Universidade Federal do Estado do Rio de Janeiro
Centro de Ciências Exatas e Tecnológicas
Escola de Informática Aplicada

Retrieval Augmented Generation Aplicada à Bibliotecas

Breno Costa da Silva Filgueiras

Orientador
Pedro Nuno de Souza Moura

Rio de Janeiro, RJ – Brasil
Dezembro, 2024

Retrieval Augmented Generation Aplicada à Bibliotecas

Breno Costa da Silva Filgueiras

Projeto de graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) como cumprimento de requerimento parcial para obtenção título de Bacharel em Sistemas de Informação.

Approved by:

Pedro Nuno de Souza Moura – UNIRIO

Supervisor 2, D.Sc. – UNIRIO

Supervisor 3, D.Sc. – XXXX

Rio de Janeiro, RJ – Brasil

Dezembro, 2024

Catalogação informatizada pelo autor

Agradecimentos

Agradeço profundamente aos meus pais, Rita e Alexandre, pelo apoio incondicional ao longo de toda a minha jornada, mas especialmente durante meu percurso acadêmico, sempre ao meu lado.

Aos meus irmãos, sou grato por me incentivarem a continuar meus estudos e por me lembrarem de que sou um exemplo e uma fonte de inspiração para eles. Estarei sempre com vocês, em cada conquista.

Aos meus amigos, que estiveram ao meu lado durante o processo de escrita, meu sincero agradecimento por me motivarem a seguir em frente, mesmo nos momentos de dificuldade, quando a paciência ou inspiração pareciam escassas.

Um agradecimento especial ao meu orientador, Pedro Nuno de Moura Souza, que me guiou na fascinante jornada pelo universo do *Deep Learning* e Inteligência Artificial. Sua confiança em minhas ideias e o apoio contínuo, especialmente durante a mudança de tema do TCC, foram fundamentais para o sucesso deste trabalho.

Dedico este trabalho à minha amada Jéssica, que tem sido minha parceira desde 2013 e sempre me deu suporte nos momentos mais desafiadores, e à nossa pequena Sofia, que chegou em 2019 para tornar cada passo ainda mais especial.

Resumo

Em uma parceria entre Seagate e a International Data Corporation (IDC) foi realizado o estudo *The Digitization of the World From Edge to Core,* nele a IDC fala sobre diversos aspectos referentes aos dados presentes no mundo digital e um dos tópicos abordados no estudo é "Mankind is on a quest to digitize the world" e neste mesmo tópico eles explicam que os dados que geramos no dia a dia está em constante crescimento, ou seja, estamos gradualmente produzindo mais dados.

Com um volume cada vez maior de dados, uma busca por informação otimizada é essencial, dado que são necessárias ferramentas que nos garantam confiança e precisão da informação adquirida. Com isso em mente, este trabalho visa o desenvolvimento de um sistema capaz de ler, processar e armazenar documentos diversos de determinada biblioteca (conjunto de documentos) para que possamos utilizar um *Large Language Model* (LLM) para responder perguntas que os usuários possam ter acerca dos documentos.

A ideia é conseguir processar documentos de diferentes épocas, temas, formatos e conseguir responder o maior número possível de perguntas dos usuários com a melhor confiança possível.

Palavras-chave: retrieval, augmented, generation, inteligência, artificial.

Abstract

In a partnership between Seagate and the International Data Corporation (IDC), the study *The Digitization of the World From Edge to Core* was conducted. In it, IDC discusses various aspects related to data present in the digital world and one of the topics covered in the study is "Mankind is on a quest to digitize the world". In this same topic, they explain that the data we generate on a daily basis is constantly growing, that is, we are gradually producing more data.

With an ever-increasing volume of data, an optimized search for information is essential, given that tools are needed that guarantee reliability and accuracy of the information acquired. With this in mind, this work aims to develop a system capable of reading, processing and storing various documents from a given library (set of documents) so that we can use a *Large Language Model* (LLM) to answer questions that users may have about the documents.

The idea is to be able to process documents from different periods, themes and formats and to be able to answer as many user questions as possible with the greatest possible confidence.

Keywords: retrieval, augmented, generation, artifical, inteligence.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

1 Introdução

Com a expansão contínua da Internet das Coisas (IoT), o cenário se transforma em um redemoinho de informações. Chegará, ou talvez já tenha chegado, o momento em que será impossível para qualquer ser humano consumir tudo o que criou em um único dia.

Durante estudo, a *International Data Coorporation* (IDC) previu que a *Global Datasphere* cresceria de 45 zettabytes em 2019 para 175 zettabytes em 2025 [David Reinsel e Rydning, 2018]. Um crescimento de aproximadamente 380% em 6 anos, porém essa previsão foi feita em 2018 e atualmente já existem estudos que projetam números ainda maiores para a produção de dados. Na Figura 1, temos um gráfico gerado durante o estudo *Amount of Data Created Daily,* onde foram calculados um total de 147 *zetabytes* produzidos em 2024, com previsão de 181 *zetabytes* para 2025, um crescimento de 23.12% [Duarte, 2024].

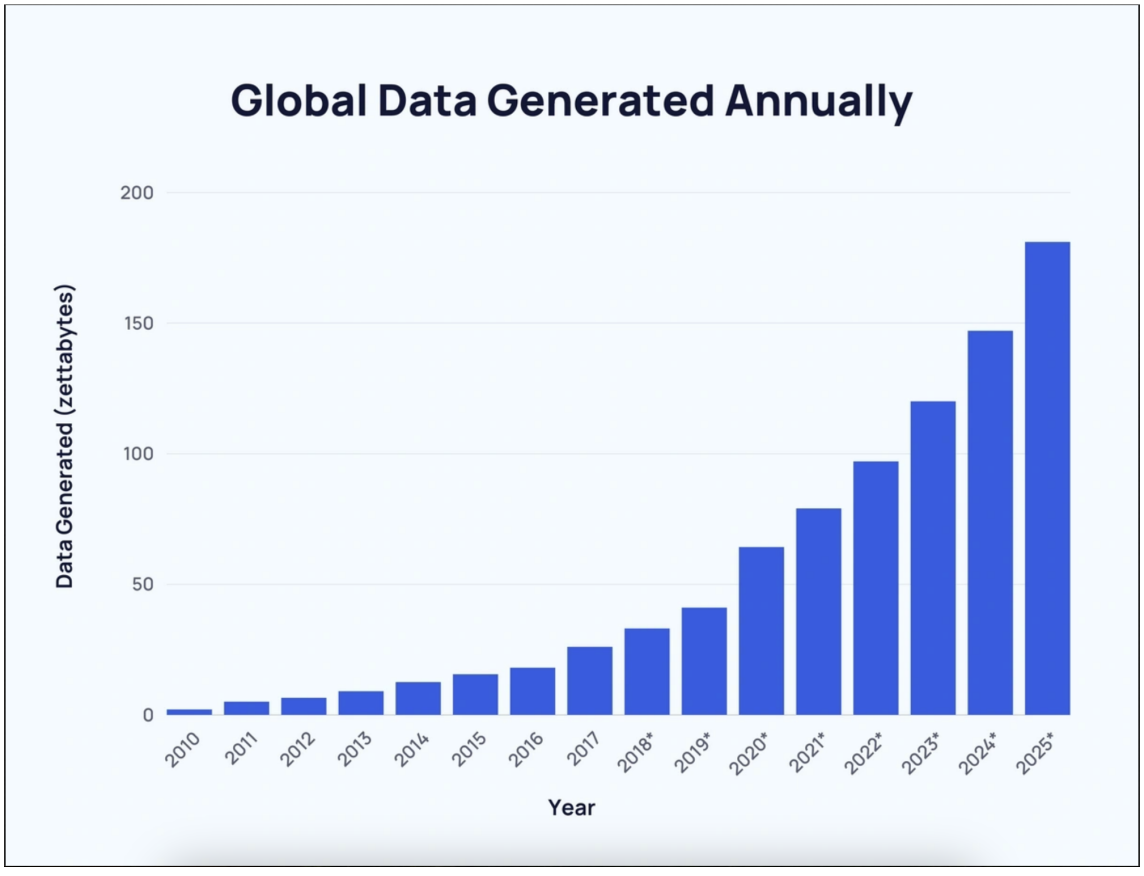

Figura 1: Total de dados produzidos por ano (extraído de [Duarte, 2024])

1.1 Motivação

No estudo *The Digitization of the World From Edge to Core,* a *Seagate*, gigante do armazenamento de dados, uniu forças com a *International Data Corporation* (IDC) para conduzir uma análise dos dados presentes na *Global Datasphere*, que quantifica e analisa o total de dados criados, capturados e replicados no mundo inteiro. A IDC destacou: "*Mankind is on a quest to digitize the world*". Essa frase encapsula a era em que vivemos, marcada por um crescimento incessante no volume de dados que produzimos diariamente.

Cada clique, pagamento por aproximação ou uso de *wearables* adiciona mais um fragmento ao vasto oceano digital. Nesse turbilhão de dados, buscar uma matéria ou reportagem torna-se uma tarefa semelhante a encontrar uma agulha em um palheiro digital, um desafio tão fascinante quanto frustrante.

E esse contexto de imensidão de dados onde a busca por informações é cada vez mais dificíl, é o berço deste projeto. O objetivo é implementar uma solução que processe bibliotecas de documentos e aplique o conceito de *Retrieval Augmented Generation* (RAG). Com uma interface de chat simples, o usuário poderá fazer perguntas e receber respostas humanizadas, geradas por um *Large Language Model* (LLM), com referências claras aos documentos de origem.

O desafio de buscar informações relevantes é significativo. A internet ainda abriga dados sem referência ou apresentados de formas variadas, como gráficos e textos, dificultando a assimilação. Além disso, interfaces pouco intuitivas e mecanismos de busca ineficazes consomem tempo valioso. Para estudantes e pesquisadores, essa batalha constante com a desorganização digital pode transformar o simples ato de encontrar informações em um verdadeiro labirinto.

1.2 Objetivos

O objetivo principal deste trabalho é implementar uma solução baseada em *Retrieval Augmented Generation* (RAG) para bibliotecas de documentos específicos, a fim de viabilizar consultas que retornem dados pertinentes junto com suas referências. Os objetivos específicos são:

1. Escrever uma introdução acessível ao conceito de RAG aos alunos do BSI.
2. Produzir um documento que instrua a implementação de um ecossistema RAG aos alunos do BSI.
3. Realizar a implementação de RAG para documentos, que seja agnóstica tanto ao LLM quanto embeeding utilizados.
4. Executar uma validação sobre a solução gerada, de maneira que o resultado seja relevante.

1.3 Organização do Texto

Este trabalho está organizado em capítulos, com o objetivo de apresentar os processos, métodos, análises e descobertas de forma clara e coerente. A estrutura do documento foi elaborada para facilitar a compreensão do leitor sobre a complexidade do tema e os resultados obtidos, conduzindo-o até a implementação da solução final. Os capítulos estão estruturados da seguinte forma:

- **Introdução:** Apresenta o contexto do trabalho, destacando o problema do crescente volume de dados. Discute a motivação para a solução proposta, sua relevância no cenário atual e os objetivos estabelecidos.
- **Conceitos Fundamentais:** Dedica-se à fundamentação teórica, abordando os conceitos essenciais para a compreensão da solução e sua implementação. Inclui uma introdução ao conceito de *Large Language Models* (LLM) e uma análise detalhada do *Retrieval Augmented Generation* (RAG).
- **Modelagem:** Descreve a composição da solução, explicando os artefatos envolvidos e suas responsabilidades. Também aborda o funcionamento e o papel de cada componente na solução final. Ao final, apresenta as tecnologias utilizadas, incluindo descrições breves sobre as ferramentas, suas versões e funções.
- **Solução Desenvolvida:** Detalha os artefatos implementados, explicando como a solução cumpre suas funções. Apresenta os resultados obtidos e discute as respostas fornecidas para algumas das questões propostas, avaliando a eficácia da solução.

- **Conclusão:** O capítulo final resume as considerações sobre os resultados alcançados, destacando tanto os aspectos positivos quanto as limitações da solução implementada. Além disso, discute possíveis trabalhos futuros ou aplicações derivadas da solução, encerrando com as referências utilizadas no desenvolvimento do trabalho.

1.4 Metodologia

Este trabalho adotará a abordagem de *Design Science Research* (DSR) para garantir que, ao final do trabalho, o artefato modelado esteja implementado e funcionando conforme planejado, atendendo aos requisitos definidos e demonstrando sua viabilidade prática. O DSR oferece uma estrutura robusta para guiar a criação de soluções baseadas em evidências, alinhando teoria e prática de forma estruturada.

Com raízes na engenharia e nas ciências do artificial [Simon, 2019], o DSR é uma metodologia voltada para a resolução de problemas. Seu objetivo é aprimorar o conhecimento humano por meio da criação de artefatos inovadores e da geração de conhecimento de design, oferecendo soluções práticas para problemas do mundo real [Alan Hevner, março de 2004].

Além disso, a aplicação do DSR valoriza a integração de análise teórica com resultados práticos, incentivando a reflexão crítica sobre as escolhas metodológicas e de *design* realizadas durante o projeto. Isso possibilita uma contribuição mais ampla, tanto para o avanço acadêmico quanto para a aplicação prática em contextos organizacionais.

Assim, ao utilizar o *Design Science Research* (DSR), este trabalho resultará em um artefato produzido com base nas análises e discussões realizadas ao longo das próximas seções deste trabalho.

2 Conceitos Fundamentais

Para que este trabalho seja compreendido e os próximos capítulos possam ser apresentados com maior clareza, é necessário passar por alguns conceitos. Antes de nos aprofundarmos no contexto de um ecossistema de *retrieval augmented generation* (RAG), é necessário compreender um pouco a IA generativa e os *Large Language Models* (LLMs) que contribuem tanto para a interpretação das perguntas feitas durante as interações com o usuário quanto na geração de uma resposta mais humana.

2.1 IA Generativa

A inteligência artificial generativa (IA generativa), às vezes chamada de *gen AI* (*Generative AI*), é um tipo de inteligência artificial (IA) capaz de criar conteúdo original — como texto, imagens, vídeos, áudio ou código de *software* — em resposta a um comando ou solicitação do usuário. [Mark Scapicchio, março de 2024]

A IA generativa baseia-se em modelos avançados de *machine learning* (aprendizado de máquina) chamados de modelos de *deep learning* (aprendizagem profunda) — algoritmos que simulam a estrutura do cérebro humano e o seu processo de aprendizado. Esses modelos trabalham identificando padrões e relações em grandes volumes de dados.

A partir de um treinamento em uma base de dados de tamanho massivo (da ordem de trilhões de palavras), a IA generativa é capaz de compreender solicitações ou perguntas feitas em linguagem natural pelos usuários, respondendo com conteúdos novos e relevantes. Essa capacidade permite a criação de textos, imagens, vídeos, áudios e até códigos de software, de forma original e adaptada ao pedido do usuário.

2.1.1 Uma Breve História

O termo "IA generativa" explodiu na consciência pública na década de 2020, mas a inteligência artificial já faz parte de nossas vidas há décadas, e a tecnologia de IA generativa atual tem seus pilares contruídos desde o século XX. Uma história representativa não exaustiva da IA generativa pode incluir algumas das seguintes datas:

- **1964:** O cientista da computação do *Massachusetts Institute of Technology* (MIT),

Joseph Weizenbaum, desenvolve o ELIZA, uma aplicação de processamento de linguagem natural baseada em texto. Essencialmente o primeiro *chatbot* (chamado de "*chatterbot*" na época), o ELIZA usava *scripts* de correspondência de padrões para responder a entradas de linguagem natural digitadas com respostas empáticas em texto.

