UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Nicolas Antero Nunes

Automatização da elaboração e análise de resumos de registro de eventos de equipamentos com o uso de grandes modelos de linguagem

Florianópolis
2024

Nicolas Antero Nunes

Automatização da elaboração e análise de resumos de registro de eventos de equipamentos com o uso de grandes modelos de linguagem

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof. Hector Bessa Silveira, Dr.
Supervisor: Lucas Vasconcelos Rocha, Eng.

Florianópolis
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Nunes, Nicolas Antero
Automatização da elaboração e análise de resumos de registro de eventos de equipamentos com o uso de grandes modelos de linguagem / Nicolas Antero Nunes ; orientador, Hector Bessa Silveira, 2024.
88 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2024.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Inteligência Artificial Generativa. 3. Grande Modelo de Linguagem (LLM). 4. Geração Aumentada por Recuperação (RAG). 5. Sumarização de Logs. I. Silveira, Hector Bessa. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

Nicolas Antero Nunes

Automatização da elaboração e análise de resumos de registro de eventos de equipamentos com o uso de grandes modelos de linguagem

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 01 de Julho de 2024.

Prof. Marcelo De Lelis, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Hector Bessa Silveira, Dr.
Orientador
UFSC/CTC/DAS

Documento assinado digitalmente
Hector Bessa Silveira
Data: 29/07/2024 09:51:26-0300
CPF: ***.846.519-**
Verifique as assinaturas em https://v.ufsc.br

Lucas Vasconcelos Rocha, Eng.
Supervisor
BIX Tecnologia

Documento assinado digitalmente
LUCAS VASCONCELOS ROCHA
Data: 26/07/2024 16:44:44-0300
CPF: ***.969.407-**
Verifique as assinaturas em https://v.ufsc.br

Thiago Raulino Dal Pont, Me.
Avaliador
UFSC

Prof. Eduardo Camponogara, Dr.
Presidente da Banca
UFSC/CTC/DAS

AGRADECIMENTOS

Primeiramente, gostaria de expressar minha profunda gratidão à minha família pelo apoio incondicional durante toda essa jornada. Agradeço ao meu pai, Carlos Augusto Rosado Nunes, à minha mãe, Rosane Antero, à parceira do meu pai, Karina, e ao parceiro da minha mãe, Ernesto. Suas palavras de incentivo e amor foram fundamentais para que eu chegasse até aqui.

Um agradecimento especial à minha namorada, Pollyana, por estar sempre ao meu lado, me apoiando e incentivando a realizar coisas que eu nem mesmo imaginava ser capaz. Seu apoio foi essencial em todos os momentos.

Expresso minha sincera gratidão ao meu orientador acadêmico, Hector Bessa Silveira, pelo apoio e pelas palavras de incentivo durante o último ano. Sua compreensão de que, apesar de nossos limites humanos, somos capazes de realizar qualquer coisa, foi uma grande inspiração.

Sou grato ao meu supervisor, Lucas Vasconcelos, que me acompanhou nos últimos dois anos na BIX Tecnologia. Aprendi muito com ele, observando e ouvindo suas ideias durante os diversos projetos que desenvolvemos juntos. Também agradeço ao Hernane Braga, pelo apoio, pelas conversas e por acreditar no meu potencial, proporcionando-me oportunidades valiosas.

Gostaria de expressar meu agradecimento à BIX Tecnologia, uma empresa que valoriza o bem-estar das pessoas e me proporcionou a oportunidade de trabalhar em projetos incríveis.

Aos meus amigos da turma de Automação 17.2, que se tornaram uma segunda família ao longo do tempo, meu muito obrigado. Vocês foram fundamentais durante toda a minha trajetória acadêmica. Aos meus amigos de Criciúma, do grupo odm, agradeço pela parceria ao longo da minha vida, pelas histórias e aventuras que compartilhamos e por serem um ponto de fuga quando necessário.

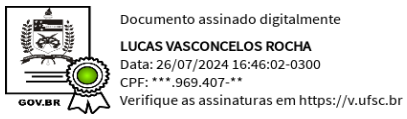
Por fim, mas não menos importante, agradeço a todos os professores da Universidade Federal de Santa Catarina, que nunca mediram esforços para compartilhar seus conhecimentos e garantir que saíssemos capacitados e preparados para o futuro.

DECLARAÇÃO DE PUBLICIDADE

Florianópolis, 01 de Julho de 2024.

Na condição de representante da BIX Tecnologia na qual o presente trabalho foi realizado, declaro não haver ressalvas quanto ao aspecto de sigilo ou propriedade intelectual sobre as informações contidas neste documento, que impeçam a sua publicação por parte da Universidade Federal de Santa Catarina (UFSC) para acesso pelo público em geral, incluindo a sua disponibilização *online* no Repositório Institucional da Biblioteca Universitária da UFSC. Além disso, declaro ciência de que o autor, na condição de estudante da UFSC, é obrigado a depositar este documento, por se tratar de um Trabalho de Conclusão de Curso, no referido Repositório Institucional, em atendimento à Resolução Normativa n° 126/2019/CUn.

Por estar de acordo com esses termos, subscrevo-me abaixo.



Lucas Vasconcelos Rocha
BIX Tecnologia

RESUMO

No cenário empresarial, a automação e a redução da dependência de trabalho manual são cruciais para a competitividade, visando eficiência e redução de custos. A adoção de soluções automatizadas tornou-se essencial para superar desafios no suporte técnico, como erros humanos e alto custo operacional, dando apoio a tomada de decisões mais precisas e ágeis relacionadas a falhas em equipamentos eletroeletrônicos. Este trabalho desenvolve uma prova de conceito focada na automatização da análise de *logs* de equipamentos. Os *logs* são registros cruciais que documentam as operações dos dispositivos, sendo fundamentais para identificar comportamentos anormais e falhas técnicas. A solução proposta utiliza grandes modelos de linguagem (LLMs) para a sumarização automática de *logs*, um processo que sumariza um número relativamente grande de *logs* em informações essenciais. Essa condensação auxilia no diagnóstico, apoiando decisões ágeis no suporte técnico. Utilizando a técnica de Geração Aumentada de Recuperação (RAG), os LLMs são capazes de acessar documentos técnicos de dispostivos eletroeletrônicos como contexto para formular respostas detalhadas sem a necessidade de retreinamento dos modelos. O projeto foi realizado com a implementação de contêineres que facilitaram a hospedagem e o acesso aos grandes modelos de linguagem de código aberto. Esta infraestrutura permitiu realizar tarefas de inteligência artificial generativa de maneira segura e eficiente, melhorando consideravelmente o suporte técnico ao reduzir o tempo necessário para diagnósticos e aumentar a precisão das respostas fornecidas aos técnicos. Para avaliar a eficácia do mecanismo de recuperação de informações, utilizou-se uma LLM para gerar um conjunto de dados de questão-resposta, minimizando a necessidade de intervenção manual por parte de analistas. Isso foi possível graças à criação de dados derivados da interpretação dos referidos documentos. A sumarização dos arquivos de *logs* foi realizada por uma cadeia de LLMs, que lidou com dados estruturados e consultas complexas, transformando a consulta inicial por meio de um fluxo sistemático até a geração da resposta final. A ferramenta desenvolvida permite interações através de uma interface que suporta tanto consultas usando a técnica RAG com base em documentos técnicos quanto a sumarização de *logs*, incluindo a capacidade de realizar o *upload* de arquivos para processamento. As métricas de avaliação utilizadas testaram diferentes configurações de *chunks* e modelos de *embeddings*, revelando que variações no tamanho dos *chunks* influenciam diretamente o desempenho dos modelos. O modelo *mixedbread-ai/mxbai-embed-large-v1* com um *chunk size* de 512 obteve os melhores resultados em termos de precisão e eficiência. Este projeto aponta que a automação com LLMs pode revolucionar o processo de suporte técnico, proporcionando respostas rápidas e precisas, reduzindo custos operacionais e melhorando a satisfação do cliente. Ele valida a aplicabilidade prática da inteligência artificial em sistemas de suporte técnico, oferecendo uma contribuição significativa ao campo da manutenção de equipamentos eletroeletrônicos e promovendo avanços importantes para a empresa cliente, para os times de suporte técnico, e para os usuários finais, minimizando o tempo de inatividade dos equipamentos e acelerando a resolução de problemas.

Palavras-chave: Inteligência Artificial Generativa, Grande Modelo de Linguagem (LLM), Geração Aumentada por Recuperação (RAG), Sumarização de Logs

ABSTRACT

In the business environment, automation and the reduction of manual labor dependency are crucial for competitiveness, aiming for efficiency and cost reduction. The adoption of automated solutions has become essential to overcome challenges in technical support, such as human errors and high operational costs, supporting more accurate and swift decision-making related to failures in electronic equipment. This work develops a proof of concept focused on automating the analysis of equipment logs. Logs are crucial records that document the operations of devices, essential for identifying abnormal behaviors and technical failures. The proposed solution utilizes large language models (LLMs) for the automatic summarization of logs, a process that condenses a relatively large number of logs into essential information. This condensation aids in diagnostics, supporting swift decisions in technical support. Using the Retrieval-Augmented Generation (RAG) technique, the LLMs can access technical documents of electronic devices as context to formulate detailed responses without the need for retraining the models. The project was carried out with the implementation of containers that facilitated the hosting and access to open-source large language models. This infrastructure allowed the performance of generative artificial intelligence tasks in a secure and efficient manner, considerably improving technical support by reducing the time needed for diagnostics and increasing the accuracy of the responses provided to technicians. To evaluate the effectiveness of the information retrieval mechanism, an LLM was used to generate a question-answer dataset, minimizing the need for manual intervention by analysts. This was made possible by creating data derived from the interpretation of the mentioned documents. The summarization of the log files was carried out by a chain of LLMs, which handled structured data and complex queries, transforming the initial query through a systematic flow until the final response generation. The developed tool allows interactions through an interface that supports both queries using the RAG technique based on technical documents and the summarization of logs, including the ability to upload files for processing. The evaluation metrics used tested different chunk configurations and embedding models, revealing that variations in chunk size directly affect the performance of the models. The model mixedbread-ai/mxbai-embed-large-v1 with a chunk size of 512 achieved the best results in terms of accuracy and efficiency. This project indicates that automation with LLMs can revolutionize the technical support process, providing quick and accurate responses, reducing operational costs, and improving customer satisfaction. It validates the practical applicability of artificial intelligence in technical support systems, offering a significant contribution to the field of electronic equipment maintenance and promoting significant advances for the client company, technical support teams, and end-users, minimizing equipment downtime and accelerating problem resolution.

