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

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

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. Como é o exemplo do 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].

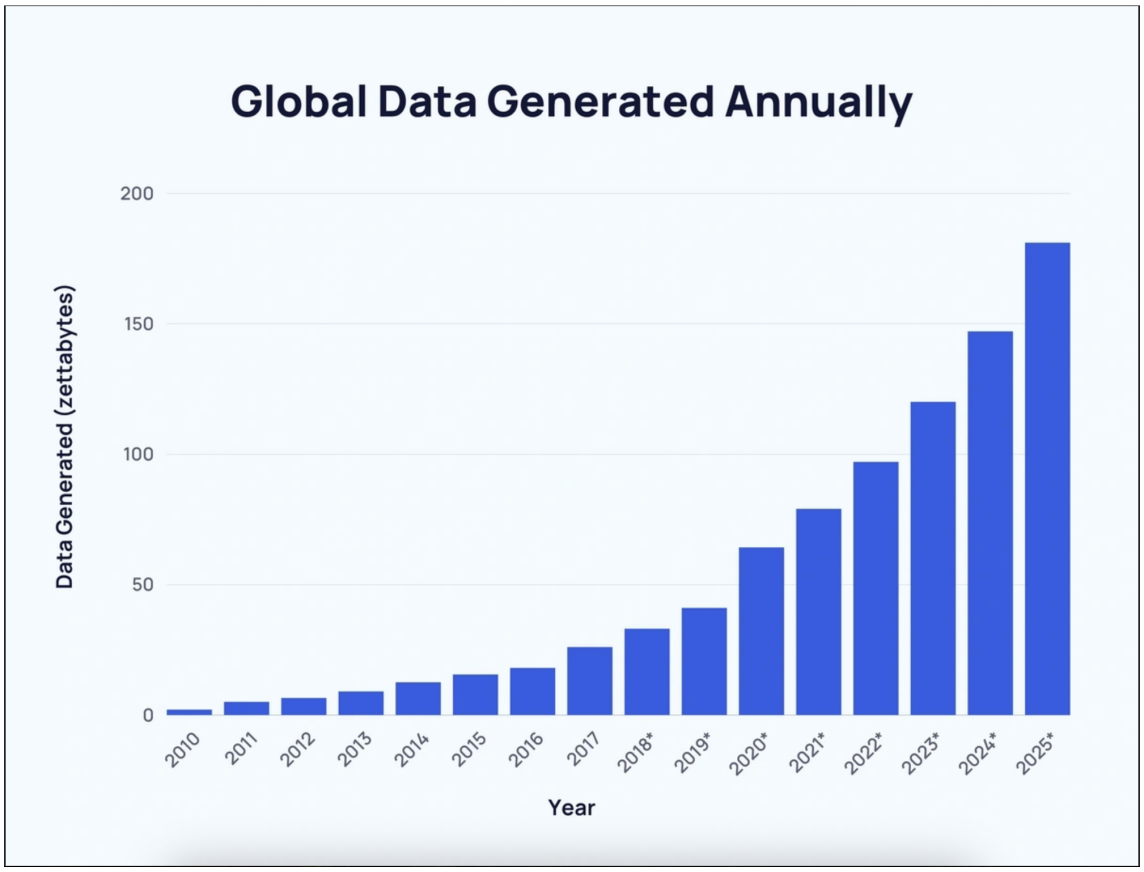

Figura 1: Total de dados produzidos por ano, [Duarte, 2024]

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.

O DSR, possui raízes na engenharia e nas ciências do artificial [Simon, 2019], 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].

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 passarmos 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, às vezes chamada de *gen 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 inteligência artificial generativa baseia-se em modelos avançados de *machine learning* (aprendizado de máquina) chamados modelos de *deep learning* (aprendizagem profunda) — algoritmos que simulam os processos de aprendizado e tomada de decisão do cérebro humano. Esses modelos trabalham identificando e codificando padrões e relações em grandes volumes de dados.