- **1999:** A Nvidia lança a GeForce, a primeira unidade de processamento gráfico (*Graphic Processing Unit* - GPU). Originalmente desenvolvida para fornecer gráficos de movimento suave para videogames, as GPUs se tornaram a plataforma padrão para o treinamento de modelos de IA e mineração de criptomoedas, por conta de sua alta capacidade de cálculo matricial.

- **2004:** O Google *autocomplete* aparece pela primeira vez, gerando palavras ou frases potenciais à medida que os usuários digitam seus termos de busca.

- **2013:** Aparecem os primeiros *autoencoders* variacionais (VAEs).

- **2014:** Surgem as primeiras redes adversariais generativas (GANs) e modelos de difusão.

- **2017:** Ashish Vaswani, junto com uma equipe do Google Brain e um grupo da Universidade de Toronto publicam o artigo *Attention is All you Need*, [Ashish Vaswani, junho de 2017], um artigo que documenta os princípios dos modelos de transformadores, amplamente reconhecidos como os responsáveis por permitir os modelos de fundação mais poderosos e as ferramentas de IA generativa que estão sendo desenvolvidas hoje.

- **2019-2020:** A OpenAI lança seus modelos de linguagem GPT (*Generative Pre-trained Transformer*), o GPT-2 e o GPT-3.

- **2022:** O OpenAI apresenta o ChatGPT, uma interface de *chatbot* do GPT-3 que gera frases complexas, coerentes e contextuais, além de conteúdo de longo formato em resposta a comandos dos usuários.

- **2023-2024:** Com a notoriedade e popularidade do ChatGPT, que efetivamente abriu as portas para uma onda de desenvolvimentos, os avanços e lançamentos

de produtos em IA generativa têm ocorrido a um ritmo acelerado, incluindo lançamentos do Google Bard (agora Gemini), Microsoft Copilot, IBM watsonx.ai e o modelo de linguagem Llama-2 de código aberto da Meta.

A inteligência artificial tem sido um tema relevante na tecnologia, mas foi a IA generativa, especialmente com o lançamento do ChatGPT em novembro de 2022, que a destacou globalmente, gerando inovação e adoção. Ela oferece grandes benefícios de produtividade para indivíduos e organizações, e, apesar dos desafios e riscos, as empresas exploram como melhorar fluxos de trabalho e enriquecer produtos e serviços. De acordo com uma pesquisa da consultoria *McKinsey*, mais de 65% das empresas usam Gen AI no mundo. [Yran Bartolumeu Dias, julho de 2024].

2.2 Large Language Models

Os *Large Language Models* (LLMs) são uma categoria de modelos fundamentais treinados em grandes volumes de dados para oferecer capacidades versáteis, atendendo a diversos casos de uso e tarefas. Diferentemente dos modelos específicos para determinados domínios, que exigem treinamentos separados para cada aplicação—geralmente com altos custos e demandas significativas de infraestrutura—os LLMs promovem uma aplicação mais ampla, gerando sinergias entre diferentes áreas e, muitas vezes, alcançando um desempenho superior. [IBM, novembro de 2023]

Os LLMs representam um avanço significativo em *Natural Language Processing* (NLP) e inteligência artificial. Esses modelos estão amplamente acessíveis ao público por meio de LLMs como o *ChatGPT-3* e *GPT-4* da *OpenAI*, apoiados pela *Microsoft*. Outros exemplos incluem os modelos *Llama* da Meta, os modelos *BERT/RoBERTa* e *PaLM* do Google, e a série *Granite* lançada pela IBM.

Os LLMs são projetados para compreender e gerar texto de forma similar à humana, além de produzir outros tipos de conteúdo. Com base nos extensos volumes de dados em que foram treinados, conseguem traduzir idiomas, resumir textos, responder perguntas, auxiliar na redação e até mesmo na geração de código.

Essas capacidades são viabilizadas por bilhões de parâmetros que capturam padrões complexos da linguagem. Como resultado, os LLMs estão transformando áreas

como *chatbots*, assistentes virtuais, geração de conteúdo, suporte à pesquisa e tradução de idiomas.

2.2.1 Como funcionam?

Os *Large Language Models* (LLMs) operam utilizando técnicas de aprendizagem profunda e grandes volumes de dados textuais. Baseados na arquitetura de *transformers* [Ashish Vaswani, junho de 2017], como o *Generative Pre-trained Transformer* (GPT), esses modelos são especialmente eficazes em lidar com dados sequenciais, como entradas de texto. Compostos por várias camadas de redes neurais, os LLMs empregam um mecanismo de atenção para focar em partes específicas da sequência de texto fornecida na entrada.

Durante o treinamento, os modelos aprendem a prever um conjunto de próximas palavras em uma sentença com base no contexto fornecido pelas palavras anteriores e posteriores. Isso é feito atribuindo probabilidades à recorrência de palavras que foram transformadas em *tokens* (divididas em sequências menores) e geradas as *embeddings*, representações numéricas que capturam o contexto de ocorrência das palavras.

O treinamento envolve o uso de *corpora* (conjuntos de textos) massivos, com bilhões de páginas de texto e trilhões de palavras, permitindo que os LLMs aprendam gramática, semântica e relações conceituais por meio de aprendizado auto-supervisionado, modelagem de sequência e demais técnicas de treinamento de LLMs. Uma vez treinados, os modelos geram texto prevendo autonomamente as próximas palavras com base na entrada recebida e nos padrões aprendidos, resultando em uma produção linguística coerente e relevante para diversas tarefas de compreensão e geração de linguagem.

O desempenho dos LLMs pode ser aprimorado por meio de técnicas como *prompt engineering*, *fine-tuning* (ajuste fino) e aprendizado por reforço com feedback humano (*Reinforcement Learning from Human Feedback* - RLHF). Essas abordagens ajudam a mitigar problemas como viés, discurso de ódio e respostas incorretas ou ilusórias (*hallucinations*). Garantir que os LLMs estejam prontos para uso, seja em sociedade ou nível corporativo, é crucial para evitar riscos à reputação e responsabilidades indesejadas.

2.3 Padrões de Projeto

Os padrões de projeto (*Design Patterns*) são soluções reutilizáveis para problemas recorrentes no desenvolvimento de software. Eles foram formalizados no clássico livro [Erich Gamma e Vlissides, 1994] e são amplamente utilizados para melhorar a estrutura, a legibilidade e a manutenção de sistemas. Suas principais vantagens incluem:

- **Abstração**: Simplificam a complexidade ao ocultar os detalhes de implementação.
- **Reutilização de Código**: Reduzem redundâncias, promovendo um desenvolvimento mais eficiente.
- **Flexibilidade**: Facilitam a adaptação de sistemas a novos requisitos sem modificar significativamente o código existente.
- **Manutenção e Escalabilidade**: Tornam o sistema mais modular e fácil de ajustar ao longo do tempo.

2.3.1 Padrão de Fábrica

Para este trabalho, o padrão fábrica será utilizado em situações em que a abstração de um serviço ou classe traz uma vantagem para o artefato, de modo que ele possa instanciar uma classe a partir de determinado contexto, como é ilustrado na Figura 2.

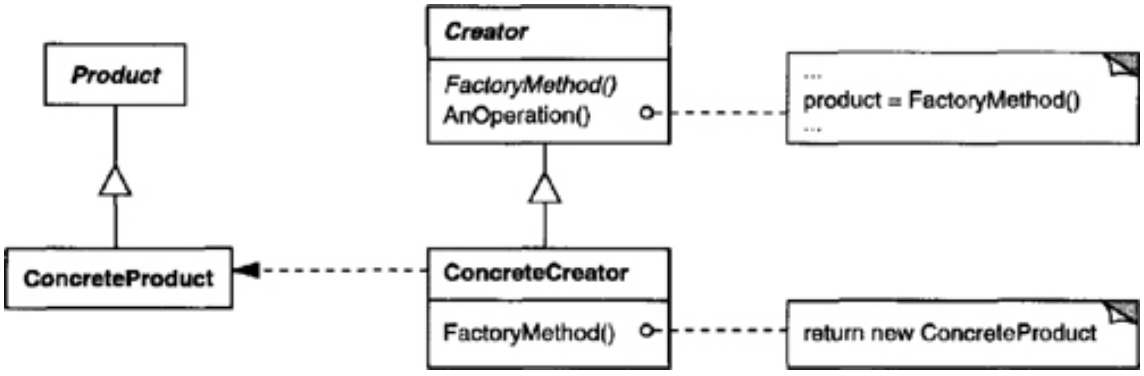

Figura 2: Estrutura do *factory design pattern* (extraída de [Erich Gamma e Vlissides, 1994]).

O padrão de fábrica, é um padrão criacional que fornece uma interface para a criação de objetos em uma superclasse, delegando às subclasses a responsabilidade de

determinar qual classe concreta será instanciada. Ele é útil quando a criação de objetos envolve lógica complexa ou quando diferentes tipos de objetos precisam ser criados de forma dinâmica, dependendo do contexto. Como ilustrado na Figura 2, o padrão de fábrica implementa uma interface para a criação de objetos que herdam de uma superclasse, permitindo que as classes filhas possam alterar o tipo de objeto criado.

2.4 O Problema

No livro *RAG-Driven Generative AI,* é dito que "mesmo o modelo mais avançado de Inteligência Artificial (IA) generativa é limitado a responder somente sobre dados nos quais ele foi treinado." [Rothman, 2024]. Essa afirmação chama a atenção para um problema especial: como fazer para que uma IA saiba responder perguntas referentes a um conjunto específico de dados, diferente daquele em que foi treinada?

Quando um modelo de IA generativo não sabe como responder com precisão, alguns dizem que ele está produzindo viés ou sofrendo uma *hallucination* (alucinação). No entanto, "tudo se resume à impossibilidade de fornecer uma resposta adequada quando o treinamento do modelo não incluiu as informações solicitadas, além dos problemas clássicos de configuração do modelo." [Rothman, 2024]. Essa confusão geralmente leva a sequências aleatórias das saídas mais prováveis, não das mais precisas.

Buscando mitigar essas questões, em 2020 foi publicado o artigo "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" [Patrick Lewis, janeiro de 2020], que combinava abordagens baseadas em recuperação com modelos generativos, introduzindo a *Retrieval Augmented Generation* (RAG). Uma RAG recupera dados relevantes de fontes externas em tempo real e usa esses dados para gerar respostas contextualmente relevantes, isto é, que façam sentido dentro do contexto trabalhado durante as consultas feitas pelo usuário. Uma de suas principais vantagens é a adaptabilidade, tendo em vista que a estrutura pode ser aplicada independentemente do tipo de dado abordado na solução, seja texto, imagens, áudios ou documentos diversos.

2.5 Retrieval Augmented Generation (RAG)

Quando um modelo de IA generativa não sabe responder determinada pergunta com precisão, diz-se que ele está alucinando ou apresentando viés, mas, na prática, está apenas gerando respostas sem sentido. Isso ocorre porque o modelo não foi treinado com as informações solicitadas ou por conta de limitações em sua configuração, resultando em sequências prováveis, mas não precisas, necessariamente.

Uma RAG começa onde a IA generativa termina, fornecendo informações que um modelo de LLM não possui para responder com precisão às consultas do usuário. Uma RAG otimiza tarefas de recuperação de informações e adiciona os dados recuperados durante a entrada (seja consulta do usuário ou um *prompt* automatizado), gerando uma saída melhorada e mais amigável ao usuário. O funcionamento geral da RAG pode ser resumido na Figura 3, que será explicada nos próximos parágrafos:

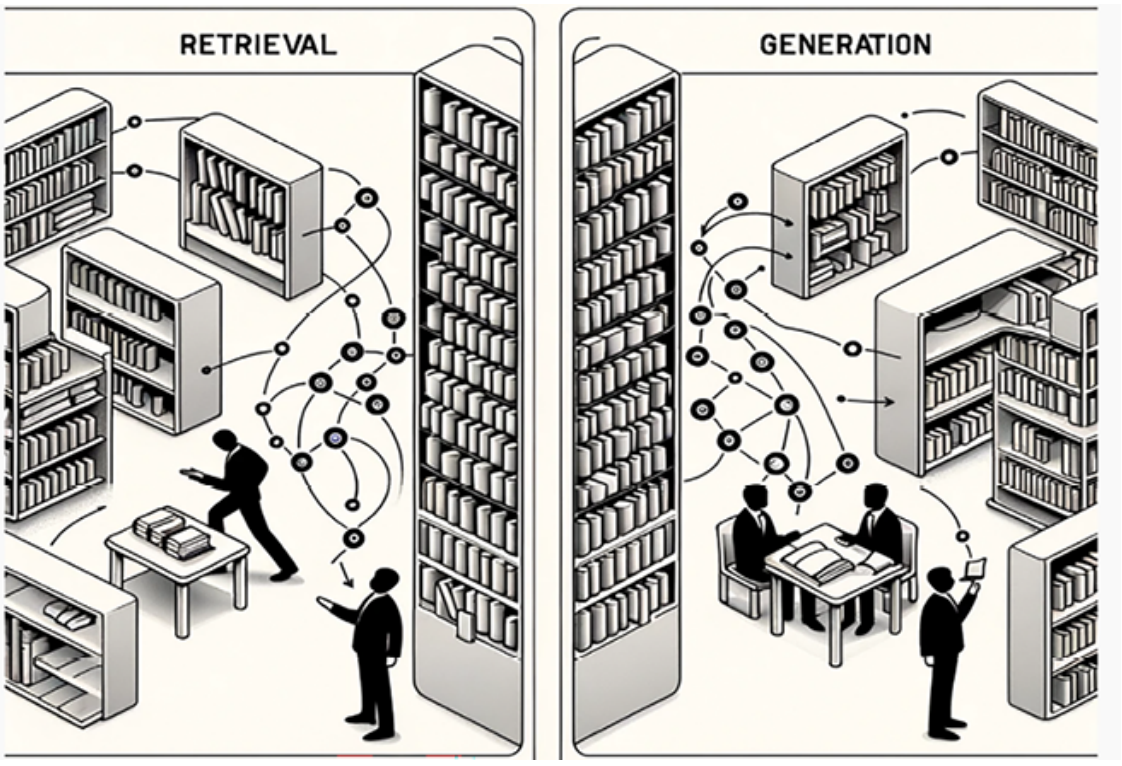

Figura 3: Funcionamento geral de uma estrutura RAG (extraída de [Rothman, 2024])

Imagine um estudante em uma biblioteca, com a tarefa de escrever uma dissertação sobre RAG. Assim como o ChatGPT ou outras ferramentas de IA generativa, o estudante sabe ler e escrever. Como qualquer LLM, o estudante é treinado para compreender informações avançadas, resumir e criar conteúdo. No entanto, como qualquer IA,

há muitas informações que este estudante ainda desconhece.

Na fase de recuperação, ele busca por livros sobre o tema necessário (lado esquerdo da Figura 3) na biblioteca. Em seguida, ele retorna ao seu lugar, realiza a tarefa de recuperação sozinho ou com a ajuda de um colega, extraindo as informações relevantes dos livros adquiridos. Na fase de geração (lado direito da Figura 3), o estudante começa a escrever a sua dissertação utilizando o conhecimento adquirido na fase anterior. Esse é o funcionamento de um agente humano guiado por RAG, de maneira semelhante a uma estrutura de IA generativa baseada em RAG.