Keywords: Generative Artificial Intelligence, Large Language Model (LLM), Retrieval Augmented Generation (RAG), Log Summarization

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

No fim de 2022, o mundo presenciou um marco relevante na história da tecnologia, o lançamento de uma plataforma chamada de ChatGPT que disponibilizou em larga escala o uso de inteligência artificial generativa para geração de textos com base em padrões pré-treinados, os grandes modelos de linguagem. As possibilidades dessa plataforma não chamaram atenção apenas da população para seu uso pessoal, mas também captaram o interesse de empresas em como elas poderiam utilizar essas tecnologias para se tornarem mais competitivas. Isso iniciou uma corrida entre empresas para desenvolverem aplicações de modo a não ficarem para trás de seus concorrentes.

Do ponto de vista empresarial, aumentar a automação e reduzir a dependência do trabalho manual são cruciais para enfrentar os desafios atuais na indústria e manter a competitividade. Além disso, minimizar erros humanos é essencial para aprimorar a eficiência e a segurança operacional. Diante dessa crescente demanda, especialmente considerando as dificuldades em treinar colaboradores para utilizar múltiplas ferramentas e realizar diversas atividades, juntamente com os altos custos operacionais associados, a empresa cliente relatou desafios significativos enfrentados pelos seus times de suporte ao cliente no diagnóstico de problemas em equipamentos eletroeletrônicos que fabricam.

Atualmente nas indústrias, o processo de diagnóstico de problemas pelos times de suporte ao cliente, é alimentado com informações enviadas pelo usuário final dos equipamentos. Nessas informações podem estar presentes o relato do problema, assim como os *logs* gerados pelo equipamento ao longo de sua operação. Os *logs* são responsáveis por registrar e armazenar eventos ocorridos durante o funcionamento dos equipamentos, sendo essa a principal fonte de informação utilizada para analisar o histórico de funcionamento e identificar comportamentos anômalos, já que esses eventos podem registrar tanto o bom funcionamento quanto o mau.

Os *Logs* entregam informações relevantes para a manutenção de um serviço, sistema ou equipamento, indicando muitas vezes eventos de erro, avisos de acontecimentos inesperados ou apenas, por segurança, acontecimentos previstos para que seja possível uma conferência futura (ELASTIC, 2024).

Não existem apenas registros de eventos para falhas, mas também para informações de acesso, de servidor, de alterações e de acontecimentos físicos que são monitorados. É importante salientar que tais registros são automatizados e salvos para análises posteriores (AMAZON WEB SERVICES, 2024).

No entanto, analisar esses *logs* não é uma atividade trivial para os times de suporte. É necessário um conhecimento profundo do equipamento e muito treinamento para saber o que procurar nessas informações, pois pode haver um número relativa-

mente grande de eventos registrados. A complexidade se torna ainda maior quando extrapolamos o problema de um equipamento para um catálogo de diversos produtos fabricados pela empresa, de forma que cada equipamento pode ter diferentes modelos, cada um possuindo suas particularidades.

A empresa cliente, uma multinacional que fabrica milhares de equipamentos para o mundo inteiro, enfrenta dificuldade para dar suporte a todos seus clientes. Os equipamentos fornecidos geram grande quantidade de dados, o que sobrecarrega os times de suporte, que levam muito tempo para analisar os *logs* e chegar a um diagnóstico para o problema. Assim, a empresa cliente visualizou nessa demanda um potencial de automatização do processo de criação de relatórios contendo o diagnóstico de problema dos equipamentos com o uso de grandes modelos de linguagem.

A sumarização de *logs* envolve a condensação de dados complexos dos registros de operações dos equipamentos em informações concisas e úteis. Estes *logs* são cruciais, pois capturam o histórico detalhado da funcionalidade dos dispositivos, incluindo registros de falhas e operações normais. Sua análise detalhada é essencial, mas desafiadora devido à sua complexidade e ao volume de dados, tornando o processo de diagnóstico lento e propenso a erros. A sumarização automatizada desses *logs* visa transformar esse cenário ao agilizar a identificação de problemas, permitindo respostas mais rápidas e precisas.

A capacidade de fornecer uma resposta automatizada “semelhante ao humano” sobre o funcionamento dos dispositivos tem potencial de simplificar significativamente o processo relatado, eliminando a necessidade de treinamento extra dos times de suporte para compreender os detalhes específicos de cada dispositivo. Essa abordagem mais automatizada não só pode reduzir a necessidade de treinamento interno para manutenção de equipamentos, mas também proporcionar muitos benefícios para os clientes e equipes de suporte.

Utilizando técnicas de Inteligência Artificial, especialmente os LLMs, a solução envolve a análise automática de registros de eventos (*logs*) e documentos técnicos para sugerir possíveis causas de problemas nos equipamentos, melhorando assim a eficiência operacional e reduzindo custos. A técnica de Geração Aumentada de Recuperação (RAG) foi aplicada para capacitar os LLMs a acessar e integrar informações de documentos técnicos relevantes durante o processo de diagnóstico.

Para implementar essa solução, foram empregados métodos e ferramentas como o LlamaIndex e o ChromaDB para gestão e recuperação de informações, além do uso do Docker para a implementação local dos modelos em contêineres, garantindo segurança e privacidade dos dados. A solução proposta foi desenvolvida e testada em um ambiente controlado, resultando em uma Prova de Conceito (PoC) que demonstrou a viabilidade da aplicação em reduzir o tempo de diagnóstico e o esforço manual necessário.

Os resultados preliminares sugerem um potencial significativo para melhorar a qualidade e a velocidade na identificação dos problemas dos equipamentos, o que poderia beneficiar diretamente os times de suporte da empresa cliente. Essa melhoria na eficiência pode levar a uma maior satisfação do cliente e a uma redução de custos operacionais. A implementação completa dessa tecnologia tem o potencial de otimizar os processos existentes e oferecer uma robusta ferramenta de apoio à decisão para o time técnico, transformando a abordagem de manutenção e suporte técnico.

1.1 A EMPRESA

A BIX Tecnologia, com sede em Florianópolis, é uma consultoria de dados fundada em 2014 pelo Engenheiro de Controle e Automação, Felipe Santos Eberhardt. O foco inicial da empresa foi oferecer serviços de inteligência de negócio, porém ao longo dos anos foi expandindo o portfólio de áreas de atuação, contando hoje com consultores e especialistas em quatro áreas da tecnologia, sendo elas engenharia de dados, desenvolvimento de sistemas, ciência de dados e, como já mencionado, inteligência de negócios. Essa diversidade de especialidades permite que a BIX Tecnologia possa desenvolver soluções completas de ponta a ponta, ou seja, desde a aquisição de dados, passando pela estruturação de fluxos de processamento, extração de valor dos dados, apresentação de forma visual, até a construção de plataformas para o cliente.

O presente trabalho foi desenvolvido no time de Ciência de Dados da BIX Tecnologia, onde o autor teve a oportunidade de participar ativamente no desenvolvimento de soluções, gerenciamento de projetos e relacionamento com clientes. Durante sua experiência, o autor teve contato com diversas tecnologias envolvidas na Ciência de Dados, como inteligência artificial, aprendizado de máquina, análise de dados e estatística. Ele pôde compreender como essas tecnologias são aplicadas em projetos reais. Além disso, o autor também teve a chance de explorar conceitos como análise preditiva e prescritiva, compreendendo como essas abordagens podem ser utilizadas para antecipar eventos futuros e fornecer recomendações acionáveis para a empresa. Essa experiência proporcionou ao autor uma visão abrangente do ciclo de um projeto de Ciência de Dados, desde a concepção até a implementação de soluções práticas em diferentes áreas de aplicação empresarial.

1.2 OBJETIVOS

O objetivo principal deste projeto é o desenvolvimento de uma prova de conceito (POC - *Proof of Concept*) que consiste em uma demonstração para validar a viabilidade da solução que a empresa pretende implementar para o problema de análise automática de *logs* de equipamentos, visando a criação de relatórios contendo o diag-

nóstico de problemas de dispositivos eletroeletrônicos de forma automatizada com o uso de inteligência artificial generativa, em específico grandes modelos de linguagem. A proposta inclui a utilização de modelos de inteligência artificial de código aberto, para desenvolver um produto mínimo viável (MVP - *Minimum Viable Product*) para análise de registros de pelo menos um dispositivo previamente selecionado pelo cliente.

Além disso, o projeto também visa explorar a aplicação desses modelos para responder a consultas sobre documentos técnicos relacionados aos equipamentos, como manuais técnicos. Assim, foram estabelecidos os seguintes objetivos específicos:

- Utilização de modelos baseados em linguagem para analisar e sumarizar o histórico de um determinado dispositivo com base nos registros armazenados em forma de *logs*;
- Permitir que a solução possa auxiliar os times de suporte na identificação de problemas relacionados aos equipamentos dos clientes;
- Utilizar manuais técnicos dos equipamentos como fonte de conhecimento, ou contexto, para que os modelos tenham maior qualidade na resposta;
- Possibilitar a criação de relatórios de forma automatizada, contendo o problema identificado pelo modelo e a fonte de conhecimento utilizada para a geração do conteúdo.

Ressalta-se que este projeto visa não apenas automatizar processos de análise e geração de relatórios, mas também melhorar a eficiência operacional, reduzir custos e oferecer suporte técnico mais eficaz para os clientes.

1.3 ORGANIZAÇÃO DO DOCUMENTO

O Capítulo 2 apresenta a fundamentação teórica sobre processamento de linguagem natural, *transformers*, *embeddings*, grandes modelos de linguagem, *retrieval augmented generation*, *prompts* e métricas de avaliação. O Capítulo 3 descreve o problema tratado e os requisitos envolvidos no projeto, assim como a solução proposta. O Capítulo 4 detalha o processo de desenvolvimento e as etapas de implementação da solução. O Capítulo 5 apresenta os resultados dos testes para avaliação das recuperações de documentos. Por fim, juntamente com uma análise dos resultados obtidos o Capítulo 6 conclui o trabalho e fornece algumas sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica para o entendimento dos principais conceitos e métodos utilizados no decorrer do trabalho.