A partir dessas informações, 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 IA generativa já faz parte de nossas vidas há décadas, e a tecnologia de IA generativa atual se baseia em avanços de aprendizado de máquina que remontam ao início do século 20. 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 o GeoForce, a primeira unidade de processamento gráfico (GPU). Originalmente desenvolvida para fornecer gráficos de movimento suave para videogames, as GPUs se tornaram a plataforma padrão para o desenvolvimento de modelos de IA e mineração de criptomoedas.

- **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, 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:** O 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 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 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 interfaces 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 transformadores [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 dos conjuntos de dados.

Durante o treinamento, os modelos aprendem a prever a próxima palavra em uma sentença com base no contexto fornecido pelas palavras anteriores. Isso é feito atribuindo probabilidades à recorrência de palavras que foram tokenizadas (divididas em sequências menores) e transformadas em embeddings, representações numéricas do contexto.

O treinamento envolve o uso de corpora massivas, com bilhões de páginas de texto, permitindo que os LLMs aprendam gramática, semântica e relações conceituais por meio de aprendizado auto-supervisionado e técnicas de zero-shot. Uma vez treinados, os modelos geram texto prevendo autonomamente a próxima palavra com base na entrada recebida e nos padrões adquiridos, 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 (*RLHF*). Essas abordagens ajudam a mitigar problemas como vieses, discurso de ódio e respostas incorretas ou ilusórias (*hallucinations*), que podem surgir devido ao treinamento em dados não estruturados. Garantir que os LLMs estejam prontos para uso em nível corporativo é crucial para evitar riscos à reputação e responsabilidades indesejadas.

2.3 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 [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 independente do tipo de dado abordado na solução, seja texto, imagens, áudios ou documentos diversos.

2.4 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 as 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 2, que será explicada nos próximos parágrafos:

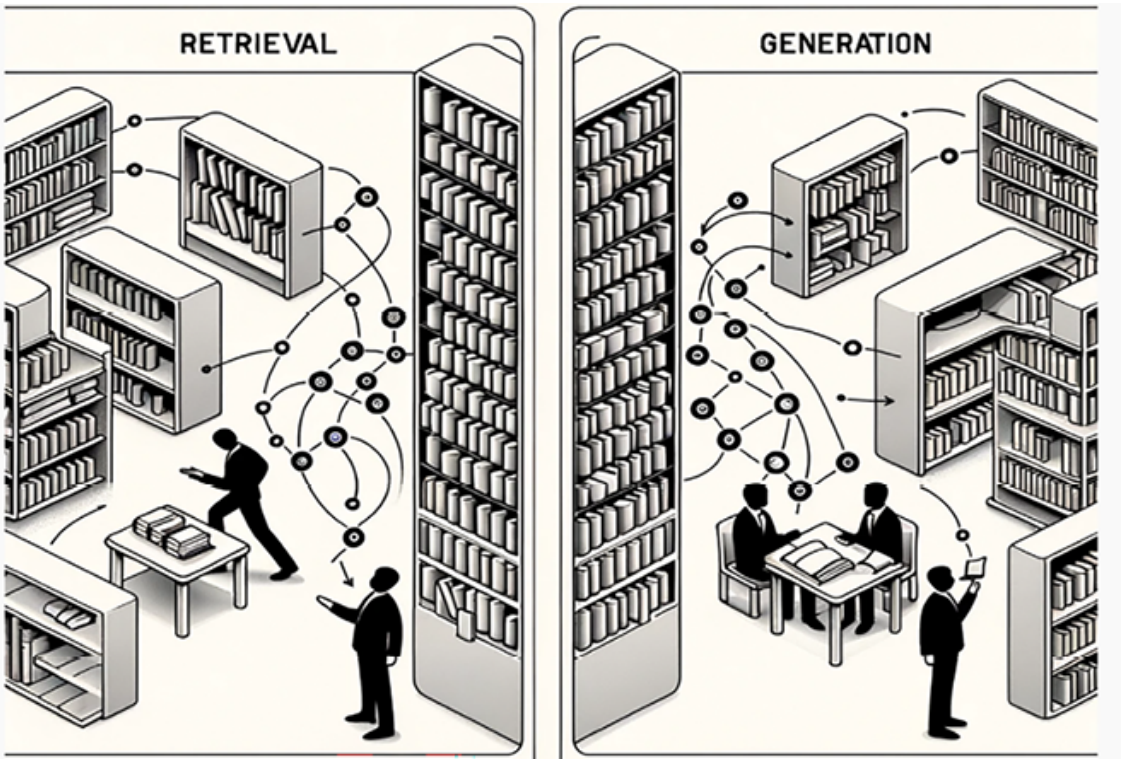

Figura 2: Funcionamento geral de uma estrutura RAG (extraída de [Rothman, 2024])

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 2) 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 2), 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.

2.5 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 3). 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 vieses), 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, a Figura 3 abaixo, ilustrando os principais componentes do *framework* RAG (independentemente do tipo implementado), será explicada.