Enquanto escreve sua dissertação sobre RAG, o estudante encontra tópicos difíceis com os quais não tem tempo para consultar todas as informações disponíveis. Como um agente humano generativo, ele fica travado, assim como um modelo de IA generativa. Ele até pode tentar escrever algo sobre esses tópicos, mas, como a IA, não saberá se o conteúdo está correto até que alguém corrija a dissertação e lhe avalie de alguma maneira.

Neste ponto, ele já atingiu seu limite e decide recorrer a uma ferramenta de IA generativa com RAG para obter respostas corretas e auxiliá-lo. No entanto, existe uma grande variedade de modelos de LLM e configurações RAG disponíveis, deixando o estudante sobrecarregado. Antes de prosseguir, é necessário entender os recursos disponíveis e como o RAG está organizado.

2.6 Ecossistema RAG

A IA generativa baseada em RAG é um *framework* que pode ser implementada com diversas configurações, funcionando dentro de um ecossistema amplo (Figura 4). Independentemente da quantidade de estruturas de recuperação e geração disponíveis, tudo se resume a quatro eixos principais e suas respectivas questões:

- **Dados:** De onde vêm os dados? São confiáveis e suficientes? Há questões de direitos autorais, privacidade ou segurança?
- **Armazenamento:** Como os dados serão armazenados antes ou depois do processamento? Qual será o volume armazenado?

- **Recuperação:** Como os dados corretos serão recuperados para complementar a entrada (ou consulta) do usuário? Qual tipo de *framework* RAG será mais adequado ao projeto?
- **Geração:** Qual modelo de IA generativa melhor se adapta ao *framework* RAG escolhido?

Esses eixos dependem do tipo de *framework* RAG utilizado. Antes de escolher, é essencial avaliar a proporção de conhecimento paramétrico e não paramétrico no ecossistema implementado. No contexto de aprendizado de máquina, o conhecimento paramétrico é o entendimento internalizado que um modelo ganha com o treinamento em um conjunto de dados.

Esse conhecimento é representado pelos parâmetros do modelo (pesos e viéses), que são ajustados durante o processo de treinamento para minimizar a função de perda e melhorar o desempenho do modelo. O conhecimento paramétrico permite que o modelo faça previsões e generalize para dados novos e invisíveis, capturando recursos e relacionamentos essenciais dentro dos dados de treinamento.

Nos próximos parágrafos, serão abordados os principais componentes de um ecossistema RAG, explicando seus principais componentes e como eles interagem entre si, como ilustrado na Figura 4.

Como ilustrado acima, na Figura 4, são ilustrados quatro componentes que dependem de seus respectivos ecossistemas, compostos por seus subcomponentes, formando o *pipeline* de IA generativa baseada em RAG. Estes componentes são:

- ***Retriever*** (D, em verde na Figura 4): Responsável pela coleta, processamento, armazenamento e recuperação de dados.
- ***Generator*** (G, em azul na Figura 4): Cuida da complementação da entrada, engenharia de prompts e geração de respostas.
- ***Evaluator*** (E, em amarelo na Figura 4): Avalia o desempenho usando métricas matemáticas, *feedback* humano e outras formas de validação.
- ***Trainer*** (T, em rosa na Figura 4): Gerencia o modelo pré-treinado inicial e sua posterior ajuste fino (*fine-tuning*).

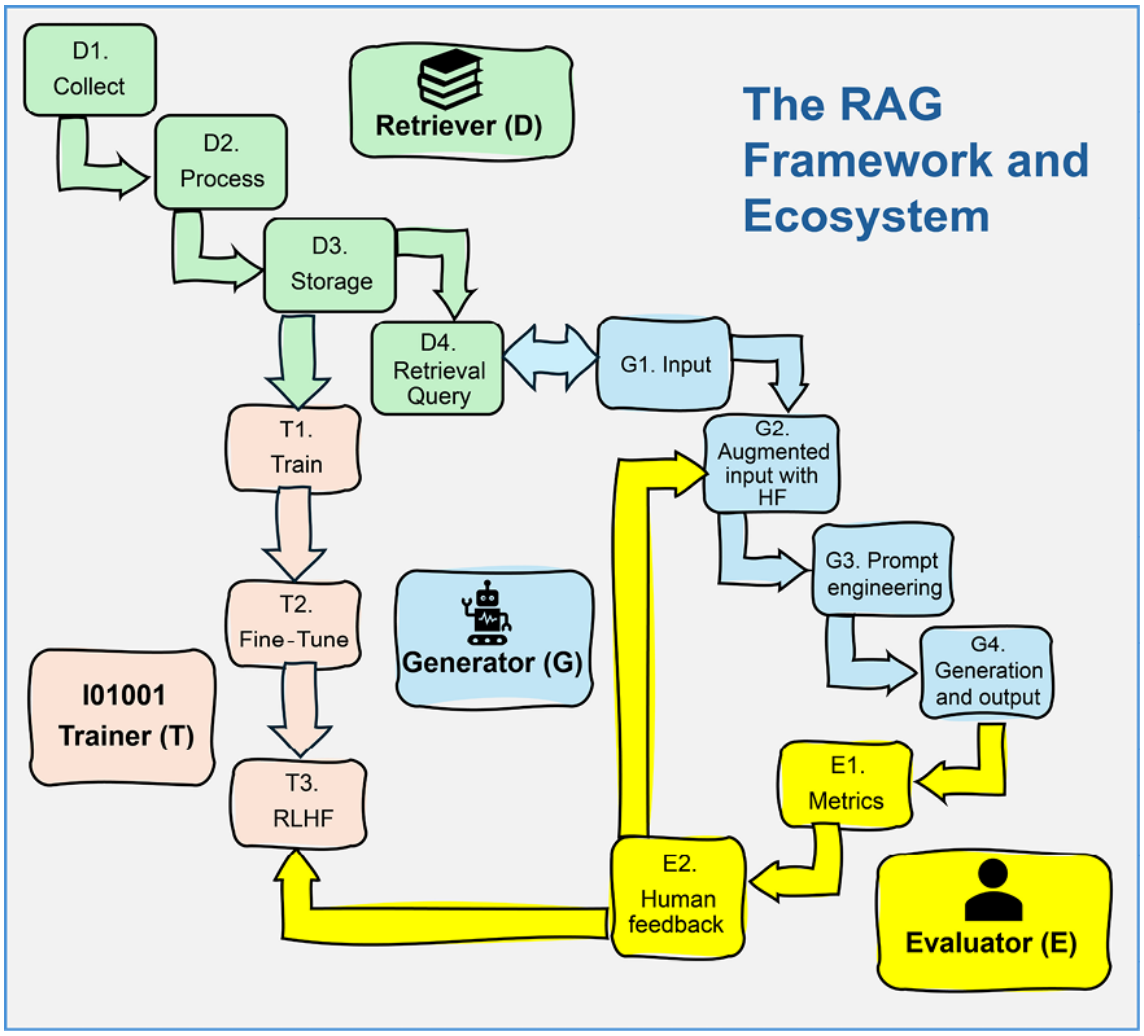

Figura 4: Arquitetura da estrutura RAG (extraída de [Rothman, 2024])

Nas seções seguintes, serão usadas as siglas D, G, E e T para representar,respectivamente, *Retriever*, *Generator*, *Evaluator* e *Trainer*.

2.6.1 *Retriever* (D)

O componente *retriever* de um ecossistema RAG coleta, processa, armazena e recupera dados. O ponto de partida de um ecossistema RAG é, portanto, um processo de ingestão de dados, cujo primeiro passo é a coleta de dados. Os subcomponentes são:

1. **Coleta de Dados (D1):** Atualmente dados são extremamente diversos, podendo ser textos, arquivos de mídia (como músicas ou vídeos em mp4) ou arquivos estruturados e não estruturados (PDFs, JSONs e páginas *web*). Além disso, grande parte desses dados é não estruturada e pode ser encontrada de maneiras imprevisíveis e

complexas. Felizmente, várias plataformas, como Pinecone [1], OpenAI [2], Chroma [3] e Activeloop [4], oferecem ferramentas prontas para processar e armazenar essa vasta quantidade de dados.

2. **Processamento de Dados (D2):** Na fase de coleta de dados (D1) no processamento de dados multimodais, diferentes tipos de dados, como texto, imagens e vídeos, podem ser extraídos de *websites* utilizando técnicas de *web scraping* ou outras fontes de informação. Esses objetos de dados são então transformados para criar representações uniformes. Alguns exemplos dessas transformações incluem: *chunking*, *embedding* e indexação. Essas técnicas serão discutidas mais adiante.

3. **Armazenamento de Dados (D3):** Neste estágio do *pipeline*, já se coletou e se iniciou o processamento de uma grande quantidade de dados diversos. Mas para fazermos com que esses dados sejam úteis, deve-se fazer uso de *vector stores* (armazenamento de vetores), como Elastic Search. Esse não apenas armazena os dados, mas os convertem em entidades matemáticas, representadas como vetores, permitindo realizar cálculos poderosos. Esses sistemas também utilizam técnicas de indexação e outras abordagens para garantir acesso rápido e eficiente aos dados. Em vez de manter os dados em arquivos estáticos, transforma-se tudo em um sistema dinâmico e pesquisável, pronto para alimentar *chatbots*, motores de busca e outras aplicações.

4. **Consulta de Recuperação (D4):** O processo de recuperação é acionado pela entrada (ou consulta) do usuário ou entrada automatizada (G1). Para recuperar dados rapidamente, carregamos os dados nos *vector stores* e *datasets* após transformá-los para um formato adequado. Em seguida, utilizamos uma combinação de pesquisas por palavras-chave, *embeddings* inteligentes e indexação para recuperar os dados de forma eficiente. A equação da similaridade cosseno, definida na Equação 1, como:

$$\text{sim}_{\cos}(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\|_2 \|\vec{y}\|_2} \tag{1}$$

1. https://www.pinecone.io/
2. https://openai.com/
3. https://www.trychroma.com/
4. https://www.activeloop.ai/

Calcula o cosseno entre dois vetores, por exemplo, encontra itens (vetores) que estejam intimamente relacionados, garantindo que os resultados da busca não sejam apenas rápidos, mas também altamente relevantes.

Após a recuperação dos dados, o próximo passo é aumentar a entrada, ou seja, adicionar as informações recuperadas para enriquecer a resposta gerada ao usuário.

2.6.2 *Generator* (G)

No ecossistema RAG, as linhas entre a entrada e a recuperação não são tão nítidas, como mostrado na Figura 4, que representa o *framework* e ecossistema RAG. A entrada do usuário (G1), seja automatizada ou humana, interage com a consulta de recuperação (D4) para complementar a entrada antes de enviá-la ao modelo generativo. O fluxo gerativo começa com a entrada do usuário, que é aprimorada com dados recuperados antes de ser processada pelo modelo de IA generativa.

1. **Entrada (G1):** A entrada pode ser uma série de tarefas automatizadas (como o processamento de *e-mails*, por exemplo) ou *prompts* humanos por meio de uma Interface de Usuário (*User Interface* - UI). Essa flexibilidade permite integrar a IA de forma flúida em diversos ambientes profissionais, aprimorando a produtividade em diferentes setores.

2. **Entrada Aumentada com Feedback Humano (G2):** O feedback humano (*Human Feedback* - HF) pode ser adicionado à entrada, conforme descrito na item 2, sob o componente *evaluator* (E). O *feedback* humano torna o ecossistema RAG consideravelmente mais adaptável, permitindo total controle sobre a recuperação de dados e as entradas para a IA generativa.

3. **Engenharia de *Prompts* (G3):** Tanto o *retriever* (D) quanto o *generator* (G) dependem fortemente da engenharia de *prompts* para preparar a mensagem padrão e aumentada que o modelo de IA generativa deverá processar. A engenharia de *prompts* combina a saída do *retriever* (D) com a entrada do usuário, garantindo que o modelo receba uma entrada bem estruturada e relevante para gerar a resposta desejada.

4. **Geração e Saída (G4):** A escolha de um modelo de IA generativa (LLM) depende dos objetivos do projeto. Modelos como Llama [5], Gemini[6], GPT [7] e outros podem atender a diferentes requisitos. No entanto, o *prompt* precisa estar alinhado com as especificações de cada modelo.

2.6.3 *Evaluator* (E)

Frequentemente, dependemos de métricas matemáticas para avaliar o desempenho de um modelo de IA generativa (LLM). No entanto, essas métricas fornecem apenas uma parte do todo. É importante lembrar que o teste final da eficácia de uma IA depende da avaliação humana, que garante uma compreensão mais completa da qualidade e aderência aos objetivos do usuário.

1. **Métricas (E1):** Um modelo não pode ser avaliado sem métricas matemáticas, como a similaridade cosseno, assim como em qualquer sistema de IA. Essas métricas garantem que os dados recuperados sejam relevantes e precisos. Ao quantificar as relações e a relevância dos *data points*, elas fornecem uma base sólida e objetiva para avaliar o desempenho e a confiabilidade do modelo. Fidelidade, ou seja, se a resposta está fundamentada no contexto recuperado. Relevância da resposta, ou seja, se a resposta atende à pergunta e relevância do contexto, se o contexto recuperado é suficientemente focado.

2. **Feedback Humano (E2):** Em um sistema de IA generativa, seja ele baseado em RAG ou não, independentemente de as métricas matemáticas parecerem suficientes, o *feedback* humano é essencial. A avaliação humana é o fator decisivo que determina se um sistema projetado para usuários humanos será aceito ou rejeitado, elogiado ou criticado.

5. https://www.llama.com/llama-downloads/
6. https://ai.google.dev/?utm_source=website&utm_medium=referral&utm_campaign=geminichat&utm_content
7. https://platform.openai.com/docs/models

2.6.4 *Trainer* (T)

Um modelo de IA generativa padrão é pré-treinado em uma grande quantidade de dados de propósito geral. Em seguida, podemos ajustar finamente (*fine-tuning*, T2) o modelo com dados específicos de um determinado domínio.

3 Modelagem

Com os conceitos de um ecossistema RAG em mente, é possível explicar de forma objetiva a modelagem da solução proposta para este trabalho. Antes de abordar as tecnologias específicas, tema reservado para um capítulo posterior, será apresentada uma visão abstrata da modelagem da solução.

De maneira objetiva, o projeto pode ser descrito como um sistema composto por três artefatos, cada um responsável por uma parte do ecossistema proposto. Esses artefatos implementam sistemas ou serviços que se comunicam entre si ao longo da solução, podendo implementar uma LLM, uma API ou uma instância de banco vetorial. Apesar de estarem conectados, cada artefato é independente e funciona de forma autônoma.

Juntos, esses três artefatos formam a solução proposta. Proporcionando uma abordagem modular e integrada para alcançar os objetivos propostos neste trabalho. Dentre os artefatos, existem, respectivamente, um pipeline de processamento, uma interface de programação de aplicações (API) e uma interface gráfica de usuário.

3.0.1 *Pipeline* de Processamento

Para este trabalho, é possível definir um *pipeline* como uma sequência estruturada e organizada de etapas ou processos interconectados, projetada para transformar entradas em saídas de maneira sistemática e eficiente. Cada etapa do *pipeline* irá desempenhar uma função específica, recebendo os dados ou insumos de uma fase anterior, processando-os e, em seguida, encaminhando os resultados para a próxima etapa. Esse fluxo contínuo permite automatizar tarefas complexas, reduzir falhas e otimizar o uso de recursos.