A Seção 2.1 traz informações sobre a metodologia CRISP-DM, utilizada no desenvolvimento de projetos de ciência de dados. Uma apresentação sobre processamento de linguagem natural é dada na Seção 2.2. Já a Seção 2.3 descreve o que são *Transformers* e como eles podem ser utilizados em modelos de processamento de linguagem natural. Na sequência, a Seção 2.4 aborda uma das técnicas de representação de palavras através de vetores. Os grandes modelos de linguagem (LLM - *Large Language Models*) e suas aplicações são apresentados na Seção 2.5. A Seção 2.6 descreve algumas das principais técnicas de recuperação de informações visando um melhor desempenho de modelos de inteligência artificial. A Seção 2.7 introduz a importância da engenharia de *prompt* para a interação com Inteligências Artificiais Generativas, sendo fundamental para melhorar a qualidade e eficácia das respostas geradas pelos modelos. Por fim, a Seção 2.8 apresenta as métricas de avaliação da recuperação de documentos que foram implementadas no desenvolvimento deste trabalho.

2.1 CRISP-DM

CRISP-DM é uma sigla em inglês que significa Cross Industry Standard Process for Data Mining, ou Processo Padrão Interindustrial para Mineração de Dados. Este padrão foi criado em 1999 e desde então se tornou a metodologia mais utilizada no desenvolvimento de projetos que aplicam mineração ou ciência de dados (WIRTH; HIPP, 2000; HOTZ, 2024).

Esta metodologia é dividida em 6 fases, sendo elas entendimento do negócio, entendimento dos dados, preparação dos dados, modelagem, avaliação e implantação. Primeiramente, o entendimento do negócio avalia os recursos disponíveis e necessários em um empreendimento, determinando o objetivo da mineração de dados e os critérios para mensurar o alcance das metas estabelecidas (SCHRÖER; KRUSE; GÓMEZ, 2021).

Em seguida, o entendimento dos dados representa a coleta e análise de dados, focando na qualidade dos mesmos e seus atributos. Assim, a preparação dos dados seleciona as informações obtidas com base em critérios de exclusão, removendo dados de baixa qualidade.

A fase de modelagem é crucial para a construção do modelo de mineração de dados. Um modelo, neste contexto, é uma representação matemática ou computacional que utiliza os dados coletados para realizar previsões, classificações ou identificar padrões. Para levantar esse modelo, são escolhidas técnicas específicas, como algo-

ritmos de aprendizado de máquina (por exemplo, regressão linear, árvores de decisão, redes neurais, etc.), que melhor se adequam ao problema em questão. Durante esta fase, diversos parâmetros do modelo são ajustados para otimizar seu desempenho. Além disso, o modelo é treinado com um conjunto de dados (conjunto de treinamento) e sua performance é testada com outro conjunto de dados (conjunto de teste) para garantir sua capacidade de generalização (GÉRON, 2019).

Na sequência, a fase de avaliação verifica a eficácia do modelo construído. Nesta fase, métricas como acurácia, erro quadrático médio (MSE), precisão, *recall* e *F1-score* são frequentemente usadas para avaliar a performance do modelo e assegurar que o modelo atenda aos objetivos definidos na fase de entendimento do negócio. A avaliação pode revelar a necessidade de ajustes ou até mesmo a revisão de fases anteriores.

Por fim, é na fase de implantação que ocorre o planejamento detalhado de como o modelo será de fato implementado no ambiente de produção. Isso inclui a integração do modelo aos sistemas existentes, a definição de um plano de monitoramento para acompanhar seu desempenho ao longo do tempo, e a manutenção contínua para assegurar que ele continue a proporcionar valor ao negócio (PROVOST; FAWCETT, 2013).

Durante todo o processo CRISP-DM, a documentação e relatórios são essenciais. Esses relatórios detalham os procedimentos realizados, os resultados obtidos e as decisões tomadas em cada fase. Enquanto a metodologia CRISP-DM pode ser interpretada como rígida devido à necessidade constante de redação de relatórios e separação de fases, ela também é considerada uma metodologia ágil, pois, apesar de separada em fases, permite que haja comunicação entre elas e revisão das mesmas. Além disso, o resultado de uma etapa define o que será feito ou planejado na fase seguinte.

2.2 PROCESSAMENTO DE LINGUAGEM NATUAL (PLN)

O Processamento de Linguagem Natural (PLN) é um método utilizado por sistemas e computadores para auxiliar na interpretação de textos em linguagem humana. É considerado um tipo de inteligência artificial criado, principalmente, através de aprendizado de máquina, com o objetivo de entender a interação entre computadores e idiomas humanos. O objetivo geral do PLN é permitir que os computadores realizem diversas tarefas relacionadas à linguagem natural, abrangendo compreensão, geração, tradução, resumo e extração de informações de dados textuais ou falados.

Para que o computador seja capaz de retornar informações relevantes ao usuário, o PLN aplicado precisa tratar o texto de forma a facilitar seu processamento. Nesse sentido, estratégias como conversão para letras minúsculas, troca de palavras derivadas por palavras originárias e primárias, e remoção de conjunções podem ser

aplicadas (CLOUDFLARE, 2024).

O PLN pode ser dividido em duas abordagens principais: simbólica e estatística. A PLN simbólica depende de regras e representações criadas manualmente por um especialista, com base no conhecimento linguístico da linguagem natural, porém possui limitações em termos de escalabilidade, robustez e adaptabilidade a novos domínios e linguagens. A PLN estatística depende de métodos orientados a dados e baseados em aprendizado de máquina e modelos probabilísticos. Ela surgiu no final da década de 1980 e no início da década de 1990 com a disponibilidade de grandes corpora e recursos computacionais. Além disso, pode lidar com dados ruidosos e incompletos, aprender com os dados e generalizar para novas situações. No entanto, a PLN estatística também tem desvantagens, como a necessidade de grandes quantidades de dados anotados, a falta de interpretabilidade e explicabilidade e a dificuldade de incorporar o conhecimento prévio e o senso comum (MANNING; SCHUTZE, 1999).

Atualmente, o PLN está presente em diversas ferramentas utilizadas com grande frequência por qualquer pessoa que utilize um computador. Entre elas estão ferramentas de buscas, tradução de textos e filtros automáticos de e-mails. As técnicas de processamento de linguagem natural geralmente são vistas como etapas de pré-processamento e não fazem parte diretamente de um modelo em si. Na recuperação de informações, o PLN desempenha um papel fundamental na identificação de documentos relevantes (CROFT; METZLER; STROHMAN, 2010).

Apesar de importante, atualmente o entendimento de textos em linguagem natural por parte das máquinas ainda não é trivial. A alta ambiguidade na comunicação humana dificulta uma programação que faça um processamento do texto de forma integralmente eficaz (VIEIRA; LOPES, 2010).

2.3 TRANSFORMERS

Em 2017, Vaswani *et al.* (2017), pesquisador da Google, publicou um artigo sobre um modelo de *deep learning*, mais precisamente, uma arquitetura de rede neural, chamada *Transformers*. Tal arquitetura superou modelos já existentes em tarefas de tradução de texto, seja pela alta qualidade de seus resultados ou pela menor necessidade de recursos em seu treinamento (TUNSTALL; VON WERRA; WOLF, 2022).

A tecnologia *Transformers* foi amplamente adotada em campos como PLN, visão computacional e processamento de fala. Sua eficiência fez com que fossem criados modelos pré treinados capazes de atingir resultados conhecidos na literatura especializada. Por isso, seu uso se tornou padrão em diversos casos, o que incentivou a criação de novos modelos que partem dos *Transformers* originais, mas apresentam resultados ainda melhores (LIN *et al.*, 2022).

Enquanto os *transformers* originais eram ineficientes no processamento de frases longas, novos modelos trazem abordagens como mecanismos de atenção aprimo-

rados e hierarquia de dados para um processamento mais leve. Os novos modelos também introduzem a ideia de viés (bias) para permitir treinamentos com dados em menor escala.

2.4 EMBEDDINGS

O termo em inglês *Embeddings* é comumente traduzido como “incorporações”. Este termo é utilizado para nomear a técnica de representação de palavras utilizando seus significados semânticos e sintáticos através de vetores no espaço n-dimensional. É uma ferramenta amplamente utilizada no contexto do processamento de linguagem natural, permitindo um melhor processamento e interpretação do texto por parte da máquina, que aplica tal conhecimento para responder perguntas, traduzir textos ou buscar informações (WANG, B. *et al.*, 2019).

Alguns modelos de *Embeddings* apresentam uma técnica mais avançada quando se trata de entender o significado de uma palavra e traduzir para um vetor. A abordagem contextual, por exemplo, consegue analisar o contexto em que a palavra está inserida e criar conexões relevantes para todo o texto analisado. Dessa forma, uma palavra que apresente um sentido diferente do ordinário em um texto facilmente interpretado por humanos, pode ser traduzido corretamente para a máquina (LIU; KUSNER; BLUNSOM, 2020).

Quando frases ou palavras são convertidas em vetores, tais vetores podem ser utilizados para calcular similaridades entre os mesmos ou classificá-los em categorias. As palavras passam por regras que determinam se as mesmas possuem relação em um dado contexto e a possibilidade de elas estarem presentes em um determinado texto. (LASSILA; LEPPÄNEN, 2024; MISHRA; VIRADIYA, 2019) . O processo de *embedding* é iniciado com a criação de *tokens*, termo em inglês para a representação de chaves. Uma frase é dividida em palavras, que então passam pelo processo de redução de conjugações ou variantes, resultando na palavra primária mais simples.

Em sequência, os *tokens* são convertidos em vetores. Uma das técnicas mais básicas para a criação dos vetores é a combinação de duas métricas estatísticas: a frequência do termo (TF) e a frequência inversa de documento (IDF). A frequência do termo é calculada dividindo o número de vezes que um termo aparece em um documento pelo número total de termos no documento. Já a frequência inversa de documento diz respeito à relevância de uma palavra no texto, sendo calculada a partir do logaritmo do quociente entre o número total de documentos em um acervo e o número de documentos que contém aquele termo específico. A multiplicação da frequência do termo pela frequência inversa de documento é capaz de retornar a relevância de uma palavra para o entendimento do texto. Palavras comuns recebem baixa relevância, enquanto palavras raras recebem grande relevância. Esta abordagem ainda não é capaz de entregar uma análise semântica das palavras, ou seja, ignora o contexto em

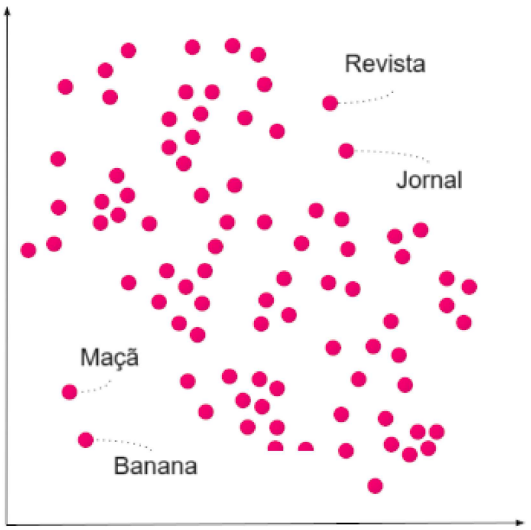
que elas estão inseridas (DEEPLEARNING.AI, s.d.).