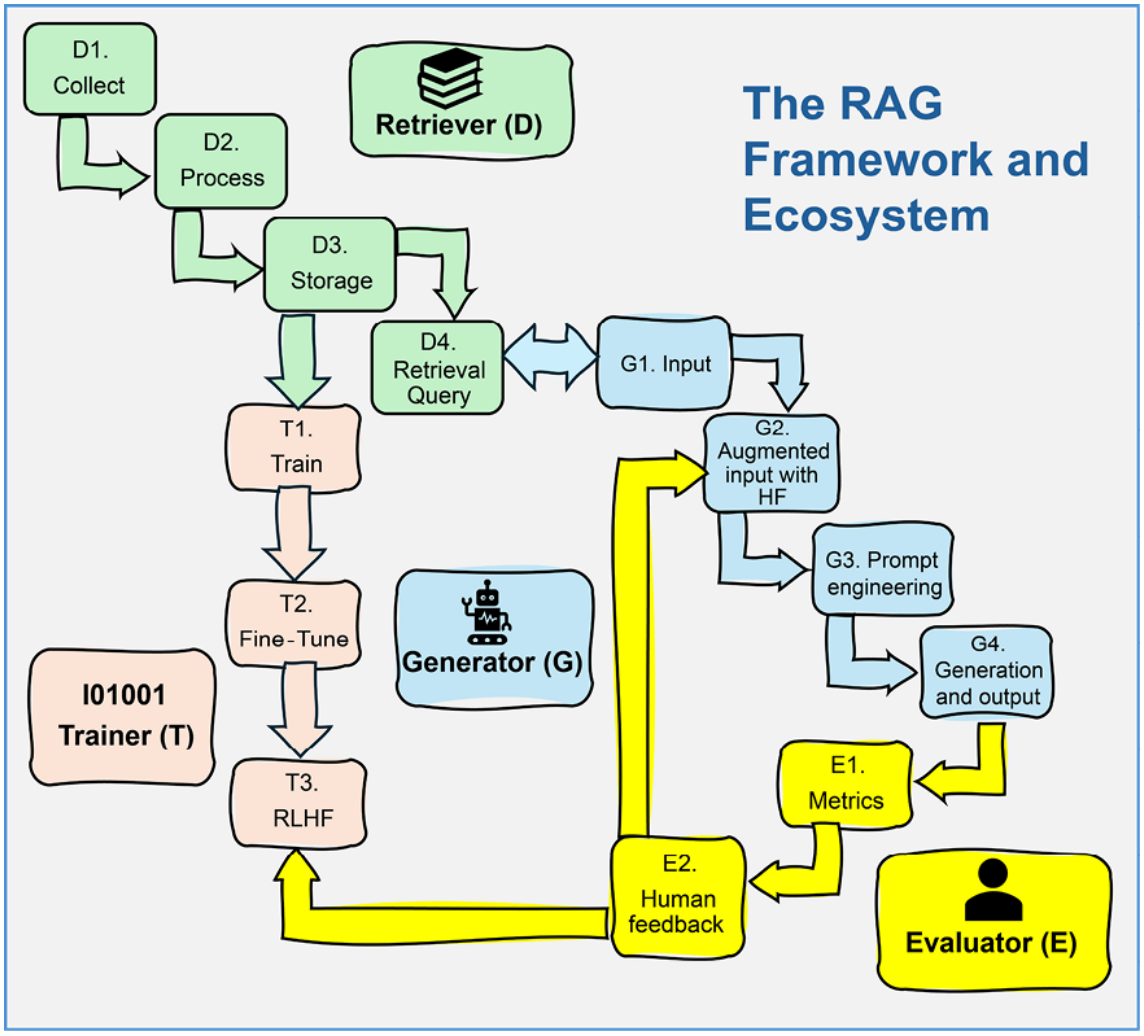

Figura 3: Arquitetura da estrutura RAG (extraída de [Rothman, 2024])

- ***Retriever*** (D, em verde na Figura 3): Responsável pela coleta, processamento, armazenamento e recuperação de dados.
- ***Generator*** (G, em azul na Figura 3): Cuida da complementação da entrada, engenharia de prompts e geração de respostas.

- ***Evaluator*** (E, em amarelo na Figura 3): Avalia o desempenho usando métricas matemáticas, *feedback* humano e outras formas de validação.

- ***Trainer*** (T, em rosa na **??**): Gerencia o modelo pré-treinado inicial e sua posterior ajuste fino (*fine-tuning*).

Esses quatro componentes dependem de seus respectivos ecossistemas, composto por seus subcomponentes, formando o *pipeline* de IA generativa baseada em RAG. Nas seções a seguir, serão usadas as siglas D, G, E e T para representar,respectivamente, *Retriever*, *Generator*, *Evaluator* e *Trainer*.

2.5.1 *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.

1. Disponível em [Pinecone, 2025]
2. Disponível em [OpenAI, 2025]
3. Disponível em [Chroma, 2025]
4. Disponível em [Activeloop, 2025]

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 similaridade cosseno, que 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.

2.5.2 *Generator* (G)

No ecossistema RAG, as linhas entre a entrada e a recuperação não são tão nítidas, como mostrado na Figura 3, que representa o *framework* e ecossistema RAG. A entrada do usuário (G1), seja automatizado ou humano, 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.

2.5.3 *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

5. Disponível em [Meta, 2024a]
6. Disponível em [Google, 2024]
7. Disponível em [OpenAI, 2024]

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.

2.5.4 *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 temos 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

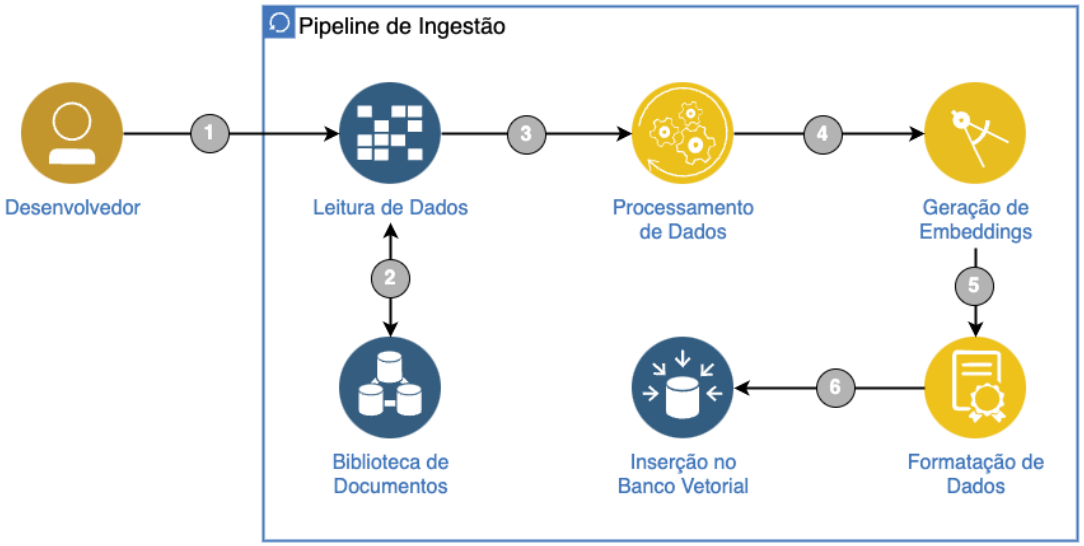

Figura 4: Visão arquitetural do pipeline de ingestão proposto no trabalho.

