acesso, democratizando o uso de bibliotecas e melhorando a eficiência do atendimento e da procura de livros. Este serviço poderá beneficiar estudantes, professores, investigadores e a comunidade em geral, incentivando o uso das bibliotecas e facilitando o acesso ao conhecimento. Também é possível expandir o escopo do projeto para incluir funcionalidades adicionais, como a verificação de informações como disponibilidade de livros, localização dentro da biblioteca e horários de funcionamento.

O projeto proporcionará à biblioteca uma solução tecnológica moderna e eficiente, que pode ser adaptada a qualquer instituição, pública ou privada, que tenha um catálogo digitalizado. Esta abordagem não apenas simplifica a interação com os recursos disponíveis, mas também contribui para a transformação digital das bibliotecas, alinhando-se às necessidades de uma sociedade cada vez mais conectada. A implementação de um *chatbot* inteligente pode ainda reduzir significativamente a carga de trabalho da equipe da biblioteca, permitindo que estes se concentrem em tarefas mais complexas ou estratégicas, como a curadoria de novos materiais e o planejamento de atividades culturais.

Com o potencial de ser um recurso de amplo alcance, o *chatbot* tem como objetivo não apenas melhorar a experiência do utilizador, mas também incentivar a adoção de tecnologias inovadoras por instituições tradicionais, promovendo um equilíbrio entre tradição e modernidade no contexto educacional e informativo.

## 1.2. Objetivos

O objetivo deste trabalho é verificar a viabilidade do desenvolvimento de um *chatbot* para bibliotecas, utilizando técnicas de *Natural Language Processing* (NLP) para oferecer interações em linguagem comum. Esse *chatbot* deverá ser capaz de proporcionar uma experiência eficiente e amigável aos utilizadores, promovendo a acessibilidade aos serviços da biblioteca e modernizando a sua interação com o público.

A proposta é criar uma solução que sirva como um canal de comunicação entre a biblioteca e os seus utilizadores, permitindo a obtenção de informações relevantes sobre os serviços e recursos disponíveis. O *chatbot* será acessível através de plataformas digitais, proporcionando flexibilidade e conveniência aos utilizadores.

Outro foco importante será a realização de uma integração robusta com a base de dados da biblioteca, garantindo que o sistema seja capaz de oferecer respostas precisas e úteis. Embora a interação principal seja informativa, o desenvolvimento do

*chatbot* deverá também contemplar uma arquitetura extensível, que permita a adição de futuras funcionalidades, como notificações personalizadas ou recomendações baseadas no histórico de interações.

Por fim, serão conduzidos testes para identificar a combinação ideal de *Large Language Model* (LLM) e avaliar a experiência de uso do *chatbot*, com o objetivo de destacar pontos de melhoria e assegurar que a solução esteja alinhada de maneira eficiente e inovadora às necessidades do público-alvo.

### 1.2.1 Objetivos de Projeto I

O principal objetivo de Projeto I foi explorar a viabilidade da utilização de um *chatbot* baseado em inteligência artificial no contexto de bibliotecas. Para isso, procurou-se:

- Compreender o estado da arte sobre a aplicação de *chatbots* em bibliotecas;
- Analisar tecnologias como NLP, LLMs e pipelines RAG;
- Desenvolver um protótipo funcional utilizando modelos locais;
- Validar a capacidade do *chatbot* em fornecer respostas úteis com base em dados extraídos de uma biblioteca digital.

### 1.2.2 Objetivos de Projeto II

Com base nas limitações encontradas na fase inicial, o Projeto II teve como principais objetivos:

- Substituir os modelos locais por soluções baseadas em API (como Mistral e Gemini) para melhorar a escalabilidade, a qualidade das respostas e o tempo de execução;
- Implementar um sistema RAG mais eficiente com *embeddings* adequados ao cenário das bibliotecas;
- Comparar o desempenho entre as duas fases (modelo local vs API), avaliando tempo de resposta, precisão e utilização de recursos;
- Finalizar o desenvolvimento do *chatbot* com foco na integração e na usabilidade da aplicação.

### 1.3. Planeamento

Este projeto, desenvolvido ao longo do ano letivo 2024/2025, centrou-se na criação de um *chatbot* para bibliotecas. A fase inicial (Projeto I) teve como principal foco a investigação e o planeamento, com o objetivo de avaliar a viabilidade da proposta. Já na segunda fase (Projeto II), procedeu-se à implementação do *chatbot* propriamente dito, com otimizações baseadas nas conclusões da primeira fase.

#### 1.3.1 Planeamento de Projeto I

O Projeto I decorreu entre setembro de 2024 e janeiro de 2025, esta fase focou nas seguintes tarefas:

- Estudo do estado da arte;
- Revisão sistemática da literatura (PRISMA);
- Escolha das ferramentas (*LangChain*, *FAISS*, *HuggingFace*);
- Implementação de um pipeline RAG com modelos locais;
- Produção de um relatório técnico parcial.

#### 1.3.2 Planeamento de Projeto II

Por sua vez, o Projeto II decorreu entre fevereiro e junho de 2025 e incluiu:

- Testes comparativos entre modelos locais e APIs;
- Migração do sistema para uso de APIs (Mistral e Gemini);
- Avaliação do tempo de resposta e desempenho dos modelos;
- Ajustes na arquitetura RAG e *embeddings*;
- Redação e finalização do relatório de projeto.

### 1.4 Alterações entre Projeto I e Projeto II

O Projeto II deu continuidade ao trabalho desenvolvido anteriormente, no entanto incorporou várias alterações significativas com base nos resultados obtidos no Projeto I. As principais mudanças foram:

- **Tecnológicas:** Transitar de modelos locais para modelos via API permitiu maior fluidez, menor tempo de resposta e redução na complexidade computacional;

- **Arquitetura:** O pipeline RAG foi melhorado com o uso de *embeddings* mais adequados ao contexto e chamadas otimizadas à base de dados;
- **Avaliação:** Foram realizados testes sistemáticos que compararam o desempenho das duas abordagens, permitindo uma escolha fundamentada.
- **Escalabilidade:** A nova versão prepara o sistema para uso em ambiente real, com possibilidade de expansão e integração futura com plataformas como WhatsApp ou Telegram.

## 2. Inteligência Artificial

O conceito IA surgiu em 1956 na Conferência *Dartmouth* a qual tinha como objetivo explorar a ideologia de que "toda característica do aprendizado ou qualquer outra característica da inteligência pode, em princípio, ser tão precisamente descrita, que uma máquina pode ser feita para simulá-la" [1].

Daquele momento em diante tentou-se desenvolver maneiras para implementar comportamentos inteligentes nas máquinas, o maior desafio da IA foi resumido com a questão "Como fazer as máquinas compreenderem as coisas?" de Minsky autor do livro "*Semantic Information Processing*" em 1968 [2].

A IA é a tecnologia que permite que as máquinas se 'comportem' de forma semelhante aos humanos em termos de aprendizagem, desde a compreensão até à tomada de decisões [3].

Aplicações ou dispositivos que possuem IA podem aprender através de experiências e novas informações, possuem autonomia o que anula a necessidade de intervenção humana. Em 2024, o foco está direcionado na IA generativa, capaz de criar diversos tipos de conteúdo.

### 2.1. Machine Learning

*Machine Learning* (ML) é uma área da inteligência artificial cujo objetivo é permitir que as máquinas imitem a capacidade humana de aprender. Desta forma, permite que realizem tarefas de forma autónoma e, que com base na experiência e no aumento dos dados disponíveis, melhorem continuamente o seu desempenho e precisão.

Segundo [4], o sistema de aprendizagem de algoritmos automáticos é composto por três partes principais: (1) Processo de Decisão, os algoritmos são utilizados para fazer previsões ou classificações, tendo em conta dados de entrada que podem ou não ser rotulados; o algoritmo cria uma estimativa sobre os padrões presentes nos dados; (2) Função de Erro, avalia a precisão das previsões e, no caso de existirem exemplos conhecidos, compara os resultados para medir a precisão do modelo; (3) Processo de Otimização do Modelo, um método em que o algoritmo analisa o erro e otimiza o processo de decisão, reduzindo a ocorrência de erros bem com a sua gravidade.

Existem diversos tipos de modelos de ML que são definidos levando em consideração a presença ou ausência de influência humana. De acordo com [5], existem modelos de ML como: (1) Supervisionados, há uma rotulação prévia dos dados, bem como uma classificação por parte dos utilizadores, permitindo ao algoritmo

avaliar a precisão do seu desempenho; (2) Não Supervisionados, os dados são utilizados em bruto, ou seja, sem rotulação prévia. Cabe ao algoritmo identificar padrões nos dados sem intervenção humana; (3) Semi-Supervisionados, os conjuntos de dados utilizados combinam informações estruturadas e não estruturadas, permitindo que o algoritmo alcance conclusões de forma independente, ao mesmo tempo que aprende a rotular dados não rotulados; (4) Por Reforço, neste tipo de modelo, utiliza-se um sistema de “recompensas e punições” para que os algoritmos aprendam através das suas próprias experiências (tentativa e erro), recebendo feedback ao longo do processo.

### 2.1.1. Deep Learning

O *Deep Learning* (DL), uma subárea do *machine learning*, possibilita que modelos computacionais compostos por várias camadas de processamento aprendam representações de dados a vários níveis de abstração. Estes métodos contribuíram de forma significativa para o avanço no reconhecimento da fala, no reconhecimento visual de objetos, na deteção de objetos e em muitas outras áreas, como a descoberta de medicamentos e a genómica [6]. Um modelo de DL é essencialmente um programa que exhibe inteligência graças ao processamento de dados sofisticado. Muitos modelos de IA, como os LLMs, utilizam DL para compreender o contexto e criar respostas realistas.

Por norma, um programa informático requer dados de entrada precisos para gerar saídas corretas. No entanto, o DL consegue processar dados imprecisos e, ainda assim, produzir resultados relevantes. Por exemplo, um programa tradicional pode determinar se dois retratos digitais são exatamente iguais, enquanto um modelo de DL pode identificar semelhanças nos indivíduos retratados, mesmo que as imagens sejam diferentes [7].

Estes modelos dependem de grandes volumes de dados, bem como de um elevado poder computacional. O aumento destes recursos tornou-os mais sofisticados. Resumidamente, o DL é uma especialização do ML. Ambos permitem que o programa aprenda de forma autónoma a partir de um conjunto de dados, mas o DL vai além, sendo capaz, por exemplo, de aprender com dados não rotulados. Estes modelos são sempre construídos com recurso a redes neuronais, ao contrário dos modelos de ML, que nem sempre utilizam esta estrutura.

O DL já possui inúmeras aplicações, e novas utilizações continuam a ser descobertas constantemente, incluindo carros autónomos, criação de imagens, NLP e *Chatbots* de IA conversacional.

## 2.2. Redes Neurais

Redes Neurais são um dos modelos de aprendizagem automática. Estas tomam decisões de forma semelhante ao cérebro humano, imitando o funcionamento dos neurónios para identificar padrões, ponderar opções e chegar a conclusões. São compostas por camadas de nós (neurónios artificiais) e incluem uma camada de entrada, uma ou várias camadas ocultas e uma camada de saída. Os nós estão interligados, e cada um possui um peso e um limiar de ativação. Quando o output de um nó excede o seu limiar, ele é ativado e transmite os dados à camada seguinte; caso contrário, não transmite nada.

Para aprender e aumentar a sua precisão, as redes neurais dependem de dados de treino. Após serem ajustadas, tornam-se ferramentas poderosas para classificar dados com rapidez. Um exemplo conhecido é o algoritmo de pesquisa da Google [8].

As redes neurais, também chamadas de *Artificial Neural Networks* (ANNs), podem, segundo [9], ser classificadas em diferentes tipos, dependendo da sua finalidade.

- **Perceptron**: A forma mais antiga de rede neuronal, criada em 1958 por Frank Rosenblatt;
- **Multi-layer Perceptrons** (MLPs): Possuem camadas de entrada, ocultas e de saída. São compostas por neurónios sigmóides, sendo ideais para problemas não lineares. Este tipo de rede serve como base para visão computacional e outros modelos;
- **Convolutional Neural Network** (CNNs): Utilizadas para reconhecer imagens e padrões. Baseiam-se em álgebra linear, como multiplicações de matrizes, para identificar padrões;
- **Recurrent Neural Networks** (RNNs): Possuem *loops* de *feedback* como principal característica. São úteis para dados temporais, como previsões de séries temporais, incluindo o mercado de ações.

## 2.2. Inteligência Artificial Generativa

A IA Generativa [10], consegue criar desde textos originais a áudio, embora não seja um conceito atual tendo surgido na década de 1960 ganhou recentemente destaque devido a praticidade das interfaces que permitem criar em segundos conteúdos de alta qualidade incluindo representações realistas de pessoas.

Esta capacidade trouxe benefícios, como dobragens aprimoradas e conteúdos educativos mais detalhados, levantou também preocupações com os *deepfakes* e ataques cibernéticos, que exploram a capacidade de imitar pessoas de forma realista.

Os *transformers*, uma arquitetura essencial para modelos modernos de NLP, foram um dos principais responsáveis pelo avanço da IA generativa, permitindo treinar LLMs sem a necessidade de rotular todos os dados. Este conceito será detalhado no Capítulo 4.

Os progressos nos LLMs, com bilhões de parâmetros, iniciaram uma nova era, onde é possível gerar textos envolventes, imagens fotorrealistas e até criar conteúdos multimodais que combinam texto, gráficos e vídeo.

### 2.3. Chatbot

De acordo com [11], um *chatbot* é um programa que simula conversas humanas. Embora nem todos os *chatbots* utilizem inteligência artificial (IA), os mais modernos recorrem a técnicas de IA conversacional, como NLP, para compreender as questões dos utilizadores e automatizar respostas.

Os primeiros *chatbots* foram desenvolvidos em 1960 [12]. Por norma, os *chatbots* utilizam conjuntos de regras predefinidas para interagir com os utilizadores. As organizações podem utilizar estes programas para melhorar as suas comunicações, abrangendo desde fluxos de trabalho até ao atendimento ao cliente.

Os *chatbots* facilitam o acesso a informações [11], respondendo instantaneamente quando questionados, seja por texto, áudio ou ambos, sem necessidade de intervenção humana. Por isso é possível identificar diversos benefícios: (1) Respostas instantâneas e acessíveis 24/7; (2) Personalização da experiência do utilizador, incluindo recomendações; (3) Redução de custos e aumento da eficiência operacional, ao libertar recursos humanos de tarefas repetitivas; (4) Automação de fluxos de trabalho complexos, como o agendamento de reuniões ou a recolha de informações detalhadas.

Nas organizações, os *chatbots* são utilizados em tarefas como assistência ao cliente em tempo real, recomendação de produtos ou serviços e automação de lembretes e notificações.

Não obstante as vantagens fornecidas pelos *chatbots*, existem desafios que precisam ser enfrentados. Riscos de segurança e privacidade que podem expor informações confidenciais, especialmente nos modelos de IA generativa, riscos de

“alucinações”, casos onde o *chatbot* pode fornecer respostas incorretas ou irrelevantes e a complexidade de integração com os sistemas existentes como as Bases de Dados.

## 2.4. Chatbots com IA Generativa

Um exemplo notável de um *chatbot* com IA Generativa é o ChatGPT [13], um sistema avançado que não só simula interações humanas, mas também é capaz de produzir diferentes tipos de conteúdo com elevado nível de qualidade. Este tipo de *chatbot* representa um marco na evolução da inteligência artificial aplicada à comunicação, pois permite uma experiência de interação mais natural, intuitiva e eficaz. Os *chatbots* que utilizam IA Generativa destacam-se por oferecerem funcionalidades avançadas, entre as quais se inclui uma compreensão mais profunda e precisa da linguagem natural. Esta capacidade permite-lhes interpretar nuances de significado, contexto e até mesmo o tom das interações [11].

Por outro lado, quando comparados aos *chatbots* tradicionais, a superioridade dos sistemas baseados em IA Generativa torna-se evidente. Enquanto os *chatbots* convencionais dependem exclusivamente de respostas pré-programadas, limitando-se a executar scripts específicos ou a responder a perguntas simples, os *chatbots* com IA Generativa têm a capacidade de criar automaticamente respostas originais. Eles conseguem lidar com perguntas inesperadas e contextos complexos, gerando novos conteúdos com base no seu treino em LLMs. Em vez de se restringirem a simular respostas humanas, estes sistemas utilizam o conhecimento adquirido durante o treino para produzir interações mais criativas e adaptativas. Isso amplia significativamente o seu leque de aplicações e utilidades.

## 2.5. Large Language Model

Um *Large Language Model* (LLM) é um tipo de modelo de aprendizagem automática concebido para compreender e gerar texto semelhante ao humano através da aprendizagem de padrões estatísticos a partir de grandes quantidades de dados de texto. Estes modelos são utilizados em várias aplicações, incluindo *chatbots* e sistemas de recomendação.

Estes modelos são capazes de processar estatisticamente textos e prever as palavras seguintes numa sequência, através da análise de padrões estatísticos. Estes modelos são versáteis, pois são treinados com um vasto conjunto de dados e podem ser ajustados para tarefas específicas [14].

Exemplos de LLMs incluem o ChatGPT (OpenAi) e o Gemini (Google) [15]. Estes modelos podem ser treinados para desempenhar diversas tarefas, sendo amplamente utilizados com IA generativa. Quando assimilam uma pergunta, são capazes de produzir texto em resposta. Para o seu treino, podem ser utilizados conjuntos de dados extensos e complexos, incluindo linguagens de programação. De facto, alguns destes modelos fornecem suporte aos programadores na escrita de código, desenvolvendo funções ou completando programas a partir de um ponto inicial.

Os LLMs também podem ser aplicados em áreas como: (1) Análise de sentimentos; (2) Investigação de ADN e (3) Serviço de apoio ao cliente.

Os LLMs são ferramentas transformadoras no *Natural Language Processing* e não só, com aplicações que abrangem vários domínios. A sua capacidade de processar e gerar texto semelhante ao humano, associada à sua adaptabilidade a várias tarefas, torna-os inestimáveis para o avanço da tecnologia e da investigação. No entanto, continua a ser crucial compreender as suas limitações e garantir a sua utilização responsável [16].

## 2.6. Modelos de Embedding

Modelos de *embedding* são um tipo de LLM, utilizados para transformar dados complexos, como texto, imagens ou áudio, em representações vetoriais numéricas. Estas representações são fundamentais para que as máquinas consigam analisar, comparar e processar informação de forma eficiente [17].

Este tipo de modelo é uma componente essencial em sistemas *Retrieval-Augmented Generation* (RAG), uma vez que permite relacionar a consulta do utilizador com conteúdos previamente indexados.

De acordo com [18], os modelos de *embedding* podem ser projetados para lidar com vários tipos de dados, sendo os mais comuns: palavras, texto, imagens, áudio e gráficos. Estes modelos analisam o significado da consulta do utilizador, convertem-na num vetor e comparam esse vetor com os *embeddings* de um conjunto de dados mais vasto. Este processo facilita a identificação da informação mais relevante, mesmo quando não existem correspondências literais entre os termos usados na consulta e os dados armazenados.

## 2.7. Retrieval-Augmented Generation (RAG)

O conceito de *Retrieval-Augmented Generation* (RAG) surgiu com o objetivo de melhorar os modelos atuais de NLP. Neste sentido, pesquisadores da Meta AI [19] combinaram modelos de linguagem com fontes de conhecimento externas, treinando-os para produzir respostas mais precisas e contextualmente relevantes.

O RAG opera através de quatro etapas principais [20]: (1) Indexação: os dados são convertidos em representações numéricas e armazenados numa base de dados vetorial; (2) Recuperação: ocorre a seleção dos documentos mais relevantes, para uma consulta específica; (3) Aumento: etapa onde as informações recuperadas são integradas na consulta original; (4) Geração: produção de uma resposta baseada na consulta original e nos dados recuperados;

A otimização ocorre porque a abordagem RAG busca informações em fontes externas, diferentes daquelas usadas durante o treinamento. Isso melhora as respostas e amplia o conhecimento do modelo sem a necessidade de treiná-lo novamente.

Um LLM sem RAG, ao receber informações do utilizador, utiliza os seus dados de treinamento para gerar a resposta. Quando o LLM utiliza RAG, o utilizador faz uma consulta e o modelo busca informações a partir de uma base externa e confiável de conhecimento. Dessa forma, as respostas geradas passam a ser mais embasadas e confiáveis [19].

### 3. Revisão Sistemática

Neste capítulo, será apresentada uma revisão sistemática da literatura sobre a aplicação de *chatbots* em bibliotecas. O objetivo desta revisão é identificar e analisar os principais estudos, tecnologias e abordagens relacionadas ao uso de *chatbots* neste contexto, tendo como foco os seus benefícios, desafios e tendências. A revisão sistemática é fundamental para garantir que este trabalho se baseie nas pesquisas mais relevantes, proporcionando uma visão abrangente do estado atual da tecnologia e das suas implicações no setor das bibliotecas.

#### 3.1. Metodologia e Processo

Para conduzir a revisão sistemática da literatura sobre o uso de *chatbots* em bibliotecas, foi aplicada a metodologia PRISMA (Preferred Reporting Items for Systematic reviews and Meta-Analyses) [21]. A escolha da metodologia PRISMA justifica-se pela sua estrutura rigorosa, que garante transparência e replicabilidade na seleção e análise dos estudos.

Para a pesquisa de trabalhos e estudos já realizados foram utilizadas as bibliotecas científicas Scopus [22] e ACM [23]. Estas plataformas digitais disponibilizam o acesso a uma vasta coleção de recursos acadêmicos, incluindo artigos científicos, revistas, livros e outros materiais de pesquisa. Para a comunidade acadêmica, oferecem acesso a publicações científicas de diversas áreas do conhecimento, facilitando a pesquisa e a obtenção de dados atualizados e relevantes para projetos e estudos acadêmicos.

No contexto deste trabalho, para efetuar a pesquisa foram selecionadas palavras-chave que descrevessem o tema abordado, sendo estas “*chatbot*”, “*library*”, “*machine learning*” e “*artificial intelligence*”.

##### 3.1.1. Questões de Pesquisa

Com o objetivo de orientar o desenvolvimento deste projeto e proporcionar uma visão clara dos resultados esperados, foram formuladas as seguintes questões de pesquisa, que abordam temas específicos e essenciais:

1. Quais são as funcionalidades mais relevantes para que um *chatbot* atenda adequadamente às necessidades dos utilizadores em bibliotecas?
2. Quais tecnologias que incorporam Inteligência Artificial foram utilizadas?
3. Quais são os principais desafios na implementação de *chatbots* para o contexto de bibliotecas?

#### 4. Qual o tipo de interação pessoa-máquina foi utilizado?

##### 3.1.2. Estratégia de Pesquisa

A pesquisa foi efetuada nas bases de dados ACM e Scopus utilizando a expressão algorítmica "*chatbot*" AND "*library*" AND ("*Artificial Intelligence*" OR "*Machine Learning*"). O processo seguiu as etapas de identificação, triagem, elegibilidade e inclusão, com aplicação de critérios de inclusão e exclusão claros para assegurar a relevância dos artigos selecionados. Foram incluídos estudos publicados nos últimos 4 anos, focados no uso de *chatbots* em contextos bibliotecários utilizando IA (Inteligência Artificial), já os estudos com temas distintos foram excluídos. Ferramentas de apoio, como o Mendeley [24], foram utilizadas para organização e extração dos dados, garantindo uma análise precisa dos resultados.

Com a utilização da expressão algorítmica mencionada no campo de pesquisa, foram obtidos 42 resultados na SCOPUS e 76 resultados na ACM. Após a eliminação de 1 documento duplicado, passaram para a próxima fase de triagem 117 artigos.

##### 3.1.3. Critérios de Inclusão

Para garantir que a revisão sistemática aborde apenas estudos relevantes, os seguintes critérios de inclusão foram estabelecidos: (1) estudos que discutem a aplicação de *chatbots* em bibliotecas, sejam estas acadêmicas ou públicas, (2) artigos e estudos publicados desde 2020, garantindo a atualidade dos dados e tecnologias discutidas, (3) trabalhos publicados em inglês.

##### 3.1.4. Critérios de Exclusão

Para manter o foco e a pertinência dos estudos selecionados, os seguintes critérios de exclusão foram aplicados: (1) estudos que não abordem especificamente o contexto de bibliotecas e (2) estudos que não abordem *chatbots* de Inteligência Artificial.

### 3.1.5. Extração e Análise dos dados

Para aplicar os critérios de inclusão, foram lidos os títulos dos artigos. Após a aplicação dos critérios foram excluídos 101, de um total de 117 artigos. Para a fase de leitura de *abstract* foram, portanto, selecionados 16 artigos. Nesta fase, 2 artigos foram excluídos por falta da integralidade do texto, 1 por não estar relacionado com IA e 4 por não mencionarem nenhum tipo de ferramenta de IA diretamente. Por fim, 9 artigos foram selecionados para leitura e análise detalhada. A Figura 1 detalha o processo de inclusão e exclusão dos artigos.