Por outro lado, *embeddings* são vetores densos que capturam a semântica das palavras com base em seu contexto. Modelos de *embeddings*, como Word2Vec ou BERT, são treinados em grandes corpora de texto para aprender representações contínuas que refletem as relações semânticas entre palavras. Diferente do TF-IDF, que é baseado em estatísticas de frequência, *embeddings* conseguem capturar nuances de significado e contexto, permitindo uma análise mais profunda do texto.

A evolução de modelos de *embeddings* foi um processo que levou anos. O modelo *Word2Vec* (MIKOLOV *et al.*, 2013) foi um dos primeiros e mais famosos modelos que utiliza o contexto de uma palavra, podendo retornar palavras que tenham similaridade contextual. A produção de vetores mais ricos em informação foi apresentada por Vaswani *et al.* (2017) com o uso de *transformers*. A partir desse trabalho, a Google introduziu modelos pré-treinados de alta eficiência, como o Sentence-BERT (REIMERS; GUREVYCH, 2019). Este modelo ganhou notoriedade pela sua eficácia e capacidade de calcular *embeddings* para frases completas, considerando o contexto das palavras.

Mais precisamente, vetores podem ser entendidos como coordenadas de um espaço multidimensional. Para um espaço de 2 dimensões, por exemplo, palavras que apresentam similaridade estão próximas, enquanto palavras sem ligação ficam distantes. Tal posicionamento pode ser visto na Figura 1. Nela, palavras como maçã e banana estão próximas, enquanto estas estão distantes de jornal e revista (TECHNOLOGIES, 2021). Considerando cada par de palavras, a distância entre elas é pequena porque elas são representadas por vetores similares. Conforme os vetores diminuem sua similaridade, as palavras ficam mais distantes no espaço.

Figura 1 – Palavras em um espaço de 2 dimensões.



Fonte: Adaptado de Technologies (2021).

2.5 GRANDES MODELOS DE LINGUAGEM - LLM

Os grandes modelos de linguagem, mais conhecidos por seu termo em inglês *Large Language Models* (LLM), são sistemas de inteligência artificial utilizados para compreender e gerar textos em linguagem natural humana. Por exemplo, ao receber a frase incompleta “O céu está...”um LLM pode completar a frase com “azul hoje” com base em padrões comuns de linguagem. Capazes de processar grande quantidade de informações, são treinados com muitos dados para que retornem uma resposta precisa, natural e dentro do contexto, se tornando úteis em muitas tarefas cotidianas (IBM, 2024b). Alguns modelos de LLM já estão inseridos nas tecnologias utilizadas por usuários comuns, sendo apresentadas por interfaces como Chat GPT (OPENAI, 2024) da OpenAI, Llama (META AI, 2024) da Meta e BERT (DEVLIN *et al.*, 2018) do Google.

Através do uso de *deep learning* e de *transformers*, um LLM pode prever qual a próxima palavra utilizada em um texto, considerando o contexto, gramática e semântica do mesmo. Esses modelos recebem parâmetros em suas configurações que representam, de maneira simplificada, um padrão de escrita humana e como uma frase pode ser criada. O conhecimento adquirido por esses modelos é capaz de atender um usuário comum rapidamente, mas não é útil para buscas aprofundadas sobre assuntos específicos (WOLFF, 2023).

A arquitetura de um grande modelo de linguagem pode ser resumida em três partes: o codificador, os mecanismos de atenção e o decodificador. O codificador, a partir de *tokens* extraídos de um texto, é capaz de aproximar palavras semelhantes no espaço vetorial, ou seja, ele determina relações entre palavras. Os mecanismos de atenção permitem que o modelo processe partes específicas de um texto, focando em palavras mais relevantes. Já o decodificador converte os *tokens* processados em texto, produzindo conteúdo em linguagem humana (DATABRICKS, 2024).

Para que isso seja possível, é necessário que o modelo seja treinado de maneira adequada. Dentro de um LMM existe uma rede neural extensa, composta por nós e camadas que se comunicam entre si através de uma definição de relevância interna, os chamados pesos. Os pesos e vieses de uma rede neural são definidos em sua configuração por parâmetros, que podem atingir bilhões em quantidade (SERVICES, 2024b).

Uma alta quantidade de dados é utilizado para que o modelo ajuste seus próprios parâmetros, até que seja capaz de prever a próxima palavra, ou *token*, com êxito. A partir disso, o modelo pode receber conjuntos menores de dados para se tornar mais eficaz em processar determinado assunto, o que é chamado de ajuste fino.

2.6 RETRIEVAL AUGMENTED GENERATION

Retrieval Augmented Generation (RAG) é o termo em inglês traduzido como Geração Aumentada por Recuperação. RAG é uma tecnologia que melhora a eficiência de modelos de inteligência artificial como o LLM. Seu objetivo é, de maneira simplificada, recuperar dados relevantes para melhor contexto e resposta do modelo LLM (NVIDIA, 2024).

Esta técnica é capaz de oferecer referências para as informações levantadas, tais como fontes de dados, artigos científicos, documentos, ou outras informações verificáveis, entregando assim maior confiança ao usuário. Além disso, ela melhora o entendimento da máquina quando há ambiguidade no texto escrito por humanos, resultando na diminuição de respostas sem conexão com a entrada, comportamento conhecido como alucinação. Por exemplo, uma alucinação ocorre quando um LLM responde a uma pergunta sobre eventos futuros com informações inventadas, como prever que “a capital do Brasil será transferida para São Paulo em 2025” sem nenhuma base factual.

Os LLMs atuais conseguem entregar bons resultados no processamento de texto em linguagem humana. Entretanto, esses modelos não são facilmente escaláveis, ou seja, há um custo considerável para retreiná-los e adicionar novas informações. Além disso, eles operam sem revisar uma memória persistente, o que contribui para alucinações. Isso ocorre porque são baseados em memórias paramétricas, que se referem às configurações iniciais e fixas do modelo treinado (LEWIS *et al.*, 2020). Memórias paramétricas armazenam conhecimento nos próprios parâmetros do modelo durante o treinamento. Por outro lado, memórias não-paramétricas podem ser atualizadas dinamicamente e acessadas durante a inferência, permitindo a incorporação de novas informações sem a necessidade de retreinar o modelo.

Ao adicionar a técnica RAG em conjunto com um LLM, há uma combinação de memórias paramétricas e não-paramétricas. Memórias não-paramétricas são externas ao modelo, como um banco de dados, e podem ter novas informações facilmente adicionadas. Por exemplo, uma base de dados de artigos científicos pode ser atualizada continuamente com novos artigos, e o modelo pode recuperar informações específicas conforme a necessidade. Já as memórias paramétricas são internas ao modelo e são adquiridas durante o processo de treinamento.

Ao receber uma pergunta ou tarefa, o modelo com RAG utiliza sua memória paramétrica para interpretar a questão e, depois, utiliza a não-paramétrica para buscar informações relevantes e entregar uma resposta mais precisa. Isso contribui para que um modelo esteja sempre atualizado e capaz de processar informações dos mais diversos nichos.

Desde a criação da RAG, várias melhorias foram implementadas. Assim, os modelos de RAG podem ser divididos em três categorias: RAG ingênua, RAG avançada

e RAG modular. A RAG ingênua refere-se ao primeiro modelo criado, recebendo esse nome por seu funcionamento direto, seguindo um fluxo simples de receber uma tarefa, buscar informações e usá-las para gerar a resposta (GAO *et al.*, 2024).

A RAG avançada trouxe métodos mais sofisticados para a recuperação de informações e produção de respostas. Ela utiliza técnicas de pesquisa mais eficazes para encontrar informações relevantes, além de algoritmos para combinar os dados encontrados, resultando em respostas mais precisas e detalhadas, considerando o contexto da entrada.

Por fim, a RAG modular é o desenvolvimento mais recente. Nesse modelo, o processo de recuperação de informações e geração de respostas é separado em módulos, permitindo que funcionem de forma independente. Essa configuração possibilita melhorias específicas e direcionadas, além de um melhor gerenciamento de memória e interpretação.

Uma grande vantagem do uso da RAG em modelos de LLM é a facilidade de implementação. Ela pode ser implementada com apenas cinco linhas de código (RIEDEL *et al.*, 2020), resultando em um método mais rápido e menos custoso de melhoria em comparação ao retreinamento de um modelo de LLM.

2.7 PROMPTS

Inteligências artificiais generativas produzem resultados com base em uma entrada, também conhecida como *prompt*. Dessa forma, a qualidade da resposta gerada é diretamente dependente da qualidade do *prompt* fornecido ao modelo. Portanto, é essencial que haja um planejamento cuidadoso do texto de entrada para que o *prompt* inserido resulte em uma resposta precisa (SERVICES, 2024a; IBM, 2024a).

A formulação de *prompts* adequados faz parte de uma área conhecida como engenharia de *prompt* e, embora seja um campo de estudo relativamente novo, tem demonstrado um crescimento significativo. A capacidade de interagir de maneira eficaz com LLMs é fundamental, e a engenharia de *prompt* oferece uma metodologia crucial para facilitar essa interação.

À medida que essa nova área de pesquisa se desenvolve, novas técnicas estão sendo criadas para otimizar o uso e as respostas dos LLMs. Além disso, é evidente que a engenharia de *prompt* está se consolidando como uma ferramenta importante para o avanço da inteligência artificial, especialmente na comunicação entre humanos e máquinas (WANG, J. *et al.*, 2023).

Algumas estratégias merecem destaque quando se trata da criação de *prompts*, como *zero-shot*, *one-shot* e *few-shot*. A técnica de *zero-shot* refere-se ao modelo que recebe apenas uma instrução em linguagem natural, sem exemplos anteriores. Este método pode não apresentar respostas de qualidade, pois o modelo de LLM deve processar informações sem qualquer contexto (BROWN *et al.*, 2020).

Por outro lado, a técnica *one-shot* fornece um único exemplo ao modelo antes que seja gerada uma resposta de acordo com o *prompt*. Assim, o modelo é condicionado a seguir um contexto, por menor que este seja. Esta abordagem se assemelha ao aprendizado humano.

Por fim, a estratégia de *few-shot* expõe o modelo a diversos exemplos. Isso permite que o modelo compreenda, de fato, novas informações e use os exemplos como guias para a geração da resposta. Esta é a técnica utilizada no projeto apresentado neste documento.