que cada um seja processado corretamente. Esse processo é modelado de acordo com a Figura 4 abaixo:

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 utilizada, 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 um determinado modelo 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* 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 de baixa dimensão dos dados de um *chunk* expecífico.

5. **Formatação de Dados**: com os *embeddings* da etapa anterior, o objeto de modelo 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 modelo 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 (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 tamanho do documento, 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, onde 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 microserviços 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 ecossistema proposto, sendo responsável por mediar a comunicação entre os diferentes componentes implementados. Sua principal função é receber e processar as requisições feitas pelos usuário na interface gráfica de usuário (GUI), garantindo que as operações sejam realizadas de forma eficiente e segura.

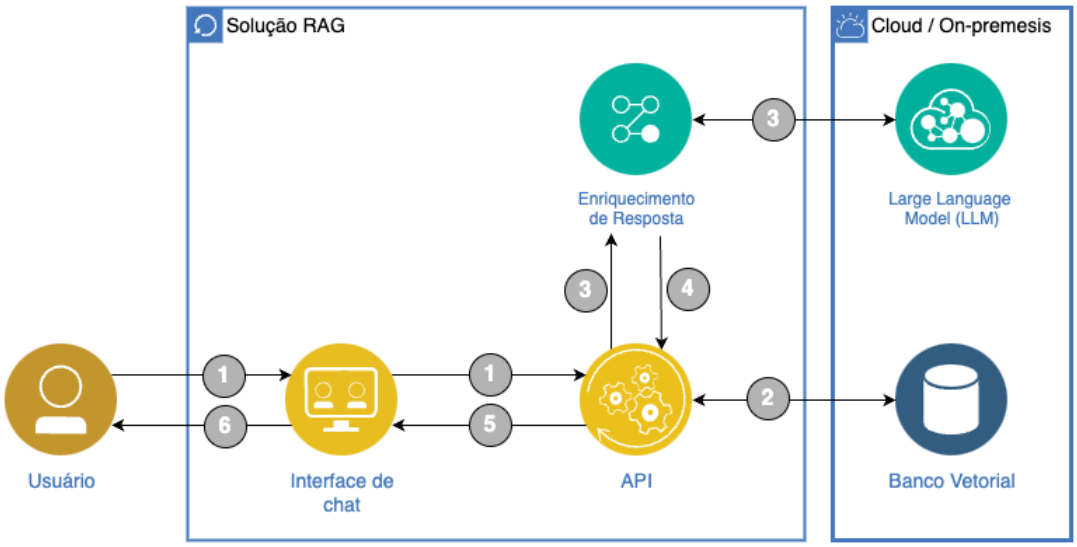

Figura 5: Visão arquitetural do fluxo da interface de usuário proposta no trabalho.

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 (GUI) é um componente essencial, responsável por receber e guiar o usuário em suas interações com o sistema. Ela oferece elementos visuais e controles intuitivos, como botões, menus e janelas, que simplificam a utilização das funcionalidades disponíveis.

Além de sua função de interação direta com o usuário, a GUI atua como um elo entre o usuário e a API do sistema. Ela interpreta as entradas do usuário, como cliques ou comandos, e as converte em requisições apropriadas para o backend, garantindo que as instruções sejam processadas corretamente.