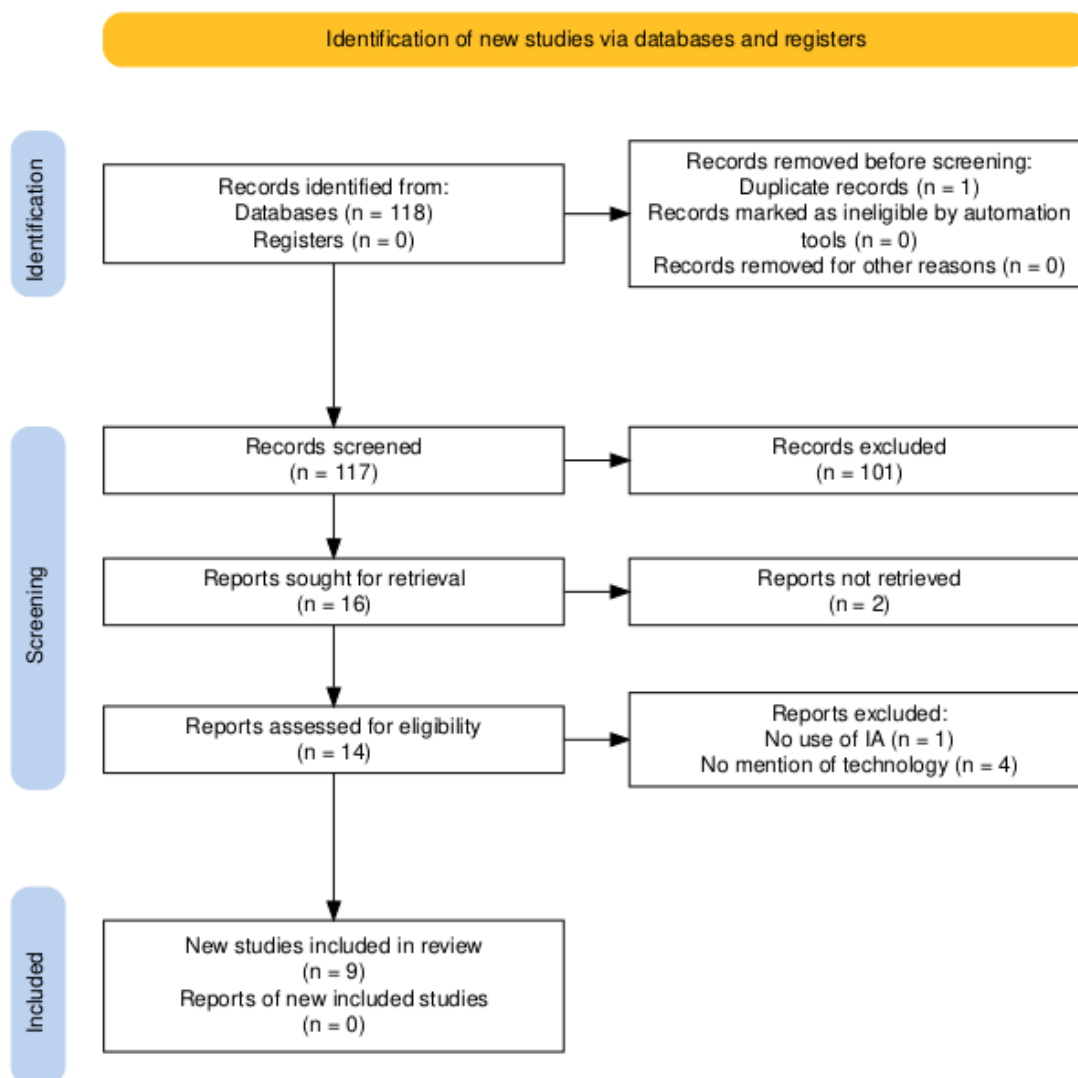


Figura 1 - Fluxograma PRISMA

### 3.2. Análise dos artigos

A Tabela 1 **Erro! A origem da referência não foi encontrada.**, apresenta uma análise dos diferentes estudos sobre tecnologias e ferramentas aplicadas a sistemas de *chatbot* e NLP. Os artigos abrangem o período de 2020 a 2024, refletindo a evolução das abordagens adotadas. A seguir, é feita a análise individual de cada artigo selecionado para leitura.

Tabela 1-Análise Artigos

Artigos	Ano de Publicação	Ferramentas/Tecnologias	Limitações Identificadas	Tipo de Interação
[25]	2024	Dialogflow, Kommunicate	Envolvimento limitado por parte dos utilizadores/ Subjetividade na interpretação	Não menciona/ Texto
[26]	2023	Python/Chroma/LangChain/ API do ChatGPT	Restrições de <i>Tokens</i> /Falta de acesso a dados em tempo real/ Alucinações	Website/Texto
[27]	2023	Unicamente teórico/ ChatGPT	Não menciona	Unicamente teórico
[28]	2024	Unicamente teórico/ ChatGPT	Relevância e precisão	Unicamente teórico

Artigos	Ano de Publicação	Ferramentas/Tecnologias	Limitações Identificadas	Tipo de Interação
[29]	2020	Rasa Stack	Dificuldade com Diálogos Complexos/ Falta de Documentação/ Dependência da Plataforma	Não menciona
[30]	2023	Unicamente teórico/ ChatGPT	Referenciação/ Alucinações/Dados desatualizados	Unicamente teórico
[31]	2022	Ivy Software	Licenças Limitadas/ Preocupações de Privacidade/ <i>Crawling</i> Limitado	Website/Texto
[32]	2024	BERT/Reconhecimento de Voz/ Flutter/Phyton/Flask/MySQL	Falta de um <i>dataset</i> amplo para treino/ Ausência de memória de contexto/Falta de suporte a múltiplos idiomas	Mobile/Texto/Voz
[33]	2023	IBM Watson Assistant/ Realidade Virtual/ Unity/ Reconhecimento de fala/Suporte de áudio	Não menciona	VR

O artigo [25] descreve o desenvolvimento de um *chatbot* de IA, destinado aos serviços de referência da Biblioteca Estadual de San José. O desenvolvimento foi efetuado com recurso ao *software* Dialogflow [34] e inclui elementos interativos e *widgets* de chat fornecidos pela plataforma Kommunicate [35]. O *chatbot* foi treinado com um conjunto de perguntas e palavras-chave, utilizando NLP e algoritmos de IA treinados, e foi programado para responder a questões básicas, estando disponível após o horário de funcionamento.

As interações com o *chatbot* foram monitorizadas durante 18 meses após o seu lançamento, e os dados recolhidos foram analisados utilizando as ferramentas analíticas oferecidas pelo Dialogflow e pela Kommunicate. Esta análise permitiu avaliar a eficácia do sistema. Durante o período inicial, foram detetadas apenas 44 interações; no entanto, os resultados demonstram que, ao fim de um ano, o número aumentou para 137 interações mensais. Na grande maioria, essas interações consistiam em perguntas sobre o horário de funcionamento e os recursos da biblioteca. Os autores observaram que apenas 10% dos utilizadores ultrapassaram essas interações básicas com o *chatbot*.

O artigo conclui que, mesmo com recursos limitados, as bibliotecas podem implementar *chatbots* de inteligência artificial com êxito.

No artigo [26], é analisado o projeto Alsha, que, segundo os autores, é uma iniciativa pioneira no contexto de *chatbots* em bibliotecas académicas, baseados na *Application Programming Interface* (API) do ChatGPT. O artigo explora o desenvolvimento do *chatbot*, recorrendo a ferramentas como Python, Chroma, LangChain e a API do ChatGPT, para se integrar nos serviços da Biblioteca da Universidade de Zayed, com o objetivo de criar um sistema funcional adaptado às necessidades dos seus utilizadores. Além disso, são traçados paralelos entre os *chatbots* baseados em regras, que são mais tradicionais, e os modelos generativos, argumentando-se que estes últimos oferecem mais vantagens em termos de flexibilidade e capacidade para lidar com consultas mais complexas. É apresentada uma revisão literária abrangente sobre o uso de *chatbots* em bibliotecas públicas e académicas, desde as primeiras implementações europeias até aos avanços das LLMs, e discute-se o potencial do ChatGPT para melhorar serviços como a busca de informações e o suporte a pesquisas. No entanto, ressalva-se a necessidade de diretrizes claras para o uso responsável da tecnologia. Na Figura 2, pode observar-se a página principal do Website da Biblioteca da Universidade de Zayed.

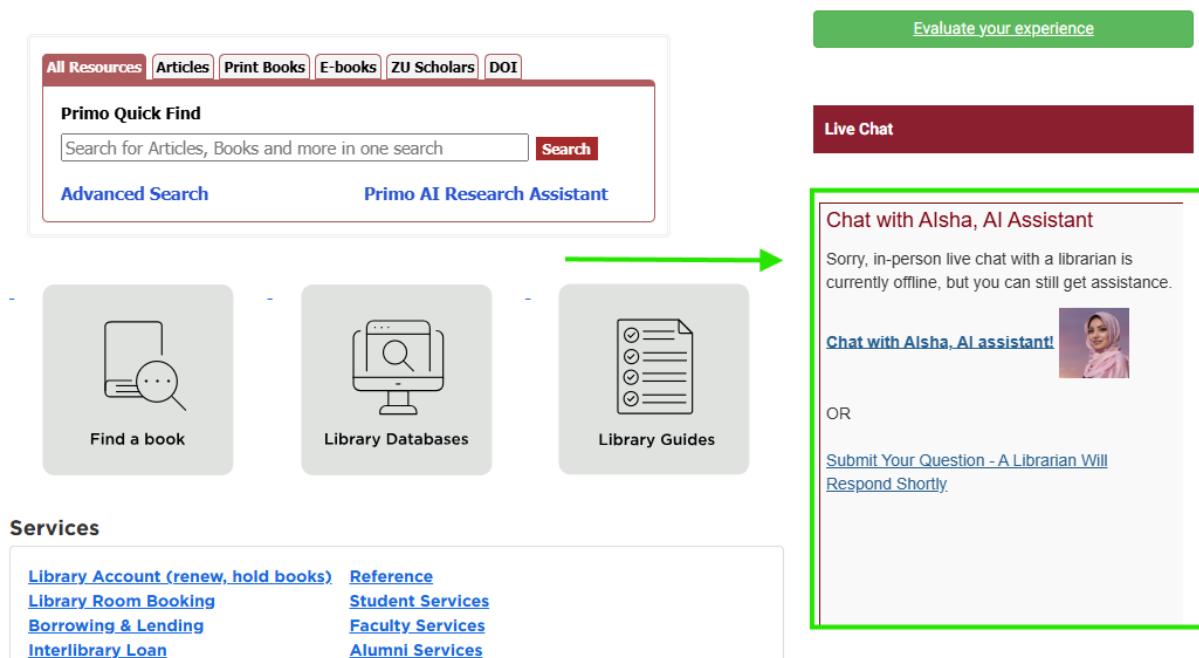


Figura 2- Website da biblioteca da Universidade de Zayed (fonte: [36])

A implementação do *chatbot* Aisha foi detalhada em etapas, incluindo coleta de dados, *fine-tuning* e desenvolvimento de uma interface. Apesar da componente técnica ter sido executada com sucesso, foram identificadas limitações significativas, como restrições de *tokens*, falta de acesso a dados em tempo real e o problema de “alucinações” do modelo. Essas limitações refletem desafios técnicos e éticos.

A Figura 3 representa a Interface do *chatbot* Aisha

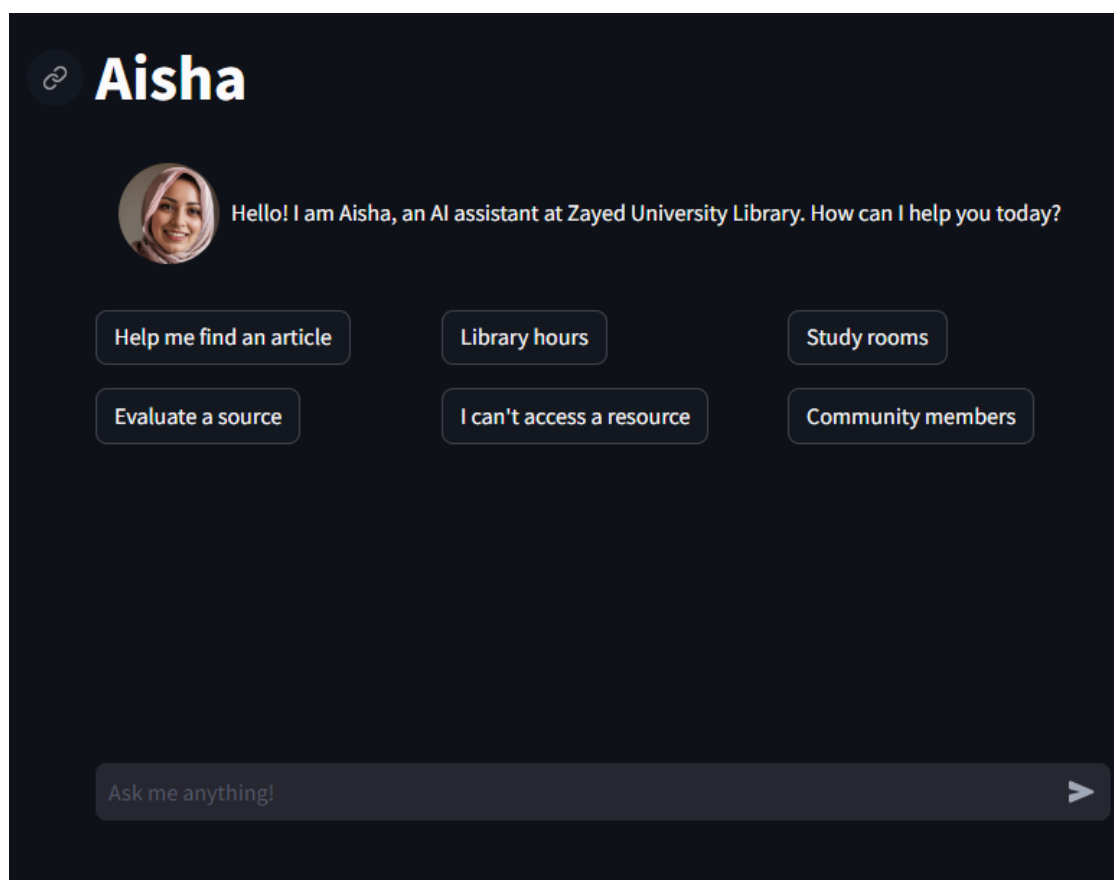


Figura 3- Interface de utilizador do chatbot Alsha (fonte [36])

O impacto potencial do *chatbot* Alsha no contexto de biblioteca foi reconhecido como transformador, possibilitando suporte personalizado e eficiente, sem comprometer o papel dos bibliotecários. No entanto, o artigo não explora o uso de *chatbots* em larga escala, ou seja, em múltiplas bibliotecas. Ainda assim, apresenta uma base sólida para avanços futuros e implementações semelhantes.

O estudo [27] apresenta os resultados de um teste simples realizado com o ChatGPT em fevereiro de 2023, utilizando *prompts* relacionados com serviços de referência bibliotecária e escrita académica. As respostas fornecidas foram comparadas com as de *chatbots* tradicionais, baseados em modelos de IA anteriores e implementados em bibliotecas. O autor contextualiza o estado atual dos *chatbots* em universidades nos Estados Unidos e relata que, até 2023, pelo menos quatro bibliotecas académicas haviam implementado algum tipo de *chatbot*.

Também são exploradas formas variadas de aplicação de *chatbots* em bibliotecas acadêmicas. Um exemplo notável citado é o da Biblioteca Nacional da Suécia, que desenvolveu um LLM para o contexto sueco [37]. Este modelo foi projetado para auxiliar investigadores na criação de *Datasets* a partir de materiais em língua sueca. O artigo destaca pontos relevantes sobre os benefícios dos *chatbots* no contexto de bibliotecas. Os *chatbots* permitem que os utilizadores interajam com as bibliotecas mesmo fora do horário de funcionamento, sendo possível conectar-se ao sistema da biblioteca e obter informações como as especificações dos livros e a sua disponibilidade. A capacidade de interagir em diferentes idiomas torna as bibliotecas acessíveis a uma comunidade mais diversificada. Embora o estudo apresente um panorama interessante sobre o potencial do ChatGPT em contexto de bibliotecas, ele limita-se ao uso direto do modelo, sem qualquer personalização conforme o contexto.

O artigo [28] apresenta uma revisão literária sobre o uso do ChatGPT, com foco no contexto das bibliotecas médicas. Discutem-se questões importantes relacionadas com a integração desta ferramenta nos serviços bibliotecários na área da medicina. O artigo apresenta uma revisão literária sobre o uso do ChatGPT, com foco no contexto das bibliotecas médicas. O texto aborda os desafios enfrentados pelas bibliotecas médicas tradicionais, principalmente em relação a motores de busca e bases de dados. Problemas como a relevância, a precisão e a acessibilidade das informações são destacados como limitações frequentes.

As funcionalidades do ChatGPT são descritas como relevantes no contexto das bibliotecas médicas, oferecendo suporte em tarefas como referências bibliográficas, auxílio em pesquisas, catalogação e classificação de materiais. Os autores destacam que o ChatGPT atua como uma ferramenta suplementar, complementando o conhecimento e as habilidades humanas, sem a intenção de os substituir.

A revisão realizada pelos autores baseou-se em palavras-chave como "ChatGPT and libraries" e "IA and libraries", evidenciando um escopo mais amplo que transcende o contexto médico, embora o foco principal do artigo permaneça na aplicação em bibliotecas de medicina. Além disso, são discutidas as limitações inerentes às ferramentas de inteligência artificial generativa. No campo médico, pequenos erros podem ter consequências graves, o que exige maior cautela na aplicação destas tecnologias. Embora o artigo ofereça uma discussão teórica relevante e um panorama geral sobre o potencial do ChatGPT em bibliotecas médicas, ele carece de exemplos práticos que validem as hipóteses levantadas. A ausência de resultados concretos limita a aplicabilidade imediata do estudo. Além disso, o foco no uso direto do ChatGPT,

sem explorar profundamente a ferramenta no que respeita à integração tecnológica, reduz o impacto da análise apresentada.

No artigo [29], o autor fornece uma visão clara sobre o potencial de evolução que os *chatbots* podem oferecer às bibliotecas. O autor refere que é possível melhorar a eficiência dos serviços e proporcionar aos utilizadores uma experiência mais interativa através de tecnologias *Open Source*, tendo como foco o Rasa Stack, uma *framework* de ML composta por dois principais componentes: o Rasa *Natural Language Understanding* (NLU), responsável pela compreensão da linguagem natural, e o Rasa Core, que consiste numa biblioteca baseada em ML responsável por decidir o comportamento e a resposta a fornecer.

O autor menciona ainda a utilização de NLP para entender as intenções dos utilizadores, permitindo que o *chatbot* responda de maneira mais assertiva. Propõe, assim, a utilização desta tecnologia para implementar *chatbots* nas bibliotecas. Para além dos benefícios, o artigo aborda também os desafios de implementação, como a necessidade de fornecer formação aos profissionais, uma vez que este é um fator crucial para que a utilização do *chatbot* atinja os seus objetivos de forma satisfatória. Ao abordar estas duas vertentes, o autor permite que os leitores compreendam tanto as oportunidades como as limitações da tecnologia.

Os autores do artigo [30] discutem as implicações do uso do ChatGPT e dos modelos de linguagem emergentes no contexto de instituições de ensino e bibliotecas, com foco nos seus impactos positivos e negativos.

Abordaram limitações do ChatGPT, como a falta de referências fidedignas, uma vez que este fornece informações sem citar as suas fontes. Casos em que a ferramenta gera respostas incorretas ou inventadas (alucinações); dados desatualizados, uma vez que o modelo foi treinado com dados disponíveis até 2021, o que limita a sua relevância em contextos que exigem informações atualizadas. O texto também aborda as questões éticas do uso do ChatGPT, como *copyright* e a possibilidade de uma utilização indevida.

Os autores sugerem que o ChatGPT pode ser útil para indicar livros relevantes, desde que essas recomendações passem por uma avaliação humana para garantir a precisão e relevância.

O artigo apresenta uma discussão teórica pertinente sobre as possibilidades e os desafios associados ao uso do ChatGPT, mas carece de exemplos práticos ou de

análises baseadas em implementações concretas. A falta de ferramentas ou metodologias concretas limita a aplicabilidade imediata do estudo.

Embora o artigo seja relevante para reflexões iniciais sobre o impacto de modelos de linguagem em bibliotecas e instituições académicas, ele se limita ao campo teórico. Estudos futuros poderiam beneficiar-se de análises práticas para explorar como essas tecnologias podem ser implementadas de maneira eficaz e ética.

O artigo [31] aborda a implementação do *chatbot* Ivy na Biblioteca Leonard Lief do Lehman College (EUA) [38]. Esta implementação teve a acessibilidade como uma das suas principais prioridades. Para tal, o design do *chatbot* foi desenvolvido de acordo com os padrões do *Web Content Accessibility Guidelines (WCAG) Version 2.1 AAA*. Foram utilizadas técnicas como esquemas de cores com contraste para garantir que a tecnologia possa ser utilizada de forma eficaz por qualquer utilizador. Na Figura 4, observa-se a página principal do Website da Biblioteca da Faculdade de Lehman.

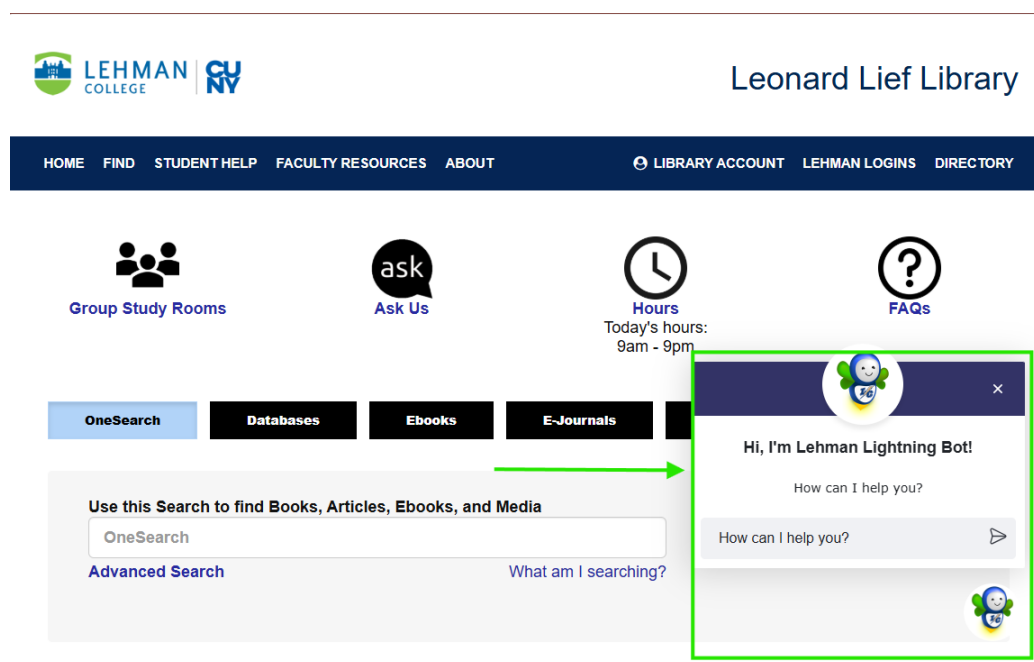


Figura 4- Website da Biblioteca da Faculdade de Lehman (fonte: [38])

O objetivo deste *chatbot* é responder a perguntas frequentes, permitindo assim que os bibliotecários possam focar a sua atenção em tarefas mais complexas. O Ivy utiliza NLP para responder de maneira mais eficaz às questões colocadas. Além de responder às questões dos utilizadores, o Ivy recolhe informações sobre estas solicitações para fornecer aos profissionais uma perceção sobre os conteúdos mais

procurados, o que permite uma gestão mais eficaz da biblioteca. A Figura 5, representa a Interface do *chatbot* Lehman Lightning.

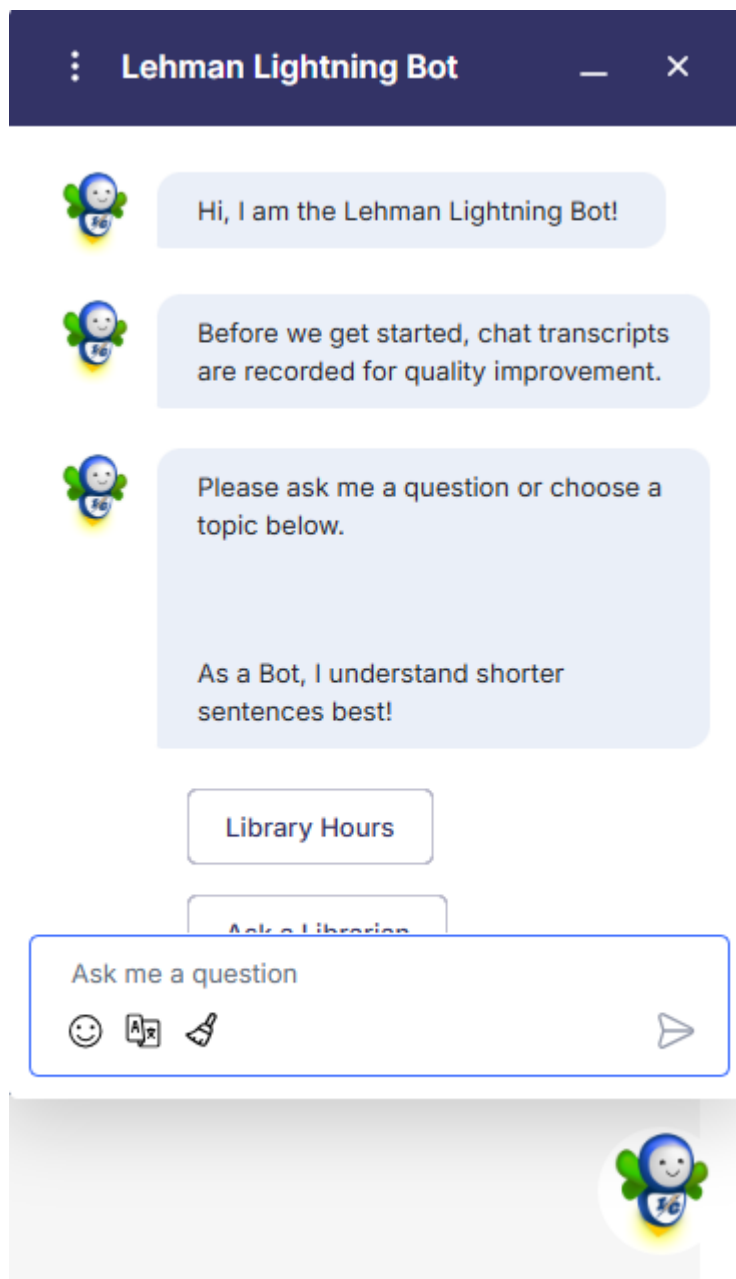


Figura 5-Interface de utilizador do chatbot Lehman Lightning (fonte: [38])

Como qualquer tecnologia, os *chatbots* precisam de manutenção para garantir o seu funcionamento. Para este fim, a biblioteca planeou monitorizar as interações com os utilizadores. O artigo destaca que, caso o Ivy não seja capaz de responder a determinada questão, um ticket com as informações do utilizador e a sua questão é enviado para os bibliotecários, para que estes possam prosseguir com a resposta. Por fim, a implementação do *chatbot* identificou também determinadas fragilidades no website da biblioteca, o que assinala que o Ivy pode ser utilizado como uma ferramenta para testar a usabilidade.