2.8 MÉTRICAS DE AVALIAÇÃO

A eficiência de alguns modelos de linguagem que utilizam recuperação de informação pode ser medida através de métricas de avaliação. Este processo é crucial para garantir bons resultados. Utilizando métodos como a Taxa de Acerto (do inglês *Hit Rate*) e a Classificação Recíproca Média (do inglês *Mean Reciprocal Rank* - MRR), é possível medir a qualidade dos dados recuperados.

O uso de métodos de avaliação é importante e necessário, pois permite que os desenvolvedores entendam como os modelos de recuperação de informação utilizados afetam o comportamento do sistema RAG. Assim, é possível identificar as melhores combinações e impulsionar o desempenho geral do sistema.

A métrica *Hit Rate* é calculada como a fração de consultas em que a resposta correta é encontrada entre os primeiros documentos recuperados, ou seja, a frequência com que o sistema acerta considerando apenas as primeiras tentativas (CONTRIBUTORS, 2024a). A fórmula para a *Hit Rate* pode ser expressa como na equação 1:

$$\text{Hit Rate} = \frac{\text{Número de acertos nas primeiras tentativas}}{\text{Número total de consultas}} \tag{1}$$

Já o MRR (*Mean Reciprocal Rank*) avalia a precisão do sistema, considerando a ordem de relevância atribuída a cada documento. Por exemplo, se o primeiro documento retornado for o mais relevante, seu Rank é 1, mas se o segundo documento for o mais relevante, seu Rank é 1/2 (CONTRIBUTORS, 2024b). A fórmula para o MRR é dada pela equação 2:

$$\text{MRR} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{\text{Rank}_i} \tag{2}$$

onde N é o número total de consultas e Rank_i é a posição do primeiro documento relevante na i-ésima consulta.

Para aplicar tais métodos, é possível gerar dados sintéticos para a formação de um conjunto de dados de avaliação inicial. Dados sintéticos são dados artificiais criados para simular condições reais sem a necessidade de acessar dados confidenciais ou

indisponíveis. Esses dados podem ser gerados através de algoritmos específicos ou modelagem estatística, baseados em padrões observados em dados reais. A geração de dados sintéticos é necessária quando não se tem acesso a um volume suficiente de dados reais ou quando se deseja testar o sistema em cenários controlados e específicos. Esta prática simplifica o processo de avaliação, sendo uma maneira rápida e eficiente de testar diferentes configurações de um sistema RAG antes de implementá-lo de fato.

3 DESCRIÇÃO DO PROBLEMA, REQUISITOS TÉCNICOS E SOLUÇÃO PROPOSTA

Neste capítulo, será abordado a contextualização dos desafios enfrentados pela empresa cliente, a descrição do problema abordado neste trabalho, bem os requisitos técnicos a serem atendidos e a solução proposta pelo autor.

Inicialmente, foi realizada uma análise do escopo de trabalho em que o projeto está inserido, ressaltando aspectos relacionados à escalabilidade, casos de uso, entendimento do processo e fonte das informações a serem utilizadas. Dessa forma, pôde-se ter um entendimento melhor das dificuldades enfrentadas pela empresa cliente.

3.1 CONTEXTUALIZAÇÃO

No ambiente empresarial atual, a automação dos processos e a redução da dependência do trabalho manual são elementos cruciais para manter a competitividade. Empresas buscam aumentar a eficiência, reduzir custos e melhorar o desempenho das operações através de soluções inovadoras e orientadas por dados. A BIX Tecnologia, uma consultoria de dados, destaca-se nesse cenário ao oferecer soluções em Engenharia de Dados, Business Intelligence, Ciência de Dados e Desenvolvimento de Software. Com mais de 60 clientes líderes de mercado, incluindo uma expansão internacional, a BIX Tecnologia é reconhecida por seu compromisso em desbloquear o potencial dos dados para impulsionar o sucesso de seus clientes.

A empresa cliente, uma fabricante multinacional de equipamentos eletroeletrônicos, enfrenta desafios significativos em seus times de suporte ao cliente. Esta empresa, que fabrica milhares de dispositivos e oferece suporte técnico em diversos países ao redor do mundo, incluindo Estados Unidos, Canadá, Reino Unido, Alemanha, China, Índia e Brasil, identificou dificuldades na redução da dependência de mão-de-obra e no treinamento de pessoal em múltiplas ferramentas e atividades.

Os desafios enfrentados incluem a necessidade de automatizar processos para melhorar a precisão no diagnóstico de problemas nos dispositivos e a capacitação do pessoal em um ambiente de suporte técnico global. A complexidade das tarefas aumentava a propensão a erros humanos, enquanto os altos custos operacionais estavam diretamente ligados à necessidade de treinamento contínuo e à dependência de mão-de-obra manual.

Para enfrentar esses desafios, a BIX Tecnologia propôs uma solução inovadora que visava automatizar e aprimorar a precisão nos diagnósticos, reduzindo assim o esforço na intervenção manual e os custos associados. Essa colaboração com a multinacional resultou em um projeto que buscava não apenas resolver os problemas imediatos, mas também criar um sistema mais eficiente e sustentável para o suporte técnico da empresa.

Essa proposta inclui a implementação de modelos avançados de Inteligência Artificial, como os baseados em grandes modelos de linguagem (LLM), dentro do ambiente da empresa cliente. Um exemplo prático é a análise automática e geração de sumários de eventos dos dispositivos fabricados pela multinacional. De forma a automatizar esse processo, auxiliar na identificação de problemas e reduzir o tempo necessário para resolver chamadas de suporte técnicas.

Em suma, a parceria entre a BIX Tecnologia e a empresa cliente exemplifica como soluções tecnológicas avançadas podem transformar a maneira como as empresas operam, resultando em maior eficiência, redução de custos e melhoria na qualidade do serviço prestado aos clientes. A BIX Tecnologia se destacou como parceira ideal devido à sua experiência comprovada em ciência de dados e desenvolvimento de *software*, além de seu histórico em projetos que transformam dados em decisões estratégicas para os negócios.

3.2 DESCRIÇÃO DO PROBLEMA

A empresa cliente enfrenta diversos desafios na manutenção da eficiência operacional e na gestão de sua força de trabalho. Entre os principais problemas identificados estão a redução do número de funcionários, a dificuldade de treinamento da equipe em várias ferramentas e atividades, além dos altos custos operacionais (OPEX). Há uma clara necessidade de aumentar a automação, não apenas para diminuir o trabalho manual, mas também para minimizar erros humanos e reduzir a necessidade de treinamento contínuo na manutenção dos dispositivos eletroeletrônicos que fabrica.

O processo de diagnóstico de problemas nos equipamentos envolve a análise de dados fornecidos pelos consumidores, que incluem descrições de problemas e registros de eventos (*logs*) gerados pelos próprios equipamentos. Tais *logs* são fundamentais para identificar o histórico de funcionamento e detectar comportamentos anômalos, pois registram eventos tanto de funcionamento correto quanto de falhas. Esses registros podem incluir informações detalhadas sobre erros de hardware, falhas de software, problemas de conectividade e outras anomalias que precisam ser interpretadas corretamente para um diagnóstico preciso. Entretanto, a análise dos *logs* é uma tarefa complexa que exige profundo conhecimento técnico e extenso treinamento da equipe de suporte ao cliente. A complexidade e a variabilidade dos dados tornam o processo demorado e oneroso, aumentando os custos e os tempos de resposta na resolução de problemas.

Os times de suporte ao cliente enfrentam várias dificuldades nesse processo. A análise manual dos *logs* é demorada e suscetível a erros, além de demandar alto custo de treinamento para que os técnicos sejam capacitados tecnicamente a lidar com a vasta gama de equipamentos e seus respectivos modelos. Cada modelo possui particularidades específicas, aumentando a complexidade do diagnóstico e prolon-

gando o tempo necessário para resolução de problemas. Isso, por sua vez, impacta negativamente a eficiência operacional e os custos da empresa cliente.

Dada essa complexidade e a diversidade de produtos fabricados pela empresa cliente, foi solicitado pela empresa cliente o desenvolvimento de uma Prova de Conceito (PoC) para validar uma abordagem automatizada no processo de sumarização e auxílio na tomada de decisão. A PoC foi especificamente solicitada para explorar o uso de grandes modelos de linguagem (LLMs) como uma potencial solução para os desafios enfrentados na resolução de problemas dos equipamentos.

Com essa PoC, espera-se que a empresa cliente possa avaliar a eficácia de utilizar LLMs para reduzir o esforço na análise, minimizar erros humanos e diminuir os custos operacionais, antes de uma possível implementação em larga escala.

3.3 SOLUÇÃO PROPOSTA

A solução proposta foi desenvolvida com base nos requisitos técnicos definidos nas reuniões iniciais com a empresa cliente. Essa solução inclui a criação de uma Prova de Conceito (PoC) que utiliza grandes modelos de linguagem (LLMs) e outras tecnologias de Inteligência Artificial (IA) para automatizar o processo de auxílio na resolução dos problemas dos equipamentos. A Prova de Conceito servirá para demonstrar a eficácia e a viabilidade dessa abordagem, possibilitando que a empresa cliente possa avaliar os benefícios antes de uma implementação em larga escala

De maneira resumida, o objetivo principal deste projeto é conseguir uma ferramenta a partir de inteligência artificial que seja capaz de sumarizar o *log* dos equipamentos e fornecer sugestões para auxiliar nas tomadas de decisões a partir dos eventos registrados. A sumarização dos *logs* envolve a condensação de grandes volumes de dados em informações essenciais e relevantes.

Para que uma inteligência artificial interprete o conteúdo dos *logs* de maneira eficiente, primeiramente, é necessário fornecer as informações específicas dos equipamentos. Tais informações são provenientes de documentos específicos, como os manuais técnicos, que precisam ser processados. Assim, é possível estender o domínio de conhecimento da IA para casos de uso específicos ou, até mesmo, utilizar dados sensíveis e privados, sem que haja um retreinamento do modelo.

Inicialmente, como o LLM é um modelo pré treinado e disponível para uso, é preciso uma forma de disponibilizar um LLM em um ambiente que possua todos os requisitos técnicos, como configuração do ambiente e compatibilidade, que possa ser executado tanto localmente quanto em ambiente na nuvem. Uma das formas de prover esse ambiente ideal é a partir da utilização de contêineres.