Esse fluxo de comunicação, ilustrado na Figura 5, possibilita que tarefas como busca e enriquecimento de respostas ocorram de maneira transparente e alinhada às necessidades do usuário. Este fluxo entre usuário e sistema, pode ser decomposto nas

seguintes etapas:

1. **Envio da *Query***: usuário realiza uma pergunta para o chat. Em seguida, a interface envia a *query* 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 query do usuário.

4. **Resposta Enriquecida**: a partir da resposta obtida anteriormente, a lista é encaminhada para uma LLM que irá analisar primeiramente um prompt focado em entregar respostas humanizadas para o usuário. Neste 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 a análise de qualquer biblioteca de documentos, ou seja, o processamento dos dados da biblioteca será realizado mesmo que ela tenha arquivos em formatos diferentes. O projeto foi contruido com base nos três artefatos previamente explicados e funciona como ilustrado na Figura 6 a seguir:

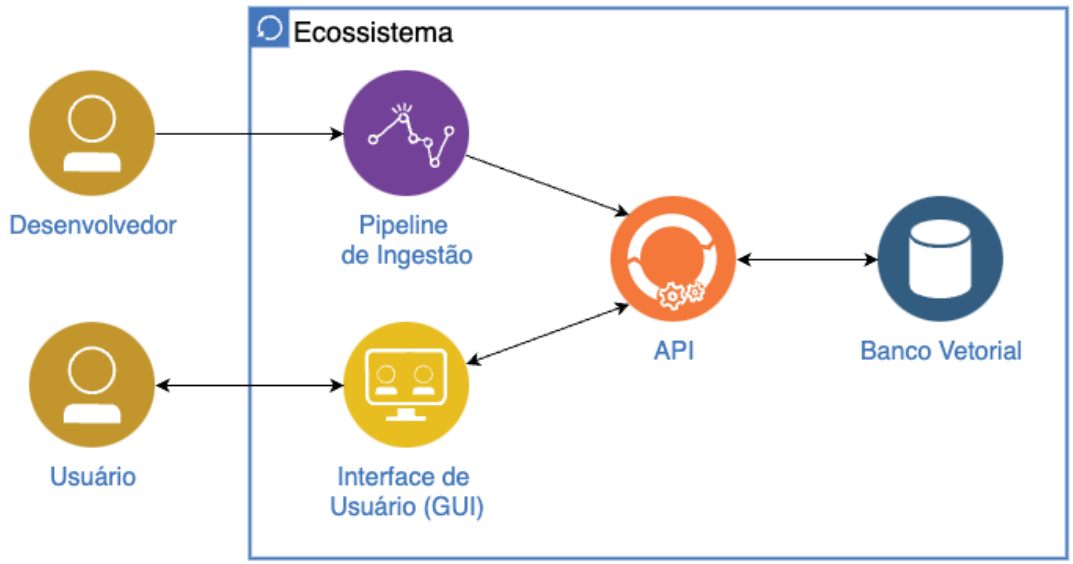

Figura 6: Visão arquitetural do ecossistema RAG proposto no trabalho

3.2 Tecnologias

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 onde 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 eficiente.

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

8. Disponível em [Foundation, 2024]
9. Disponível em [IBM, 2024]

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 flexibilidade 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,

10. Disponível em [Index, 2024]
11. Disponível em [Company, 2024]

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 implementem 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 utilizando 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

12. Disponível em [Meta, 2024b]
13. Disponíveis em [Meta, 2024a]

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 linguagem de grande porte 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 para a camada de rede e o Pydantic para a validação de dados, garantindo alta performance e confiabilidade.

Em resumo, 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.

14. Disponível em [Tiangolo, 2018]

3.2.7 Streamlit

O Streamlit [15] 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 *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.

Em resumo, o Streamlit é uma ferramenta poderosa e acessível para a criação de aplicativos *web* interativos em Python, ideal para cientistas de dados e engenheiros de aprendizado de máquina que desejam compartilhar e visualizar seus projetos de forma rápida e eficaz.