No artigo [32], um *chatbot* de inteligência artificial é proposto e implementado como uma aplicação móvel destinada a responder a questões relacionadas com a biblioteca e a realizar recomendações de livros.

O algoritmo *Bidirectional Encoder Representations from Transformers* (BERT) foi utilizado para classificar as intenções das mensagens dos utilizadores. O modelo foi treinado com elevada precisão (99,14%), demonstrando eficiência no reconhecimento e na categorização de intenções em mensagens de entrada.

O *chatbot* implementa um sistema de reconhecimento de voz que converte áudio em texto, permitindo interações por texto e voz.

A aplicação móvel foi desenvolvida em Flutter, garantindo compatibilidade com os sistemas Android e iOS. O backend foi implementado em Python, utilizando o *framework* Flask para gerir as comunicações via API. O armazenamento dos dados foi realizado em MySQL.

Perguntas não respondidas são armazenadas numa base de dados para serem analisadas posteriormente pelos administradores, que podem incorporar essas consultas ao modelo através de novos dados de treino.

O modelo BERT apresentou uma precisão de 99,14% na classificação de intenções, destacando a sua capacidade de identificar e processar mensagens com elevada fiabilidade.

A integração do modelo BERT com reconhecimento de fala adiciona um diferencial ao *chatbot*, tornando-o mais acessível e funcional para uma variedade de utilizadores. Além disso, a funcionalidade de captura de perguntas não respondidas é um aspeto prático que facilita o treino contínuo do modelo e aumenta a eficiência do *chatbot* ao longo do tempo.

No entanto, a falta de um *dataset* amplo para treino limita o modelo e o facto de o *chatbot* não possuir memória de contexto, dificulta a sua capacidade de lidar com referências a mensagens anteriores na conversa.

Em suma, o artigo fornece uma abordagem sólida e bem detalhada para o desenvolvimento de um *chatbot* em contexto de biblioteca utilizando tecnologias modernas como o BERT. Apesar de ser funcional, o sistema poderia beneficiar da inclusão de memória de contexto e suporte a múltiplos idiomas, sendo que este último não foi mencionado. Investigações futuras podem explorar o uso de *Reinforcement Learning* (RL) para melhorar a memória contextual e a implementação de conjuntos de dados mais diversificados para expandir a sua aplicabilidade.

O artigo [33] explora a implementação de um *chatbot* de bibliotecário virtual no contexto de realidade virtual (VR), integrando IA através do IBM Watson Assistant [39]. O objetivo principal é investigar a aceitação dos utilizadores, analisando fatores como utilidade percebida (PU), facilidade de uso percebida (PEOU), prazer percebido (PE) e curiosidade e intenção de uso (ITU), com base no modelo de aceitação tecnológica (TAM).

O estudo concentra-se num ambiente académico, avaliando como a combinação de IA e VR pode transformar o acesso a serviços de biblioteca, melhorar a experiência do utilizador e aliviar a carga de trabalho dos bibliotecários.

O *chatbot* foi desenvolvido utilizando o IBM Watson Assistant, uma ferramenta de IA conversacional, e foi treinado com 4 anos de dados de chat ao vivo. O *designing intents* (entender as intenções dos utilizadores e criar respostas para elas) e entidades foi elaborado para responder a questões rotineiras, como empréstimos, horários de funcionamento e outras informações, enquanto questões complexas eram encaminhadas para bibliotecários. As funcionalidades incluem suporte de áudio, reconhecimento de fala e avatares 3D criados em Unity [40] para melhorar a interação

Um grupo de mais de 60 pessoas (estudantes, professores e funcionários) interagiu com o *chatbot* num ambiente de realidade virtual. Após a experiência, foi preenchido um questionário baseado no TAM. Os resultados mostram que a PU foi fortemente influenciada pela PEOU e PE, especialmente entre estudantes; a PEOU teve um impacto mais relevante entre estudantes do que entre professores e funcionários. A curiosidade impactou a PU para os estudantes, mas não para os demais grupos.

Este artigo apresenta várias limitações. A amostra é diminuta, o que pode não refletir plenamente a diversidade dos utilizadores. Além disso, o contexto específico deste estudo limita a possibilidade de generalização para outros ambientes.

O artigo destaca-se pela relevância do estudo sobre a integração de IA e VR em bibliotecas acadêmicas, com uma análise detalhada dos fatores que influenciam a aceitação do *chatbot*. Contudo, faltam dados comparativos que evidenciem os benefícios em relação aos sistemas tradicionais. Para além disso, a dependência de ferramentas proprietárias, como o IBM Watson, pode limitar a replicação e adaptação noutras instituições.

Este trabalho oferece uma perspetiva valiosa sobre como os *chatbots* em VR podem melhorar os serviços de biblioteca.

### 3.3. Discussão dos Resultados

A análise dos artigos sobre a implementação de *chatbots* em bibliotecas revela avanços significativos na aplicação da inteligência artificial para melhorar os serviços de referência, as interações com os utilizadores e a acessibilidade. Mesmo com diferenças em termos de tecnologias e ferramentas, alguns padrões destacam-se, o que permite comparar as diferentes abordagens, ferramentas, tecnologias e resultados obtidos.

As ferramentas empregadas para o desenvolvimento dos *chatbots* variam amplamente, mas uma tecnologia em questão emerge entre as outras. Apesar de ser novidade no mercado da inteligência artificial, o ChatGPT é referenciado em 4 dos 9 artigos ([26] [27] [28] [30]). Este número expressa a relevância e a importância que tem a ferramenta em questão, evidencia também a sua flexibilidade. Mesmo assim, as ferramentas utilizadas para o desenvolvimento dos *chatbots* referenciados neste trabalho variam amplamente. O artigo que explora o uso do Rasa Stack [29] demonstrou como bibliotecas podem implementar tecnologias de IA de forma acessível. O uso do IBM Watson Assistant integrado em ambientes de VR [33] proporcionou uma experiência imersiva e inovadora no contexto do estudo.

Os estudos enfatizaram os impactos positivos dos *chatbots*, incluindo a acessibilidade ampliada, permitindo interações com as bibliotecas mesmo fora dos horários de funcionamento; personalização do serviço e engajamento em diferentes tipos de plataformas, sejam elas por websites, mobile ou VR.

Embora os avanços sejam notáveis, os estudos apontaram desafios comuns, como falta de memória contextual em casos em que o *chatbot* processa as mensagens de forma isolada; limitações técnicas e éticas e dependência de ferramentas proprietárias.

Alguns estudos oferecem resultados práticos que evidenciam a eficácia dos *chatbots*, seja o caso do aumento da utilização por parte dos utilizadores [25] ou da precisão na classificação dos *intents* [32]

As aplicações futuras devem procurar resolver algumas limitações. A adoção de técnicas avançadas, como RAG, para implementar memória de contexto poderia melhorar a experiência e a continuidade das conversas. A inclusão de suporte multilíngue aumentaria a acessibilidade para diferentes comunidades, a expansão dos canais de interação para plataformas amplamente utilizadas, como o Telegram [41] e o WhatsApp[42], bem como a integração com assistentes de voz, aumentaria o alcance e a conveniência para os utilizadores.

Além disso, a análise da literatura revelou que nenhum dos artigos analisados utilizou RAG na implementação de *chatbots* para bibliotecas. No entanto, observou-se que algumas das limitações identificadas nos estudos poderiam ser mitigadas com o uso desta abordagem, tais como: (1) ausência de memória contextual, que impede o *chatbot* de compreender o histórico da conversa; (2) alucinações em modelos generativos, em que o *chatbot* gera respostas incorretas por não ter acesso a dados atualizados; (3) dificuldade em fornecer respostas precisas sobre acervos bibliográficos em tempo real, devido à ausência de integração com bases externas.

Com base nesses desafios, optou-se por incorporar RAG no projeto AlbiBooks como solução para melhorar a recuperação e geração de respostas. Ao integrar um mecanismo de busca em bases de dados antes da geração de texto, o *chatbot* poderá fornecer respostas mais precisas e atualizadas, reduzindo o risco de informações desatualizadas ou imprecisas.

Em conclusão, os *chatbots* representam um avanço significativo na modernização das bibliotecas, com benefícios claros para a acessibilidade e a eficiência. No entanto, a implementação eficaz requer atenção às limitações técnicas, aos custos e às questões éticas. O futuro desses sistemas depende de melhorias em memória contextual, integração com sistemas existentes e personalização para diferentes contextos. Com investigação contínua, os *chatbots* podem modernizar as bibliotecas, tornando-as mais tecnológicas e acessíveis.

## 4. Tecnologias e Ferramentas

Para a implementação deste projeto, planeja-se o uso de várias tecnologias e ferramentas. Neste capítulo estas serão apresentadas, assim como a justificativa da escolha de cada uma.

### 4.1. Python

Python é uma linguagem de programação interpretada de alto-nível orientada a objetos com semântica dinâmica. É uma linguagem atrativa para o desenvolvimento rápido de aplicações. A sintaxe de Python é fácil de aprender e de manter. A linguagem suporta módulos e pacotes, o que encoraja a modularidade da programação e a reutilização de código. O interpretador e o número extensivo de bibliotecas estão disponíveis de forma gratuita [43].

Desenvolver aplicações de inteligência artificial é uma atividade complexa e que demanda tempo. Entretanto há várias bibliotecas para este fim que facilitam o desenvolvimento de aplicações de IA e que são compatíveis com Python. As bibliotecas são conjuntos de códigos pré-escritos que podem ser reutilizados ao serem importados e ao acionarem as suas funções. Este é um dos motivos pelo qual Python é amplamente utilizado pela comunidade de desenvolvimento de IA [44]. A versão do Python utilizada é a 3.11.11.

### 4.2 Google Colab

Um projeto com Inteligência artificial e LLM requer uma potência computacional significativa [45]. Levando em consideração este requisito e a imprescindibilidade de trabalhar em colaboração, foi escolhido trabalhar com o Google Colab [46] no que se refere a parte prática deste projeto. É uma ferramenta gratuita do Google, baseada no projeto de código aberto Jupyter [47], que provê recursos computacionais, como a Unidade de Processamento Gráfico (GPU), a Unidade de Processamento Tensor (TPU) e bibliotecas da linguagem Python. Outro ponto positivo desta ferramenta é a possibilidade de ser utilizada através do próprio navegador. Os ficheiros, ou “notebooks”, são salvos no Google Drive e podem ser compartilhados com outros utilizadores. Inclusive, os ficheiros referente a parte prática deste projeto pode ser acedido em:

<https://drive.google.com/drive/folders/1LvSxQAQj8szOxQ4BrfVmqNoEmslteLWY?usp=sharing>

Apesar de ser uma ferramenta gratuita, possui as suas limitações. Os recursos disponíveis são limitados e dependem da demanda da ferramenta, o que significa que podem não estar sempre disponíveis ou podem ser limitados em momentos de maior utilização. Além disso, há opções pagas que oferecem um maior número de recursos, como mais tempo de execução, mais memória RAM e mais espaço de armazenamento. Para este projeto apenas foram utilizados recursos gratuitos, com as opções de Unidade Central de Processamento CPU e GPU. Além disso, esta ferramenta fornece o Gemini, um *chatbot* de IA generativa, de forma integrada para ajudar na construção e na depuração do código.

Com o Google Colab, é possível criar documentos compostos por células, nas quais cada uma pode conter código ou texto (em formato Markdown). Imagens podem ser incorporadas tanto como parte da documentação quanto como saída gerada pelo código. Além disso, ele elimina a necessidade de instalações e configurações complexas, pois, como já foi referido, corre diretamente no navegador. Inclui diversas bibliotecas de Python pré-instaladas [48].

### 4.3 Hugging face

Hugging Face é um centro de modelos de IA de última geração. É conhecido principalmente pela sua vasta gama de modelos de código aberto baseados em *transformers* que se destacam no NLP, visão computacional e tarefas de áudio. A plataforma oferece diversos recursos e serviços destinados a programadores, investigadores, empresas e qualquer pessoa interessada em explorar modelos de IA para os seus próprios casos de utilização.

A plataforma oferece uma ampla gama de recursos, destacando-se duas categorias principais (1) Modelos: A Hugging Face aloja um vasto repositório de modelos de IA pré-treinados que são facilmente acessíveis e altamente personalizáveis. A plataforma é orientada para a comunidade e permite que os utilizadores contribuam com os seus próprios modelos, fornecendo assim uma seleção diversificada e em constante crescimento; e (2) *Datasets*: A Hugging Face tem uma biblioteca de milhares de conjuntos de dados que pode utilizar para treinar, avaliar e melhorar os seus modelos. Estes variam de *benchmarks* de pequena escala a conjuntos de dados maciços do mundo real que abrangem uma variedade de domínios, como dados de texto, imagem e áudio. Tal como um centro de modelos, a plataforma apoia as contribuições da comunidade e fornece as ferramentas necessárias para pesquisar, transferir e utilizar dados nos seus projetos de aprendizagem automática.

Foi nesta plataforma que foram obtidos, o tutorial [49], no qual este projeto se baseou, o *dataset* utilizado [50] e os LLMs implementados nesta fase prática do projeto.

## 4.4 Transformers

*Transformers* é a nova arquitetura de rede neural, simples, mas poderosa, introduzida pelo Google Brain em 2017 [51]. Baseia-se no mecanismo de atenção em vez da computação sequencial que normalmente se observa nas RNNs [52].

A biblioteca *Transformers*, desenvolvida pela Hugging Face, é uma ferramenta de código aberto que fornece uma ampla gama de modelos pré-treinados focados principalmente em NLP. Estes modelos abrangem várias tarefas em modalidades como o processamento de linguagem natural, a visão por computador, o áudio e a aprendizagem multimodal. A utilização de modelos de código aberto pré-treinados pode reduzir os custos, reduzir o tempo necessário para treinar modelos do zero e proporcionar um maior controlo sobre os modelos que implementa [53].

A biblioteca é construída com foco em desempenho, usabilidade e acessibilidade. É compatível com Pytorch[54] e TensorFlow,[55] bibliotecas populares para computação numérica e desenvolvimento de modelos de ML e DL. Também há suporte para ajuste fino de modelos *transformers* em uma ampla gama de tarefas de NLP. A biblioteca fornece ferramentas para processar os dados específicos e prepará-los para o modelo, que, depois de prontos, pode ser feito o ajuste fino do modelo [48][56].

## 4.5. Pytorch

Pytorch é uma biblioteca de ML de código aberto para Python, usada para aplicações como NLP. É conhecida pela facilidade de uso, visualização de gráficos computacionais, construção de gráficos dinâmicos e pelo uso contínuo de GPUs para computação. Esses recursos tornam-na uma escolha popular entre pesquisadores e desenvolvedores no campo de DL [54].

O PyTorch distingue-se pelo seu excelente suporte para GPUs e pela sua utilização de autodiferenciação em modo inverso, que permite que os gráficos de computação sejam modificados em tempo real. Isto torna-a uma escolha popular para experimentação e prototipagem rápida [57].

Como referenciado anteriormente, trabalhos com o uso de modelos de DL exigem grande capacidade computacional, e o ideal neste caso é utilizar hardware que seja capaz de satisfazer esta necessidade. No caso do uso de GPUs, neste trabalho, a

utilização desta biblioteca é indispensável, embora também seja possível usá-la em situações em que se utilize apenas a CPU, com as devidas reduções de desempenho.

## 4.6. LangChain

LangChain é um *framework* de código aberto projetado para simplificar a criação de aplicações usando LLMs. Disponível em bibliotecas baseadas em Python e JavaScript [58], o LangChain oferece ferramentas e APIs que facilitam o processo de construção de aplicações orientadas por LLMs, como *chatbots* e agentes virtuais.

Este *framework* serve como uma interface genérica para praticamente qualquer LLM, fornecendo um ambiente de desenvolvimento centralizado para construir aplicações de LLM e integrá-las com fontes de dados externas e fluxos de trabalho de *software*. Isto permite que desenvolvedores de IA criem aplicações baseadas na combinação de LLMs, como o GPT-4, com fontes externas de computação e dados.

O LangChain segue um pipeline geral onde, um utilizador faz uma pergunta ao modelo. A representação vetorial da pergunta é usada para realizar uma busca de similaridade em uma base de dados vetorial, de onde as informações relevantes são recuperadas. Essas informações são então alimentadas ao modelo de linguagem, que gera uma resposta ou executa uma ação [59].

Este processo é particularmente relevante para a técnica de RAG, que combina a capacidade de geração dos LLMs com a recuperação de informações externas. O RAG permite que os modelos acessem e incorporem conhecimentos específicos ou atualizados que podem não estar presentes nos seus dados de treinamento originais [60].

O LangChain pode facilitar a maioria dos casos de uso para LLMs e NLP, como *chatbots*, busca inteligente, resposta a perguntas, serviços de sumarização ou até mesmo agentes virtuais capazes de automação de processos robóticos. Além disso, embora os modelos de base (como os que alimentam os LLMs) sejam pré-treinados em conjuntos de dados massivos, eles não são oniscientes. Se uma tarefa específica requer acesso a informações contextuais específicas, como documentação interna ou conhecimento de domínio, os LLMs precisam ser conectados a essas fontes de dados externas [61].

Essa capacidade de integração com fontes externas é crucial, pois permite que as aplicações baseadas em LLMs reflitam uma consciência em tempo real de eventos atuais ou informações específicas do domínio, superando as limitações temporais dos dados de pré-treinamento do modelo.

## 4.7. Facebook AI Similarity Search

FAISS é uma biblioteca de código aberto projetada para busca de similaridade eficiente e agrupamento de vetores densos. Ela oferece recursos avançados que a tornam especialmente útil para lidar com grandes conjuntos de dados multimídia e permite aplicações como sistemas de recomendação e busca de imagens [62][63].

A biblioteca destaca-se pela sua capacidade de construir índices e realizar buscas com notável velocidade e eficiência de memória. Esta biblioteca utiliza algoritmos sofisticados, como agrupamento *k-means* e quantização de produtos, para organizar e recuperar vetores de forma eficiente, garantindo que as buscas de similaridade sejam rápidas e precisas [62].

As características principais desta biblioteca são (1) escalabilidade, já que foi projetada para lidar com grandes vetores de dados; (2) velocidade, pois tem estrutura de dados e algoritmos otimizados, e permite o uso de GPU, o que é crucial para aplicações que precisam de respostas rápidas, como *chatbots*; (3) precisão, a biblioteca oferece flexibilidade, balanceando velocidade e precisão; (4) versatilidade, FAISS é capaz de tratar diferentes tipos de dados e converte-os em vetores de representação.

A FAISS se destaca como uma ferramenta poderosa para acelerar pesquisas de similaridade de vetores densos, oferecendo variedade de funcionalidades e otimizações para operações de busca eficientes e eficazes em diversos cenários de aplicação.

## 4.8. Visual Studio Code

O Visual Studio Code (VS Code) é um editor de código fonte, gratuito e de código aberto desenvolvido pela Microsoft. O VS Code oferece funcionalidades simples como edição de código com suporte a várias linguagens de programação, terminal integrado e controle de versão.

A princípio, é uma ferramenta simples; entretanto, disponibiliza uma extensa loja de extensões, o que permite adicionar funcionalidades ao editor de código [64], [65], [66].

## 4.9. GitHub

Em termos simples, GitHub é um website e um serviço baseado em nuvem projetado para ajudar programadores a armazenar, manter, rastrear e controlar as alterações feitas no código de maneira eficiente. Ele utiliza o sistema de controle de versões Git [67], que permite que programadores registrem e gerenciem historicamente todas as modificações realizadas em seus projetos. Isso é especialmente útil para rastrear alterações, reverter mudanças indesejadas e colaborar com outras pessoas de forma integrada.

O GitHub oferece a hospedagem de repositórios Git diretamente na nuvem, o que elimina a necessidade de servidores locais para armazenamento de código. Essa funcionalidade facilita o acesso remoto, garantindo que indivíduos ou equipes de desenvolvimento possam trabalhar nos mesmos projetos de qualquer lugar do mundo. Além disso, a plataforma inclui recursos adicionais, como a criação de *issues* para rastreamento de tarefas, revisão de código por meio de pull requests e integração com outras ferramentas de desenvolvimento, aumentando significativamente a produtividade e a organização do trabalho em equipe.

Para indivíduos, o GitHub funciona como uma espécie de portfólio digital, permitindo compartilhar projetos com a comunidade global, receber feedback e até demonstrar habilidades para potenciais empregadores. Já para equipes de desenvolvimento, a plataforma é um ponto central para colaboração e gestão, tornando os processos mais transparentes e organizados. Dessa forma, o GitHub não é apenas uma ferramenta de controle de versões, mas também uma solução completa para o desenvolvimento moderno de *software* [68].

Além disso, todo o código referente ao pipeline está disponível em: <https://github.com/gabrielPereira-ipcb/AlbiBooks>

## 4.10. Web Scraping

*Web scraping* é uma técnica computacional utilizada para extrair dados de websites. Este processo envolve a navegação por páginas web, a extração de dados relevantes e o seu posterior armazenamento. Segundo [69], o processo inicia-se com o envio de uma solicitação ao servidor do website; em resposta, o servidor devolve os dados da página solicitada, como texto e imagens. As ferramentas de *web scraping* analisam então esses dados, podendo ser programadas para localizar e extrair informações específicas. Os dados extraídos são, por fim, armazenados num ficheiro ou numa base de dados.

É possível extrair qualquer tipo de dado, desde mecanismos de pesquisa até informações governamentais [70]. A maioria dos websites disponibiliza os seus dados para *scrapers*, bem como para outras formas de recolha automatizada de informação. No entanto, a acessibilidade desses dados nem sempre é garantida. Muitas vezes, os dados não estão disponíveis no formato desejado ou o website pode não fornecer uma API que facilite a sua extração. Em situações como estas, o *web scraping* torna-se essencial para obter as informações pretendidas, contornando limitações impostas pela estrutura ou pela política do site. Ainda assim, importa referir que a recolha automatizada de dados pode levantar questões legais e éticas, sendo necessário avaliar cuidadosamente os termos de utilização dos websites em causa.

### 4.10.1. Web Scrapers

Existem diversos formatos de *Web Scrapers*. Em [69] são referidos alguns dos principais:

- **Extensões de navegador:** Complementos fáceis de utilizar, como o *Web Scraper* para o Chrome [71], que permitem extrair dados diretamente do navegador através de uma interface simples. Estão, contudo, limitadas a tarefas de pequena escala e ao próprio navegador.
- **Raspagem programática (com *script*):** Envolve o desenvolvimento de código para recolha de dados da web, sendo ideal para tarefas complexas e em larga escala. Oferece flexibilidade, mas requer conhecimentos de programação, bem como manutenção frequente.
- **Ferramentas dedicadas de raspagem:** Plataformas como o *Octoparse* [72], permitem configurar raspagens complexas com pouca ou nenhuma programação. São práticas e escaláveis, embora possam implicar custos elevados.

- **Automação Robótica de Processos (RPA):** Utiliza robôs de software que imitam ações humanas para extrair dados da *web*, sem necessidade de programação. Ferramentas como o *Automate* [73], integram-se facilmente com outros sistemas e são eficazes para tarefas repetitivas.

#### 4.11. Modelo mistral-embed

Para selecionar o modelo de *embedding* a utilizar neste projeto realizou-se uma breve pesquisa com o objetivo de identificar qual o modelo gratuito que apresentava as características mais adequadas às necessidades do sistema. Optou-se pelo modelo ***mistral-embed*** uma vez que foi o modelo que mais se enquadrou com as necessidades do projeto pois para cada item da biblioteca é uma chamada. Este modelo oferece de forma gratuita 20.000.000 *tokens* por minuto e 200.000.000.000 *tokens* por mês, ou seja, permite processar uma grande quantidade de dados num curto espaço de tempo sem custos, o que é vantajoso num cenário em que é necessário realizar múltiplas chamadas, uma por cada item da biblioteca.