Com a LLM disponível para utilização, é possível realizar perguntas para o modelo e receber uma resposta, porém nosso desejo é que o modelo seja capaz de responder perguntas específicas aos equipamentos da empresa cliente. Por conta des-

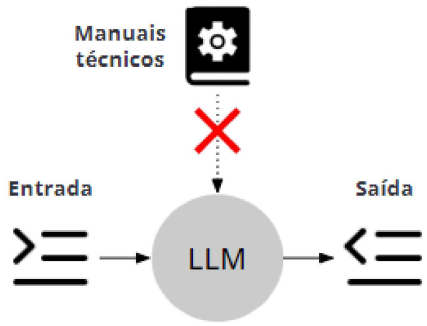


Figura 2 – Representação da LLM sem fonte de conhecimento adicional.

ses documentos serem publicados em ferramentas internas a empresa, é improvável que durante o treinamento de um grande modelo de linguagem essas informações estivessem na coleção usada no treino. Dessa forma, como visto na Figura 2 , a LLM é incapaz de responder perguntas específicas dos equipamentos de forma correta e sem alucinação.

Essa limitação no escopo de conhecimento dos modelos, não é um impeditivo para que uma LLM possa responder perguntas nichadas. Para isso é possível utilizar o RAG para fornecer novas informações a LLM e enriquecer as respostas.

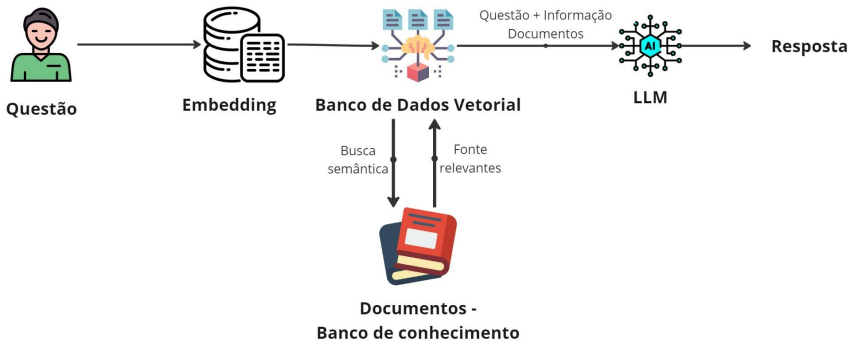
O RAG recebe um documento e extrai seu texto, esse texto é então vetorizado em uma representação numérica, de forma que posteriormente seja possível realizar uma busca semântica usando a pergunta do usuário, também vetorizada, e toda coleção de documentos processados. O documento mais relevante após a busca é então utilizado como entrada para uma LLM gerar uma resposta com o contexto correto.

Na Figura 3 vemos a arquitetura conceitual do fluxo do RAG. Nela podemos ver as várias etapas envolvidas, como a fonte de dados externos, que é o novo conhecimento que será informado ao LLM; o modelo de incorporação, que converte os dados textuais, tanto da nova fonte de dados, quanto da consulta do usuário, em representações vetoriais; o banco de dados de vetores, no qual são armazenadas as incorporações para criar uma biblioteca para recuperar informações; e, por fim, um grande modelo de linguagem, que terá como entrada um mapeamento do conteúdo recuperado relativo à consulta do usuário e, com isso, retornará um texto gerado a partir das novas informações às quais o modelo não tinha conhecimento prévio.

Na Figura 4, é ilustrado o processo RAG (Retrieval-Augmented Generation) para resposta de perguntas em três etapas principais.

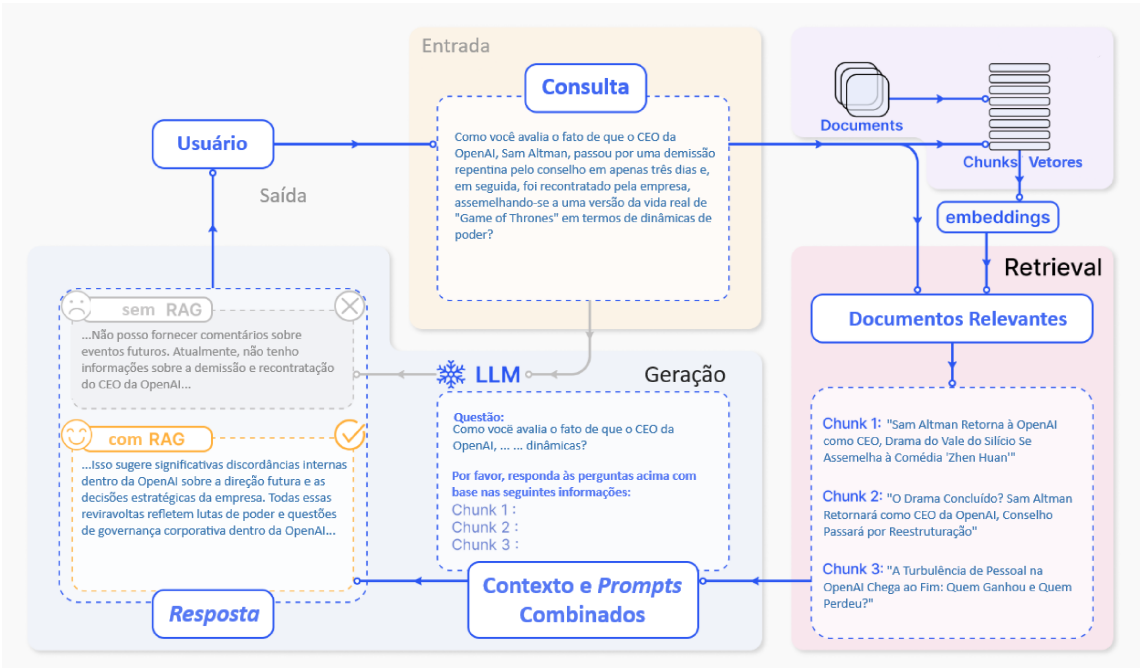
Primeiramente, os documentos são divididos em *chunks*, que são pequenas partes ou segmentos dos documentos originais. Esses *chunks* são então codificados em vetores por meio de uma técnica chamada de *embeddings*. *Embeddings* são representações numéricas dos *chunks* que capturam suas características semânticas, permitindo que similaridades entre textos possam ser calculadas de forma eficiente.

Figura 3 – RAG - Fluxo de funcionamento.



Fonte: Autor.

Figura 4 – Representação de RAG aplicado a resposta de questões.



Fonte: Adaptado de Gao *et al.* (2024) .

Esses vetores são armazenados em um banco de dados vetorial, que permite uma recuperação rápida e precisa de informações relevantes.

Em seguida, ocorre o processo de *retrieval* (recuperação de texto). Com base na consulta fornecida pelo usuário (neste contexto, o usuário é a pessoa que faz a pergunta ao sistema), os *chunks* mais relevantes são recuperados do banco de dados vetorial. Essa recuperação é feita com base na similaridade semântica entre a consulta e os *embeddings* dos *chunks*. O objetivo é selecionar os *chunks* que contêm informações mais pertinentes para responder à pergunta do usuário.

Por fim, a consulta do usuário e os *chunks* recuperados são combinados e inseridos em um grande modelo de linguagem (LLM) para gerar a resposta final. O

LLM utiliza o contexto fornecido pelos *chunks* relevantes para produzir uma resposta mais informada e contextualizada.

No diagrama, a consulta do usuário exemplificada é: “Como você avalia o fato de que o CEO da OpenAI, Sam Altman, passou por uma demissão repentina pelo conselho em apenas três dias e, em seguida, foi recontratado pela empresa, assemelhando-se a uma versão da vida real de ’Game of Thrones’ em termos de dinâmicas de poder?”

Os *chunks* recuperados para essa consulta são:

Chunk 1: “Sam Altman Retorna à OpenAI como CEO, Drama do Vale do Silício Se Assemelha à Comédia ’Zhen Huan”’ Chunk 2: “O Drama Concluído? Sam Altman Retornará como CEO da OpenAI, Conselho Passará por Reestruturação” Chunk 3: “A Turbulência de Pessoal na OpenAI Chega ao Fim: Quem Ganhou e Quem Perdeu?” A resposta gerada sem o uso de RAG poderia ser limitada, com uma afirmação como: “Não posso fornecer comentários sobre eventos futuros. Atualmente, não tenho informações sobre a demissão e recontratação do CEO da OpenAI...”

Enquanto que a resposta gerada com o uso de RAG seria mais elaborada e informada, como: “Isso sugere significativas discordâncias internas dentro da OpenAI sobre a direção futura e as decisões estratégicas da empresa. Todas essas reviravoltas refletem lutas de poder e questões de governança corporativa dentro da OpenAI...”

Portanto, o uso de RAG melhora substancialmente a qualidade das respostas fornecidas pelo modelo, utilizando informações relevantes recuperadas dos documentos originais.

A tecnologia desenvolvida até então deve estar disponível em uma interface para que os usuários consigam validar a ferramenta. Essa interface deve possuir um campo para realizar perguntas e retornar respostas utilizando os manuais como fonte de informação.

Com esta parte da ferramenta desenvolvida, pode-se iniciar o desenvolvimento do módulo responsável pela sumarização dos *logs* dos equipamentos. Devido ao limite do tamanho da pergunta que um LLM é capaz de receber, os arquivos de *log* não podem fazer parte do corpo da pergunta, já que podem ser muito extensos. Por isso é necessário filtrar apenas as informações relevantes para a pergunta, que estão contidas no arquivo do *log*.

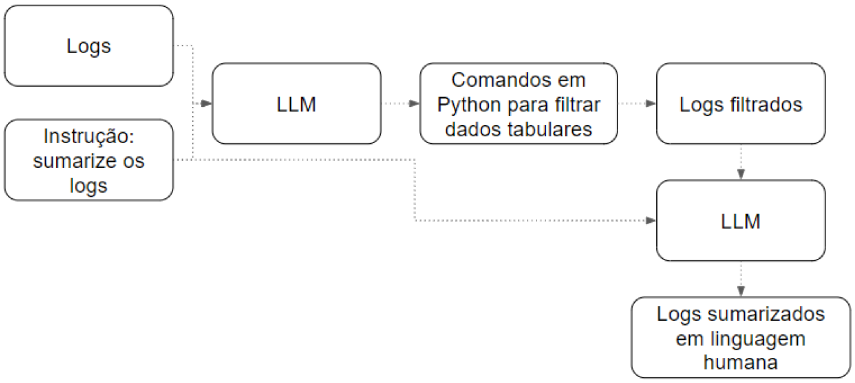
Para realizar a filtragem, a pergunta do usuário deve ser convertida em uma linguagem de consulta a partir de um LLM. Em sequência, a consulta é aplicada no arquivo de *log*, de forma que o resultado contenha apenas as informações relevantes, que é então passada para uma outra LLM em conjunto com a pergunta para gerar a resposta desejada, ou seja, o *log* sumarizado.