15. Disponível em [Inc., 2024]

4 Solução Desenvolvida

Com os conceitos fundamentais por trás de um ecosistema RAG, discutidos na Seção 2, este capítulo seguirá expondo o processo de desenvolvimento do ecosistema 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, não é viável processar todos os documentos de uma biblioteca devido as limitações de hardware enfrentadas durante a implementação. Para sanar este problema, foi selecionado um conjunto de dados específico que permitisse o processamento de todos os documentos.

Este conjunto de dados é formado por todos os trabalhos de conclusão de curso do ano de 2023 publicados pela secretaria 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 trabalhos de conclusão de curso 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. E para obter uma leitura e processamento de documentos agnóstica a extensões de documentos, segue-se um Design Pattern, mais especificamente o factory design pattern.

O factory design pattern 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. Neste contexto, foi implementada uma fábrica de extratores, onde todo extrator possui uma instância dos clientes provedores de serviços do LlamaIndex (Subseção 3.2.3) e Docling (Subseção 3.2.2), assim como variáveis pré definidas e os métodos de extração.

No entanto, cada extrator implementa o método de extração a sua própria maneira, assim é possível implementar 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 extraído 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 a 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 a 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: Modelo 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, podemos dar continuidade a partição dos documentos em *chunks*. Neste projeto, cada *chunk* foi configurado para ter no máximo 2048 tokens. Cada *chunk* gerado será lido e terá o *embedding* gerado para a inserção no banco vetorial, neste projeto o modelo de *embedding* utilizado

é o ***impira/layoutlm-document-qa*** [16], pois é um modelo voltado para implementações de RAG e já treinado para melhor atender nosso caso de uso.

Prosseguindo com o particionamento dos documentos em *chunks*, ao final da partição obtemos o modelo definido no Algoritmo 2 a seguir:

```
{
    "_id": "chave única do chunk",
    "extension": "extensão do arquivo",
    "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 a 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: Modelo de dados para inserção no banco vetorial

Este processo de leitura e processamento irá se repetir até que o último documento seja devidamente processado. Uma vez processados, os documentos estarão no formato proposto pelo modelo no Algoritmo 2 e estarão 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.

16. Disponível em [Impira, 2024]

A inserção no banco será 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 modelo no Algoritmo 2. Ao desserializar o conteúdo do arquivo *.pickle* é feita uma requisição para API contendo os objetos desserializados e o tamanho dos *batches* configurados para inserção. Neste contexto, um *batch* é o tamanho do grupo de objetos que serão inseridos no banco simultaneamente. A implementação da API será discutida na Subseção 4.3.

4.3 Interface de Programação de Aplicações (API)

Como visto na Subseção 3.0.2, uma API tem como principal função é permitir que um programa acesse recursos e funcionalidades de outro, facilitando a integração e a troca de dados. Neste projeto, a API é utilizada principalmente no intuito de restringir o acesso direto ao banco vetorial. Para isso foi utilizado a biblioteca *fastAPI* (dicutida na Subseção 3.2.6) para implementar uma API capaz de atuar no ecossistema proposto neste trbalho.

Para garantir que a API irá sanar todos os pontos de acesso ao banco vetorial, é necessário listar todos os cenários em que o acesso ao banco é necessário. Esses cenários são:

1. **Ingestão de Dados**: necessário para popular o banco, descrito na Subseção 3.0.1
2. **Busca por contexto**: necessário para enriquecimento da resposta, descrito na Subseção 3.0.3

4.4 Interface Gráfica de Usuário

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna.

Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

5 Conclusão

5.1 Considerações Finais

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

5.2 Limitações

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum

pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

5.3 Trabalhos Futuros

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

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

Company, The Search AI. 2024. *Elasticsearch.* 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.* https://www.seagate.com/files/www-content/our-story/trends/files/dataage-idc-report-final.pdf. Último acesso em: 19 de dezembro de 2024.

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

Foundation, Python Software. 2024. *Python.* 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*. 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/.

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/.

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

Simon, Herbert A. 2019. *The Sciences of the Artificial.* Reissue of the third edition with a new introduction by John Laird. MIT Press.

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.