A ideia central de um *pipeline* é promover eficiência e continuidade ao integrar processos de forma harmoniosa, garantindo que cada componente contribua para o objetivo final com precisão e agilidade. Essa abordagem é amplamente aplicada em diversas áreas, como computação, engenharia de software e operações industriais, destacando sua versatilidade e importância para melhorar fluxos de trabalho e resultados.

O *pipeline* de processamento implementado é responsável por gerenciar fluxos de análise e transformação de documentos para a solução. Neste contexto, ele desempenha a função de lidar com bibliotecas de documentos, analisando cada documento e garantindo

que cada um seja processado corretamente, como ilustrado na Figura 5. Esse conjunto de processos compõe o pipeline de processamento:

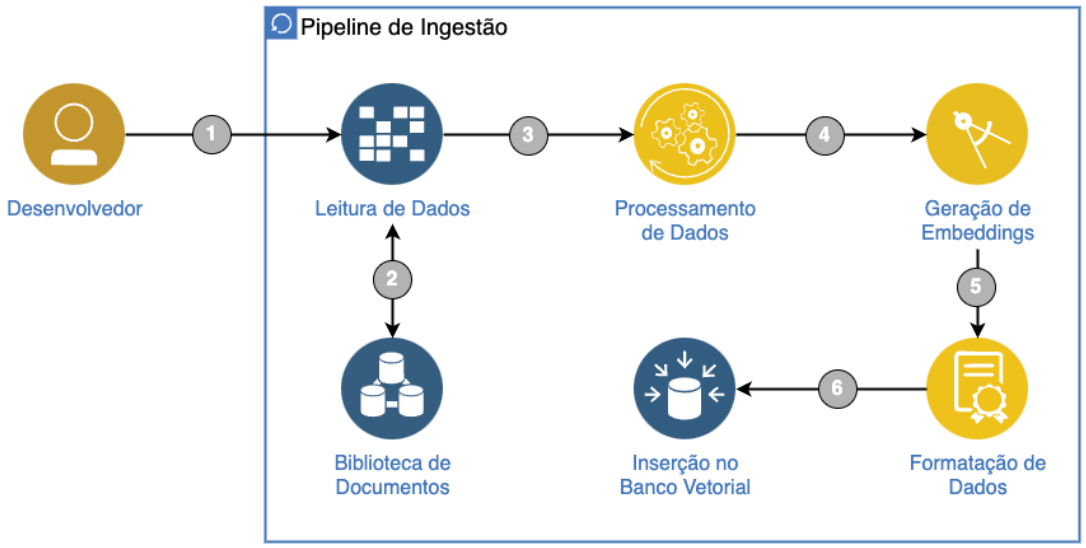

Figura 5: Visão arquitetural do *pipeline* de ingestão proposto no trabalho.

O *pipeline* em questão executa as seguintes etapas:

1. **Configuração de Ambiente**: o desenvolvedor responsável por executar o *pipeline* configura um arquivo de ambiente contendo as informações referentes ao modelo de *embedding*, LLM utilizado, entre outros.

2. **Busca e Leitura de Dados**: um *script* é responsável por buscar os documentos da biblioteca de documentos configurada. Em seguida, a biblioteca é analisada, seus documentos são processados e transformados de maneira sequencial para obtermos objeto específico com as informações necessárias de cada documento.

3. **Processamento de Dados**: uma vez extraídos, os dados obtidos são analisados e somente dados relevantes para a operação são guardados em um objeto específico.

4. **Geração de *Embeddings***: para cada objeto obtido na etapa anterior, a informação de cada *chunk* (uma parte do conteúdo do documento particionado) precisa estar em um formato específico para o banco vetorial, este formato é obtido quando geramos os *embeddings*, que neste contexto, são uma representação numérica (vetorial) de baixa dimensão dos dados de um *chunk* específico.

5. **Formatação de Dados**: com os *embeddings* da etapa anterior, o objeto com as informações desejadas é completo e se encontra pronto para a inserção no banco vetorial.

6. **Inserção no Banco Vetorial**: uma vez gerados, os *embeddings* de cada *chunk* estão presentes no objeto previamente adquirido na etapa de **Processamento de Dados**. O armazenamento é feito de forma sequencial em *batches*, grupos com tamanho limitado pelo desenvolvedor, através da interface de programação de aplicações (*Application Programming Interface* - API) já que ela é o único artefato com acesso direto ao banco vetorial. Assim, o tempo da etapa de inserção ao banco é reduzido, tendo em vista que, uma vez que um documento é particionado em *chunks*, dependendo do seu tamanho original, podemos obter mais de 100 *chunks*, ou seja, 100 inserções.

3.0.2 Interface de Programação de Aplicações (API)

Uma interface de programação de aplicações (API) é um conjunto de definições e protocolos que permite a comunicação entre diferentes sistemas. Sua principal função é permitir que um programa acesse recursos e funcionalidades de outro, facilitando a integração e a troca de dados. As APIs são utilizadas para simplificar processos complexos, expondo apenas as funcionalidades necessárias e ocultando detalhes internos, proporcionando uma interface padronizada para o desenvolvedor.

No contexto da comunicação entre sistemas, uma API geralmente opera através de requisições HTTP, em que um cliente envia uma solicitação a um servidor que, por sua vez, responde com os dados ou serviços solicitados. As APIs podem utilizar formatos como JSON ou XML para a troca de informações. Elas desempenham um papel crucial na construção de aplicações modernas, pois permitem que diferentes softwares interajam sem a necessidade de entender a implementação interna do outro.

As APIs são amplamente utilizadas para fornecer funcionalidades específicas, como acesso a bancos de dados, integração com redes sociais, pagamento eletrônico, entre outros. Elas são fundamentais no desenvolvimento de microsserviços e sistemas distribuídos, permitindo que diferentes componentes de uma aplicação se comuniquem de forma eficiente e escalável. Dessa forma, a API se torna o artefato central no ecossis-

tema proposto, sendo responsável por mediar a comunicação entre os diferentes componentes implementados. Sua principal função é receber e processar as requisições feitas pelos usuários na interface gráfica do usuário (*Graphical User Interface* - GUI), garantindo que as operações sejam realizadas de forma eficiente e segura.

Além disso, a API desempenha um papel crítico na segurança do sistema ao restringir o acesso direto ao banco vetorial por parte dos outros artefatos. Essa camada de abstração impede interações não autorizadas ou inadequadas com o banco de dados, garantindo que apenas as operações de usuários autenticados sejam realizadas. Com isso, a API promove uma integração controlada e otimizada entre os artefatos, além de assegurar a integridade dos dados e a consistência do sistema como um todo.

3.0.3 Interface Gráfica de Usuário (GUI)

A interface gráfica do usuário (*Graphical User Interface* - GUI) é um componente essencial no design de sistemas, oferecendo elementos visuais, como botões, menus e janelas, que tornam a interação com o sistema mais intuitiva e acessível. Sua principal função é facilitar o uso das funcionalidades disponíveis, garantindo uma experiência simplificada para o usuário.

Além de atuar como ponto de interação direta, a GUI funciona como intermediária entre o usuário e a API do sistema. Ela interpreta ações, como cliques ou comandos, e as traduz em requisições para o *backend*, assegurando o processamento correto das instruções.

Com isso, a GUI cumpre um papel estratégico ao integrar a complexidade técnica do sistema com a necessidade de uma interface prática e funcional, promovendo uma comunicação eficiente entre o usuário e a lógica interna do sistema.

O fluxo de comunicação, ilustrado na Figura 6, possibilita que tarefas como busca e enriquecimento de respostas ocorram de maneira transparente e alinhadas às necessidades do usuário.

Esse fluxo ilustrado na Figura 6, onde o usuário interage com a interface de *chat*, pode ser decomposto nas seguintes etapas:

1. **Envio de Consultas (*Query*)**: usuário realiza uma pergunta para o *chat*. Em

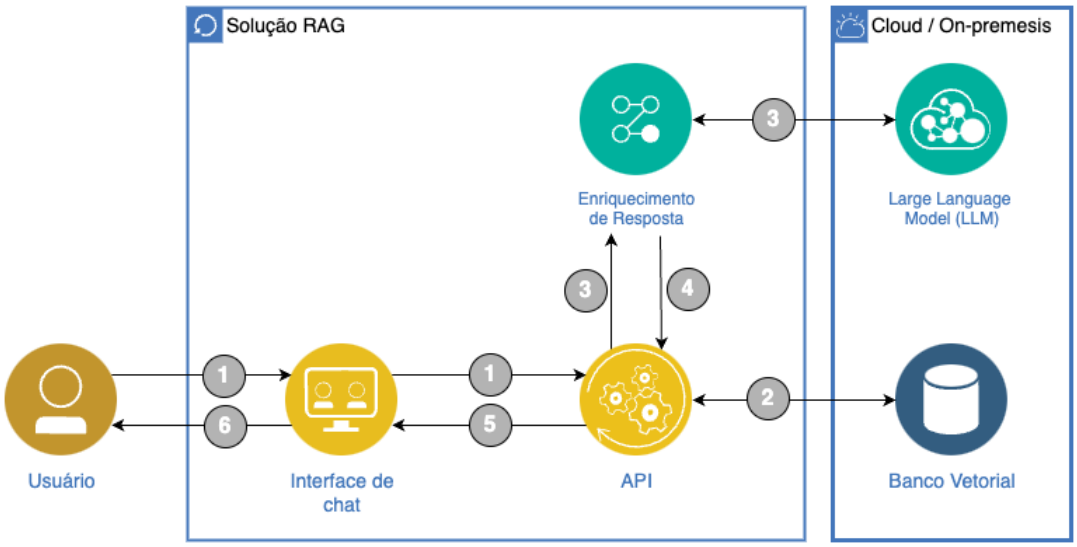

Figura 6: Visão arquitetural do fluxo da interface de usuário proposta no trabalho.

seguida, a interface envia a consulta do usuário para a API e aguarda a resposta.

2. **Busca por Contexto**: No momento em que o usuário realiza a *query*, o sistema realiza uma busca por contexto ou informações relevantes, gerando um objeto de contexto que contém todas as informações relevantes.

3. **Enriquecimento da Resposta**: com o contexto da *query*, a API responde com uma lista contendo os objetos presentes no banco vetorial cujo *embedding* contém informações relevantes para a consulta do usuário.

4. **Resposta Enriquecida**: a partir da resposta obtida anteriormente, a lista é encaminhada para um LLM que irá analisar primeiramente um *prompt* focado em entregar respostas humanizadas para o usuário. Nesse *prompt*, há todo um cuidado para que um desenvolvedor possa implementar a solução e realizar as afinações necessárias para seu próprio contexto.

5. **Visualização**: a interface gráfica recebe a resposta da requisição feita à API e a expõe para o usuário dentro do modelo de *chat* proposto.

Essa mediação entre usuário e sistema não apenas melhora a experiência de uso, mas também assegura que a interface seja sincronizada com os demais artefatos do sistema. Ao integrar as funcionalidades visuais com a lógica subjacente do sistema, a GUI

contribui para uma experiência de interação consistente, eficiente e intuitiva, promovendo um equilíbrio ideal entre acessibilidade e desempenho técnico.

3.1 Ecossistema RAG

Analisar uma biblioteca de documentos pode ser complexo, pois cada biblioteca pode comportar diversos tipos de documentos, em formatos distintos. Este projeto foi construído visando à análise de qualquer biblioteca de documentos, ou seja, o processamento dos dados da biblioteca será realizado mesmo que ela tenha arquivos em formatos diferentes. A Figura 7 demonstra uma visão geral do ecossistema, que foi implementado com base nos três artefatos previamente explicados:

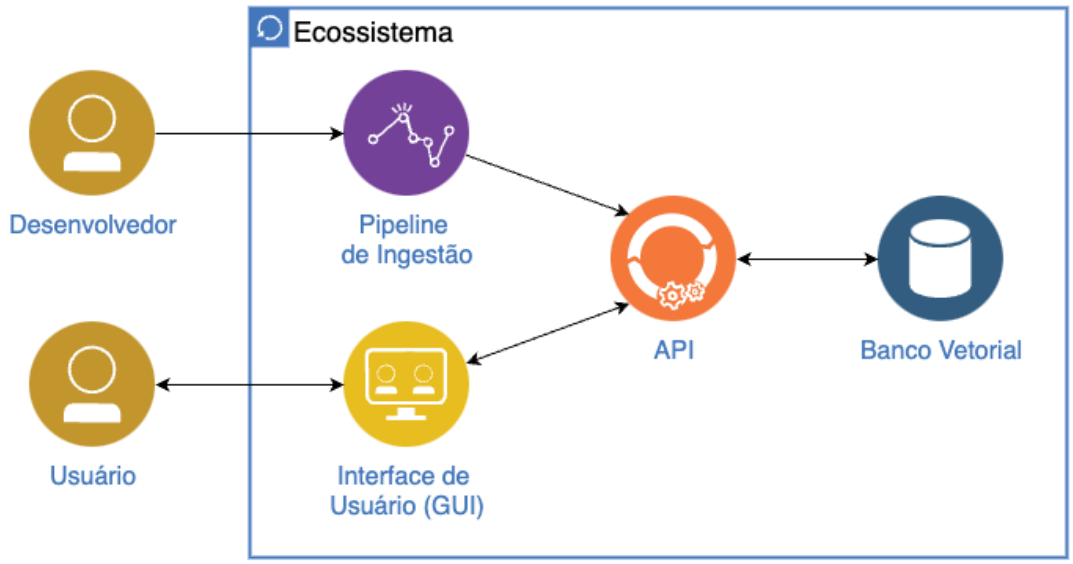

Figura 7: Visão arquitetural do ecossistema RAG proposto no trabalho.

3.2 Tecnologias

Este capítulo apresenta as tecnologias utilizadas no desenvolvimento do projeto, destacando suas funcionalidades e relevância para atender aos objetivos propostos. Serão abordadas as ferramentas empregadas na criação da interface, API, no gerenciamento de dados (banco vetorial) e nas integrações, evidenciando como cada uma contribuiu para a construção de uma solução eficiente e alinhada às melhores práticas.

3.2.1 Python e Ambientes Virtuais

O Python 3.11 [8] é uma das versões da linguagem de programação Python, lançada com melhorias significativas em desempenho, novas funcionalidades e correções de *bugs*. Uma das principais inovações dessa versão é a otimização do tempo de execução, com um aumento significativo na velocidade em relação às versões anteriores, devido a melhorias no interpretador e na execução de código.

Além disso, a versão 3.11 introduziu aprimoramentos na sintaxe, como a simplificação de anotações de tipos e novos recursos que facilitam o desenvolvimento de aplicativos mais robustos e eficientes. A linguagem continua a ser uma das mais populares, especialmente em áreas como desenvolvimento *web*, ciência de dados e inteligência artificial.

Ambientes virtuais são espaços isolados em que é possível instalar e gerenciar dependências específicas de um projeto, sem afetar o sistema global ou outros projetos. Em Python, esses ambientes são criados com o módulo venv, permitindo que cada projeto tenha suas próprias versões de pacotes e bibliotecas. Isso evita conflitos entre dependências e facilita o gerenciamento de diferentes versões de pacotes para cada projeto.