Por fim, o *log* sumarizado pode ser passado como entrada no RAG, que já possui as informações provenientes dos manuais técnicos, em conjunto com a pergunta do usuário, para que a resposta do LLM seja enriquecida e possa auxiliar nas tomadas

de decisão. O sistema final deve ser apto a importar *logs* no formato *csv*. Com isso, os colaboradores podem realizar testes com futuros *logs* que serão gerados.

Conforme ilustrado na Figura 5, o processo inicia-se com os *logs* de equipamentos e instruções em texto (por exemplo, "Sumarize o *log* do equipamento"). Esses dados são enviados para um modelo de linguagem (LLM) que converte a instrução em uma consulta na sintaxe SQL/Pandas. Essa consulta é então aplicada aos *logs* de equipamentos, resultando em *logs* filtrados. Em seguida, os logs filtrados são novamente processados pelo modelo de linguagem (LLM) para gerar uma versão sumarizada dos *logs*. Dessa forma, a solução assegura que os *logs* de equipamentos sejam filtrados e resumidos de maneira eficiente, atendendo às instruções especificadas.

Figura 5 – Processamento e Sumarização de Logs de Equipamentos Utilizando LLM.



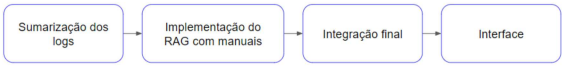
Fonte: Autor.

A solução proposta deve resolver os problemas enfrentados pela empresa cliente, melhorando a eficiência na identificação de falhas em equipamentos, automatizando processos complexos e eliminando a necessidade de treinamento especializado dos modelos, o que, em conjunto, aumenta a eficiência das operações.

Com isso, de maneira resumida, como visto na Figura 6, a solução propõe dividir o desenvolvimento da solução em quatro etapas. A primeira etapa tem como foco a utilização dos dados dos registros de eventos de equipamentos, de forma que a LLM fosse capaz de sumarizar os registros sem o uso dos manuais como contexto. A etapa seguinte prosseguiu com a utilização de documentos com informações sobre os equipamentos, por exemplo, manuais, para o desenvolvimento de uma ferramenta com a qual pudéssemos testar diferentes abordagens de RAG. Por fim, as duas últimas etapas são destinadas ao desenvolvimento de uma interface que integrasse o resultado do desenvolvimento das duas primeiras etapas e possibilitasse o usuário interagir com a solução.

Os dois primeiros módulos, apesar de serem desenvolvidos e funcionarem de forma independente, serão utilizados de forma conjunta para atingir o resultado esperado, no qual a interface desempenhará o papel de integrar o usuário e os módulos de

Figura 6 – Etapas da solução.



Fonte: Autor.

documentos e de registro de eventos dos equipamentos, assim como na Figura 7.

Figura 7 – Fluxo com integração entre os dois módulos.

Fonte: Autor.

3.4 REQUISITOS TÉCNICOS A SEREM ATENDIDOS

Os requisitos técnicos da solução foram definidos através de diversas reuniões iniciais entre a BIX e a empresa cliente, com o objetivo de alinhar as necessidades e expectativas do projeto.

A proposta de solução deve incluir também uma fase de pesquisa para confirmar ou refutar as seguintes hipóteses:

- Uso de Modelos de IA: Verificar se é possível utilizar o mesmo modelo de inteligência artificial tanto para análise de documentos técnicos (como manuais), quanto para registros de funcionamento dos equipamentos. Esta hipótese visa determinar a flexibilidade e adaptabilidade do modelo de IA.
- Memória e Contexto: Analisar se o *chatbot* pode fornecer respostas precisas sem treinamento específico, mas com acesso a informações ou contexto relacionado ao dispositivo. Esta abordagem testa a capacidade do modelo de utilizar informações contextuais para melhorar a precisão das respostas.
- Execução Local de Modelos: Explorar a viabilidade de executar modelos de IA localmente, atendendo necessidades da empresa cliente. Isso inclui a avaliação de servidores proprietários de tamanho médio para suportar a carga de processamento necessária.
- Adequação de Modelos de Código Aberto: Determinar se modelos de IA de código aberto são suficientes para a aplicação proposta, comparando sua performance com modelos proprietários.

As expectativas para a PoC incluem a entrega de um serviço na forma de um sistema de resposta automática que receba um arquivo ou fluxo de registros e explique

o que ocorreu com o dispositivo durante o período registrado; responda a perguntas sobre os dispositivos com base no manual técnico (opcionalmente) e possua uma interface simples para interação do usuário, capaz de apresentar os resultados de maneira clara.

O time de desenvolvimento da BIX determinou que a PoC deve atender os seguintes requisitos funcionais:

- Análise de Logs: Capacidade de receber arquivos com *logs* dos equipamentos e explicar detalhadamente os eventos que ocorreram durante o período em que os registros foram realizados. Isso inclui identificar e descrever falhas e anomalias.
- Consulta a Manuais Técnicos: Responder a perguntas sobre os dispositivos com base nas informações contidas nos manuais técnicos. Por exemplo, a PoC deve ser capaz de fornecer sugestões de solução baseadas na documentação existente para cada tipo de equipamento.
- Interface para Interação do Usuário: Desenvolver uma interface amigável que permita a visualização clara e organizada dos resultados e sugestões fornecidas pela LLM.

Para garantir a implementação e operação eficaz da PoC, os seguintes requisitos de infraestrutura deverão ser satisfeitos:

- Servidores e Capacidade de Processamento: A PoC deve ser capaz de ser executada tanto localmente quanto em servidores proprietários de tamanho médio.
- Armazenamento de Dados: Sistemas de armazenamento adequados para manter os *logs* e dados utilizados no treinamento e operação dos modelos de IA.
- Segurança e Privacidade: Implementação de medidas de segurança e privacidade para proteger os dados sensíveis dos clientes da empresa.
- Integração com Sistemas Existentes: Facilidade de integração com os sistemas e processos existentes na empresa cliente, minimizando a necessidade de alterações na infraestrutura atual.

As expectativas para a PoC incluem a entrega de um serviço que receba um arquivo ou fluxo de registros e explique o que ocorreu com o dispositivo durante o período registrado, responda a perguntas sobre os dispositivos com base no manual técnico e possua uma interface simples para interação do usuário, capaz de apresentar os resultados de maneira clara.

4 DESENVOLVIMENTO E IMPLEMENTAÇÃO

Este capítulo apresenta o processo de desenvolvimento do projeto de fim de curso. Na Seção 4.1 é apresentado as ferramentas utilizadas para auxiliar no desenvolvimento. A Seção 4.5 detalha a aplicação da técnica *Retrieval Augmented Generation* (RAG) com manuais de equipamentos para enriquecer respostas de modelos de linguagem. A Seção 4.5.6 aborda a criação de uma interface de usuário no Streamlit para interação com o sistema de *Retrieval Augmented Generation* (RAG), permitindo consultas e visualização de respostas. A Seção 4.6 descreve o uso de LLMs para criar automaticamente um conjunto de dados de perguntas e respostas para avaliação de modelos de recuperação de informações. Na Seção 4.3 é abordado a utilização de uma cadeia de LLMs para sumarizar *logs* tabulares, destacando as limitações do uso direto de LLMs para esses dados. Por fim, a Seção 4.7 detalha a integração do módulo de sumarização de *logs* com o módulo de *Retrieval Augmented Generation* (RAG), destacando melhorias na interface e na modularização de código para facilitar análises de eventos.

O projeto foi desenvolvido utilizando a linguagem de programação Python, juntamente com uma variedade de módulos e ferramentas de código aberto disponibilizadas para a comunidade. Essas ferramentas apresentam abstrações de códigos e técnicas e foram criados para executar funções específicas, visando facilitar e acelerar o desenvolvimento de aplicações. A análise dos resultados obtidos será apresentada no próximo capítulo, fornecendo uma avaliação do desempenho na recuperação de documento.

4.1 FERRAMENTAS

4.1.1 Llamaindex

LlamaIndex é um *framework* que facilita a orquestração e desenvolvimento de aplicações que utilizam grandes modelos de linguagem, pois permite a integração entre fonte de dados personalizados e LLMs. Ao utilizar o *framework*, o modelo pode utilizar os dados disponibilizados para auxiliar na geração de uma resposta. Tais dados podem ser privados e não ter sido utilizados no treinamento do modelo, portanto, sem o *framework*, não podem ser uma fonte de auxílio para gerar a resposta (LLAMAINDEX, 2024b).

O LlamaIndex permite o carregamento de dados de forma simplificada, através do suporte a diversas fontes de dados como banco de dados, APIs, e diferentes formatos de arquivos como PDF e Excel. Esses carregadores facilitam ao desenvolvedor a realização da conexão com as fontes de dados e a ingestão em aplicações com LLM.

Além da ingestão de dados, essa ferramenta oferece diversas abstrações que são essenciais na técnica de RAG, como o armazenamento de documentos, a inde-

xação de dados, a interface de consulta e a avaliação das recuperações. Maiores detalhes sobre cada uma dessas etapas e implementações serão exploradas ao longo do texto.

Uma das vantagens em utilizar esse *framework* é o suporte para implantação local, oferecendo benefícios de privacidade, pois os dados não precisam ser compartilhados com terceiros. Ele permite alta disponibilidade e operação *offline* de aplicações baseadas em LLM. Os usuários podem testar diferentes LLMs de código aberto localmente.

4.1.2 Chroma

O ChromaDB é um sistema de armazenamento de vetores de código aberto, projetado para armazenar e recuperar *embeddings* de vetores, juntamente com metadados associados. Esses *embeddings* são essenciais para quaisquer modelos de linguagem e podem ser usados em mecanismos de busca semântica que operam com dados textuais (CHROMA, 2024).

Essa ferramenta é interessante e uma boa opção de utilização devido a sua simplicidade de configuração e a possibilidade de adaptação para casos de uso avançados. Para utilizar o ChromaDB, é possível fazer sua instalação usando gerenciadores de pacote em Python, que no caso do autor foi o *pip*. Ao utilizar o ChromaDB, podemos criar coleções, que se assemelham a tabelas de um banco de dados relacional e consistem em um agrupamento dos documentos utilizados, os metadados respectivos e o *embedding* criado.