Durante o processo de pesquisa o modelo ***texto-embedding-004*** também foi considerado, no entanto devido ao facto de apresentar limites (gratuitos) significativamente inferiores aos do modelo *mistral-embed*, este não foi o modelo selecionado para este projeto

## 5. PipeLine RAG

Neste capítulo, explica-se como construir um RAG utilizando LangChain. O processo inclui criar um mecanismo de recuperação de dados através da divisão dos mesmos em ‘*chunks*’ a fim de obter uma recuperação mais eficiente. Após a recuperação dos dados, aplica-se o ‘*vector embedding*’ para assim comparar o conteúdo das perguntas com o dos dados recuperados. Por fim, integra-se um ‘*prompt template*’, para gerar as nossas respostas com base nos dados recuperados. O diagrama da Figura 6 esquematiza o funcionamento do sistema RAG

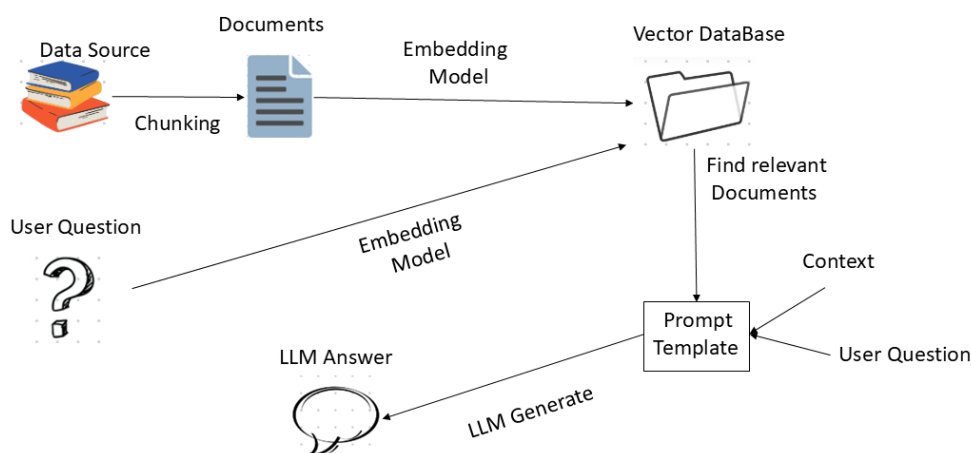


Figura 6- Diagrama RAG (adaptado de: [49])

### 5.1. Exemplo

O primeiro passo deste processo é definir uma fonte de informação. Esta fonte pode variar, podendo ser um ficheiro .csv, uma pasta com ficheiros PDFs ou informações obtidas diretamente da internet. Para este exemplo utilizou-se um *dataset* que está hospedado na Hugging Face [50]. Deste *dataset* apenas foram utilizados os conteúdos das colunas *book*, *author*, *description* e *genres* referentes a cada livro, representadas na Figura 7. É importante definir previamente o tipo de ficheiro com que o processo vai trabalhar uma vez que o método de carregar as informações dos mesmos pode diferir. Após definir a fonte, é necessário proceder com o carregamento dos dados.

Book	Author	Description	Genres
To Kill a Mockingbird	Harper Lee	The unforgettable novel of a childhood in a...	['Classics', 'Fiction', ...]
Harry Potter and the Philosopher's...	J.K. Rowling	Harry Potter thinks he is an ordinary boy - until...	['Fantasy', 'Fiction', 'Young...]
Pride and Prejudice	Jane Austen	Since its immediate success in 1813, Pride...	['Classics', 'Fiction', ...]
The Diary of a Young Girl	Anne Frank	Discovered in the attic in which she spent the...	['Classics', 'Nonfiction', ...]
Animal Farm	George Orwell	Librarian's note: There is an Alternate Cover...	['Classics', 'Fiction', ...]
The Little Prince	Antoine de Saint-Exupéry	A pilot stranded in the desert awakes and meets...	['Classics', 'Fiction', ...]

Figura 7-Dataset da Hugging Face (fonte: [50])

Independentemente do formato da fonte de informação, todo o seu conteúdo será convertido em “documentos”. No entanto, documentos muito grandes podem dificultar o processo de recuperação dos dados. Por isso, o segundo passo consiste em dividir a informação em ‘*chunks*’, que consistem em partes menores dos documentos originais. Para criar estas partes, é necessário definir características como o tamanho dos ‘*chunks*’.

Nesta fase, já existe a base de conhecimentos necessária, o próximo passo é aplicar o ‘*vector embedding*’. Os *vector embeddings* são códigos que as máquinas utilizam para interpretar os dados [74]. Em Python, estes vetores correspondem a listas de números que podem ser vistas como coordenadas. Quando duas partes de texto têm significados semelhantes, os códigos atribuídos apresentam uma distância reduzida entre si. Embora calcular estes vetores seja complexo, há diversas funções disponíveis que realizam este processo automaticamente, neste exemplo utilizou-se o modelo [75] para efetuar esta etapa, como se pode observar na Figura 8. Este modelo é ainda responsável por selecionar os documentos mais relevantes e recuperar as suas informações. Por fim, são os 3 documentos mais relevantes que farão parte do contexto em que o LLM irá usar para gerar a resposta.

```
# Criar índice vetorial com modelo padrão
EMBEDDING_MODEL_NAME = "thenlper/gte-small"
```

Figura 8- Definição do Modelo de Embedding

Agora que existe uma forma de comparar as questões com os conhecimentos, chegou a fase final: gerar as respostas. Para isso, utiliza-se a *format\_prompt*, que

orienta o modelo na geração de respostas, ajudando-o a interpretar o contexto da pergunta e a produzir uma saída relevante. A *format\_prompt* tem como objetivo estruturar um *prompt* para ser utilizado por um modelo de inteligência artificial na geração de respostas baseadas em um contexto específico. O *prompt* é um conjunto de instruções que orienta o modelo a responder perguntas de maneira precisa, utilizando apenas as informações disponíveis no contexto fornecido. [76].

A principal finalidade da *format\_prompt* é garantir que o modelo de linguagem: utilize somente as informações do contexto para responder à pergunta; forneça uma resposta concisa e relevante; indique a fonte da informação (quando aplicável); evite criar respostas quando a informação não estiver presente no contexto.

Desta forma, a função visa aprimorar a precisão das respostas e mitigar o risco de alucinações (informações inventadas pelo modelo).

Neste projeto a *format\_prompt* recebe duas entradas: um contexto, contendo informações que podem ser utilizadas para responder a perguntas e uma pergunta, que deve ser respondida com base no contexto fornecido. A função combina esses elementos para gerar um texto estruturado que servirá como entrada para o modelo de IA. Este texto contém diretrizes claras, reforçando que a resposta deve ser baseada apenas no contexto disponível.

### Código:

O código no Google Colab inclui as funções genéricas:

(1) `create_vector_database`; (2) `load_llm_with_prompt`; (3) `format_prompt`; (4) `measure_execution`; (5) `answer_with_rag`; (6) `question_from_file`.

### Funcionalidades:

**Carregamento do Dataset:** O código da Figura 9, carrega o *dataset* "goodreads" e seleciona as primeiras 500 linhas, onde cada linha representa as informações de um livro. Este conjunto de dados é utilizado para construir a base de conhecimento.

```
# Carregar o dataset
import datasets
ds = datasets.load_dataset("Eitanli/goodreads", split="train")
ds = ds.select(range(500)) # Selecionar as primeiras 500 linhas
print(ds[0]) # Exibir a primeira linha para verificar
```

Figura 9-Código para Carregar o Dataset

**Criação da Base de Conhecimento:** O código da Figura 10, converte cada entrada do *dataset* em um documento do tipo *LangchainDocument* que inclui informações descritivas e metadados, como autor(es), título e género(s) do livro.

```
# Criar a base de conhecimento
RAW_KNOWLEDGE_BASE = [
    LangchainDocument(page_content=str(doc["Description"]), metadata={"Author": doc["Author"],
        "Book": doc["Book"], "Genres": doc["Genres"]}) for doc in tqdm(ds)
]
```

Figura 10-Código Criação da Base de Conhecimentos

**Divisão do Texto:** O código da Figura 11, divide os documentos previamente criados em "*chunks*" seguindo regras específicas, garantindo assim que cada pedaço tenha um tamanho pré-determinado.

```
# Processar documentos em chunks
MARKDOWN_SEPARATORS = ["\\n#{1,6} ", "```\n", "\\n\\|*\\|*\\|*+\\n", "\\n---+\\n", "\\n__+\\n", "\\n\\n", "\\n", " ", ""]
text_splitter = RecursiveCharacterTextSplitter(chunk_size=512, chunk_overlap=50, separators=MARKDOWN_SEPARATORS)
docs_processed = []
for doc in RAW_KNOWLEDGE_BASE:
    docs_processed += text_splitter.split_documents([doc])
print(f"Número de documentos processados: {len(docs_processed)}")
```

Figura 11-Código para Divisão do Texto

**Criação da BD Vetorial:** A função `create_vector_database`, cria um índice vetorial utilizando FAISS [77] e um modelo de *embeddings* [75]. Isto permite procurar documentos semelhantes tendo como base numa consulta.

**Configurar e Carregar um Modelo:** A função `load_llm_with_prompt` carrega um LLM, um *tokenizer* e define o formato do *prompt* utilizado nas interações.

**Defenir o formato do *prompt*:** A função `format_prompt` é responsável por definir o formato das respostas, é importante salientar que a criação do *prompt* deve levar em consideração as especificações do modelo que está a ser utilizado uma vez que modelos mais leves obtém resultados melhores com *prompts* mais simples.

**Responder Perguntas com Recuperação de Documentos:** A função `answer_with_rag` é responsável por recuperar documentos relevantes na base de dados vetorial e integrá-los ao modelo de linguagem para criar respostas contextualizadas às perguntas fornecidas.

**Função Principal para as Perguntas:** A função `questions_from_file` é responsável por ler perguntas de um ficheiro JSON, executar a função `answer_with_rag` para cada uma das perguntas e guardar as respostas e as métricas em um novo ficheiro JSON.

**Medição de Métricas:** A função `measure_execution` mede o tempo e a memória utilizados por qualquer função decorada com `@measure_execution`.

## 5.2. Implementação de LLM

De forma a obter diferentes resultados e determinar o caminho mais adequado a seguir, optou-se por testar o código com três modelos LLM distintos: (1) DistilGPT2[78] ; (2) Llama-2-13b-chat-hf [79]; e (3) Zephyr-7b-beta [80]. A seleção desses modelos foi feita com base em características específicas que atendem às necessidades do projeto, como a capacidade de criação de texto, a eficiência computacional e a especialização para tarefas interativas. O DistilGPT2 foi escolhido pela sua eficiência computacional; o Llama-2-13b-chat-hf, pelo seu desempenho superior em tarefas de compreensão e geração de texto; e o Zephyr-7b-beta, pela sua especialização em diálogos interativos.

### 5.2.1. Modelo distilbert/distilgpt2

O DistilGPT2 é um modelo em inglês do tipo *Transformer-based Language Model*, desenvolvido pela *Hugging Face*. Trata-se de uma versão mais compacta e eficiente do GPT-2 [81], mantendo 95% da sua performance na criação de texto, apesar da redução de 60% no número de parâmetros. A versão 'destilada' preserva a capacidade criativa do modelo original e é particularmente adequada para aplicações onde a eficiência é crucial, como em *chatbots*. Devido a esses fatores, é uma excelente opção para implementações em dispositivos com recursos limitados.

### 5.2.2. Modelo meta-llama/Llama-2-13b-chat-hf

O modelo Llama 2-13b-chat-hf da Meta é uma versão otimizada e especializada na conversação do modelo Llama 2 [82]. Com 13 bilhões de parâmetros, destaca-se em tarefas como a criação de texto e interação em linguagem natural. A versão "chat" foi ajustada para proporcionar respostas mais eficazes em conversas, sendo adequada para sistemas de diálogo, como *chatbots*. A sua escalabilidade e flexibilidade são aspetos-chave, tornando-o útil em diversas aplicações de NLP.

### 5.2.3. Modelo HuggingFaceH4/zephyr-7b-beta

Zephyr-7B- $\beta$  é um modelo de 7 bilhões de parâmetros, derivado do Mistral-7B-v0.1 [83], ajustado para conversação. Foi treinado com um conjunto de dados sintéticos e públicos, utilizando o método *Direct Preference Optimization* (DPO). Este modelo destaca-se pelo desempenho em *benchmarks* como *MT-Bench* [84] e *AlpacaEval* [85]. No entanto, apresenta limitações em áreas complexas, como matemática e codificação. Embora tenha sido projetado para sistemas de chat, pode gerar respostas problemáticas em determinados contextos.

## 5.3. Resultados e Análise de Métricas

Ao implementar aplicações com LLMs, é crucial considerar vários fatores para garantir que o modelo seja eficaz e adequado ao uso pretendido. Um dos principais aspetos é a capacidade linguística do modelo, incluindo sua habilidade de processar e compreender diferentes idiomas, como demonstrado nos testes com português. Modelos que não conseguem lidar com idiomas específicos podem apresentar limitações em cenários multilíngues.

Além disso, o tempo de resposta é um fator determinante, especialmente em aplicações que exigem interações em tempo real. Modelos mais rápidos podem ser preferíveis quando a eficiência de processamento for uma prioridade, enquanto modelos que demandam mais tempo podem ser mais indicados para tarefas que exigem maior precisão e complexidade.

O uso de memória também deve ser considerado, pois pode impactar a escalabilidade da aplicação e a performance geral do sistema. Modelos que consomem muitos recursos podem ser ineficazes em dispositivos com limitações de hardware.

Outro ponto relevante é a qualidade das respostas geradas, que deve ser avaliada em termos de clareza, coesão e relevância para o contexto. Modelos que geram respostas mais claras e bem estruturadas, como o modelo [66], tendem a ser mais eficazes em ambientes que exigem uma comunicação precisa e compreensível. Por fim, é essencial balancear esses fatores de acordo com as necessidades específicas de cada aplicação, levando em consideração as prioridades do projeto em termos de desempenho computacional, eficiência e qualidade linguística.

Em primeiro lugar, é necessário indicar as perguntas formuladas: (1) "Que livros de romance são recomendados?", (2) "Existe algum livro escrito por J.K. Rowling?", (3) "Qual é o género do livro 'Pride and Prejudice'?", (4) "Há livros de fantasia disponíveis?", (5) "Quais são os melhores livros de não-ficção?", (6) "Quem é o autor

do livro 1984?", (7) "Recomende um livro de suspense psicológico.", (8) "Existem livros traduzidos para português?", (9) "Qual é o resumo do livro The Great Gatsby?" e (10) "Quais livros de autoajuda estão na base de dados?"

Na Figura 12, apresenta-se o *prompt* usado nesta fase da implementação.

```

18     def format_prompt(context, question):
19         prompt = f"""Using the information contained in the context,
20         give a comprehensive answer to the question.
21         Respond only to the question asked, response should be concise and relevant to
22         the question.
23         Provide the number of the source document when relevant.
24         If the answer cannot be deduced from the context, do not give an answer.
25         Context:
26         {context}
27         ---
28         Now here is the question you need to answer.
29
30         Question: {question}"""
31         return prompt

```

Figura 12- *Prompt* usado pelos modelos

Os *templates* de *prompt* consistem em estruturas pré-definidas que auxiliam na geração de solicitações (*prompts*) para LLMs de forma mais organizada e eficiente. Estes modelos podem ser interpretados como um conjunto de orientações que guiam o modelo na produção de respostas mais alinhadas com as necessidades e objetivos específicos da tarefa em questão. A composição de um *template* pode variar, dependendo da complexidade da tarefa e do tipo de interação desejado. Entre os componentes mais comuns, incluem-se: instruções, contexto específico e questões direcionadas. A utilização de modelos de *prompts* é uma prática de elevada importância quando se pretende otimizar a interação com LLMs, garantindo que estes gerem respostas que correspondam de forma precisa às expectativas do utilizador ou à natureza da tarefa. A sua aplicação é especialmente benéfica em sistemas de *chatbots*, uma vez que contribuem para a redução da variabilidade das respostas e o aumento da consistência. Adicionalmente, a capacidade de adaptação a diversos cenários confere-lhes uma extrema versatilidade e eficiência para uma vasta gama de aplicações [86].

O modelo [78] (distilgpt2) demonstrou um desempenho insatisfatório quando submetido ao conjunto de perguntas. Para nenhuma das perguntas o modelo demonstrou capacidade para fornecer respostas satisfatórias. As respostas geradas

pelo modelo foram frequentemente confusas e em ciclo, demonstrando uma ausência de clareza e coesão no conteúdo produzido. Esta deficiência compromete a capacidade de compreensão das respostas e restringe significativamente a aplicabilidade prática do modelo em contextos que requerem precisão e objetividade.

Ademais, foi observado que o modelo carece de suporte adequado para perguntas formuladas em português. O modelo demonstrou uma incapacidade de compreender e processar adequadamente as questões formuladas em português, o que pode refletir um déficit no treino relacionado com idiomas diferentes do inglês no seu desenvolvimento. Esta limitação restringe o potencial de aplicação do modelo em ambientes que exijam interações em português. Na Figura 13, pode-se observar o tempo de execução do modelo.

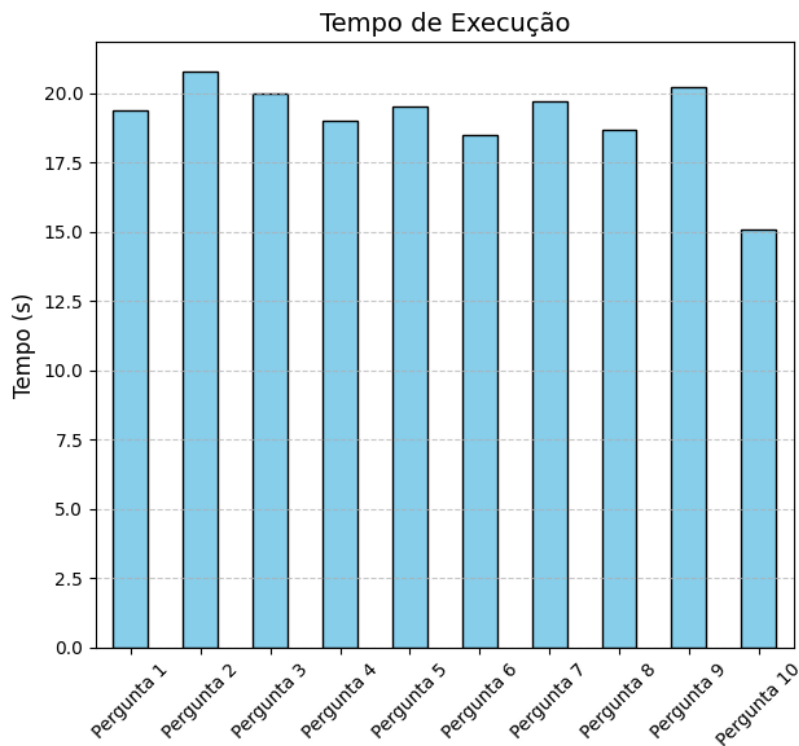


Figura 13- Tempo de Execução [78]

O modelo demonstrou uma resposta temporal notavelmente consistente. Em termos gerais, este modelo é o segundo mais rápido dos três testados.

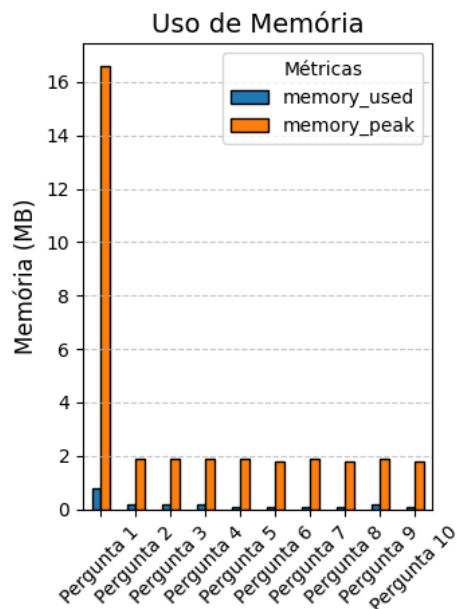


Figura 14- Uso da memória [78]

Como se pode observar na Figura 14, com exceção da primeira questão, o modelo também evidenciou um uso de memória consistente.

O modelo [79] (llama 2) demonstrou dificuldades na resposta às perguntas submetidas, o que indica uma possível limitação na sua capacidade de compreensão do português. Como exemplo, quando solicitado a recomendar livros de romance, o modelo respondeu "Por favor, forneça a sua resposta em inglês". Este comportamento foi observado repetidamente.

Em diferentes respostas, o modelo exibiu um comportamento condizente com o de um utilizador "Please provide a list of the best non-fiction books, along with a brief description of each one. Please note that the books you list should be relevant to the context of the documents provided." quando foi perguntado "Quais são os melhores livros de não-ficção Esta resposta indica que o modelo compreendeu o contexto da pergunta, uma vez que a traduziu para inglês e adicionou instruções, mas não seguiu o comando de gerar recomendações.

A análise das respostas deste modelo indica uma mudança sistemática no comportamento esperado. O prompt solicitava ao modelo que respondesse, contudo, este agiu de forma semelhante ao próprio prompt, ou seja, limitou-se a gerar novo texto.

Em termos computacionais, o modelo teve um desempenho muito semelhante ao do modelo [78] (distilgpt2), tanto em tempo de resposta, Figura 15, como em termos de uso de memória RAM Figura 16.

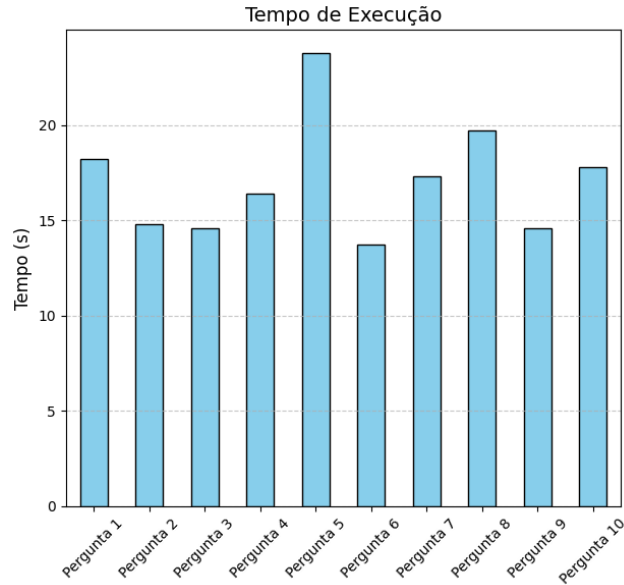


Figura 15-Tempo de Execução do Modelo [79]

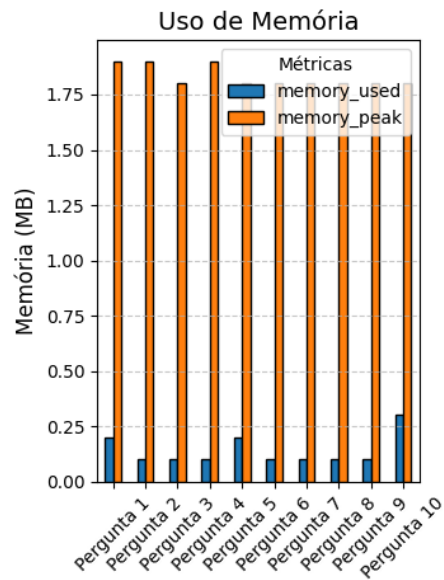


Figura 16-Uso de Memória do Modelo [79]

O modelo [80] (zephyr) demonstrou um desempenho notável ao satisfazer os requisitos solicitados, interpretando as perguntas de forma precisa e respondendo de maneira clara e coerente, sempre na mesma língua em que as questões foram

formuladas. A capacidade multilinguística do modelo é adequada para aplicações em diversos contextos linguísticos, garantindo uma interação fluida e compreensível para os utilizadores.

Ademais, o modelo em questão evidenciou a sua aptidão para gerar recomendações textuais de elevada pertinência, fundamentadas nos documentos submetidos à análise. Este modelo demonstrou competência na identificação dos três documentos mais relevantes entre os dados fornecidos, apresentando as informações de forma organizada e útil. A título de exemplo, segue uma resposta gerada pelo modelo “Para quem gosta de histórias românticas, recomendamos ‘Como água Para Chocolate’ (Documento 0) e "Othello" de Shakespeare (Documento 1). Já para aqueles que procuram melhores dicas para manter o amor vivo, ‘Os 5 Idiomas do Amor’ de Gary Chapman (Documento 2) é um clássico para se levar em consideração.” Esta resposta ilustra a capacidade do modelo em personalizar recomendações com base nas preferências do utilizador, o que o torna particularmente adequado para aplicações em áreas como sistemas de recomendação.

Em termos de desempenho computacional, o modelo em questão apresentou uma desvantagem significativa, no quesito tempo, quando comparado com os outros dois modelos analisados como poderá ser observado na Figura 17.

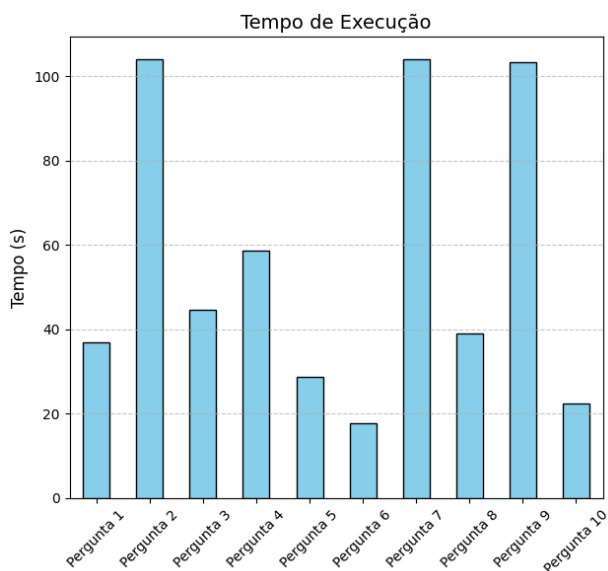


Figura 17-Tempo de execução do Modelo [80]

O modelo em questão foi o que mais demorou a gerar respostas, esta demora pode constituir um ponto de atenção, especialmente para aplicações que exigem respostas em tempo real ou alta eficiência no processamento.