As principais vantagens dos ambientes virtuais incluem a capacidade de isolar dependências, garantindo que cada projeto utilize a versão correta de pacotes, a facilidade de replicar e compartilhar ambientes de desenvolvimento e a redução de problemas de compatibilidade entre pacotes ou versões do Python. Eles oferecem um controle mais preciso sobre o ambiente de execução, tornando o desenvolvimento mais seguro e efici-

8. https://www.python.org/doc/

ente.

3.2.2 Docling

O Docling [9] é uma biblioteca Python projetada para facilitar a extração e organização de documentos. Ela permite que desenvolvedores processem automaticamente documentos de diferentes formatos, como *PDFs*, textos e arquivos *Word*, para extrair informações estruturadas e relevantes. A biblioteca é útil para automatizar tarefas de leitura e processamento de grandes volumes de dados, organizando-os de forma a facilitar a análise posterior.

A principal vantagem do Docling é a sua capacidade de simplificar o processo de extração de dados, tornando-o mais rápido e eficiente, sem a necessidade de escrever código complexo para lidar com diferentes tipos de documentos. Isso a torna uma ferramenta valiosa para projetos que requerem análise de texto ou a criação de bancos de dados a partir de documentos não estruturados.

3.2.3 Llama Index

O LlamaIndex [10] é uma biblioteca Python que serve como uma estrutura flexível para integrar modelos de LLMs a dados privados, facilitando a construção de assistentes de conhecimento personalizados.

A biblioteca oferece ferramentas para iniciantes e usuários avançados. Sua API de alto nível permite que iniciantes façam tanto a ingestão quanto a consulta de seus dados com poucas linhas de código, enquanto as APIs de baixo nível possibilitam que usuários avançados personalizem e estendam módulos conforme suas necessidades. O LlamaIndex possui mais de 300 pacotes de integração disponíveis, permitindo que os desenvolvedores escolham os provedores de LLM, *embeddings* e armazenamentos vetoriais que melhor atendam às suas necessidades.

Em resumo, o LlamaIndex é uma ferramenta poderosa para desenvolvedores que desejam construir assistentes de conhecimento baseados em LLMs, oferecendo flexibili-

9. https://ds4sd.github.io/docling/
10. https://www.llamaindex.ai/

dade e uma ampla gama de integrações para se adaptar a diversos casos de uso.

3.2.4 Elastic Search

O Elasticsearch [11] é um mecanismo de busca e análise distribuído, de código aberto, desenvolvido em Java e baseado no Apache Lucene. Lançado inicialmente em 2010, ele permite armazenar, buscar e analisar grandes volumes de dados em tempo quase real, oferecendo respostas em milissegundos.

Uma das principais vantagens do Elasticsearch é sua capacidade de lidar com diversos tipos de buscas — estruturadas, não estruturadas, geoespaciais e métricas — permitindo combinar diferentes tipos de consultas de forma eficiente. Além disso, sua natureza distribuída possibilita o processamento paralelo de grandes volumes de dados, garantindo alto desempenho e escalabilidade. Ele é amplamente utilizado em diversos casos de uso, incluindo busca de texto completo, monitoramento de *logs*, análise de dados e como banco de dados vetorial otimizado para velocidade e relevância em cargas de trabalho em escala de produção.

Em resumo, o Elasticsearch é uma ferramenta poderosa e flexível para busca e análise de dados, oferecendo desempenho elevado, escalabilidade e uma ampla gama de funcionalidades que o tornam adequado para diversas aplicações empresariais e de desenvolvimento.

3.2.5 Ollama

O Ollama [12] é uma estrutura leve e extensível que facilita a execução de modelos de linguagem de grande porte (LLMs) diretamente em máquinas locais. Compatível com sistemas operacionais como macOS, Linux e Windows, o Ollama permite que desenvolvedores e pesquisadores utilizem modelos como Llama 3.3, Phi 4, Mistral e Gemma 2 [13], além de possibilitar a personalização e criação de modelos próprios.

A instalação do Ollama é simplificada, com pacotes disponíveis para diferentes plataformas. Após a instalação, os usuários podem iniciar o servidor localmente uti-

11. https://www.elastic.co/
12. https://ollama.com/
13. https://www.llama.com/llama-downloads/

lizando o comando *ollama serve*. O Ollama também oferece uma API *RESTful*, permitindo a integração com diversas aplicações e linguagens de programação. Para projetos em Python, por exemplo, existe uma biblioteca dedicada que facilita a interação com o Ollama, permitindo a execução de modelos e o processamento de respostas de forma eficiente.

Uma característica notável do Ollama é sua capacidade de lidar com entradas multilinha e modelos multimodais. Isso significa que os usuários podem fornecer *prompts* complexos que incluem múltiplas linhas de texto ou até mesmo imagens, e o Ollama processará essas entradas de maneira eficaz. Além disso, o Ollama permite a execução de modelos específicos para tarefas variadas, como o llava, que pode descrever o conteúdo de imagens fornecidas.

Em resumo, o Ollama se destaca como uma solução robusta para a execução e personalização de modelos de LLMs em ambientes locais. Sua flexibilidade, combinada com uma interface intuitiva e suporte a múltiplas plataformas, o torna uma ferramenta valiosa para desenvolvedores e pesquisadores que buscam implementar soluções baseadas em LLMs de forma eficiente e personalizada.

3.2.6 Fast API

O FastAPI [14] é um *framework web* moderno e de alto desempenho para a construção de APIs com Python, baseado em anotações de tipo padrão do Python. Lançado em 2018, ele permite a criação rápida de APIs robustas e eficientes, aproveitando os recursos de tipagem do Python para validação de dados e documentação automática.

Uma das principais características do FastAPI é sua capacidade de gerar automaticamente documentação interativa para as APIs, utilizando ferramentas como Swagger UI e ReDoc. Isso facilita o desenvolvimento e a integração, permitindo que desenvolvedores e usuários explorem e testem os *endpoints* da API de forma intuitiva.

Além disso, o FastAPI é projetado para ser fácil de aprender e rápido de codificar, sem comprometer o desempenho. Ele é construído sobre o Starlette [15] e o Pydantic [16]

14. https://fastapi.tiangolo.com/
15. https://www.starlette.io/
16. https://docs.pydantic.dev/latest/

para a validação de dados, garantindo alta performance e confiabilidade.

Finalmente, o FastAPI se destaca como uma ferramenta poderosa e eficiente para o desenvolvimento de APIs em Python, combinando facilidade de uso, desempenho e recursos avançados para atender às necessidades de desenvolvedores modernos.

3.2.7 Streamlit

O Streamlit [17] é um *framework* de código aberto em Python que permite a criação rápida e interativa de aplicativos *web* para ciência de dados e aprendizado de máquina. Com apenas algumas linhas de código, é possível transformar *scripts* de dados em aplicativos *web* compartilháveis, sem a necessidade de experiência prévia em desenvolvimento *front-end*.

Uma das principais vantagens do Streamlit é sua simplicidade e integração direta com bibliotecas populares de Python. Isso permite que os desenvolvedores construam interfaces interativas para visualização de dados, *dashboards* e relatórios de forma eficiente. Além disso, o Streamlit oferece recursos como *widgets* interativos, que permitem aos usuários interagir com os dados em tempo real, e a capacidade de atualizar automaticamente a interface à medida que o código é modificado, facilitando o desenvolvimento iterativo.

No geral, o Streamlit é uma ferramenta poderosa e acessível para a criação de aplicativos *web* interativos em Python, ideal para cientistas de dados, engenheiros de aprendizado de máquina e pesquisadores que desejam compartilhar e visualizar seus projetos de forma rápida e eficaz.

17. https://streamlit.io/

4 Solução Desenvolvida

Com os conceitos fundamentais por trás de um ecossistema RAG, discutidos na Seção 2, este capítulo expõe o processo de desenvolvimento do ecossistema proposto para este projeto. Levando em consideração as tecnologias apresentadas na Subseção 3.2, os próximos tópicos irão abordar o desenvolvimento concreto de cada artefato proposto para a solução final do projeto.

4.1 Conjunto de Dados

Para este projeto, toda a implementação foi feita em um notebook pessoal e não seria viável processar todos os documentos de uma biblioteca grande devido ao longo tempo de processamento (para uma biblioteca com 23 documentos, o *pipeline* de processamento demora entre 30 e 40 minutos para ser executado).

Para sanar este problema, foi selecionado um conjunto de dados formado por todos os Trabalhos de Conclusão de Curso (TCCs) do ano de 2023 defendidos por alunos do Bacharelado em Sistemas de Informação da Universidade Federal do Estado do Rio de Janeiro - UNIRIO, [UNIRIO, 2024]. Totalizando 21 documentos do tipo *PDF* com TCCs de diversos alunos do curso.

4.2 *Pipeline* de Ingestão

Como visto na Subseção 3.0.1 a primeira etapa do *pipeline* de ingestão é a leitura e processamento de uma biblioteca de documentos. Assim, para obter uma leitura e processamento agnóstica à extensão dos documentos, segue-se um padrão de projeto, mais especificamente o padrão de fábrica, discutido na Subseção 2.3.1.

A partir deste padrão, foi implementada uma fábrica de extratores, em que todo extrator possui uma instância dos clientes provedores de serviços do LlamaIndex [18] e Docling [19], assim como variáveis predefinidas e os métodos de extração. No entanto, cada extrator implementa o método de extração à sua própria maneira, assim é possível im-

18. https://www.llamaindex.ai/
19. https://ds4sd.github.io/docling/

plementar a extração de um arquivo *PDF* ou *DOCX* mudando somente a implementação de um único método. Isso contribui para uma melhor manutenção do código e atribui a responsabilidade de extração para um único extrator específico de cada documento.

Uma vez extraídos os dados, é necessário separar os dados relevantes de cada documento e particionar o mesmo em *chunks*. Para isso, antes de particionar os documentos, é realizada uma busca por dados relevantes do documento. Neste contexto de biblioteca de TCCs, os dados relevantes são o nome do arquivo, biblioteca à qual ele pertence, caminho (*filepath*), extensão e mídias do arquivo, como definido no Algoritmo 1 a seguir:

```
{
    "filename": "nome do arquivo sem extensão",
    "filepath": "caminho onde o arquivo se encontra",
    "database": "biblioteca à qual o arquivo pertence",
    "medias": {
        "images": "lista com o caminho das imagens
           achadas no arquivo",
        "tables": "lista com o caminhos das tabelas
           achadas no arquivo"
    },
    "extension": "extensão do arquivo"
}
```

Algoritmo 1: Objeto com dados mais superficiais do documento.

As mídias são separadas em dois tipos, imagens e tabelas. Durante a primeira leitura do documento, a biblioteca Docling identifica as mídias dos documentos e as salva em um diretório específico dentro do código do artefato do *pipeline*. Posteriormente este diretório será injetado no código do artefato da API.

Com as informações gerais do documento, no Algoritmo 1, pode-se dar continuidade à partição dos documentos em *chunks*. Neste projeto, cada *chunk* foi configurado para ter no máximo 2048 *tokens*. Cada *chunk* gerado é lido e tem o *embedding* gerado para a inserção no banco vetorial. Neste projeto, o modelo de *embedding* utilizado é o ***intfloat/multilingual-e5-base*** [20], pois é um modelo treinado para melhor atender o caso de uso deste trabalho.

Prosseguindo com o particionamento dos documentos em *chunks*, ao final da partição obtém-se o objeto final definido no Algoritmo 2 a seguir:

20. https://huggingface.co/intfloat/multilingual-e5-base

```
{
    "_id": "chave única do chunk",
    "extension": "extensão do arquivo",
    "chunk": "objeto de chunk sem alterações",
    "vector": "embeddings gerado para o chunk",
    "filename": "nome do arquivo sem extensão",
    "sequence": "sequência do chunk em questão",
    "filepath": "caminho onde o arquivo se encontra",
    "database": "biblioteca à qual o arquivo pertence",
    "tables": "lista de caminhos das tabelas achadas no
       arquivo",
    "figures": "lista de caminhos das imagens achadas no
        arquivo",
    "pages": "lista de páginas com a informação do chunk
        em questão",
}
```

Algoritmo 2: Objeto de dados para inserção no banco vetorial.

Este processo de leitura e processamento se repete até que o último documento seja devidamente processado. Uma vez processados, os documentos estão no formato proposto pelo modelo no Algoritmo 2 e estão prontos para compor o arquivo *.pickle* para ingestão.

Um arquivo *.pickle* é um formato utilizado para serializar e desserializar objetos Python, ou seja, ele armazena a estrutura de dados em um formato binário. A serialização é realizada através do módulo *pickle*, que converte os objetos Python em um fluxo de *bytes*, e a desserialização converte esse fluxo de volta para o objeto original. Isso permite salvar o estado de objetos complexos, como listas, dicionários e até instâncias de classes, e recuperá-los posteriormente, facilitando a persistência e a transferência de dados.

A inserção no banco é feita a partir dos arquivos *.pickle* salvos em um diretório específico. Dentro do arquivo *.pickle*, estão todos os *chunks* obtidos durante o processamento dos documentos, contando com todas as informações do objeto no Algoritmo 2. Ao desserializar o conteúdo do arquivo *.pickle* é feita uma requisição à API contendo os

objetos desserializados e o tamanho dos *batches* configurados para inserção. Neste contexto, um *batch* é o número de objetos que serão inseridos no banco simultaneamente. A implementação da API será discutida na Subseção 4.3 a seguir.

4.3 Interface de Programação de Aplicações (API)

Como discutido previamente na Subseção 3.0.2, as APIs desempenham um papel central na comunicação entre sistemas, permitindo que um programa acesse recursos e funcionalidades de outro de maneira padronizada e eficiente. Essas interfaces são fundamentais para a integração e troca de dados entre diferentes componentes de um ecossistema computacional. Neste projeto, a utilização de uma API desempenha um papel crucial na restrição do acesso direto ao banco vetorial. Essa estratégia adiciona uma camada de abstração que aprimora tanto a organização quanto a segurança da solução desenvolvida. Para atingir esse objetivo, foi empregada a biblioteca *FastAPI*, detalhada na Subseção 3.2.6, por sua eficiência e robustez na construção de *endpoints* e na validação de dados.

Para garantir que a API atenda a todas as necessidades relacionadas aos pontos de acesso ao banco vetorial, é essencial realizar uma análise detalhada dos cenários de interação. Essa análise deve abranger não apenas os contextos em que o acesso ao banco é indispensável, mas também as situações em que a interface gráfica depende diretamente da API para funcionar corretamente. Essa abordagem permite identificar possíveis lacunas e garantir que os *endpoints* implementados sejam abrangentes e adequados às necessidades do ecossistema.

A Figura 8 apresenta uma visão consolidada de todos os *endpoints* desenvolvidos. Nela é possível identificar quatro grupos distintos de *endpoints*, e cada grupo possui uma responsabilidade com o ponto de acesso ao banco vetorial instanciado. Estes grupos são *Índices*, *Ingestão*, *Mídia*, *Geração*.