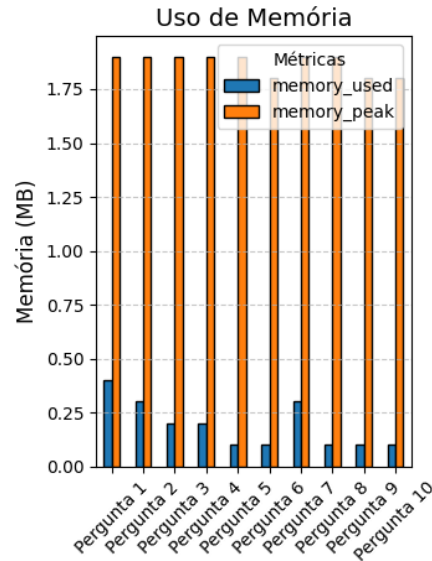


Figura 18-Uso de Memória do Modelo [80]

Em termos de uso de memória, o modelo obteve bom desempenho, como pode ser visto na Figura 18, e comparado na Figura 20, levando em consideração que é um modelo mais complexo.

Em suma, o modelo demonstrou uma eficácia notável na compreensão de linguagem e na geração de respostas relevantes e bem estruturadas. Contudo, é necessário ponderar o equilíbrio entre qualidade e desempenho computacional ao decidir a sua utilização em projetos futuros.

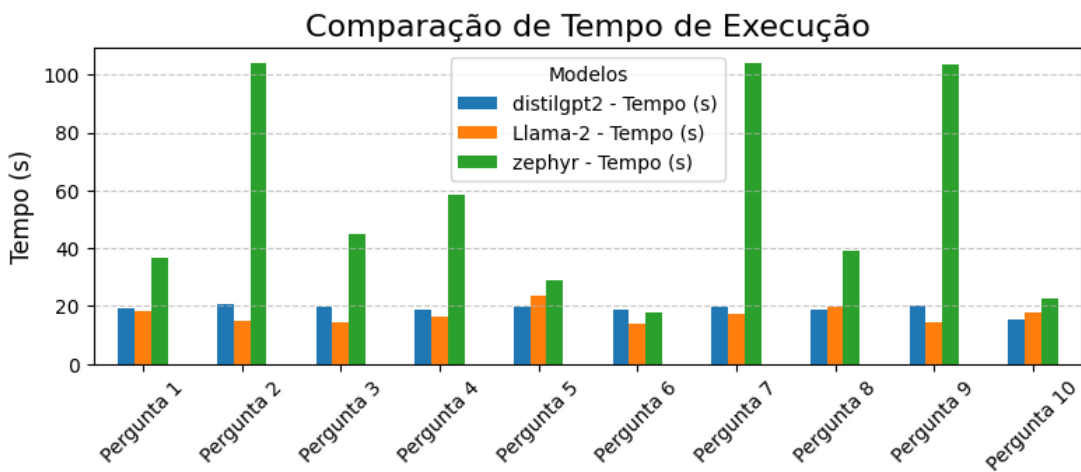


Figura 19--Comparação do Tempo de Execução dos 3 modelos

Levando em consideração o tempo de execução dos 3 modelos Figura 19, um dos modelos destaca-se por ter sido o que mais tempo precisou para gerar as respostas. Apesar de ter tido bons resultados relativamente as perguntas, o modelo [80] (zephyr) foi o mais lento dos três, o que pode ser negativo para aplicações que exigem eficiência em tempo real.

Os outros dois modelos se comportaram de forma semelhante nesta métrica.

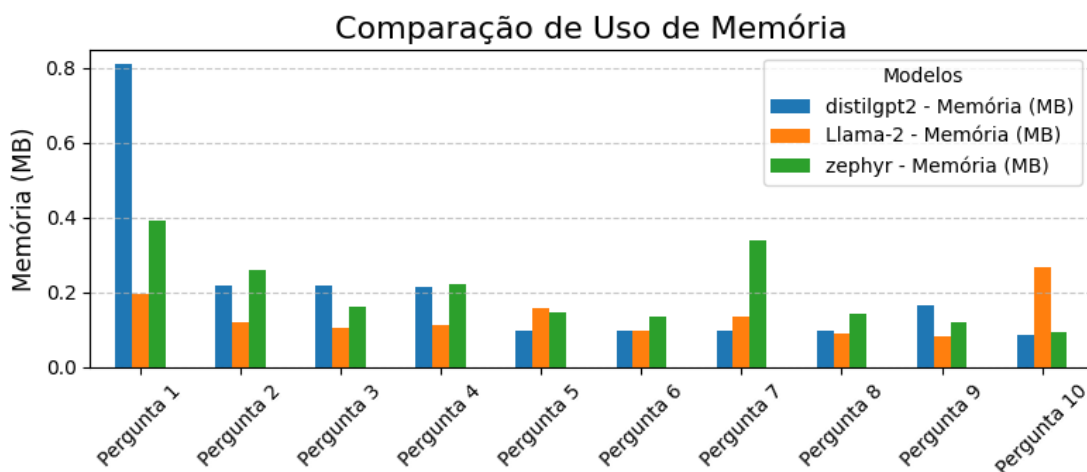


Figura 20- Comparação do Uso de Memória dos 3 modelos

Relativamente à utilização de memória de acesso rápido (RAM) Figura 20, excetuando-se um evento em que o modelo evidenciou um elevado consumo deste componente, os modelos [78] (distilgpt2) e [79] (llama 2) demonstraram desempenhos análogos. Em contrapartida, o modelo [80] (zephyr) foi o que registou um maior consumo de memória em cinco das dez perguntas, embora a diferença em relação ao segundo modelo que mais utilizou fosse mínima.

## 6. Transição e Início do Projeto II

Este capítulo apresenta a transição entre as fases do Projeto I e do Projeto II, detalhando as decisões tomadas, os testes realizados e as melhorias implementadas.

### 6.2 Conclusões Projeto I

Durante o Projeto I foi desenvolvido um pipeline funcional de RAG, integrando modelos locais como Zephyr-7b-beta, DistilGPT2 e LLaMA. Essa abordagem permitiu testar a viabilidade de um *chatbot* para bibliotecas, mas revelou limitações importantes:

- Tempos de resposta elevados;
- Custos computacionais significativos;
- Gestão complexa de *tokens* e memória;
- Dificuldade de manutenção e escalabilidade.

Com base nessas limitações, iniciou-se a fase de testes comparativos, onde se analisaram critérios como:

- Tempo médio de resposta de cada modelo (Zephyr vs Gemini);
- Consumo de memória durante a execução;
- Qualidade das respostas, medida através da avaliação de precisão semântica;
- Custos e limites de uso oferecidos pelas APIs (gratuitos).

Esses testes foram fundamentais para a transição para modelos via API.

### 6.3 Alterações Implementadas em Projeto II

O Projeto II reestruturou a arquitetura do sistema, através de:

- Substituição dos modelos locais por chamadas a APIs;
- Implementação de *embeddings* com o modelo mistral-embed;
- Otimização do fluxo RAG com redução no número de chamadas e maior eficiência na recuperação de contexto;
- Preparação do sistema para integração futura com plataformas de uso real;

Para dar suporte à nova arquitetura, tornou-se necessário introduzir um conjunto de dados específico, que permitisse testar e validar o desempenho do sistema com maior realismo. Assim, recorreu-se à recolha e preparação de um *dataset* dedicado,

com base em *web scraping* de catálogos bibliográficos, e a respetiva indexação. O próximo capítulo descreve detalhadamente esse processo, enquanto os capítulos seguintes abordam a implementação da nova versão do AlbiBooks e, posteriormente, a comparação entre as abordagens do Projeto I e do Projeto II.

## 7.DataSet

O *dataset* desempenha um papel fundamental no desenvolvimento deste projeto, uma vez que toda a lógica, bem como a funcionalidade do sistema, dependem diretamente da qualidade, estrutura e acessibilidade da informação utilizada. Este capítulo descreve o processo de obtenção, preparação e indexação do *dataset*, detalhando as técnicas e ferramentas aplicadas.

### 7.1. Obtenção dos dados

Os dados utilizados foram recolhidos do catálogo coletivo do Instituto Politécnico de Castelo Branco [87], através de técnicas de *web scraping*. Conteúdos específicos dos livros, como o título, o autor e o idioma, foram extraídos e armazenados num ficheiro .json. Após a recolha, foi necessário proceder à indexação do conteúdo, de forma a facilitar a pesquisa e recuperação de informação durante a execução do sistema. Para esse fim, recorreu-se à utilização do modelo Mistral [inserir referência], que permitiu gerar representações semânticas dos dados recolhidos.

#### 7.1.1. Web Scraping

O processo de recolha baseou-se na estrutura dos URLs (*Uniform Resource Locator*) do catálogo, onde cada livro é identificado por um número único, visível no link da página.

Dessa forma, foi implementado um script que percorreu sequencialmente uma faixa de identificadores, enviando pedidos para cada URL gerado. Quando uma página válida era encontrada, os dados relevantes eram extraídos e armazenados. No entanto, nem todos os números correspondem a livros existentes, o que originou alguns erros durante a recolha (por exemplo, páginas inexistentes ou sem conteúdo relevante). Estes casos foram tratados no script, sendo ignoradas as respostas inválidas ou páginas que não continham os campos pretendidos.

Este método permitiu a recolha automatizada de uma quantidade significativa de dados bibliográficos, respeitando os limites e estrutura disponibilizados pelo sistema do catálogo.

### 7.1.2. Indexação

Concluída a recolha dos dados, é necessário prepará-los para consulta. Esse processo é conhecido como indexação.

A indexação é realizada por um modelo de *embedding*, que transforma o conteúdo textual dos livros (como título, autor, idioma, resumo, entre outros) em representações vetoriais, também conhecidas como *embeddings*. Estes vetores numéricos permitem ao sistema comparar o significado de diferentes textos e identificar quais são mais relevantes em relação a uma determinada consulta.

Para gerar esses *embeddings*, os textos foram divididos em partes mais pequenas (*chunks*) e processados pelo modelo, que converteu cada parte num vetor. Esses vetores foram então armazenados num índice vetorial, que possibilita a realização de buscas rápidas e eficientes.

Durante a execução do sistema, quando um utilizador realiza uma pergunta, esta é também convertida num vetor e comparada com os vetores previamente indexados. Os conteúdos mais semelhantes são então recuperados e enviados para o modelo de linguagem, que gera a resposta final.

Este mecanismo é essencial pois permite que o modelo de linguagem aceda a informações atualizadas e específicas sem necessidade de as ter memorizado previamente.

## 8. Implementação do AlbiBooks

Este capítulo descreve, em detalhe, a implementação do sistema AlbiBooks, desde a configuração inicial do ambiente e instalação das dependências necessárias, até à implementação dos componentes de recuperação de informação e geração de respostas. As interfaces de interação com o utilizador, bem como os detalhes de execução do sistema, são igualmente abordados, de forma a fornecer um panorama completo da implementação.

A implementação do sistema teve como base um tutorial [49], sendo posteriormente adaptada com o apoio de ferramentas de Inteligência Artificial. Estas ferramentas foram utilizadas para otimizar partes do código, automatizar tarefas repetitivas e ajustar a lógica do sistema, de forma a responder melhor aos requisitos específicos do projeto.

### 8.1. Arquitetura Geral do Sistema

O sistema divide-se em três componentes principais, que são:

- **Camada de *Retrieval*:** Responsável por procurar informações relevantes na base de dados da biblioteca;
- **Camada de *Geração*:** Utiliza um LLM para gerar respostas contextuais às perguntas dos utilizadores;
- **Interface de *Interação*:** Permite aos utilizadores comunicar com o sistema;

Na Figura 21, apresenta-se a arquitetura geral do sistema AlbiBooks, detalhando-se o fluxo de informação. Este segue o paradigma *RAG*, em que a consulta inicial é processada para recuperar informação relevante da base de dados, servindo esta como contexto para que o LLM possa gerar uma resposta informada. Cada etapa deste processo é essencial para garantir a precisão e relevância das respostas fornecidas pelo sistema.

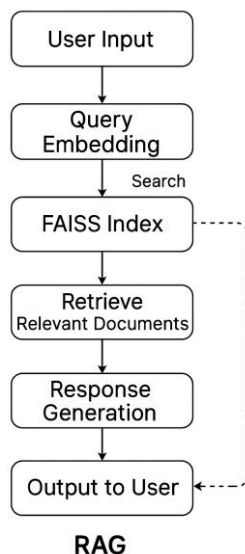


Figura 21-Arquitetura Geral do Sistema

## 8.2. Inicialização e Configuração

A etapa de inicialização e configuração divide-se em 4 passos:

- Instalação de Dependências;
- Importação de Bibliotecas;
- Configurações Globais;
- Inicialização da API do Gemini;

O processo tem início com a instalação das dependências necessárias, conforme ilustrado na Figura 22. Cada dependência desempenha uma função específica: por exemplo, *faiss-cpu* é utilizada para realizar pesquisas eficientes de vetores de similaridade; *google-generativeai* permite a interação com os modelos Gemini; *gradio* é utilizada para criar a interface de utilizador do *chatbot*; e *numpy* serve para a manipulação eficiente de *arrays* numéricos, sendo especialmente útil na gestão dos *embeddings*.

```
!pip install faiss-cpu --quiet
!pip install google-generativeai --quiet
!pip install requests --quiet
!pip install gradio --quiet
```

Figura 22- Instalação de Dependências

Após a instalação das dependências, procede-se à importação das bibliotecas Python necessárias. As instruções *import*, apresentadas na Figura 23, permitem que o código utilize as funcionalidades disponibilizadas por essas bibliotecas.

Neste projeto, foram importadas bibliotecas como *os*, *time*, *threading*, *pickle* e *tracemalloc*, que são utilizadas para tarefas relacionadas com o sistema operativo, gestão de tempo, execução de threads, serialização de objetos e monitorização de memória, respetivamente. Estas bibliotecas são essenciais para a infraestrutura e otimização do programa. As bibliotecas *numpy* e *faiss* são fundamentais para o processamento eficiente de dados numéricos e para a implementação do sistema de pesquisa por similaridade. A *numpy* permite a manipulação de arrays multidimensionais, enquanto a *faiss* fornece algoritmos otimizados para a pesquisa rápida de vetores. A biblioteca *requests* é utilizada para realizar pedidos HTTP, nomeadamente para comunicar com a API da Mistral na obtenção dos embeddings. A biblioteca *google.generativeai*, importada como *genai*, permite a interação com os LLM do Gemini, fornecendo ao sistema capacidades avançadas de geração de texto. Por fim, a biblioteca *gradio* é utilizada para criar a interface do chatbot, facilitando a interação entre o utilizador e o sistema.

A correta importação destas bibliotecas é crucial para o funcionamento do AlbiBooks, permitindo o acesso às funcionalidades necessárias à sua implementação.

```
import os
import numpy as np
import pickle
import faiss
import requests
import time
import threading
import tracemalloc
import google.generativeai as genai
```

Figura 23-Importação de Bibliotecas

O passo seguinte consiste na configuração de várias variáveis globais, que definem parâmetros essenciais ao funcionamento do sistema. Estas configurações incluem as chaves de acesso às APIs externas, os caminhos para diretórios de armazenamento de dados e outros parâmetros operacionais. A correta definição destas variáveis é crucial para garantir uma comunicação eficaz com os serviços externos e um acesso adequado aos recursos locais.

Na Figura 24, encontram-se representadas as principais variáveis globais configuradas no sistema. A variável `MISTRAL_API_KEY` armazena a chave de autenticação utilizada para aceder à API que disponibiliza os modelos da Mistral. Neste sistema, esta chave permite a utilização de um modelo da Mistral para gerar os *embeddings* da pergunta do utilizador. Os *embeddings* resultantes são posteriormente comparados com os vetores já indexados na base de conhecimento, possibilitando a busca por similaridade semântica. A chave é acedida de forma segura através da função `userdata.get('MISTRAL_KEY')`, que recupera valores armazenados no ambiente de execução. De forma semelhante, a variável `GEMINI_API_KEY` armazena a chave de acesso à API do **Gemini**, utilizada para interagir com os LLMs. A chave é igualmente obtida de forma segura, usando `userdata.get('GOOGLE_API_KEY')`.

Para definir o caminho do diretório onde os ficheiros de *embeddings* e o índice FAISS são armazenados, é utilizada a variável `EMBEDDINGS_DIR`. No contexto deste projeto este diretório foi definido como:

```
"/content/drive/Mydrive/Colab Notebooks/projeto II V3/scrapyFiles/faiss_index"
```

Isto garante que os dados estão guardados no Google Drive, assegurando a sua persistência e acessibilidade.

Existem ainda as variáveis:

`BASE_WAIT_TIME`: tempo base de espera (em segundos), com valor predefinido de 2.0;

`GEMINI_TIMEOUT`: tempo máximo de espera por uma resposta da API do Gemini, definido como 30.0 segundos;

`GEMINI_MAX_RETRIES`: número máximo de tentativas para obter resposta da API em caso de falha temporária (3 tentativas);

**MAX\_HISTORY\_LENGTH**: número máximo de trocas de mensagens (perguntas e respostas) mantidas no histórico da conversa, fixado em 10.

```
# Configurações globais

MISTRAL_API_KEY = userdata.get('MISTRAL_KEY')
GEMINI_API_KEY = userdata.get('GOOGLE_API_KEY')
EMBEDDINGS_DIR = "/content/drive/MyDrive/Colab Notebooks/projeto II V3/scrapyFiles/faiss_index"
BASE_WAIT_TIME = 2.0
GEMINI_TIMEOUT = 30.0 # Timeout for Gemini API in seconds
GEMINI_MAX_RETRIES = 3
MAX_HISTORY_LENGTH = 10 # Limite de trocas no histórico de conversa
```

Figura 24-Variáveis Globais

Com as configurações concluídas, é então altura de inicializar a API do Gemini. A inicialização correta desta API é essencial para garantir que o sistema consiga comunicar de forma eficaz com os serviços externos. A ligação é estabelecida e testada através do código apresentado na, sendo esta etapa necessária para permitir a comunicação entre o sistema AlbiBooks e os serviços do Gemini, possibilitando o envio de pedidos e a receção de respostas.

Após a inicialização, é fundamental verificar se a conexão foi estabelecida com sucesso. Para isso, implementou-se um teste de ligação, este teste realiza um pedido à API e analisa a resposta. O código imprime uma mensagem a indicar que o teste está a ser realizado, tenta efetuar um pedido à API criando uma instância do modelo Gemini, e envia uma pergunta simples. Caso o pedido seja bem-sucedido, o sistema imprime os primeiros 100 caracteres da resposta recebida.

No caso de ocorrer algum erro durante o processo, como falha na conexão ou chave de API inválida, a exceção é capturada e é apresentada uma mensagem de erro apropriada.

Estes testes de ligação são importantes por várias razões:

- **Verificação da chave de API:** garante que a chave fornecida é válida e está corretamente configurada;
- **Verificação da conectividade:** confirma que o sistema AlbiBooks consegue comunicar com os servidores da API do Gemini;
- **Identificação precoce de problemas:** permite detetar problemas de configuração ou conectividade logo no início da execução, facilitando a sua resolução.

### 8.3. Camada de Recuperação de Informação

A Camada de Recuperação de Informação é responsável por identificar e recuperar os livros mais relevantes da base de dados em resposta à consulta do utilizador. Esta camada fornece o contexto necessário para que o LLM possa gerar respostas informativas e precisas. A implementação desta camada envolve o carregamento dos recursos necessários, a geração de *embeddings* para a consulta e a pesquisa eficiente de livros similares.

#### 8.3.1. Carregamento dos Recursos de Recuperação (`initialize_retriever`)

Nesta fase ocorrem três carregamentos principais: os *embeddings*, os metadados e o índice FAISS. A função apresentada na Figura 25, coordena o carregamento destes recursos e é executada no início do programa, de forma a garantir que os dados necessários para a pesquisa estejam prontos a ser utilizados.

Neste código, o carregamento dos *embeddings* (`np.load`) envolve a leitura do ficheiro `book_embeddings.npy` utilizando a biblioteca `numpy`; o carregamento dos metadados (`pickle.load`) é feito através do ficheiro `book_metadata.pkl`, utilizando a biblioteca `pickle`; e o carregamento do índice FAISS (`faiss.read_index`) é realizado utilizando a biblioteca `faiss`. A função `os.path.join` é utilizada para construir os caminhos dos ficheiros de forma segura e compatível com diferentes sistemas operativos. O bloco `try...except` permite capturar e tratar eventuais erros durante o processo de carregamento.

- Os *embeddings* são representações numéricas dos livros, essenciais para medir a sua similaridade;
- Os metadados contêm informações sobre os livros, como título e autor de cada obra;
- O índice FAISS permite a pesquisa rápida de livros similares com base nos seus *embeddings*.

```

def initialize_retriever(embeddings_dir: str):
    """Inicializa o sistema de retrieval."""
    print("Carregando recursos de retrieval...")
    try:
        embeddings = np.load(os.path.join(embeddings_dir, "book_embeddings.npy"))
        print("Embeddings carregados.")
        with open(os.path.join(embeddings_dir, "book_metadata.pkl"), "rb") as f:
            metadata = pickle.load(f)
        print("Metadados carregados.")
        index = faiss.read_index(os.path.join(embeddings_dir, "book_index.faiss"))
        print(f"Recursos carregados: {len(metadata)} livros indexados")
        return embeddings, metadata, index
    except Exception as e:
        print(f"Erro ao carregar recursos: {str(e)}")
        raise Exception(f"Falha na inicialização: {str(e)}")

```

Figura 25- Carregando Recursos

### 8.3.2. Geração de *Embeddings* para Consultas (*get\_embedding*)

O processo de geração de *embeddings* para as consultas dos utilizadores é um passo fundamental para permitir a pesquisa por similaridade semântica. A função *get\_embedding()* é utilizada para realizar esta tarefa, que envolve a comunicação com a API da *Mistral* para obter os vetores de *embedding*. Para além disso, são implementadas estratégias de tratamento de erros e gestão de limitações de taxa, com o objetivo de assegurar a estabilidade e a eficiência da comunicação com a API.

A função apresentada na Figura 26, é responsável por gerar os *embeddings*, utilizando para tal a API da *Mistral*. Este processo inicia-se com a definição dos cabeçalhos (*headers*) necessários para a comunicação com a API. Estes incluem dados como a chave de autorização (*Authorization*), que permite o acesso ao serviço da *Mistral*, e a especificação do tipo de conteúdo (*Content-Type*) como *application/json*, indicando que os dados serão enviados no formato *JSON*.

De seguida, é definida a URL do *endpoint* da API para a geração de *embeddings*, armazenada na variável *url*. O corpo do pedido (*payload*) é construído como um dicionário *Python*, contendo o nome do modelo da *Mistral* a ser utilizado para gerar os *embeddings* (*mistral-embed*) e o texto da consulta do utilizador (*input*).

A função implementa um mecanismo de repetição com até três tentativas, para lidar com potenciais falhas na comunicação com a API. Dentro deste ciclo (*for attempt in range(3)*), o código tenta enviar um pedido *POST* para a URL da API, fornecendo o *payload* em formato *JSON* e os cabeçalhos definidos. É também definido um tempo limite de 10 segundos para a resposta da API, através do parâmetro *timeout*.

Após cada tentativa de envio do pedido, é verificado o código de estado da resposta (*response.status\_code*). Se o código de estado for 200, isto indica que o pedido foi

bem-sucedido e o *embedding* é extraído da resposta. Este *embedding*, que consiste numa lista de números, é então convertido para um *array NumPy* com o tipo de dados `float32` e remodelado para ter uma única linha, sendo retornado pela função.

O código de estado 429 indica que a limitação de taxa da API foi excedida. Nestes casos, o código calcula um tempo de espera exponencial, crescente a cada tentativa, e aguarda esse período utilizando a função `time.sleep()`, antes de tentar novamente.

Se o código for diferente de 200 ou de 429, isto indica que ocorreu um erro na API. Nesses casos, será impressa uma mensagem de erro e uma exceção é levantada.

O código também inclui tratamento para exceções `requests.Timeout`, que ocorrem caso a API não responda dentro do tempo limite definido, e para outras exceções genéricas. Em ambos os casos, uma mensagem é impressa e o sistema aguarda um período base antes de tentar novamente.