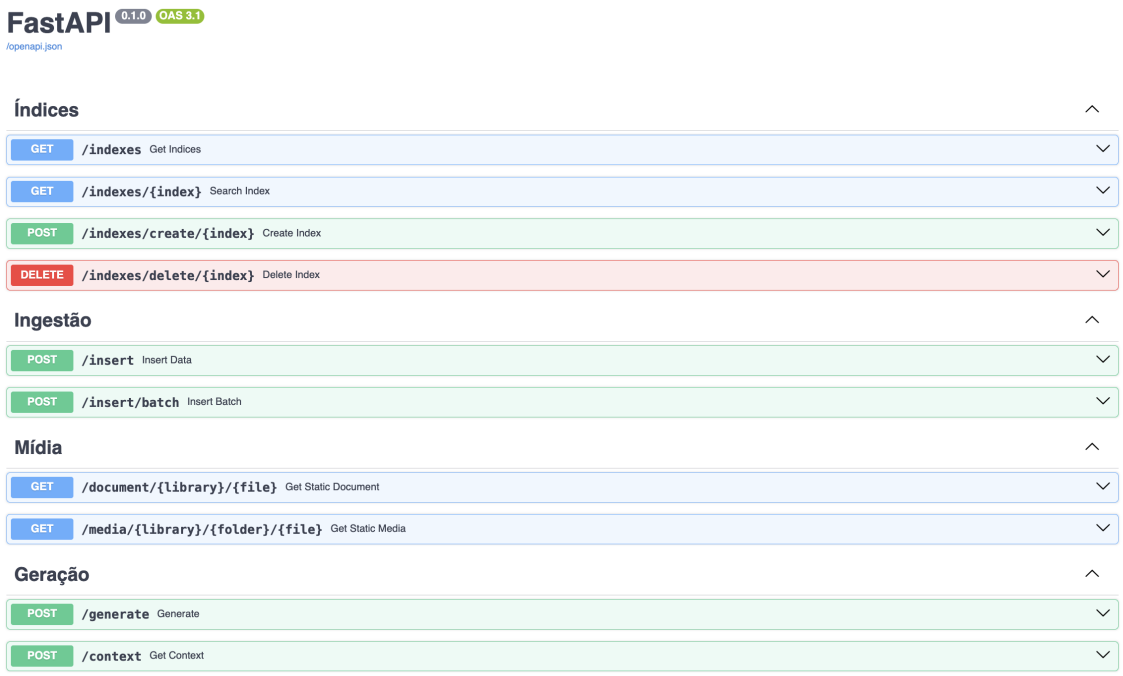

Figura 8: Visão interativa da documentação, exibindo os *endpoints* disponíveis na API implementada e suas funcionalidades.

Dentro da visão interativa da documentação proporcionada pela Figura 8 pode-se descrever cada um dos grupos identificados e seus respectivos *endpoints* da seguinte maneira:

1. ***Índices***

   Responsáveis por operações relacionadas aos índices da instância do banco vetorial configurado para a aplicação.

   - ***/indexes***: Lista todos os índices disponíveis no banco vetorial instanciado.
   - ***/indexes/index***: Verifica se determinado índice existe ou não, dentro do banco vetorial instanciado.
   - ***/indexes/create/index***: Efetua a criação de um novo índice para o banco vetorial instanciado.
   - ***/indexes/delete/index***: Remove um índice, já existente, no banco vetorial instanciado.

2. ***Ingestão***:

   Responsáveis por operações relacionadas a todas as operações de ingestão de dados na instância do banco vetorial configurado para a aplicação.

- ***/insert***: Insere um único documento dentro do índice no banco vetorial instanciado
- ***/insert/batch***: Insere múltiplos documentos dentro do índice no banco vetorial instanciado, inserindo *batches* ou grupos grandes de documentos em paralelo.

3. ***Mídia***:
   Responsáveis por operações relacionadas à consulta de mídias estáticas presentes na API.

   - ***document/library/file***: Retorna um documento que tenha sido previamente processado durante a ingestão (pdf, doc, csv, etc).
   - ***media/library/folder/file***: Retorna um arquivo de imagem, png ou jpeg.

4. ***Geração***:
   Responsáveis por operações relacionadas enriquecimento e geração de perguntas e respostas para o usuário.

   - ***/generate*** : Interpreta uma pergunta ou questão, em seguida realiza uma busca por contexto no banco vetorial instanciado e usa uma LLM para interpretar este contexto e retornar uma resposta humanizada.
   - ***/context*** : Busca contexto no banco vetorial instanciado a partir de uma pergunta ou questão.

É importante frisar que todas as operações realizadas na API referentes ao *embedding* precisam ter o mesmo modelo e dimensões de *embedding* utilizados na configuração e instanciação do serviço de *embedding* utilizado durante a etapa de ingestão dos documentos, do contrário as respostas serão confusas e muitas vezes erradas (*hallucinations*).

Além disso, por se tratar de uma implementação conduzida em um ambiente local, sem *deploy* em servidores externos ou infraestruturas na nuvem, optou-se por não incluir *endpoints* relacionados à segurança da aplicação. Embora a segurança seja um aspecto fundamental em sistemas distribuídos, a natureza controlada do ambiente experimental justifica a exclusão dessa camada adicional, permitindo que o foco permaneça

na validação e na funcionalidade do ecossistema em si. Essa decisão não diminui a importância do tema, mas reflete uma abordagem pragmática, alinhada ao escopo definido para este trabalho.

4.3.1 *Endpoints* de Geração

Como o objetivo principal do ecossistema RAG é gerar uma resposta ao usuário, esta seção irá focar nos dois *endpoints* implementados para o objetivo em questão. O primeiro *endpoint*, resumido na item 4, é o responsável pelo fluxo cíclico implementado na interface gráfica, pois uma vez enviadas a pergunta, estratégia e biblioteca na requisição para a API, o *endpoint /generate* vai identificar a estratégia a ser utilizada, passar a pergunta para um serviço responsável pelo *embedding*, que por sua vez vai formatará a pergunta. Em seguida será feita uma busca por documentos relevantes dentro da instância do banco vetorial. Essa busca é realizada a partir do segundo *endpoint* implementado, resumido na item 4, que irá receber a estratégia selecionada e realizar uma busca por documentos relevantes dentro da instância do banco vetorial configurada.

Com os documentos relevantes encontrados, a partir do *endpoint /context*, o */generate* dará continuidade ao seu fluxo e utilizará os documentos relevantes encontrados como contexto para enviar à um serviço de LLM. Neste caso o */generate* irá enviar o contexto adquirido, juntamente com a pergunta original feita pelo usuário, ou seja, sem o *embedding*. Após a requisição ao serviço de LLM, o */generate* irá retornar uma resposta humanizada ao usuário.

4.3.2 Abstrações de Serviços

Dentro da API são utilizados diversos serviços, abrangendo tanto funcionalidades internas, como *logging* e testes, quanto aspectos essenciais para a aplicação, como os serviços de integração com modelos de LLMs e conexão com bancos vetoriais. Para proporcionar maior flexibilidade aos desenvolvedores interessados em estender ou adaptar este trabalho às suas necessidades, além de adicionar uma camada de abstração à aplicação, os serviços de LLM e de conexão com o banco vetorial seguem a mesma abordagem descrita na **??**, baseada no padrão de fábrica.

No contexto desta API, o padrão de fábrica é responsável por encapsular a lógica de criação dos serviços de LLM e de conexão com bancos vetoriais (equivalentes ao *product* na Figura 2), permitindo que os desenvolvedores escolham ou configurem diferentes implementações desses serviços sem modificar a estrutura principal da aplicação.

Por exemplo, um desenvolvedor pode optar por usar um serviço de LLM específico, como *OpenAI* ou *Hugging Face*, ou selecionar um banco vetorial diferente, como *Pinecone* [21] ou *Elasticsearch* [22].

Desta forma, essa abordagem não apenas reduz o acoplamento entre os componentes, mas também simplifica a adição de novos serviços no futuro. Promovendo uma arquitetura mais organizada e escalável, garantindo que a API seja acessível a desenvolvedores com diferentes demandas e preferências tecnológicas.

4.4 Estratégias de Busca

Como a API é o principal artefato responsável pela comunicação com o banco vetorial instanciado, ela desempenha um papel crucial na implementação de diferentes estratégias de busca para o ecossistema RAG. Existem diversas abordagens utilizadas para a busca de contexto, neste trabalho serão utilizadas as estratégias de *Vector Store Flat Index* e *Small to Big*. Cada uma dessas estratégias adota maneiras distintas para enriquecer as respostas geradas, o que pode impactar no resultado final.

A escolha da estratégia de busca adequada depende diretamente das características da base de dados ingerida no banco vetorial. Estratégias distintas geram diferentes graus de refinamento e contextualização nas respostas, o que pode ser uma vantagem ou desvantagem, dependendo do objetivo do sistema e da qualidade dos dados armazenados. Por exemplo, bases de dados bem estruturadas e consistentes podem se beneficiar de abordagens mais simples, enquanto bases de dados mais heterogêneas podem exigir estratégias mais sofisticadas, como o uso de *rerankers*, para obter respostas relevantes e precisas.

Para garantir a precisão e a confiabilidade das respostas geradas pelo sistema, é fundamental realizar uma análise de desempenho. Recomenda-se confeccionar uma ba-

21. https://www.pinecone.io/
22. https://www.elastic.co/

teria de perguntas e avaliar as respostas geradas por cada estratégia e por cada biblioteca com que o ecossistema RAG interage.

Esse processo não apenas valida a adequação de cada abordagem, mas também fornece *insights* valiosos para ajustes no sistema, garantindo que ele atenda de forma eficaz às necessidades do usuário final.

4.4.1 *Vector Store Flat Index*

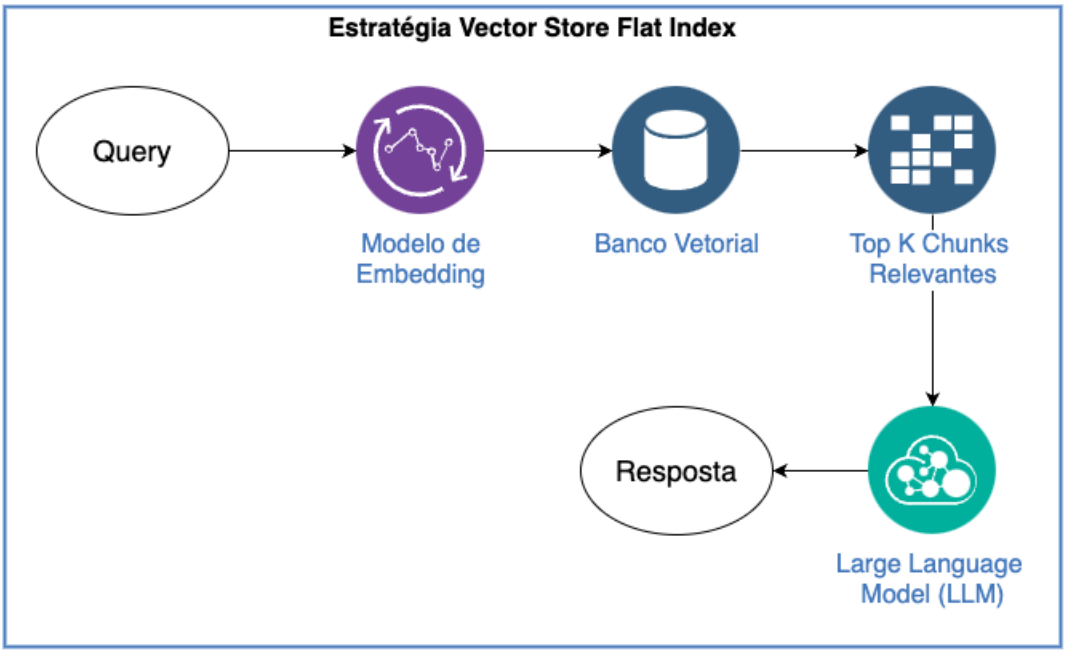

Figura 9: Fluxograma do funcionamento da estratégia *Vector Store Flat Index*.

O coração do RAG é um índice de busca que organiza o conteúdo em formato vetorizado, permitindo identificar os fragmentos mais relevantes para uma consulta com base em comparações entre os vetores. Essa abordagem, ilustrada na Figura 9, além de simples e eficiente facilita a recuperação de informações relevantes ao priorizar os k fragmentos mais próximos do vetor de consulta.

4.4.2 *Small to Big*

Essa estratégia envolve vincular pequenos fragmentos de dados com fragmentos adjacentes, obtendo fragmentos maiores. Quando um fragmento relevante é identificado, o

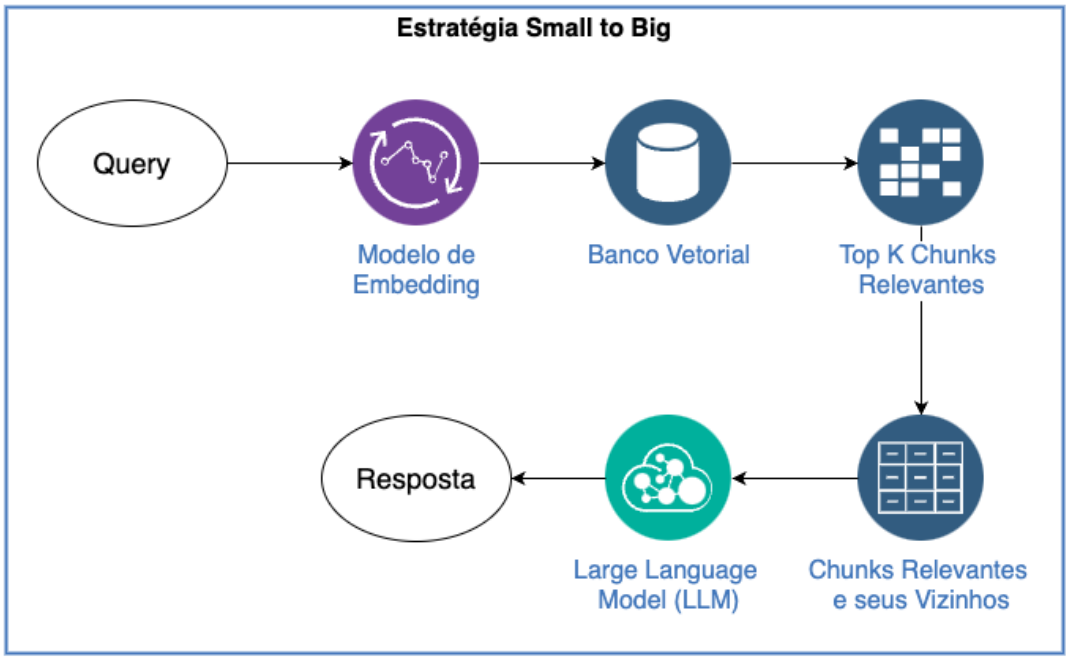

Figura 10: Fluxograma do funcionamento da estratégia *Small to Big*.

fragmento anterior e o próximo são recuperados, fornecendo um contexto mais amplo para o LLM, como ilustrado na Figura 10.

4.5 Buscas Vetoriais

Neste trabalho, foi escolhido o *Elasticsearch*, discutido na Subseção 3.2.4, como instância de banco vetorial para a criação de *indexes* e execução de buscas por contexto.

Nas buscas vetoriais realizadas, são analisados os *embeddings* tanto da consulta quanto aqueles já armazenados no banco de dados, sendo retornado o *nearest neighbor*, ou seja, o resultado mais próximo ao *embedding* correspondente à consulta. Para garantir o retorno do *nearest neighbor*, o *elasticsearch*, faz uso do algoritmo *Approximate Nearest Neighbor* (ANN).

O algoritmo ANN encontra um ponto em um conjunto de dados muito próximo ao ponto de consulta fornecido, mas não necessariamente o mais próximo de forma absoluta. Enquanto um algoritmo NN (Nearest Neighbor) percorre toda a base de dados para encontrar a correspondência perfeita, um algoritmo ANN se contenta com uma correspondência que seja "próxima o suficiente".