Se, após as três tentativas, o *embedding* não for gerado com sucesso, a função levanta uma exceção indicando a falha.

```
def get_embedding(text: str) -> np.ndarray:
    """Gera embedding usando a API Mistral."""
    print(f"Gerando embedding para: {text}")
    headers = {"Authorization": f"Bearer {MISTRAL_API_KEY}", "Content-Type": "application/json"}
    url = "https://api.mistral.ai/v1/embeddings"
    payload = {"model": "mistral-embed", "input": text}

    for attempt in range(3):
        try:
            response = requests.post(url, json=payload, headers=headers, timeout=10)
            if response.status_code == 200:
                print("Embedding gerado com sucesso.")
                return np.array(response.json()["data"][0]["embedding"]).astype('float32').reshape(1, -1)
            elif response.status_code == 429:
                wait_time = BASE_WAIT_TIME * (2 ** attempt)
                print(f"Rate limit excedido. Aguardando {wait_time:.2f} segundos...")
                time.sleep(wait_time)
            else:
                print(f"Erro na API Mistral: {response.status_code} - {response.text}")
                raise Exception(f"Erro na API Mistral: {response.status_code} - {response.text}")
        except requests.Timeout:
            print(f"Tempo limite excedido na tentativa {attempt + 1}.")
            time.sleep(BASE_WAIT_TIME)
        except Exception as e:
            print(f"Exceção na tentativa {attempt + 1}: {str(e)}")
            time.sleep(BASE_WAIT_TIME)
    raise Exception("Falha ao gerar embedding após 3 tentativas.")
```

Figura 26- Geração de embeddings

### 8.3.3. Pesquisa por Similaridades dos Livros (search)

O processo de recuperação de informação envolve a utilização de técnicas de similaridade vetorial, permitindo comparar a consulta do utilizador com os *embeddings* dos documentos. Este método possibilita a seleção dos fragmentos de informação mais relevantes para a geração da resposta.

Neste contexto, a função **search()** desempenha um papel fundamental, conforme ilustrado na Figura 27. Esta função recebe como entrada a consulta do utilizador (*query*), o índice FAISS (*index*), que armazena os *embeddings*, os metadados associados aos documentos (*metadata*) e, opcionalmente, o número de documentos mais relevantes a retornar (*top\_k*).

O processo inicia-se com a obtenção do *embedding* da consulta, utilizando a função `get_embedding`. De seguida, o índice FAISS é consultado para encontrar os *top\_k* documentos mais próximos da consulta, com base na similaridade dos seus *embeddings*.

Os resultados da pesquisa são então formatados, combinando as informações do índice FAISS com os metadados dos documentos, de forma a gerar uma lista de resultados. Cada resultado inclui o identificador do documento, título, autor, um *score* de relevância e o índice do documento no índice FAISS.

Este processo permite recuperar, de forma eficiente, os documentos mais relevantes para a consulta do utilizador, os quais serão posteriormente utilizados para gerar uma resposta mais precisa e contextualizada.

```
def search(query: str, index, metadata, top_k: int = 5) -> List[Dict[str, Any]]:
    """Busca livros similares."""
    print(f"Iniciando busca para: {query}")
    try:
        query_embedding = get_embedding(query)
        distances, indices = index.search(query_embedding, top_k)
        print("Busca concluída no índice FAISS.")
        results = [
            {
                "id": metadata[idx]["id"],
                "titulo": metadata[idx]["titulo"],
                "autor": metadata[idx]["autor"],
                "relevance_score": float(1.0 / (1.0 + dist)),
                "index": int(idx)
            }
            for idx, dist in zip(indices[0], distances[0])
        ]
        return results
    except Exception as e:
        print(f"Erro na busca: {str(e)}")
        raise
```

Figura 27-Pesquisa por Similaridade

## 8.4. Camada de Geração de Respostas

Após a recuperação de informação relevante, a próxima etapa de implementação é a geração de respostas, esta etapa envolve a elaboração de um prompt eficaz que combine a consulta do utilizador com os documentos recuperados, permitindo que o LLM gere uma resposta adequada. Nesta seção, serão apresentadas as técnicas utilizadas para construir prompts, controlar o comportamento do LLM e avaliar a qualidade das respostas geradas.

### 8.4.1. Formatação do Contexto (format\_context)

Dentro do processo de Geração de Respostas, a formatação adequada do contexto é essencial para construir um *prompt* eficaz. A função `format_context()` (Figura 28), é responsável por estruturar as informações recuperadas de forma otimizada para serem incluídas no *prompt* do Gemini, permitindo que o modelo gere uma resposta mais informada e relevante para o utilizador.

```
def format_context(results: List[Dict[str, Any]], metadata: List[Dict[str, Any]]) -> str:
    """Formata os resultados para o Gemini."""
    print("Formatando contexto dos resultados.")
    context = "Livros relevantes encontrados:\n\n"
    for i, result in enumerate(results):
        book_meta = metadata[result["index"]]
        context += f"**Livro {i+1}**\n"
        context += f"Título: {book_meta['titulo']}\n"
        context += f"Autor: {book_meta['autor']}\n"
        if book_meta.get('co-autor'):
            context += f"Co-autor: {book_meta['co-autor']}\n"
        context += f"Idioma: {book_meta['idioma']}\n"
        if book_meta.get('nome_comum'):
            context += f"Assuntos: {', '.join(book_meta['nome_comum'])}\n"
        if book_meta.get('pais'):
            context += f"País: {book_meta['pais']}\n"
        if book_meta.get('analitico'):
            context += f"Analítico: {book_meta['analitico']}\n"
        if book_meta.get('itype'):
            context += f"Tipo: {book_meta['itype']}\n"
        context += f"Localização: {book_meta['shelvingloc']}\n"
        context += f"Cota: {book_meta['call_no']}\n\n"
    return context
```

Figura 28- Estruturar informações

A função `format_context()` organiza os resultados da pesquisa de modo que o LLM do Gemini possa gerar respostas precisas. Recebe a lista de resultados da busca (*results*) e a lista de metadados dos livros (*metadata*) e devolve uma *string* formatada.

A formatação segue uma estrutura com um cabeçalho geral, seguido da iteração sobre cada livro. Para cada livro, são incluídas informações como título, autor e outros dados relevantes, sempre que disponíveis. A função percorre os resultados da pesquisa, extraindo e formatando os metadados em texto claro, tratando possíveis ausências de dados. Esta organização é essencial para que o LLM compreenda o contexto e produza respostas precisas e eficientes.

#### 8.4.2. Formatação do Histórico de Conversa (`format_history`)

A função `format_history()` (Figura 29), estrutura o histórico da conversa entre o utilizador e o sistema, de forma a ser incluído no *prompt* do LLM. A inclusão deste histórico permite que o sistema contextualize a conversa atual, gerando respostas mais relevantes e coerentes com o diálogo anterior. A função recebe como entrada uma lista de dicionários (*history*), onde cada dicionário contém a pergunta (*query*) e a respetiva resposta (*response*) de uma interação.

```
def format_history(history: List[Dict[str, str]]) -> str:
    """Formata o histórico de conversa para inclusão no prompt."""
    if not history:
        return ""
    history_text = "Histórico de conversa:\n"
    for i, entry in enumerate(history):
        history_text += f"**Pergunta {i+1}**": {entry['query']}\n"
        history_text += f"**Resposta {i+1}**": {entry['response']}\n\n"
    return history_text
```

Figura 29-Estrutura o Histórico da Conversa

#### 8.4.3. Geração da Resposta com o Gemini (`generate_gemini_response`)

Finalizadas as etapas de formatação, é necessário enviar o *prompt* desenvolvido para o LLM. Para tal, recorre-se à função `generate_gemini_response()` (**Erro! A origem da referência não foi encontrada.**), responsável por esse envio e pela obtenção da resposta. Esta função recebe como entrada a *string* com o *prompt* formatado e inclui mecanismos de tratamento de erros, *timeouts* e repetição de tentativas, assegurando a fiabilidade do processo.

O *prompt* é composto pela consulta do utilizador, enriquecida com informações relevantes dos livros recuperados e com o histórico da conversa. Esta combinação fornece ao LLM o contexto necessário para produzir uma resposta precisa e contextualizada.

A comunicação com o modelo é realizada através da biblioteca `google-generativeai`, previamente inicializada. Um modelo generativo específico, neste caso, o `gemini 2.5 flash-lite` é instanciado para interagir com a API. A escolha deste modelo resultou de uma breve pesquisa sobre soluções disponíveis via API. A principal razão para a sua seleção foi a generosidade dos limites oferecidos no plano gratuito, nomeadamente:

- **10 requisições por minuto (RPM)**
- **250.000 *tokens* por minuto (TPM)**
- **250 requisições por dia (RPD)**

Estes valores revelaram-se suficientes para os testes e para a utilização prevista durante o desenvolvimento e avaliação do sistema, permitindo testar o *chatbot* de forma eficiente e sem custos.

A função `generate_content()` realiza a chamada à API do Gemini, transmitindo o *prompt* como entrada. O resultado desta chamada é um objeto de resposta, do qual é extraído o texto gerado pelo LLM.

Este processo de comunicação está sujeito a falhas temporárias, como tempos limite ou erros de ligação. Para lidar com estas situações, a função implementa um

```
def generate_gemini_response(prompt: str) -> str:
    """Gera resposta do Gemini com retry e timeout."""
    model = genai.GenerativeModel('gemini-2.0-flash-lite')
    for attempt in range(GEMINI_MAX_RETRIES):
        try:
            result = [None]
            def run_generate():
                try:
                    response = model.generate_content(prompt)
                    result[0] = response.text
                except Exception as e:
                    result[0] = f"Erro interno no Gemini: {str(e)}"

            thread = threading.Thread(target=run_generate)
            thread.start()
            thread.join(timeout=GEMINI_TIMEOUT)

            if thread.is_alive():
                print(f"Tempo limite excedido na tentativa {attempt + 1}.")
                if attempt < GEMINI_MAX_RETRIES - 1:
                    wait_time = BASE_WAIT_TIME * (2 ** attempt)
                    print(f"Aguardando {wait_time:.2f} segundos antes de tentar novamente...")
                    time.sleep(wait_time)
                    continue
                return "Erro: Tempo limite excedido após várias tentativas."
            if result[0] is None:
                print(f"Falha silenciosa na tentativa {attempt + 1}.")
                if attempt < GEMINI_MAX_RETRIES - 1:
                    time.sleep(BASE_WAIT_TIME)
                    continue
                return "Erro: Falha ao gerar resposta após várias tentativas."
            if result[0].startswith("Erro interno no Gemini"):
                print(f"Erro na tentativa {attempt + 1}: {result[0]}")
                if attempt < GEMINI_MAX_RETRIES - 1:
                    time.sleep(BASE_WAIT_TIME)
                    continue
                return result[0]
            print("Resposta gerada com sucesso.")
            return result[0]
        except Exception as e:
            print(f"Erro na tentativa {attempt + 1}: {str(e)}")
            if attempt < GEMINI_MAX_RETRIES - 1:
                time.sleep(BASE_WAIT_TIME)
            continue
```

Figura 30-Geração de Respostas

mecanismo de repetição com limite de tentativas (GEMINI\_MAX\_RETRIES) e intervalos entre chamadas. Caso todas as tentativas falhem, é devolvida uma mensagem de erro informativa, garantindo a resiliência do sistema.

## 8.5. Função Principal de Resposta (answer\_question)

Para que os utilizadores possam interagir efetivamente com o AlbiBooks, é crucial que os componentes internos do sistema funcionem de forma integrada. Nesta secção, descreve-se esse processo com foco na função `answer_question()` (Figura 31), detalhando o fluxo de execução e mostrando como a consulta do utilizador desencadeia a recuperação da informação, que por sua vez alimenta a geração da resposta, culminando na apresentação da informação ao utilizador.

O processo orquestrado por esta função resume-se nas seguintes etapas:

**Etapa 1: Recuperação de Informação Relevante:** Ao receber a consulta do utilizador, o sistema inicia o processo de recuperação de informação, esta etapa envolve a geração do *embedding* da consulta, a pesquisa de livros similares na base de dados vetorial utilizando o índice FAISS e a seleção dos resultados mais relevantes.

**Etapa 2: Geração da Resposta com o Gemini:** Com os resultados da recuperação em mãos, o sistema procede à geração da resposta, esta etapa inclui a formatação do contexto dos resultados e do histórico da conversa num *prompt* adequado, que é então enviado ao LLM para gerar uma resposta textual.

**Etapa 3: Tratamento de Casos sem Resultados:** Após a etapa de recuperação, é possível que nenhum livro relevante seja encontrado para a consulta do utilizador. A função `answer_question()` lida com estes casos, garantindo que uma resposta apropriada seja apresentada ao utilizador, em vez de prosseguir para a etapa de geração com um contexto vazio.

**Etapa 4: Atualização do Histórico de Conversa:** De forma a manter o contexto das interações, o histórico da conversa é atualizado após cada pergunta e resposta. A pergunta do utilizador e a resposta gerada pelo sistema são armazenadas, permitindo que conversas subseqüentes sejam contextualizadas.

**Etapa 5: Medição de Desempenho:** Adicionalmente, o desempenho da função é monitorizado através do decorador `measure_execution`. Este decorador regista métricas importantes como o tempo de execução da função, auxiliando na análise e otimização do sistema.

```

@measure_execution
def answer_question(query: str, embeddings, metadata, index, history: List[Dict[str, str]], top_k: int = 10) -> str:
    """Gera resposta com base na busca, incluindo histórico de conversa."""
    print(f"Processando pergunta: {query}")
    try:
        search_results = search(query, index, metadata, top_k=top_k)
        if not search_results:
            print("Nenhum resultado encontrado.")
            response = "Não encontrei livros relevantes para sua consulta."
        else:
            context = format_context(search_results, metadata)
            print("Contexto formatado, gerando resposta com Gemini.")
            print(f"Prompt enviado ao Gemini (primeiros 500 caracteres):\n{context[:500]}...")
            history_text = format_history(history)
            prompt = f"""
            Você é um assistente de biblioteca amigável. Responda à pergunta do usuário com base no contexto abaixo e no histórico de conversa, se relevante.
            Se não houver informação suficiente, seja honesto e sugira algo útil.
            Sempre refira a localização do livro(a biblioteca onde se encontra).

            **HISTÓRICO:**
            {history_text}

            **CONTEXTO:**
            {context}

            **PERGUNTA:**
            {query}

            **RESPOSTA:**
            """
            response = generate_gemini_response(prompt)

        # Atualizar histórico
        history.append({"query": query, "response": response})
        if len(history) > MAX_HISTORY_LENGTH:
            history.pop(0)
        return response
    except Exception as e:
        print(f"Erro ao gerar resposta: {str(e)}")
        response = f"Ocorreu um erro: {str(e)}. Tente novamente ou reformule sua pergunta."
        history.append({"query": query, "response": response})
        if len(history) > MAX_HISTORY_LENGTH:
            history.pop(0)
        return response

```

Figura 31- Fluxo de execução

## 8.6. Interfaces de Interação

Para compreender melhor o sistema em si é importante analisar e sua execução do ponto de vista do utilizador. Esta secção apresenta um panorama do fluxo de interação entre o utilizador e o *chatbot*, desde a entrada da consulta até à saída da resposta.

### 8.6.1. Interface de Consola (perguntar\_ao\_assistente)

A função `perguntar_ao_assistente()` (Figura 32), implementa uma forma simples de interação entre utilizador e sistema através da linha de comandos. Quando executada, esta função inicia um ciclo de interação contínuo, permitindo que o utilizador questione e receba respostas em tempo real.

O primeiro passo da função é exibir uma mensagem de boas-vindas ao utilizador, esta mensagem indica como interagir com o assistente e a palavra-chave para encerrar a secção (*sair*). De seguida, uma lista vazia chamada *history* é inicializada.

Esta lista tem como objetivo armazenar o histórico da conversa, registrando cada pergunta feita pelo utilizador e a sua respetiva resposta.

A interação principal ocorre dentro de um *loop while...True*, que se mantém em execução até que o utilizador decida sair. A cada iteração do *loop*, as seguintes ações são realizadas:

- **Visualização do Histórico:** O histórico da conversa atual é exibido no console. Para cada entrada no histórico, a pergunta do utilizador e os primeiros 100 caracteres da resposta do sistema são apresentados, isto permite ao utilizador recordar o contexto da conversa.
- **Recebimento da Pergunta:** O sistema solicita a introdução da pergunta através da função *input()*. A pergunta inserida é armazenada na variável *pergunta*.
- **Verificação de Comando de Saída:** A entrada do utilizador é convertida para minúsculas e comparada com a palavra 'sair'. Se a entrada for 'sair', uma mensagem de encerramento é exibida e o *loop* é interrompido com o comando *break*, finalizando a execução da função.
- **Processamento da Pergunta:** Se a entrada não for o comando 'sair', a função *answer\_question()* é chamada, esta recebe a pergunta do utilizador, os recursos de recuperação (*embeddings*, metadados e índice) e o histórico da conversa atual como argumentos. Processa a pergunta, recupera informações relevantes e gera uma resposta. Além da resposta, ela também retorna métricas sobre o processo, armazenadas na variável *metrics*.
- **Exibição da Resposta:** A resposta gerada pela função *answer\_question()* é exibida no console para o utilizador.

O *loop* continua permitindo que o utilizador faça múltiplas perguntas numa única sessão, com o histórico da conversa a ser mantido e utilizado para contextualizar as respostas.

```
def perguntar_ao_assistente(embeddings, metadata, index):
    """Permite fazer perguntas ao assistente diretamente no console com histórico."""
    print("Bem-vindo ao Assistente de Biblioteca! Digite sua pergunta ou 'sair' para encerrar.")
    history = []
    while True:
        print(f"\nHistórico de conversa:")
        for i, entry in enumerate(history):
            print(f"Pergunta {i+1}: {entry['query']}")
            print(f"Resposta {i+1}: {entry['response'][:100]}...")
        pergunta = input("\nDigite sua pergunta: ")
        if pergunta.lower() == 'sair':
            print("Encerrando o assistente.")
            break
        response, metrics = answer_question(pergunta, embeddings, metadata, index, history)
        print(f"Resposta: {response}")
```

Figura 32-Interação com o Utilizador

### 8.6.2. Interface Gráfica com Gradio (*chatbot\_interface*)

A função *chatbot\_interface()* (Figura 33), cria uma interface gráfica minimalista utilizando a biblioteca *Gradio* (*gr*). Esta interface facilita a interação dos utilizadores com o *chatbot*, proporcionando uma experiência de conversação intuitiva em formato de chat.

A interface é construída com a classe *gr.ChatInterface*, que abstrai a complexidade da implementação, oferecendo componentes visuais padrão, como a caixa de entrada de mensagens e a área de exibição do histórico da conversa. A instanciação desta classe é feita com os seguintes parâmetros principais:

- ***fn=respond***: Função responsável por processar as mensagens do utilizador e gerar as respostas;
- ***type='messages'***: Define o formato da interface como conversacional, apresentando perguntas e respostas em sequência;
- ***flagging\_mode='manual'***: Permite que os utilizadores sinalizem conversas para revisão, com opções configuradas em *flagging\_options*;
- ***save\_history=True***: Garante que o histórico da conversa seja mantido durante a sessão.

O objeto demo criado representa a aplicação *Gradio* com esta configuração e é devolvido por *chatbot\_interface()*, podendo ser lançado num navegador web.

A função *respond()* constitui o núcleo da interatividade da interface, funcionando como elo entre a entrada do utilizador e a lógica principal do sistema AlbiBooks. Esta função recebe dois argumentos:

- *message*: A mensagem de texto inserida pelo utilizador;
- *chat\_history*: Lista que mantém o histórico da conversa durante a sessão.

No seu interior, decorrem as seguintes ações:

- **Processamento da Pergunta**: A função *answer\_question()* é chamada com os dados necessários, tratando a consulta e gerando a resposta.
- **Simulação de Digitação (Opcional)**: Um ciclo percorre os caracteres da resposta, com um pequeno atraso (*time.sleep(0.01)*) entre cada um, simulando o efeito de “digitação”. O uso de *yield* transforma *respond()* num gerador, permitindo exibir a resposta de forma progressiva na interface.

- **Apresentação da Resposta:** A resposta é devolvida caractere a caractere e apresentada na área de chat, juntamente com a mensagem original do utilizador, atualizando visualmente o histórico da conversa.

```
def chatbot_interface(embeddings, metadata, index):
    """Cria uma interface Gradio mínima para o assistente de biblioteca."""
    history = []

    def respond(message, chat_history):
        # Chame a função do seu chatbot para gerar a resposta
        response, metrics = answer_question(message, embeddings, metadata, index, history)

        # Simule um atraso para dar a impressão de que o chatbot está "pensando"
        for i in range(len(response)):
            time.sleep(0.01)
            yield response[: i + 1]

    demo = gr.ChatInterface(
        respond, # Use a função modificada aqui
        type='messages',
        flagging_mode="manual",
        flagging_options=["Like", "Spam", "Inappropriate", "Other"],
        save_history=True,
    )
    return demo # Return the demo object
```

Figura 33- Interface Gráfica (Gradio)

Em resumo, `chatbot_interface()` é responsável pela configuração e apresentação da interface, enquanto `respond()` executa o processamento das mensagens e a geração das respostas, integrando de forma eficiente a interface gráfica com o motor do assistente AlbiBooks.

## 8.7. Configuração do Ambiente e Execução

Esta seção descreve o processo de execução e lançamento da interface gráfica do utilizador (GUI) do *chatbot* AlbiBooks. O código presente nesta seção, dentro do bloco `if __name__ == "__main__":` (Figura 34), demonstra como a função `chatbot_interface()` é utilizada para criar e iniciar a aplicação *Gradio*, tornando o *chatbot* acessível aos utilizadores através de um navegador web.

```
if __name__ == "__main__":
    demo = chatbot_interface(embeddings, metadata, index) # Call the function and assign the result to demo
    demo.launch(debug = True)
```

Figura 34-Execução

Após a execução da aplicação, é disponibilizada uma interface simples e funcional, onde o utilizador pode introduzir perguntas em linguagem natural e obter respostas baseadas no conteúdo indexado. Esta interface permite testar e utilizar o sistema de forma intuitiva, sem necessidade de conhecimentos técnicos.

A Figura 35 apresenta um exemplo real da interface em funcionamento, ilustrando a interação entre o utilizador e o sistema.

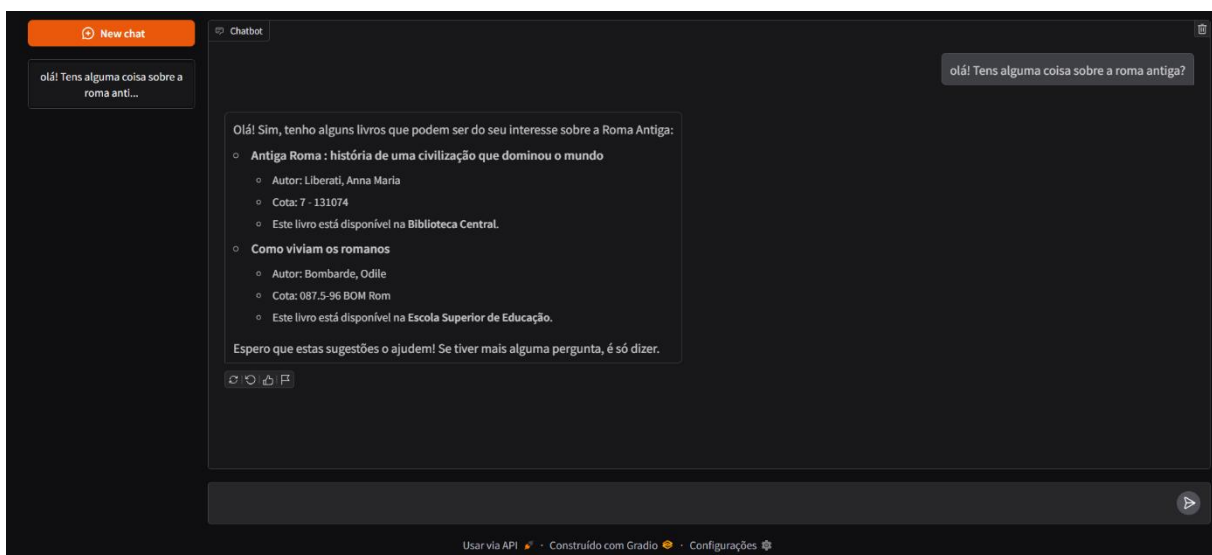


Figura 35-Interface (gradio) do AlbiBooks

## 9. Comparação entre Fases do Projeto

Este capítulo descreve e compara as diferenças entre a primeira fase deste projeto (referenciada no Capítulo 5) e a segunda fase (referenciada no Capítulo 6). A fase final corresponde a uma versão mais robusta e evoluída do sistema. O principal objetivo é detalhar a evolução do sistema de *chatbot* para bibliotecas, focando nos aspetos técnicos e nas decisões de desenvolvimento tomadas ao longo do percurso.

### 9.1 Visão Geral da Evolução da Arquitetura

A transição da Fase I para a Fase II representou uma evolução significativa na arquitetura do sistema. Na Fase I, a abordagem era de um sistema unificado e não modular de RAG, onde o *pipeline* era maioritariamente gerido pela biblioteca *LangChain*. Nessa fase, o carregamento de um subconjunto de dados do *dataset* iniciava o fluxo de dados, e o foco principal eram as descrições textuais dos livros. Por contraste, na Fase II a arquitetura foi reorganizada para permitir maior independência entre os componentes, passando a integrar APIs externas e a permitir maior controlo sobre o pipeline de dados. Esta reestruturação facilitou a manutenção, a expansão futura e uma melhor performance geral.

Em síntese, a diferença entre as duas fases reflete uma evolução técnica do projeto, guiada por decisões informadas com base nas limitações observadas na fase inicial e nas necessidades práticas de um sistema mais completo e adaptável ao contexto real de uma biblioteca. Na Tabela 2, é possível observar as diferenças entre fases.