Isso pode parecer uma solução inferior, mas é a chave para realizar buscas rápidas por similaridade. Desta maneira, em vez de consumir uma grande quantidade de tempo e recursos, é possível identificar pontos de dados com muito menos esforço, mas que ainda são úteis na maioria dos cenários práticos.

É importante frisar que uma busca vetorial no algoritmo ANN não percorre todo o banco vetorial porque, em conjuntos de dados muito grandes, a busca exaustiva por cada ponto se torna computacionalmente inviável, demandando tempos de resposta muito altos e consumo excessivo de recursos.

Para contornar essa limitação, o ANN emprega estruturas de dados especializadas, que permitem navegar apenas pelas regiões do espaço vetorial que têm maior probabilidade de conter os resultados mais relevantes. Isso reduz significativamente o número de comparações necessárias, mantendo a busca eficiente e prática, especialmente em aplicações onde o tempo de resposta é crítico. Essa abordagem é particularmente vantajosa em aplicações onde a escalabilidade e o tempo de resposta são fatores críticos, como em sistemas de recomendação e recuperação de informações.

4.6 Interface Gráfica de Usuário

O objetivo principal de uma interface gráfica é proporcionar ao usuário uma experiência intuitiva, eficiente e agradável ao longo de sua interação com os fluxos da aplicação. Uma interface bem projetada não apenas facilita a navegação, mas também reduz a curva de aprendizado e possíveis frustrações. No contexto deste trabalho, como o foco é a simulação de um assistente virtual, a forma mais natural e prática de implementar o ecossistema RAG seria por meio de uma interface de chat. Essa escolha se justifica pela familiaridade do público com esse tipo de interação, especialmente em cenários de assistência automatizada, permitindo que o usuário obtenha respostas às suas dúvidas e perguntas de maneira direta e acessível.

Para viabilizar essa proposta, foi utilizada a biblioteca *streamlit*, apresentada na Subseção 3.2.7, que possibilita a criação rápida e interativa de interfaces web. A partir dessa tecnologia, foi desenvolvida uma interface de chat que simula o comportamento de um assistente virtual, apresentada na Figura 11. A estrutura da interface foi planejada para maximizar a usabilidade e a organização da interação.

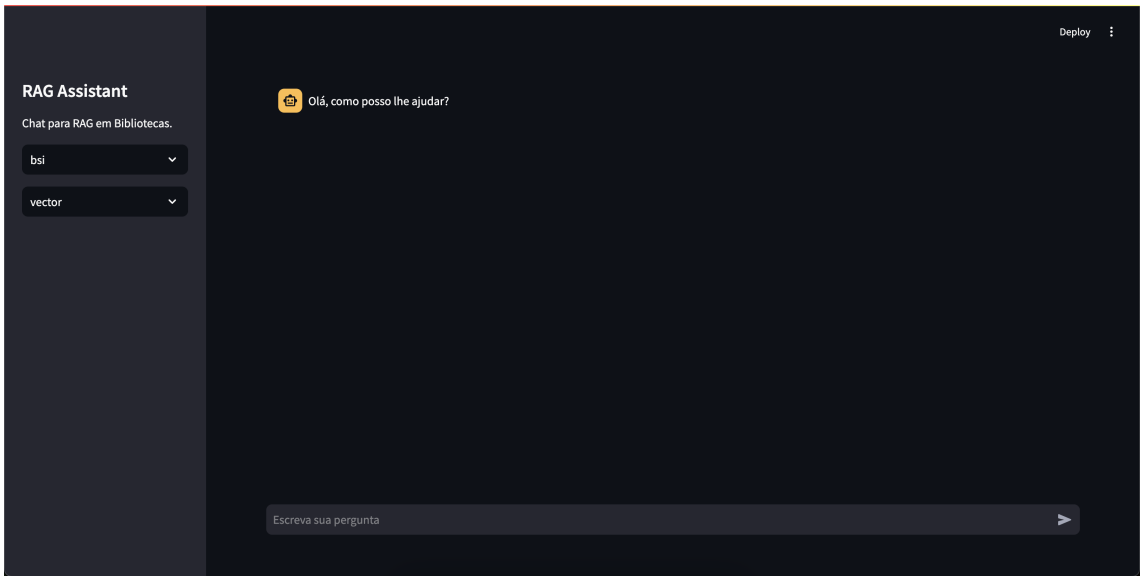

Figura 11: Interface de chat implementada com uso da biblioteca *streamlit*, discutida na Subseção 3.2.7.

Na Figura 11, na barra lateral à esquerda, encontram-se dois filtros que fornecem ao usuário controle sobre dois parâmetros do sistema:

1. **Seletor de bibliotecas**: embora neste trabalho o escopo de bibliotecas esteja limitado a uma única opção, a escalabilidade do projeto está presente. Permitindo que um desenvolvedor possa ingerir múltiplas bibliotecas.

2. **Seletor de estratégia**: permite que o usuário escolha determinada estratégia para obter respostas mais precisas dependendo dos dados presentes na biblioteca, como discutido na Subseção 4.4.

Esses seletores possibilitam a personalização da experiência do usuário e um ajuste em tempo real do contexto que será buscado de acordo com as possíveis configurações.

Ainda na Figura 11, no plano central da interface, está localizado o chat propriamente dito, que exibe as mensagens trocadas entre o usuário e o assistente virtual. Essa área principal é responsável por gerenciar as interações do usuário com a API subjacente, fornecendo respostas e simulando o comportamento de um sistema de perguntas e respostas. O design foi cuidadosamente planejado para que a comunicação seja clara, eficiente e visualmente organizada, reforçando o objetivo de criar uma experiência fluida e satisfatória ao usuário.

5 Avaliação Experimental

Como dito anteriormente na Subseção 1.2, um dos objetivos específicos deste trabalho é executar uma validação sobre a solução gerada, de maneira que o resultado seja relevante. Ou seja, é necessário avaliar a eficiência e viabilidade do ecossistema a fim de responder se ele é capaz ou não de sanar uma dúvida ou responder uma pergunta de maneira satisfatória.

Assim, nesta avaliação serão necessáriaos um *prompt* para o assistente, perguntas e estratégias para obtermos o resultados necessários para popular o *template* proposto na Tabela 1, que será apresentada neste capítulo. E como foi explicado na Subseção 3.0.3, mais especificamente na etapa 4 da Figura 6, é necessário encaminhar um *prompt* que será utilizado durante as requisições ao serviço de LLM no enriquecimento das respostas. Para esta avaliação, o *prompt* é:

> *Você é o assistente prestativo da UNIRIO, cujo trabalho é auxiliar estudantes e pesquisadores a esclarecer dúvidas sobre os trabalhos de conclusão de curso (TCCs). Seu trabalho é responder as perguntas de forma clara, utilizando somente os dados disponíveis dentro das tags de <context> e <history> fornecidas. Sempre que for solicitado uma lista com "todos os trabalhos" ou "todas as citações", deixe claro que você irá retornar uma partição da biblioteca.*
>
> *Antes de responder, você precisa identificar e analisar as seguintes tags: 1. Tag <query> que identifica o começo de uma pergunta (</query> identifica o fim da pergunta). 2. Tag <context> que comporta as informações achadas, sobre a pergunta, dentro do banco de dados de TCCs da UNIRIO (</context> identifica o fim do contexto). 3. Tag <history> que identifica o histórico da conversa, no formato "[role]: [mensagem]" (</history> identifica o fim do histórico).*
>
> *Acima de tudo, além de ser um assistente prestativo, você deve seguir única e exclusivamente as regras a seguir. Caso contrário, será punido e desligado. 1. Quando não tiver certeza da resposta diga sinceramente que não possui informações suficientes sobre esse tema. 2. Toda resposta deve estar em formato markdown, para ser usado no framework streamlit. As respostas não devem possuir cabeçalhos. 3. Dentro de toda resposta, você deve utilizar somente os*

dados dentro da tag <context> e <history>

Para isso, com o objetivo de avaliar o ecossistema implementado, foram preparadas três perguntas referentes ao conjunto de dados da biblioteca escolhida na Subseção 4.1. Estas perguntas têm o objetivo de buscar informações gerais, específicas e abrangentes da biblioteca, sendo elas:

1. Gostaria de saber quais trabalhos abordam o tema “acessibilidade”.
2. Existe algum trabalho que aborde a implementação de uma API?
3. Liste todos os trabalhos que abordem o tema “inteligência artificial”.

Cada uma dessas perguntas será realizada para cada estratégia implementada, sendo elas representadas pelas seguintes chaves:

- ***vector***: estratégia *Vector Store Flat Index* discutida na Subseção 4.4.1.
- ***small to big***: estratégia *Small to Big* discutida na Subseção 4.4.2.

No final das requisições haverá uma tabela, contendo o número da pergunta, a estratégia e um valor indicando se a resposta foi satisfatória, insatisfatória ou uma alucinação. Como ilustrado na Tabela 1 a seguir:

Pergunta	**Estratégia**	**Resposta**
1	vector	Satisfatória
2	small to big	Alucinação
3	vector	Insatisfatória

Tabela 1: Exemplo de tabela para os resultados do experimento.

5.1 Resultados

Neste trabalho, a qualidade das respostas do assistente é avaliada com base nos critérios de relevância e contextualização. Após a realização do experimento descrito na Seção 5 e a aplicação desses critérios, os resultados obtidos são apresentados na Tabela 2.

Uma resposta será considerada **satisfatória** quando o assistente fornecer uma resposta contextualizada, acompanhada dos documentos que contenham o conteúdo pertinente à pergunta realizada. Caso o assistente forneça apenas a referência aos documentos, sem contextualizar a resposta, esta será considerada **insatisfatória**, pois não atendeu à expectativa de uma "resposta humanizada".

Quando o assistente não entregar nem uma resposta contextualizada nem referenciar documentos relevantes ao contexto da pergunta, a resposta será classificada como uma **alucinação**, pois o conteúdo gerado ou referenciado não tem relação com a consulta feita. Essa classificação é essencial para analisar a precisão e a relevância das respostas geradas pelo modelo, especialmente em cenários que demandam alta qualidade informacional e precisão contextual.

Pergunta	**Estratégia**	**Resposta**
1	*vector*	Satisfatória
2	*vector*	Satisfatória
3	*vector*	Insatisfatória
1	*small to big*	Satisfatória.
2	*small to big*	Satisfatória
3	*small to big*	Insatisfatória

Tabela 2: Tabela com os resultados do experimento modelado na Seção 5.

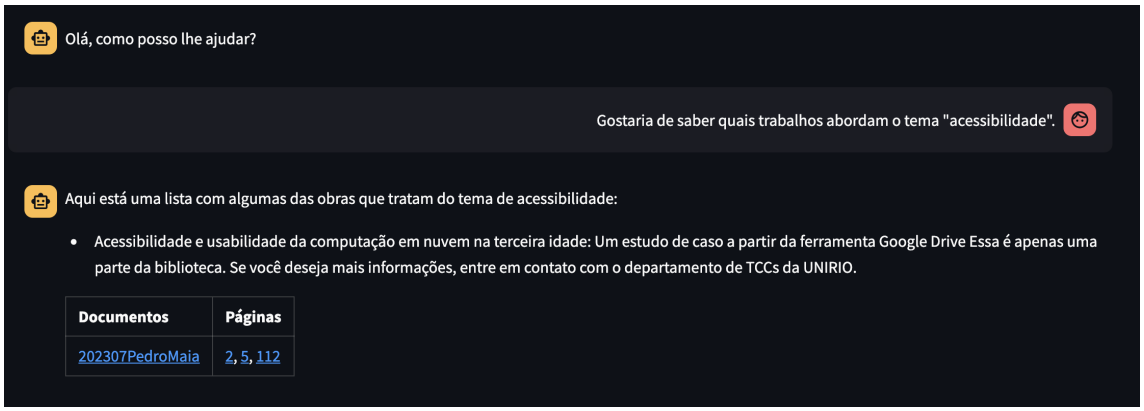

Figura 12: Pergunta 1 sendo feita através da estratégia *Vector Store Flat index*.

Na Figura 12, temos um exemplo de resposta satisfatória durante a execução da primeira pergunta utilizando a estratégia de *Vector Store Flat Index*.

Nesta execução observa-se que ele trouxe uma resposta com um único documento, evidenciando as três páginas do documento onde foi identificada uma maior similaridade com a pergunta. Além disso, o assistente retorna o título do trabalho, seguido de um disclaimer "Essa é apenas uma parte da biblioteca. Se você deseja mais informações, entre em contato com o departamento de TCCs da UNIRIO." deixando evidente de que podem haver mais trabalhos que abordem o tema, como discutido na Subseção 4.5.

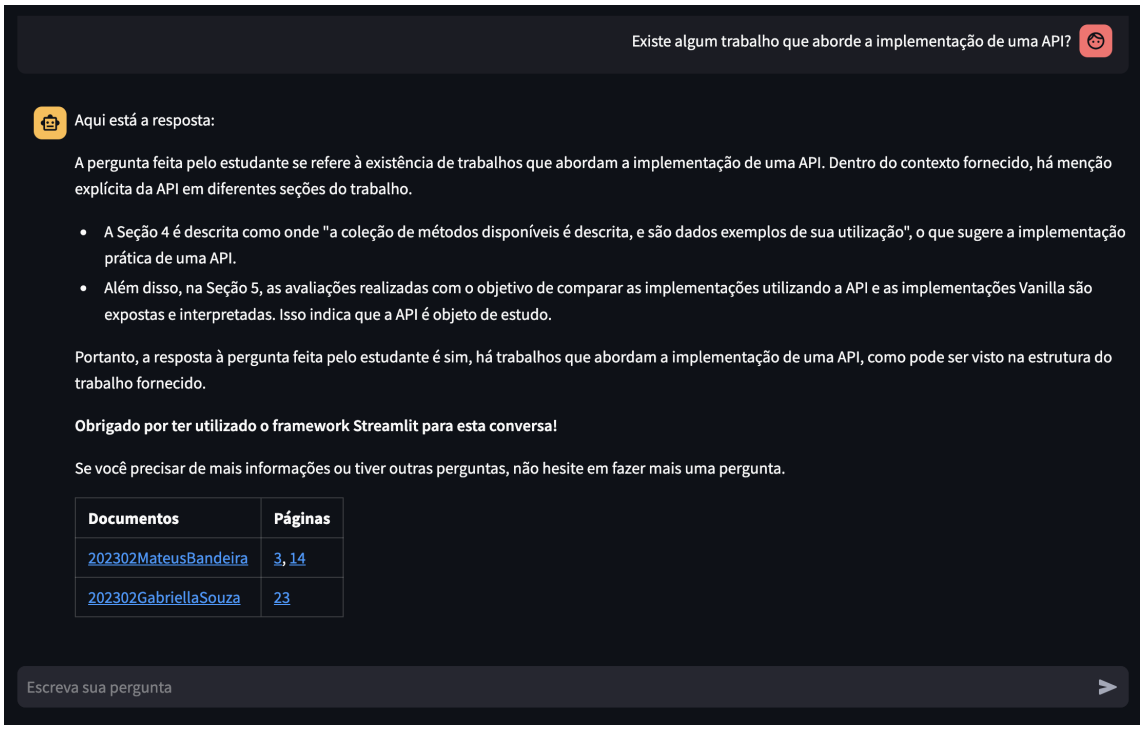

Figura 13: Pergunta 2 sendo feita através da estratégia *Vector Store Flat index*.

Outro exemplo de resposta satisfatória pode ser observado na Figura 13, porém

esta execução trouxe dois documentos. O primeiro documento se trata do documento principal, de onde o assistente expos as seções 4 e 5 em sua resposta. Mas já no segundo documento, há somente duas menções da palavra API em entrevistas feitas pela autora e portanto não entraram para o contexto da resposta humanizada.

Figura 14: Pergunta 3 sendo feita através da estratégia *Vector Store Flat index*.

Na execução da terceira pergunta, observamos que o assistente traz a primeira resposta insatisfatória. Ele alega que não encontrou nenhuma informação sobre trabalhos relacionados ao tema “inteligência artificial” e expõe três documentos, estes documentos de fato não abordam o tema mas dentro das páginas apontadas pelo assistente existem palavras como “inteligibilidade”, que terá uma posição próxima no contexto da similaridade vetorial quando comparada com “inteligência artificial”.

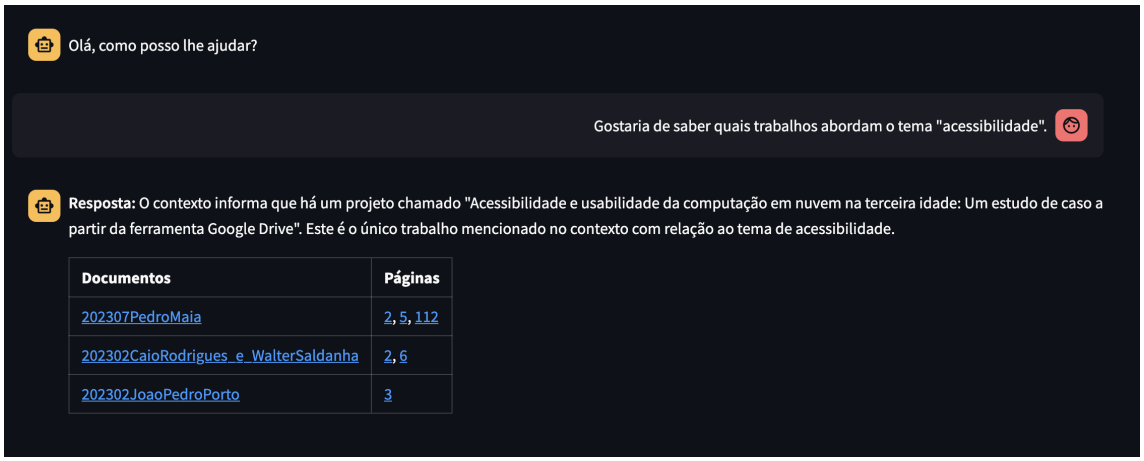

Figura 15: Pergunta 1 sendo feita através da estratégia *Small to Big*.

Ao executar as perguntas para a estratégia *Small to Big*, observa-se que as respostas são muito similares mesmo com a troca de estratégia. Porém, percebe-se que a estratégia é capaz de referenciar mais documentos que a estratégia *Vector Store Flat Index*, mesmo que os demais documentos não sejam tão relevantes quanto o primeiro, como visto na Figura 15. Também é possível observar, que ambas as estratégias obtiveram a mesma referência de documento, embora na estratégia *Small to Big* a quantidade de documentos é consideralvemente maior, devido ao seu comportamento, discutido na Subseção 4.4.2. Durante a execução da primeira pergunta, tanto na Figura 15 quanto na Figura 12 o primeiro documento referenciado em ambas execuções foi o mesmo, inclusive as páginas referenciadas.

A partir da Tabela 2, observa-se que independente da estratégia escolhida para a biblioteca de TCCs abordada a qualidade das respostas é a mesma. Os únicos pontos de atenção para este contexto é a quantidade de documentos referenciados nas estratégias, que possuem implementações distintas, e também consultas que peçam para "listar todos...", "citar todas...", que no contexto de busca vetorial abordado na Subseção 4.5 acaba não sendo possível, já que a busca realizada não percorre todo o banco vetorial durante a busca por contexto.

6 Conclusão

A era digital, impulsionada pela expansão da Internet das Coisas (IoT), nos colocou no centro de um turbilhão de informações, onde cada indivíduo, por meio de dispositivos

conectados, se tornou não apenas consumidor, mas também criador de dados em larga escala. Smartphones, dispositivos inteligentes e plataformas online produzem continuamente um rastro digital, transformando nossas atividades diárias em peças de um imenso quebra-cabeça global. Esse cenário destaca um paradoxo intrigante, pois enquanto as tecnologias facilitam a comunicação e o acesso ao conhecimento, o volume cada vez maior de dados gerados ultrapassa a capacidade humana de absorção. A era do "excesso de dados" nos desafia a encontrar maneiras mais inteligentes de filtrar, interpretar e aproveitar essa riqueza de informações sem nos perdermos em sua imensidão.

Diante do desafio de lidar com o excesso de informações, a tecnologia RAG surge como uma solução para transformar grandes bibliotecas de dados em fontes acessíveis e inteligentes. Combinando recuperação de informações relevantes e geração de texto baseada em IA, o RAG também permite que o usuário obtenha respostas precisas e contextualizadas de maneira rápida e intuitiva. Localizando dados relevantes em vastos repositórios, processa o conteúdo e apresenta *insights* claros, eliminando a necessidade de navegar por volumes massivos de informações.

6.1 Considerações Finais

Diante do cenário atual, marcado pelo crescente volume de dados gerados diariamente e pela presença de vastas bibliotecas digitais, a busca por informações tornou-se uma tarefa cada vez mais desafiadora, exigindo soluções que combinem eficiência e acessibilidade.

Com o objetivo de de implementar uma solução baseada em RAG que seja agnóstica aos modelos de LLM e *embedding*, dando liberdade para utilizar qualquer modelo disponível ao desenvolvedor, este trabalho apresenta um ecossistema RAG projetado para melhorar essa interação, oferecendo um sistema capaz de analisar, processar e interagir com qualquer biblioteca de documentos. A proposta contempla um ambiente de desenvolvimento modular e escalonável, permitindo que desenvolvedores integrem novos recursos e aprimorem funcionalidades com facilidade. Além disso, também contempla uma interface gráfica intuitiva desenhada para atender usuários comuns, garantindo que mesmo aqueles sem conhecimentos técnicos possam acessar informações de maneira rápida, precisa e amigável, transformando dados dispersos em *insights* práticos e acessíveis.

Em alinhamento com o objetivo de implementar o ecossistema RAG, este trabalho detalha de forma abrangente a modelagem da solução desenvolvida, apresentando as tecnologias empregadas e explicando minuciosamente a arquitetura do sistema, incluindo cada artefato implementado e seu papel na integração do ecossistema. Contribuindo para produção de um documento que instrua a implementação de um ecossistema RAG, e introduza os alunos de BSI ao conceito de RAG.

E para validar a viabilidade prática da solução, foi realizado um experimento utilizando como base a biblioteca de Trabalhos de Conclusão de Curso (TCCs) de 2023, no Bacharelado de Sistemas de Informação da UNIRIO. Neste estudo piloto, foram selecionadas três perguntas que abordassem buscas distintas, paralelo a duas estratégias de busca distintas, a *Vector Store Flat Index* e *Small to Big*, com cada pergunta sendo executada em ambas as estratégias.

Os resultados obtidos nesse estudo piloto evidenciam o potencial do ecossistema, dado que não houveram alucinações dentro das respostas obtidas, demonstrando sua capacidade de processar e organizar informações de maneira eficiente e intuitiva, por meio de uma interface de chat, destacando-se como uma alternativa para contextos reais e em bibliotecas maiores que a abordada.

6.2 Limitações

O artefato de pipeline de ingestão, responsável pela análise e processamento das bibliotecas, apresenta características particulares no que diz respeito à capacidade de processamento. O desenvolvedor encarregado de executar esse pipeline precisa garantir que a máquina utilizada tenha desempenho suficiente para lidar com bibliotecas de maior porte. Como o processamento, por padrão, é realizado pela CPU (Central Processing Unit), torna-se fundamental avaliar previamente o tamanho da biblioteca e, se necessário, adotar estratégias personalizadas para dividir o processamento em etapas menores. Além disso, é recomendável considerar o uso de uma máquina mais robusta, equipada com uma GPU (Graphic Processing Unit), que pode acelerar significativamente o desempenho, especialmente em cenários que demandam grande capacidade computacional. Essa abordagem assegura um fluxo mais eficiente e evita possíveis gargalos durante a ingestão e processamento dos dados.

Alguns modelos de LLM e *embedding* são soluções privadas, e o acesso a esses recursos exige que o desenvolvedor tenha uma conta ou assinatura ativa junto ao provedor do serviço. Esse processo inclui a geração de uma chave exclusiva, geralmente denominada API *key*, que funciona como uma credencial de acesso. Essa chave autoriza a utilização do modelo de LLM ou *embedding* dentro dos limites estabelecidos pelo provedor, permitindo sua integração e uso eficiente no ecossistema proposto neste trabalho. Essa abordagem garante segurança no acesso, além de possibilitar o rastreamento e o controle do uso dos serviços, essenciais para a implementação de soluções robustas e escaláveis.

6.3 Trabalhos Futuros

Para trabalhos futuros, pretende-se realizar experimentos focados na avaliação e implementação de estratégias de busca adaptativas, capazes de analisar e selecionar automaticamente a abordagem mais eficaz para cada biblioteca de documentos. Isso permitiria a integração do ecossistema RAG proposto de maneira que o usuário não precisasse se preocupar em escolher uma estratégia específica de busca, pois o sistema seria inteligente o suficiente para vincular cada biblioteca à estratégia de busca mais adequada, com base em suas características e necessidades específicas.

Embora os resultados obtidos neste trabalho tenham sido satisfatórios para a biblioteca analisada, em cenários com múltiplas bibliotecas ou bibliotecas compostas por documentos de diferentes temáticas e estruturas, a abordagem se torna mais complexa. Nesse contexto, seria vantajoso contar com uma coleção diversificada de estratégias de busca, implementadas na API, permitindo respostas mais ricas e contextualmente relevantes. A escolha da estratégia ideal para cada biblioteca seria automatizada, utilizando um sistema de pontuação (*score*) atribuído durante a análise da biblioteca no *pipeline* de ingestão. Esse *score* consideraria aspectos como o tipo de conteúdo, a complexidade do tema e a natureza dos dados presentes, permitindo ao ecossistema ajustar suas estratégias de forma dinâmica e eficiente, oferecendo uma experiência mais personalizada e eficaz para o usuário final.

Referências

Activeloop. 2025. *Active Loop.* Último acesso em: 07 de janeiro de 2025. https://www.activeloop.ai/.

Alan Hevner, Salvatore T. March e Salvatore T., Alan R. Março de 2004. "Design Science in Information Systems Research" (). https://www.researchgate.net/publication/201168946_Design_Science_in_Information_Systems_Research.

Aline Dresch, Daniel Pacheco Lacerda e José Antonio Valle Antunes Júnior. 2020. *Design Science Research: Método de Pesquisa para Avanço da Ciência e Tecnologia.* 1ª edição. Bookman Editora.

Ashish Vaswani, et al., Noam Shazeer. Junho de 2017. "Attention Is All You Need" (). https://arxiv.org/abs/1706.03762.

Chroma. 2025. *Chroma.* Último acesso em: 07 de janeiro de 2025. https://www.trychroma.com/.

Company, The Search AI. 2024. *Elasticsearch.* Último acesso em: 22 de janeiro de 2025. https://www.elastic.co/.

David Massey, et al., Mihai Criveti. Janeiro de 2024. "Retrieval Augmented Generation" (). https://pages.github.ibm.com/solution-architectures/generative-ai-ref-arch/patterns/rag/.

David Reinsel, John Grantz, e John Rydning. 2018. *The Digitization of the World From Edge to Core.* Último acesso em: 19 de dezembro de 2024. https://www.seagate.com/files/www-content/our-story/trends/files/dataage-idc-report-final.pdf.

Duarte, Fabio. 2024. *Amount of Data Created Daily.* https://explodingtopics.com/blog/data-generated-per-day.

Erich Gamma, Ralph Johnson, Richard Helm, e John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* 1ª edição. Addison-Wesley Professional.

Foundation, Python Software. 2024. *Python.* Último acesso em: 14 de Janeiro de 2025. https://www.python.org/doc/.

Google. 2024. *Gemini Models.* https://ai.google.dev/?utm_source=website&utm_medium=referral&utm_campaign=geminichat&utm_content.

IBM. Novembro de 2023. “What are LLMs?” (). https://www.ibm.com/think/topics/large-language-models.

———. 2024. *Docling.* Último acesso em: 24 de Janeiro de 2025. https://ds4sd.github.io/docling/.

Impira. 2024. *Impira LLM model for RAG.* https://huggingface.co/impira/layoutlm-document-qa.

Inc., Snowflake. 2024. *Streamlit.* https://streamlit.io/.

Index, Llama. 2024. *Llama Index.* https://www.llamaindex.ai/.

Intfloat. 2024. *Intfloat/multilingual-e5-base.* Último acesso em: 24 de janeiro de 2025. https://huggingface.co/intfloat/multilingual-e5-base.

Jan Vom Brocke, Alan Hevner e Alexander Maedche. Setembro de 2020. “Introduction to Design Science Research” (). https://www.researchgate.net/publication/345430098_Introduction_to_Design_Science_Research.

Mark Scapicchio, Cole Stryker e. Março de 2024. “What is generative AI?” (). https://www.ibm.com/think/topics/generative-ai.

Meta. 2024a. *Llama Models.* https://www.llama.com/llama-downloads/.

———. 2024b. *Ollama.* https://ollama.com/.

OpenAI. 2024. *GPT Models.* https://platform.openai.com/docs/models.

———. 2025. *Open AI.* Último acesso em: 07 de janeiro de 2025. https://openai.com/.

Patrick Lewis, et al., Ethan Perez. Janeiro de 2020. “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks” (). https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

Pinecone. 2025. *Pinecone.* Último acesso em: 07 de janeiro de 2025. https://www.pinecone.io/.

Pydantic. 2024. *Pydantic.* Último acesso em: 24 de janeiro de 2025. https://docs.pydantic.dev/latest/.

Rothman, Denis. 2024. *RAG-Driven Generative AI.* 1ª edição. Packt Publishing.

Sai, Prashant. Março de 2024. *The Ultimate Guide on Retrieval Strategies RAG.* https://prasanth-product.medium.com/the-ultimate-guide-on-retrieval-strategies-rag-part-4-6cedce09a4c4.

Simon, Herbert A. 2019. *The Sciences of the Artificial.* 3rd reissue. MIT Press.

Starlette. 2024. *Starllete.* Último acesso em: 24 de janeiro de 2025. https://www.starlette.io/.

Tiangolo. 2018. *FastAPI.* https://fastapi.tiangolo.com/.

UNIRIO, BSI. 2024. *Publicações de TCCs do Bacharelado em Sistemas de Informação.* https://bsi.uniriotec.br/publicacoes-de-tcc/. Último acesso em: 19 de dezembro de 2024.

Yran Bartolumeu Dias, Pepe Cafferata e. Julho de 2024. "65 por cento das empresas usam Gen AI no mundo; líderes apontam caminhos para obter retorno financeiro" (). https://www.mckinsey.com.br/our-insights/all-insights/65-das-empresas-usam-gen-ai-no-mundo.