Tabela 2- Diferenças entre Projeto I e Projeto II

Aspeto	Fase I	Fase II
<b>Arquitetura</b>	Sistema não modular (notebook único)	Arquitetura modular (notebooks separados)
<b>Ingestão de Dados</b>	Subconjunto do dataset Goodreads (500 registos)	Ficheiro local com +18.000 registos reais da biblioteca do IPCB
<b>Modelo de Embedding</b>	thenlper/gte-small local com LangChain	mistral-embed via API da Mistral
<b>Indexação</b>	Indexação FAISS local, não persistente	FAISS persistente com ficheiros .faiss, .pkl e .npy

Aspeto	Fase I	Fase II
<b>LLM Utilizado</b>	Modelos open-source (Zephyr, LLaMA, etc.)	Gemini 2.5 Flash via API Google
<b>Pipeline</b>	Orquestrado por LangChain	Separação clara entre indexação e conversação
<b>Interação com o Utilizador</b>	Via Consola	Interface gráfica com Gradio
<b>Nº de Perguntas para Teste</b>	10	100
<b>Idioma das Perguntas</b>	Inglês	Português
<b>Persistência de Dados</b>	Sem persistência entre sessões	Artefactos guardados no Google Drive

A evolução da Fase I para a Fase II trouxe uma arquitetura mais modular e robusta, com melhorias significativas:

**Separação da indexação:** O processo de indexação foi desacoplado do chatbot, permitindo execução 'offline', melhor modularidade, manutenção facilitada e reutilização dos artefactos de indexação.

**Uso de modelos de IA via API:** A transição para modelos de *embedding* e *LLMs* baseados em serviços externos permitiu acesso a modelos mais potentes, reduzindo a carga computacional local, embora introduza dependência de rede.

**Abandono do Langchain:** O pipeline RAG foi reimplementado diretamente em *Python*, oferecendo maior controlo e transparência no fluxo de dados, apesar de exigir mais trabalho manual.

**Interface gráfica com Gradio:** Implementação de UI interativa que melhora a experiência do utilizador e permite recolha de feedback.

**Gestão do histórico da conversação:** Permite manter contexto ao longo de múltiplas trocas, tornando a interação mais natural.

**Persistência do índice no Google Drive:** Garantia de salvaguarda dos dados entre sessões, facilitando o fluxo de trabalho.

**Fonte de dados específica:** Substituição do dataset genérico por um ficheiro JSON representativo do catálogo da biblioteca, aumentando a relevância do chatbot.

**Mecanismos de robustez para APIs:** Inclusão de lógicas de retry, backoff e timeout para aumentar a fiabilidade perante falhas temporárias ou limites das APIs.

## 9.2. Estratégias de Recuperação e Embeddings

Entre a Fase I e a Fase II do projeto, verificou-se uma evolução clara na abordagem à representação e recuperação de informação. Na Fase II, a substituição do modelo de *embedding* local por um serviço via API trouxe *embeddings* com maior dimensão e potencial semântico, permitindo uma representação mais completa dos dados bibliográficos. Ao invés do *chunking* de descrições, agora o sistema utiliza um *embedding* consolidado para cada livro, agregando múltiplos metadados, o que melhora a precisão nas consultas e respostas. A mudança para embeddings de livros completos e a utilização da distância L2 (com a subsequente transformação em *relevance\_score*) representam uma tentativa de alinhar a recuperação com a natureza multifacetada dos dados bibliográficos.

Adicionalmente, a métrica de similaridade mudou de cosseno para distância Euclidiana, o que implica cuidados na normalização dos vetores. A construção do contexto para o modelo de linguagem também foi aprimorada, passando de uma simples concatenação de texto para uma formatação estruturada e clara dos metadados, facilitando a geração de respostas mais relevantes e contextualizadas. Estas alterações tornaram o sistema mais robusto, modular e escalável, embora introduzam a necessidade de gestão cuidadosa das chamadas à API e dos custos associados.

Estas mudanças refletem uma evolução não apenas tecnológica, mas também metodológica, ao integrar uma visão mais abrangente sobre os dados bibliográficos. A transição para *embeddings* consolidados, a adoção da distância Euclidiana e a formatação estruturada do contexto são componentes que, em conjunto, fortalecem a relevância e precisão das respostas geradas pelo sistema. Outro aspeto relevante prende-se com a quantidade e densidade do contexto construído: enquanto na Fase I o parâmetro `num_docs_final` controlava o número de *chunks*, na Fase II é o parâmetro `top_k` que define o número de livros completos cujos metadados são formatados para o contexto. Apesar de o número de unidades poder ser semelhante, a densidade informacional dos *embeddings* consolidados por livro é consideravelmente superior, resultando num contexto mais rico e eficaz.

### 9.3. Análise do Desempenho

A comparação entre as fases do projeto permite uma análise fundamentada nas mudanças arquiteturais e técnicas implementadas.

No que respeita ao desempenho, a passagem para uma arquitetura modular e a adoção de modelos via API resultaram numa redução significativa da latência observada na Fase I, que estava condicionada ao processamento local. Esta nova configuração permite escalabilidade e melhor aproveitamento dos recursos de computação na nuvem, otimizando o tempo de resposta.

Em termos de gestão de *tokens*, o novo sistema beneficia de estratégias que controlam de forma mais eficaz o contexto enviado para o modelo de linguagem. A substituição do *chunking* por *embeddings* consolidados e a parametrização do número de documentos relevantes para a construção do contexto promovem um uso mais eficiente dos *tokens*, reduzindo custos e aumentando a coerência das respostas.

Quanto à precisão, a incorporação de *embeddings* que agregam múltiplos metadados e a melhoria na formatação do contexto fornecem ao modelo uma visão mais rica e estruturada da informação, resultando em respostas mais alinhadas com as necessidades do utilizador. Além disso, a robustez introduzida no acesso às APIs minimiza erros de comunicação que poderiam prejudicar a qualidade da interação.

Esta análise qualitativa sugere que as decisões tomadas para a Fase II oferecem um sistema mais robusto, eficiente e apto para lidar com as exigências práticas do contexto bibliotecário, preparando o terreno para futuras avaliações quantitativas que confirmarão estes ganhos.

#### 9.3.1 Análise Qualitativa do Desempenho

Esta subsecção apresenta uma avaliação baseada em casos reais do comportamento do sistema, destacando erros, respostas mais relevantes e perspectivas para melhorias. O conjunto de perguntas utilizado durante os testes foi gerado com recurso a ferramentas de Inteligência Artificial, tendo por base o catálogo bibliográfico da biblioteca do IPCB.

**Erros significativos identificados:** Após 100 testes realizados, foi detetado um erro significativo. Foi colocada uma pergunta sobre obras de um determinado coautor, o sistema respondeu que não existia nenhuma obra correspondente, quando, na realidade, existia na base de dados um livro com esse coautor. Este foi o único erro relevante encontrado. À primeira vista, poderia sugerir uma limitação na forma como o sistema lida com campos de coautoria; no entanto, noutra pergunta semelhante, o

sistema identificou corretamente o coautor, o que indica que o problema poderá estar relacionado com a formulação específica da pergunta ou com os dados em causa.

**Recomendações úteis mesmo sem correspondência direta:** Durante o processo de teste observou-se que, em situações em que o sistema não encontrava exatamente aquilo que era solicitado, este continua a apresentar sugestões relacionadas. Ou seja, em vez de dar uma resposta em branco, o *chatbot* recomenda obras com temáticas semelhantes, ajudando o utilizador a encontrar conteúdo relevante mesmo que não coincida de forma exata com o pedido.

**Formatação inconsistente nas respostas geradas:** Num dos testes, o sistema respondeu a uma pergunta sobre periódicos científicos em educação especial, referindo uma obra com o título “Lista das publicações periódicas existentes / Escola Superior de Educação de Coimbra”. Embora a instituição mencionada não corresponda diretamente ao contexto do *chatbot* (IPCB), esta informação encontra-se corretamente no catálogo, fazendo parte do título da obra.

O verdadeiro problema observado está na formatação da resposta: o sistema de recuperação devolve os 10 resultados mais relevantes com etiquetas automáticas (“Livro 1”, “Livro 2”, etc.). O modelo de linguagem inclui essas marcas textuais diretamente na resposta final (ex: “Livro 2: ...”) Figura 36, sem as reestruturar, o que pode gerar confusão para o utilizador.

Isto sugere a necessidade de rever o *prompt* utilizado na construção do contexto enviado ao modelo, removendo essas etiquetas antes da geração da resposta.

```
--- Pergunta 25 ---
ID da Pergunta: q25_educacao_periodicos_especial
Pergunta: Quais periódicos científicos sobre 'educação especial' a Escola Superior de Educação subscreve ou possui?

Resposta: Com base nas informações fornecidas, não foi encontrado um periódico científico específico sobre "educação especial" diretamente listado como uma subscrição ou posse da Escola Superior de Educação.

No entanto, a **Escola Superior de Educação** possui o livro:
* **Livro 2:** "Lista das publicações periódicas existentes / Escola Superior de Educação de Coimbra", Cota: A 018 ESC Lis. Este é um catálogo das publicações periódicas que a biblioteca possui e pode ser um recurso útil para verificar se há alguma subscrição relevante sobre o tema "educação especial".

Sugiro que consulte a "Lista das publicações periódicas existentes" diretamente na **Escola Superior de Educação** para verificar se algum periódico específico sobre educação especial está incluído.

Métricas:
Tempo de Execução: 8.7381 segundos
Memória Usada: 0.06 MB
Memória Máxima: 0.31 MB
-----
```

Figura 36-Pergunta ID 25 (Inconsistência)

**Exemplo Positivo: Pergunta ID 82:** A Pergunta ID 82 (Figura 37), produziu uma resposta particularmente relevante, onde o sistema recomendou duas obras

relacionadas com a Segunda Guerra Mundial, incluindo um livro que não estava claramente assinalado como fotografia, mas que se revelou relevante. Esta resposta demonstrou a capacidade do *chatbot* de interpretar o conteúdo semântico do título e fornecer uma sugestão útil, revelando um comportamento inteligente mesmo em casos mais complexos.

```

--- Pergunta 82 ---
ID da Pergunta: q82_central_historia_segunda_guerra_fotos
Pergunta: A Biblioteca Central tem livros com fotografias da 'Segunda Guerra Mundial'?

Resposta: Sim, a Biblioteca Central possui livros que podem conter fotografias relacionadas à Segunda Guerra Mundial:

* **The armed forces of world war II**
  * Autor: Mollo, Andrew
  * Este livro, que aborda as forças armadas da Segunda Guerra Mundial e vestuário militar, é provável que inclua ilustrações ou fotografias.
  * Localização: Biblioteca Central
  * Cota: 391 - 1326-04

* **Imagens e memórias da Escola Técnica de Enfermeiras : 1940-1988**
  * Autor: Corrêa, Beatriz de Mello
  * Embora o foco principal seja a história da enfermagem, o título indica a presença de imagens ("Imagens e memórias") e o período de 1940-1988 abrange a Segunda Guerra Mundial, o que significa que pode conter fotografias desse período.
  * Localização: Biblioteca Central
  * Cota: 616 - 153999

Sugiro que verifique estes títulos diretamente na Biblioteca Central para confirmar o tipo específico de conteúdo visual que procura.

```

Figura 37-Pergunta ID 82

### 9.3.2 Análise Quantitativa do Desempenho

Para complementar a análise qualitativa apresentada, será incluída uma análise quantitativa que compara métricas essenciais entre as duas fases do projeto. Os gráficos seguintes, apresentarão dados como Tempo de Execução e Uso de Memória referentes a fase II do projeto. De forma a permitir visualizar de forma clara e objetiva as melhorias resultantes da evolução da arquitetura e das escolhas técnicas implementadas, será realizada uma comparação entre os dados obtidos nesta fase (fase II) e os dados obtidos na fase I. Esta comparação fornecerá uma visão detalhada do impacto das alterações e servirá como base para futuras otimizações do sistema.

Entretanto é importante salientar que para os testes da fase II do projeto foram utilizadas 100 perguntas de teste, de forma a ter mais robustez nos resultados.

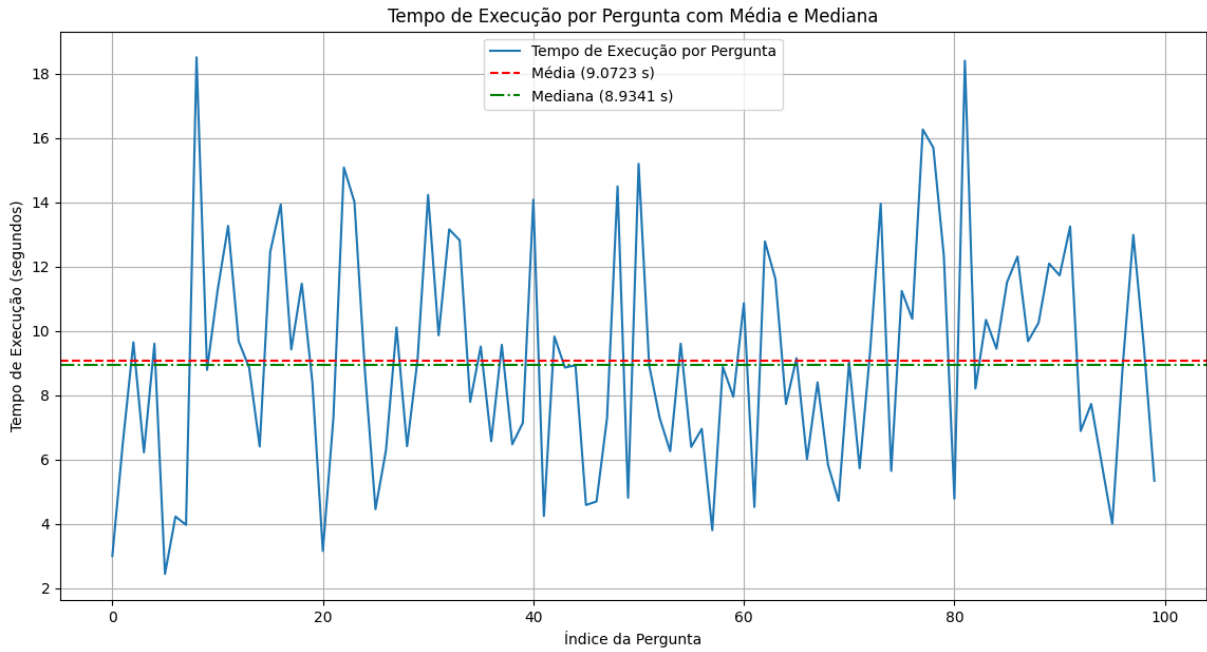


Figura 38- Tempo de Execução Projeto II

Em comparação com os resultados obtidos na fase I (Figura 19), o tempo de resposta obtido pelo modelo da fase II (Figura 38Figura 39) apresentou tempos de resposta significativamente mais rápidos.

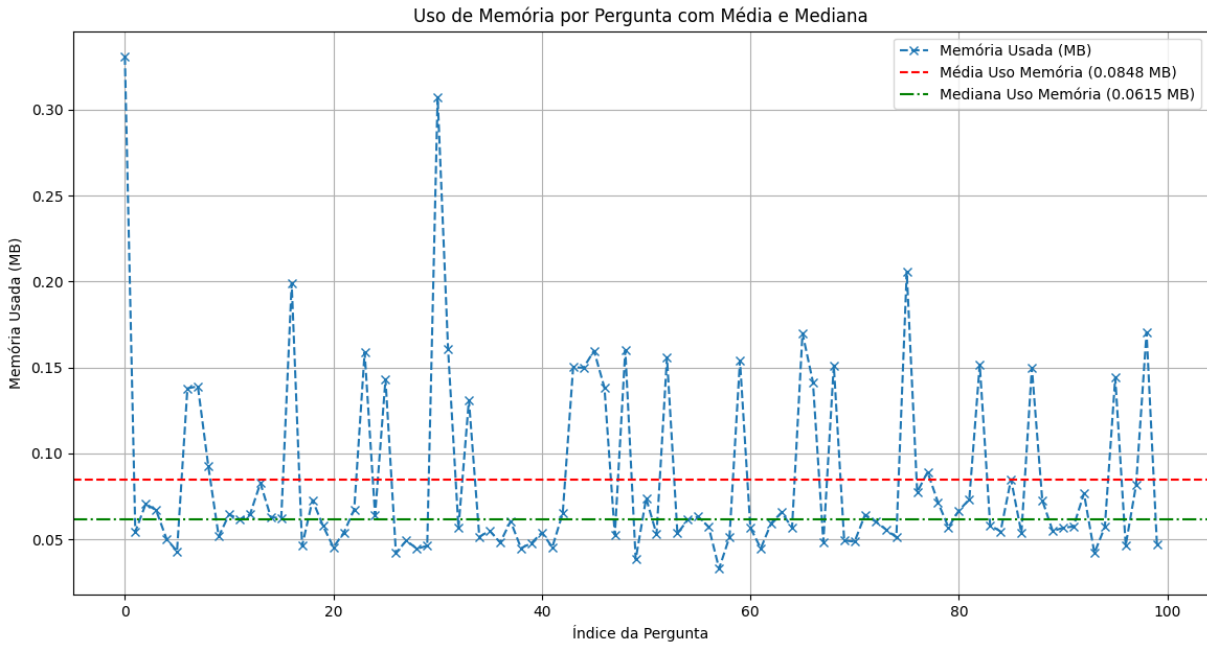


Figura 39-Uso de Memória Projeto II

Em comparação com os resultados obtidos na fase I referente ao uso de memória, Figura 20, na fase II pode-se observar que houve alguns picos de uso de memória que teve performance pior que o modelo de melhor eficiência da fase I. Entretanto, em modo geral, em média o uso de memória do modelo da fase II (Figura 39), esteve abaixo do melhor modelo da primeira fase do projeto, o que atesta a eficácia da arquitetura com API.

## 9.4 Alterações nos Dados

A eficácia de um sistema RAG depende fortemente da natureza e do tratamento dos dados subjacentes. No projeto AlbiBooks, entre as fases I e II, registaram-se mudanças significativas relativas à fonte, volume, qualidade e preparação dos dados para a geração de *embeddings*, que serão detalhadas a seguir.

A Tabela 3, sintetiza os principais aspetos relacionados com a fonte, volume, qualidade dos dados, estratégias de *chunking* e normalização aplicadas em cada etapa. Esta comparação destaca a evolução significativa na preparação dos dados, fundamental para a melhoria do desempenho do sistema RAG.

Tabela 3- Alterações nos Dados entre Projeto I e Projeto II

Aspeto	Fase I	Fase II
<b>Fonte</b>	<i>Dataset</i> público <i>Goodreads</i> (500 registos)	Catálogo da biblioteca do IPCB (18.266 registos)
<b>Volume</b>	Limitado a 500 registos	Muito maior, 18.266 registos
<b>Qualidade dos Dados</b>	Genérico	Mais relevante e específica, inclui cota e localização
<b>Estratégia de <i>Chunking</i></b>	Divisão em <i>chunks</i> de 512 <i>tokens</i>	Documento único por livro, com todos os metadados

### 9.4.1 Fonte, Volume e Qualidade dos Dados

Na Fase I, os dados provinham do *dataset* público, disponível na plataforma Hugging Face. Este conjunto continha 500 registos limitados, possuindo um carácter

genérico e pouca especificidade para um contexto bibliotecário real. Já na Fase II, a fonte mudou para um ficheiro JSON local contendo um catálogo real de uma biblioteca, com 18.266 registos, incluindo metadados essenciais como localização física e tipo de item. Esta mudança representa um avanço substancial na relevância e aplicabilidade dos dados ao *chatbot*.

#### 9.4.2 Estratégia de Chunking e Formatação para Embedding

Enquanto na Fase I se utilizou uma abordagem de fragmentação (*chunking*) dos textos das descrições com o *RecursiveCharacterTextSplitter* para gerar *embeddings* de pequenos segmentos, na Fase II optou-se por consolidar todos os metadados relevantes de cada livro num único texto formatado para gerar um *embedding* integrado. Esta alteração visa capturar uma representação vetorial mais integral do item bibliográfico.

### 9.5 Ajustes no Fluxo de Interação com o Utilizador

A experiência do utilizador em aplicações conversacionais é fundamental para garantir a adoção e eficácia do sistema. No projeto AlbiBooks, observou-se uma evolução significativa na interação entre a Fase I e a Fase II, passando de uma abordagem técnica para uma mais centrada no utilizador.

#### 9.5.1 Interface e Modo de Interação

- **Fase I:** A interação era feita exclusivamente através da consola, com submissão de perguntas via execução de células de código Python e retorno das respostas em texto simples na consola ou em ficheiros JSON. Esta abordagem, embora adequada para programadores, não é intuitiva para utilizadores finais.
- **Fase II:** Introduziu-se uma interface gráfica baseada no Gradio, que oferece uma caixa de texto para perguntas e uma área de diálogo que exhibe o histórico da conversação de forma clara. Adicionalmente, as respostas do *chatbot* são apresentadas com efeito de escrita progressiva para aumentar a naturalidade da interação. A interface Gradio facilita testes e demonstrações sem necessidade de programar uma interface robusta.

-

### 9.5.2 Gestão do Histórico de Conversa

- **Fase I:** Cada pergunta era processada isoladamente, sem gestão de histórico no código, exigindo implementação manual para manter contexto.
- **Fase II:** Implementou-se uma gestão explícita do histórico através de uma lista Python que guarda as últimas 10 mensagens, formatada para ser incluída no *prompt* enviado ao modelo Gemini. Esta gestão permite uma melhor compreensão de perguntas de seguimento e maior coerência nas respostas, embora aumente a latência pela maior quantidade de *tokens*.
- 

### 9.5.3 Orientação do Comportamento do *Chatbot*

- **Fase I:** O comportamento do LLM baseava-se unicamente no seu treino prévio e no conteúdo recuperado, com instruções básicas para o estilo e tom na língua inglesa.
- **Fase II:** O *prompt* foi enriquecido com instruções que definem o *chatbot* como um assistente de biblioteca amigável, honesto perante falta de informação e com foco em fornecer localização dos livros. Esta engenharia de *prompt* contribuiu para uma persona mais definida e respostas alinhadas com as expectativas do utilizador. Nesta fase o *prompt* já estava em português.
- 

## 9.6 Ferramentas e Bibliotecas Utilizadas

A seleção de ferramentas e bibliotecas é crucial para um projeto de software. No AlbiBooks, a evolução da Fase I para a Fase II envolveu uma mudança significativa no conjunto tecnológico, refletindo o amadurecimento dos objetivos, a complexidade crescente e a necessidade de maior robustez e desempenho.

A Tabela 4, apresenta uma síntese das principais ferramentas utilizadas em cada fase e as justificativas para as alterações implementadas.

Tabela 4-Diferenças do Conjunto Tecnológico de Projeto I e Projeto II

<b>Ferramenta/Biblioteca</b>	<b>Fase I</b>	<b>Fase II</b>	<b>Justificativa da Mudança</b>
<b>Langchain</b>	Framework de orquestração e abstração para pipeline RAG	Não utilizado	Removido para maior controlo direto das APIs e lógica customizada; maior flexibilidade e personalização.
<b>Transformers (Hugging Face)</b>	Modelos locais (Zephyr 7B, embeddings thenlper/gte-small)	Substituído por APIs (Mistral, Gemini)	Permite acesso a modelos estado da arte, redução da carga computacional local, maior desempenho e atualização constante.
<b>Faiss-CPU</b>	Índice vetorial local para embeddings	Mantido	Ferramenta robusta e eficiente para busca por similaridade que permanece adequada em ambas as fases.
<b>Torch / accelerate / bitsandbytes</b>	Suporte local para deep learning e quantização	Não utilizado	Inutilizado com a migração para execução via APIs externas, eliminando a necessidade de recursos locais.
<b>Google-generativeai SDK</b>	Não utilizado	Introduzido para interagir com Gemini	Integração direta com a API Gemini para geração de linguagem e gestão do contexto.

Ferramenta/Biblioteca	Fase I	Fase II	Justificativa da Mudança
<b>Requests</b>	Não utilizado	Utilizado para chamadas REST Mistral	Necessário para integração com API de <i>embeddings</i> Mistral.
<b>Gradio</b>	Não utilizado	Utilizado para interface gráfica	Facilita a construção rápida de interfaces interativas, melhorando usabilidade e demonstração do sistema.
<b>Numpy / Pickle</b>	Não utilizado	Utilizado para persistência de dados	Essencial para armazenar eficientemente <i>embeddings</i> e metadados, suportando indexação offline.
<b>Utilitários Python (json, time, threading, tracemalloc, etc.)</b>	Utilizados para manipulação de dados e medição de desempenho	Mantidos e expandidos para novas necessidades	Suporte básico e avançado para funcionalidades internas, performance e gestão de chamadas API.

Estas mudanças permitiram ao projeto passar de um protótipo local e experimental para uma aplicação escalável e alinhada com as necessidades reais do utilizador, enquanto explora os benefícios das mais recentes tecnologias de IA via APIs.

## 9.7. Justificações Técnicas para as Principais Decisões

A transição da Fase I para a Fase II do projeto AlbiBooks envolveu decisões arquiteturais e tecnológicas fundamentais para superar limitações anteriores e otimizar robustez, desempenho e experiência do utilizador.

### 9.7.1. Mudança de Modelos de *Embedding*

O modelo local utilizado na Fase I era eficiente para atestar a viabilidade da construção do *chatbot*, mas limitado para o domínio especializado da biblioteca e para o aumento do volume de dados. Além disso, a indexação local era computacionalmente pesada. O modelo via API utilizado na Fase II oferece melhor capacidade semântica e desempenho, resultando numa recuperação de informação mais precisa e relevante para metadados bibliotecários. Reduz a carga computacional local e simplifica a manutenção. Em contrapartida, torna o sistema dependente de serviços externos, sujeito a latências de rede, custos associados à API, além da necessidade de segurança na gestão de chaves. No entanto, esta decisão resultou em melhorias significativas na qualidade dos *embeddings* e na escalabilidade da indexação, aumentando a relevância do sistema.

### 9.7.2. Mudança de LLMs

Os modelos da Fase I exigiam muitos recursos e apresentavam latências elevadas, com capacidades limitadas para raciocínios complexos e compreensão contextual prolongada. O modelo Gemini, utilizada na Fase II, traz melhor fluidez, coerência, raciocínio e seguimento de instruções detalhadas, elevando a qualidade das respostas e simplificando a infraestrutura. Esta alteração resultou em respostas mais precisas e contextuais, experiência conversacional mais natural, e viabilizou um *prompt engineering* sofisticado.

### 9.7.3. Separação do Processo de Indexação

Na Fase I, a indexação e a conversação estavam no mesmo fluxo, tornando o processo lento e pouco modular. Com a indexação 'offline' da Fase II, é possível obter um arranque rápido do *chatbot*, maior modularidade, manutenção facilitada e gestão independente dos artefactos (índice FAISS, *embeddings*, metadados). Embora seja necessário gerir o fluxo e armazenamento dos artefactos entre processos, com atualização manual da indexação, esta alteração trouxe maior eficiência, escalabilidade e robustez operacional.

### 9.7.4. Alteração da Fonte e Estrutura dos Dados para *Embeddings*

A Fase I utilizava um *dataset* genérico do Goodreads focado em descrições, pouco relevante para o contexto bibliotecário. A utilização do catálogo específico da biblioteca

com *embeddings* únicos por livro na Fase II permitiu melhor semântica e utilidade prática, aumentando a relevância e precisão do *chatbot* para respostas específicas da biblioteca.

#### 9.7.5. Introdução de Interface Gráfica (Gradio) e Gestão de Histórico

A Fase I possuía uma interação puramente através da consola e sem histórico, limitando a conversação a interações únicas. A implementação da interface Gradio na Fase II facilita o uso por utilizadores não técnicos e permite demonstração interativa. A gestão do histórico (armazenamento e envio das últimas interações) permite diálogos mais naturais e contextualizados. Contudo, o histórico aumenta o número de *tokens* enviados, impactando custo e latência. Embora o Gradio impeça uma personalização avançada da interface, esta decisão melhorou a experiência do utilizador e tornou os diálogos mais sofisticados.

#### 9.7.6. Abandono do Langchain e Implementação Direta da Lógica RAG

O *Langchain* facilitou a prototipagem na Fase I, mas limitava a flexibilidade em personalizações específicas e interações com APIs. A implementação direta da lógica RAG permite controlo granular sobre pedidos, respostas, *prompts* e histórico, facilitando otimizações, robustez e transparência. Embora a quantidade de código a desenvolver tenha aumentado, a alteração resultou numa integração mais fina e flexível com APIs, maior robustez e melhor desempenho do sistema.

### 9.8. Conclusões

O projeto AlbiBooks evoluiu significativamente da Fase I para a Fase II, passando de um protótipo básico para um sistema mais robusto e especializado. A Fase I utilizou ferramentas *open-source* e uma arquitetura integrada, mas dependia de dados genéricos e tinha limitações de desempenho e usabilidade.

Na Fase II, a adoção de APIs especializadas para *embeddings* e geração de texto, a modularização do sistema, o uso de dados reais da biblioteca e a introdução de uma interface gráfica melhoraram a precisão, eficiência e experiência do utilizador. A implementação direta da pipeline RAG também aumentou o controlo e a robustez do sistema.

Assim, o AlbiBooks consolidou-se como uma solução eficaz para consultas bibliotecárias, demonstrando avanços importantes na aplicação de IA generativa e recuperação de informação para um problema real.

## 10. Conclusão

O projeto AlbiBooks foi desenvolvido ao longo de duas fases distintas, sendo a fase I correspondente ao Projeto I e a fase II ao Projeto II. O objetivo principal foi criar um *chatbot* inteligente, baseado em técnicas de NLP, capaz de interagir com a base de dados de uma biblioteca e apoiar o utilizador na pesquisa de livros e conteúdos.

Na primeira fase, foi realizada uma revisão sistemática da literatura sobre *chatbots* em bibliotecas e tecnologias associadas, utilizando a metodologia PRISMA. Com base nessa investigação, foi desenvolvido um protótipo funcional com um pipeline RAG, recorrendo a modelos de linguagem locais (*DistilGPT2*, *LLaMA* e *Zephyr*). Esta abordagem permitiu validar a viabilidade do projeto, bem como revelar várias limitações, nomeadamente ao nível do tempo de execução, da escalabilidade e do consumo de recursos computacionais.

Na segunda fase, essas limitações foram endereçadas com a reestruturação completa do sistema. Foram adotados modelos via API (como Gemini) e *embeddings* mais eficientes (*mistral-embed*), bem como introduzido um *dataset* personalizado recolhido por *web scraping*. A nova arquitetura do pipeline mostrou-se mais leve, escalável e adequada a um uso real, preparando o sistema para uma possível integração futura com plataformas externas.

### 10.1. Trabalho Futuro

Apesar dos avanços alcançados, o projeto AlbiBooks apresenta várias oportunidades de desenvolvimento futuro. Uma das prioridades será a realização de testes com utilizadores reais, de forma a avaliar a usabilidade e a eficácia do *chatbot* em contextos práticos. A integração do sistema com um portal web institucional será também um passo importante para garantir a aplicabilidade da solução em ambientes reais. Para além disso, será relevante melhorar a personalização das respostas, através da incorporação de perfis de utilizador e preferências de leitura, o que poderá aumentar significativamente a utilidade do sistema.

Outra linha de evolução passa pela disponibilização de informações para além daquelas que se encontram na base de dados atual, permitindo que o *chatbot* recorra a conteúdos da internet como complemento às informações indexadas. Desta forma, o sistema poderá fornecer respostas mais completas sem depender exclusivamente do conteúdo previamente carregado.

Adicionalmente, seria útil desenvolver um sistema de CRUD (*Create, Read, Update, Delete*) para a base de dados vetorial, permitindo a atualização constante do

conteúdo indexado de forma mais simples e eficiente. Atualmente, o sistema não possui uma funcionalidade que facilite essa manutenção regular, o que pode limitar a escalabilidade e a fiabilidade da informação fornecida ao utilizador.

Do ponto de vista técnico, propõe-se ainda aprofundar a avaliação do desempenho do sistema, utilizando métricas automáticas que permitam medir a precisão e a relevância das respostas.

Por fim, será vantajoso manter o sistema atualizado e explorar a utilização de LLMs *open source* mais recentes, desde que compatíveis com os recursos computacionais disponíveis, contribuindo assim para uma solução mais autónoma e economicamente sustentável.

## 11.Referências


- [1] “(5) 1956: O Marco de Dartmouth e o Nascimento Oficial da Inteligência Artificial | LinkedIn.” Accessed: Jan. 10, 2025. [Online]. Available: <https://www.linkedin.com/pulse/1956-o-marco-de-dartmouth-e-nascimento-oficial-da-luc%C3%ADola-coelho-xc18f/>
- [2] “Inteligência Artificial – Observador.” Accessed: Jan. 10, 2025. [Online]. Available: <https://observador.pt/explicadores/inteligencia-artificial/>
- [3] “What Is Artificial Intelligence (AI)? | IBM.” Accessed: Jan. 10, 2025. [Online]. Available: <https://www.ibm.com/think/topics/artificial-intelligence>
- [4] “O que é Machine Learning (ML)? - I School Online.” Accessed: Jan. 16, 2025. [Online]. Available: <https://ischoolonline.berkeley.edu/blog/what-is-machine-learning/>
- [5] “Difference Between Supervised, Unsupervised, & Reinforcement Learning | NVIDIA Blog.” Accessed: Jan. 16, 2025. [Online]. Available: <https://blogs.nvidia.com/blog/supervised-unsupervised-learning/>
- [6] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature* 2015 521:7553, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.
- [7] “What is deep learning? | Cloudflare.” Accessed: Jan. 23, 2025. [Online]. Available: <https://www.cloudflare.com/learning/ai/what-is-deep-learning/>
- [8] “Google.” Accessed: Jan. 16, 2025. [Online]. Available: <https://www.google.com/>
- [9] “What is a Neural Network? | IBM.” Accessed: Jan. 16, 2025. [Online]. Available: <https://www.ibm.com/think/topics/neural-networks>
- [10] “O que é Gen AI? IA Generativa Explicada | TechTarget.” Accessed: Jan. 10, 2025. [Online]. Available: <https://www.techtarget.com/searchenterpriseai/definition/generative-AI>
- [11] “O que é um chatbot? | IBM.” Accessed: Jan. 03, 2025. [Online]. Available: <https://www.ibm.com/think/topics/chatbots>
- [12] “O que é um chatbot? - Explicação sobre chatbots de IA - AWS.” Accessed: Jan. 04, 2025. [Online]. Available: <https://aws.amazon.com/pt/what-is/chatbot/>
- [13] “ChatGPT | OpenAI.” Accessed: Jan. 10, 2025. [Online]. Available: <https://openai.com/chatgpt/overview/>
- [14] M. R. Douglas, “Large Language Models,” Jul. 2023, Accessed: Jan. 26, 2025. [Online]. Available: <http://arxiv.org/abs/2307.05782>

- [15] “Introducing Gemini: Google’s most capable AI model yet.” Accessed: Jan. 10, 2025. [Online]. Available: <https://blog.google/technology/ai/google-gemini-ai/#sundar-note>
- [16] H. Naveed *et al.*, “A Comprehensive Overview of Large Language Models”.
- [17] “Escolhendo um modelo de incorporação | Pinecone.” Accessed: May 03, 2025. [Online]. Available: <https://www.pinecone.io/learn/series/rag/embedding-models-rundown/>
- [18] “O que é um modelo de incorporação? | Coursera.” Accessed: May 03, 2025. [Online]. Available: <https://www.coursera.org/articles/embedding-model>
- [19] “O que é geração aumentada de recuperação (Retrieval-Augmented Generation, RAG)?” Accessed: Jan. 10, 2025. [Online]. Available: <https://neuralmind.ai/2024/03/05/o-que-e-geracao-aumentada-de-recuperacao-retrieval-augmented-generation-rag/>
- [20] “Retrieval Augmented Generation (RAG): A Complete Guide - WEKA.” Accessed: Jan. 30, 2025. [Online]. Available: <https://www.weka.io/learn/guide/ai-ml/retrieval-augmented-generation/>
- [21] “The PRISMA statement.” Accessed: Oct. 25, 2024. [Online]. Available: <http://www.prisma-statement.org/>
- [22] “Scopus.”
- [23] “ACM Digital Library.” Accessed: Dec. 03, 2024. [Online]. Available: <https://dl.acm.org/>
- [24] “Mendeley .” Accessed: Oct. 25, 2024. [Online]. Available: <https://www.mendeley.com/search/>
- [25] M. Chase, “Academic Libraries Can Develop AI Chatbots for Virtual Reference Services with Minimal Technical Knowledge and Limited Resources,” *Evid Based Libr Inf Pract*, vol. 19, no. 2, pp. 136–138, 2024, doi: 10.18438/eblip30523.
- [26] Y. Lappalainen and N. Narayanan, “Aisha: A Custom AI Library Chatbot Using the ChatGPT API,” *Journal of Web Librarianship*, vol. 17, no. 3, pp. 37–58, 2023, doi: 10.1080/19322909.2023.2221477.
- [27] X. Chen, “ChatGPT and Its Possible Impact on Library Reference Services,” *Internet Reference Services Quarterly*, vol. 27, no. 2, pp. 121–129, 2023, doi: 10.1080/10875301.2023.2181262.

- [28] B. D. Lund, D. Khan, and M. Yuvaraj, "ChatGPT in medical libraries, possibilities and future directions: An integrative review," *Health Info Libr J*, vol. 41, no. 1, pp. 4–15, 2024, doi: 10.1111/hir.12518.
- [29] M. Bagchi, "Conceptualising a library chatbot using open source conversational artificial intelligence," *DESIDOC Journal of Library and Information Technology*, vol. 40, no. 6, pp. 329–333, 2020, doi: 10.14429/djlit.40.6.15611.
- [30] A. B. Houston and E. M. Corrado, "Embracing ChatGPT: Implications of Emergent Language Models for Academia and Libraries," *Technical Services Quarterly*, vol. 40, no. 2, pp. 76–91, 2023, doi: 10.1080/07317131.2023.2187110.
- [31] M. Ehrenpreis and J. DeLooper, "Implementing a Chatbot on a Library Website," *Journal of Web Librarianship*, vol. 16, no. 2, pp. 120–142, 2022, doi: 10.1080/19322909.2022.2060893.
- [32] T.-J. Ng, K.-W. Ng, and S.-C. Haw, "Lib-Bot: A Smart Librarian-Chatbot Assistant," *International Journal of Computing and Digital Systems*, vol. 15, no. 1, pp. 1–11, Jul. 2024, doi: 10.12785/ijcds/160101.
- [33] P. Safadel, S. N. Hwang, and J. M. Perrin, "User Acceptance of a Virtual Librarian Chatbot: an Implementation Method Using IBM Watson Natural Language Processing in Virtual Immersive Environment," *TechTrends*, vol. 67, no. 6, pp. 891–902, 2023, doi: 10.1007/s11528-023-00881-7.
- [34] "Dialogflow Documentation | Google Cloud." Accessed: Dec. 03, 2024. [Online]. Available: <https://cloud.google.com/dialogflow/docs?hl=pt-br>
- [35] "AI Powered Customer Service Automation Platform | Kommunicate." Accessed: Dec. 03, 2024. [Online]. Available: <https://www.kommunicate.io/>
- [36] "Library & Learning Commons." Accessed: Jan. 30, 2025. [Online]. Available: <https://www.zu.ac.ae/main/en/library/index>
- [37] "Sweden's National Library Turns Page to AI | NVIDIA Blog." Accessed: Dec. 03, 2024. [Online]. Available: <https://blogs.nvidia.com/blog/sweden-library-ai-open-source/>
- [38] "Leonard Lief Library - Lehman College." Accessed: Jan. 30, 2025. [Online]. Available: <https://lehman.edu/library/>
- [39] "IBM watsonx Assistant Virtual Agent." Accessed: Jan. 30, 2025. [Online]. Available: <https://www.ibm.com/products/watsonx-assistant>
- [40] "Plataforma de desenvolvimento em tempo real do Unity | 3D, 2D, engine VR e AR." Accessed: Jan. 30, 2025. [Online]. Available: <https://unity.com/pt>

- [41] “Telegram Web.” Accessed: Jan. 30, 2025. [Online]. Available: <https://web.telegram.org/>
- [42] “What Is WhatsApp? – Guide To The International Messaging App.” Accessed: Jan. 10, 2025. [Online]. Available: <https://www.forbes.com/sites/technology/article/what-is-whatsapp/>
- [43] “What is Python? Executive Summary | Python.org.” Accessed: Jan. 10, 2025. [Online]. Available: <https://www.python.org/doc/essays/blurb/>
- [44] “Why Is Python the Best Choice for AI and Machine Learning?” Accessed: Jan. 10, 2025. [Online]. Available: <https://www.turing.com/kb/python-best-adapted-to-ai-and-machine-learning>
- [45] “Computational Power and AI - AI Now Institute.” Accessed: Jan. 22, 2025. [Online]. Available: <https://ainowinstitute.org/publication/policy/compute-and-ai>
- [46] “colab.google.” Accessed: Jan. 22, 2025. [Online]. Available: <https://colab.google/>
- [47] “Project Jupyter | Home.” Accessed: Jan. 22, 2025. [Online]. Available: <https://jupyter.org/>
- [48] “Why and how to use Google Colab | TechTarget.” Accessed: Jan. 22, 2025. [Online]. Available: <https://www.techtarget.com/searchenterpriseai/tutorial/Why-and-how-to-use-Google-Colab>
- [49] “Advanced RAG on Hugging Face documentation using LangChain - Hugging Face Open-Source AI Cookbook.” Accessed: Jan. 22, 2025. [Online]. Available: [https://huggingface.co/learn/cookbook/advanced\\_rag#advanced-rag-on-hugging-face-documentation-using-langchain](https://huggingface.co/learn/cookbook/advanced_rag#advanced-rag-on-hugging-face-documentation-using-langchain)
- [50] “Eitanli/goodreads · Datasets at Hugging Face.” Accessed: Jan. 22, 2025. [Online]. Available: <https://huggingface.co/datasets/Eitanli/goodreads>
- [51] A. Vaswani *et al.*, “Attention Is All You Need,” *Adv Neural Inf Process Syst*, vol. 2017-December, pp. 5999–6009, Jun. 2017, Accessed: Jan. 22, 2025. [Online]. Available: <https://arxiv.org/abs/1706.03762v7>
- [52] “Recurrent neural network - Wikipedia.” Accessed: Jan. 22, 2025. [Online]. Available: [https://en.wikipedia.org/wiki/Recurrent\\_neural\\_network](https://en.wikipedia.org/wiki/Recurrent_neural_network)
- [53] “Hugging Face Transformers: Leverage Open-Source AI in Python – Real Python.” Accessed: Jan. 22, 2025. [Online]. Available: <https://realpython.com/huggingface-transformers/>

- [54] “PyTorch Definition | DeepAI.” Accessed: Jan. 22, 2025. [Online]. Available: <https://deepai.org/machine-learning-glossary-and-terms/pytorch>
- [55] “TensorFlow.” Accessed: Jan. 28, 2025. [Online]. Available: <https://www.tensorflow.org/?hl=pt-br>
- [56] “🤖 Transformers.” Accessed: Jan. 22, 2025. [Online]. Available: <https://huggingface.co/docs/transformers/index>
- [57] “What is PyTorch? | Data Science | NVIDIA Glossary.” Accessed: Jan. 22, 2025. [Online]. Available: <https://www.nvidia.com/en-eu/glossary/pytorch/>
- [58] “What is JavaScript.” Accessed: Jan. 28, 2025. [Online]. Available: [https://www.w3schools.com/whatis/whatis\\_js.asp](https://www.w3schools.com/whatis/whatis_js.asp)
- [59] “Introduction to LangChain - GeeksforGeeks.” Accessed: Jan. 22, 2025. [Online]. Available: <https://www.geeksforgeeks.org/introduction-to-langchain/>
- [60] “Langchain — RAG — Retrieval Augmented Generation | by Airton de Sousa Lira Junior | Medium.” Accessed: Jan. 22, 2025. [Online]. Available: <https://airtonlirajr.medium.com/langchain-rag-retrieval-augmented-generation-edc34451c7c6>
- [61] “What Is LangChain? | IBM.” Accessed: Jan. 22, 2025. [Online]. Available: <https://www.ibm.com/think/topics/langchain>
- [62] “What Is Faiss (Facebook AI Similarity Search)? | DataCamp.” Accessed: Jan. 22, 2025. [Online]. Available: [https://www.datacamp.com/blog/faiss-facebook-ai-similarity-search?utm\\_source=google&utm\\_medium=paid\\_search&utm\\_campaignid=21374847033&utm\\_adgroupid=165153430762&utm\\_device=c&utm\\_keyword=&utm\\_matchtype=&utm\\_network=g&utm\\_adpostion=&utm\\_creative=726052822690&utm\\_targetid=aud-1903815585993:dsa-2222697810678&utm\\_loc\\_interest\\_ms=&utm\\_loc\\_physical\\_ms=1011719&utm\\_content=DSA~blog~Artificial-Intelligence&utm\\_campaign=240617\\_1-sea~dsa~tofu\\_2-b2c\\_3-ptbr-lang-en\\_4-prc\\_5-na\\_6-na\\_7-le\\_8-pdsh-go\\_9-nbe\\_10-na\\_11-na-jan25&gad\\_source=1&gclid=Cj0KCQiAy8K8BhCZARIsAKJ8sfTBII CczVqF-Efaxfjdwei4GzWm7wKea9hHG3m4IUJNTVO0VguCQMAaAnIbEALw\\_wcB](https://www.datacamp.com/blog/faiss-facebook-ai-similarity-search?utm_source=google&utm_medium=paid_search&utm_campaignid=21374847033&utm_adgroupid=165153430762&utm_device=c&utm_keyword=&utm_matchtype=&utm_network=g&utm_adpostion=&utm_creative=726052822690&utm_targetid=aud-1903815585993:dsa-2222697810678&utm_loc_interest_ms=&utm_loc_physical_ms=1011719&utm_content=DSA~blog~Artificial-Intelligence&utm_campaign=240617_1-sea~dsa~tofu_2-b2c_3-ptbr-lang-en_4-prc_5-na_6-na_7-le_8-pdsh-go_9-nbe_10-na_11-na-jan25&gad_source=1&gclid=Cj0KCQiAy8K8BhCZARIsAKJ8sfTBII CczVqF-Efaxfjdwei4GzWm7wKea9hHG3m4IUJNTVO0VguCQMAaAnIbEALw_wcB)
- [63] “Faiss: A library for efficient similarity search - Engineering at Meta.” Accessed: Jan. 22, 2025. [Online]. Available: <https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/>

- [64] “Introdução ao Visual Studio Code - DevMedia.” Accessed: Jan. 10, 2025. [Online]. Available: <https://www.devmedia.com.br/introducao-ao-visual-studio-code/34418>
- [65] “Visual Studio: IDE e Editor de Código para Desenvolvedores de Software e Teams.” Accessed: Jan. 10, 2025. [Online]. Available: <https://visualstudio.microsoft.com/pt-br/#vscode-section>
- [66] “VS Code - O que é e por que você deve usar? | Blog da TreinaWeb.” Accessed: Jan. 10, 2025. [Online]. Available: <https://www.treinaweb.com.br/blog/vs-code-o-que-e-e-por-que-voce-deve-usar>
- [67] “Git - What is Git?” Accessed: Jan. 10, 2025. [Online]. Available: <https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F>
- [68] “What Is GitHub? A Beginner’s Introduction to GitHub.” Accessed: Jan. 10, 2025. [Online]. Available: <https://kinsta.com/knowledgebase/what-is-github/>
- [69] “O que é Web Scraping? | Um Guia Completo.” Accessed: May 03, 2025. [Online]. Available: <https://automate.fortra.com/resources/guides/what-is-web-scraping>
- [70] “O que é Web Scraping? Como Extrair Legalmente o Conteúdo da Web.” Accessed: May 03, 2025. [Online]. Available: <https://kinsta.com/pt/base-de-conhecimento/o-que-e-web-scraping/>
- [71] “Web Scraper - The #1 web scraping extension.” Accessed: May 03, 2025. [Online]. Available: <https://webscraper.io/>
- [72] “Web Scraping Tool & Free Web Crawlers | Octoparse.” Accessed: May 03, 2025. [Online]. Available: <https://www.octoparse.com/>
- [73] “Microsoft Power Automate – Plataforma de Automatização de Processos | Microsoft.” Accessed: May 03, 2025. [Online]. Available: <https://www.microsoft.com/pt-pt/power-platform/products/power-automate?market=pt>
- [74] “Embeddings de vetores em aplicações RAG | por Faraaz Khan | The Deep Hub | Medium.” Accessed: Jan. 21, 2025. [Online]. Available: <https://medium.com/thedeephub/vector-embeddings-in-rag-applications-9ea8043c172b>
- [75] “thenlper/gte-small · Rosto Abraçado.” Accessed: Jan. 24, 2025. [Online]. Available: <https://huggingface.co/thenlper/gte-small>
- [76] “Prompt Templates |  LangChain.” Accessed: Jan. 21, 2025. [Online]. Available: [https://python.langchain.com/docs/concepts/prompt\\_templates/](https://python.langchain.com/docs/concepts/prompt_templates/)

- [77] “Faiss |  LangChain.” Accessed: Jan. 22, 2025. [Online]. Available: <https://python.langchain.com/docs/integrations/vectorstores/faiss/>
- [78] “distilbert/distilgpt2 · Hugging Face.” Accessed: Jan. 22, 2025. [Online]. Available: <https://huggingface.co/distilbert/distilgpt2>
- [79] “meta-llama/Llama-2-13b-chat-hf · Rosto Abraçado.” Accessed: Jan. 22, 2025. [Online]. Available: <https://huggingface.co/meta-llama/Llama-2-13b-chat-hf>
- [80] “HuggingFaceH4/zephyr-7b-beta · Hugging Face.” Accessed: Jan. 22, 2025. [Online]. Available: <https://huggingface.co/HuggingFaceH4/zephyr-7b-beta>
- [81] “openai-community/gpt2 · Hugging Face.” Accessed: Jan. 30, 2025. [Online]. Available: <https://huggingface.co/openai-community/gpt2>
- [82] “meta-llama/Llama-2-7b · Hugging Face.” Accessed: Jan. 30, 2025. [Online]. Available: <https://huggingface.co/meta-llama/Llama-2-7b>
- [83] “mistralai/Mistral-7B-v0.1 · Hugging Face.” Accessed: Jan. 30, 2025. [Online]. Available: <https://huggingface.co/mistralai/Mistral-7B-v0.1>
- [84] “MT Bench - a Hugging Face Space by lmsys.” Accessed: Jan. 22, 2025. [Online]. Available: <https://huggingface.co/spaces/lmsys/mt-bench>
- [85] “AlpacaEval Leaderboard.” Accessed: Jan. 22, 2025. [Online]. Available: [https://tatsu-lab.github.io/alpaca\\_eval/](https://tatsu-lab.github.io/alpaca_eval/)
- [86] “Quick reference |  LangChain.” Accessed: Jan. 27, 2025. [Online]. Available: [https://python.langchain.com/v0.1/docs/modules/model\\_io/prompts/quick\\_start/](https://python.langchain.com/v0.1/docs/modules/model_io/prompts/quick_start/)
- [87] “Bibliotecas | Instituto Politécnico de Castelo Branco catálogo.” Accessed: May 03, 2025. [Online]. Available: <https://catalogo.ipcb.pt/>