Diferentes modelos podem ser utilizados para criação dos *embeddings*. Esses *embeddings* são utilizados para realizar a busca por similidade de vetores, no qual um documento é recuperado com base na consulta de um usuário, processo esse chamado de recuperação, do inglês *retrieval*. O processo de *retrieval* se inicia com a consulta do usuário, que é utilizada para realizar uma busca no *embedding* armazenado no Chroma pelo texto mais relevante para a consulta. Esse texto é, então, fornecido à LLM de preferência.

4.1.3 Hugging Face

Hugging Face é uma plataforma líder em inteligência artificial, especialmente conhecida por suas contribuições ao processamento de linguagem natural (PLN). Ela oferece uma variedade de ferramentas, bibliotecas e modelos pré-treinados de PLN, como BERT, GPT e Transformer, que revolucionaram o campo de pesquisa. A plataforma não só fornece acesso fácil a esses modelos, mas também promove um ambiente colaborativo para pesquisadores e desenvolvedores avançarem na tecnologia de PLN (HUGGINFACE, 2024).

O ecossistema abrangente apoia o aprendizado e a aplicação de PLN, fornecendo recursos extensivos, incluindo repositórios de código e modelos de IA, conjuntos de dados e aplicações usando o que é mais recente tecnologia em inteligência artificial. A abordagem de código aberto do Hugging Face para IA, comparada ao GitHub, atraiu mais de 50.000 organizações que utilizam suas ferramentas para diversas aplicações de IA (TECHCRUNCH, 2023).

O crescimento do Hugging Face também é impulsionado por parcerias estratégicas, como com a Dell Technologies, para facilitar a implementação de modelos de IA generativa em implementações locais (NEWSWIRE, 2023). Além disso, o Hugging Face fornece recursos educacionais e suporta variadas aplicações em texto, áudio e visão computacional através de seus modelos de aprendizado de máquina. A missão da plataforma de democratizar a IA se reflete através de suas contribuições de código aberto, desenvolvimento profissional e uma cultura de apoio à comunidade.

4.1.4 Google Cloud Platform

A Google Cloud Platform (GCP) é uma plataforma que oferece um conjunto de serviços de computação em nuvem, disponibilizando máquinas virtuais e módulos para processamento e gerenciamento de aplicações em ambiente virtual. Alguns dos serviços incluem armazenamento, redes, big data, aprendizado de máquina (ML), análise, Internet das Coisas (IoT), segurança e ferramentas para desenvolvedores (GOOGLE, 2024).

O GCP é utilizado por empresas com diversos níveis de maturidade, não importando se é uma startup ou uma grande empresa, A plataforma oferece diferentes configurações de máquinas, que podem se adequar ao objetivo de utilização. Dessa forma, é amplamente utilizada para construir, testar e implantar aplicativos e serviços na nuvem, permitindo escalabilidade e gerenciamento de recursos.

4.2 IMPLEMENTAÇÃO DA LLM EM CONTÊINER

De acordo com os requisitos do cliente, uma das exigências é que os modelos e todo o conjunto de ferramentas utilizadas no desenvolvimento do projeto estivessem desconectados da internet, ou seja, uma solução em servidores locais, ou do inglês, *on-premise*. Por conta disso, o desenvolvimento do projeto foi iniciado com testes para validar não só a possibilidade de implementar os grandes modelos de linguagem em máquina local, mas também os requisitos para processamentos, tempo de resposta e custos envolvidos, caso o cliente deseje hospedar a solução na nuvem.

Além disso, outro requisito é apenas a utilização de modelos *Open Source* que possam ser implementados em máquina local. Modelos *Open Source*, assim como *softwares Open Source*, são projetos no qual o código fonte é disponibilizado

para utilização de forma gratuita para todos que desejarem utilizar e contribuir para o aprimoramento da tecnologia. Isso foi solicitado pelo cliente de modo a proteger os dados ao serem processados por esses modelos para inferência. O contrário dos projetos *Open Source*, são os projetos *Closed Source*, que são projetos mantidos por empresas privadas, utilizando infraestrutura própria para disponibilizar e servir os serviços. Dessa forma, ao utilizar modelos *Closed Source*, como por exemplo, os modelos da OpenAI, tal qual o GPT, a conexão é feita através de uma API. Assim, para utilização, os dados devem ser enviado para API e processados na infraestrutura da empresa que fornece o serviço. Empresas que desejam manter seus dados protegidos, dado a sensibilidade dos dados, tendem a não enviar suas informações para empresas terceiras.

Para isso foi utilizado apenas modelos *Open Source* que podem ser processado em máquina local. Para que o modelo possa ser utilizado posteriormente pelo cliente, foi utilizado a tecnologia de encapsulamento de aplicações ou sistemas operacionais com o uso de contêineres. Esses contêineres possibilitam o desenvolvedor criar uma imagem que reproduz o ambiente de desenvolvimento, de forma que a aplicação possa funcionar de maneira correta, assim como o desenvolvedor projetou inicialmente. Isso é realizado para que não haja conflito de versões de aplicações ou sistemas operacionais. Com isso, uma aplicação sempre irá funcionar nas mesmas configurações de ambiente, a menos que o desenvolvedor realize e disponibilize uma nova versão da imagem com novas configurações.

Nesse projeto foi utilizado o *software* de código aberto Docker (DOCKER, 2024) para realizar o encapsulamento da aplicação em um contêiner. Para a criação do contêiner é necessário a utilização de uma imagem, que é um arquivo contendo as instruções da configuração do ambiente projetado pelo desenvolvedor. Essa imagem é um arquivo compartilhável e portátil, podendo ser implantada em diferentes locais. Dessa forma, uma imagem permite que outras máquinas possam rodar a aplicação encapsulada em um contêiner, sem que haja uma configuração e *download* de *softwares* que são requisitos para executar a aplicação desenvolvida. Um Docker também permite que sua aplicação possa ser executada em plataformas na nuvem, ou seja, em serviços que oferecem máquinas virtuais *on demand*, para processamento e execução de aplicações.

4.2.1 Encapsulamento de uma LLM

Como um dos requisitos do cliente é que a PoC possa ser executada em ambiente local e que não houvesse compartilhamento de dados por chamadas em API, foi necessário utilizar modelos que pudessem ser executados em máquina local, ou seja, modelos de código aberto.

Implantar um grande modelo de linguagem manualmente não é uma tarefa trivial.

Existem diversos desafios técnicos envolvidos nesse processo, como configuração do ambiente, compatibilidade e gerenciamento de pacotes, uso eficiente de recursos computacionais, otimização do funcionamento dos modelos, atualizações dos modelos, entre outros.

De modo a resolver e facilitar desenvolvedores na utilização de LLMs, existe uma ferramenta de código aberto chamada Ollama que possibilita executar e compartilhar LLMs localmente, seja através de *download* de um *software* ou com a utilização de um contêiner disponibilizado por meio de uma interface de linha de comando (CLI). O Ollama oferece suporte e manutenção a vários modelos com diferentes configurações de parâmetros e tamanhos, incluindo os de códigos abertos, como Llama2, Mistral, e modelos de código fechado, como OpenAI.

Nessa PoC, foi optado pela utilização de dois grandes modelos de linguagem disponibilizados publicamente para uso, de forma que tenhamos opções de geração de respostas, sendo o primeiro o LLama2 13B, desenvolvido pela empresa Meta, e o segundo o modelo Mistral 7B Instruct, desenvolvido pela empresa Mistral AI. Esse modelos foram escolhidos devido a boa reputação na comunidade e também pelo bom desempenho na tabela de classificação de modelos da *LMSYS Chatbot Arena*, uma plataforma de avaliação de LLMs com base em *feedbacks* humanos (CHIANG *et al.*, 2024). É importante frisar que, constantemente, novos modelos são treinados e os modelos escolhidos brevemente serão ultrapassados.

4.2.1.1 Implementação dos modelos na CPU

Para avaliar a implementação dos modelos em uma CPU, realizamos testes de desempenho em dois LLMs selecionados: Mistral 7B Instruct v2 e LLama2 13B. Esses testes foram projetados para medir o tempo de resposta, o uso do núcleo do processador e o consumo de memória. Os resultados desses testes estão resumidos na Tabela 1, fornecendo uma das métricas de desempenho de cada modelo em diferentes configurações de máquina.

Tabela 1 – Métricas de desempenho dos modelos implementados na CPU.

	Tempo de resposta [s]	**Núcleos do processador**	**Memória RAM [GB]**
Modelo 1: mistral 7B - instruct v2	120	8	12
	50	32	32
Modelo 2: Llama2 13B	240	8	12
	240	8	32
	60	32	32

Fonte: Autor.

Para fazer a implantação do modelo, basta ter o Docker instalado e executar os comando apresentados na Seção de Código 4.1 no terminal:

```
1
2 docker run -d -v ollama:/root/.ollama -p 11434:11434 --name ollama
    ollama/ollama}
3
4 docker exec -it ollama ollama run mistral:7b-instruct-v0.2-q4\_0
```

Seção de Código 4.1 – Comando para executar um Docker

Após a execução será liberado um IP local no computador para chamadas de geração de texto, no seguinte endereço:

- http://localhost:11434/api/generate

Como o tempo de processamento para fazer inferência e gerar resposta com os modelos testados utilizando a CPU foi alto, foi alinhado com o cliente a implantação dos mesmos modelos na nuvem da GCP, de forma que seja configurado com máquinas mais robustas e com a utilização de GPU. O uso de GPUs aceleram significativamente o processamento de inferência, especialmente para modelos grandes, tornando-os mais viáveis para aplicativos em tempo real. Seu uso local é adequado para carregar dados e modelo menores, mas o desempenho pode ser limitado.

4.2.1.2 Implementação dos modelos no GCP usando GPU

Para avaliar a implementação dos modelos no Google Cloud Platform (GCP) usando GPUs, realizamos testes de desempenho nos dois mesmos modelos de linguagem: Mistral 7B Instruct v2 e LLama2 13B. Esses testes mediram o tempo de resposta, dadas as configuração do sistema, o tipo de máquina, as especificações da GPU, o número de CPUs, o uso da memória e os custos associados. Os resultados, resumidos na Tabela 2, fornecem uma comparação das métricas de desempenho de cada modelo em diferentes configurações de GCP.

Tabela 2 – Desempenho de modelos implementados no GCP usando GPU.

	Modelo 1: mistral 7B - instruct v2	**Modelo 2: LLama2 13B**
Tempo de resposta [s]	20	20
Sistema operacional	Debian otimizado para aprendizagem profunda com CUDA 121	Sistema operacional otimizado para contêineres com Kubernetes e CUDA 11.4
Tipo da máquina	g2-standard-24	n1-standard-1
Placas de vídeo	2 GPUs NVIDIA L4	1 GPU NVIDIA T4 Virtual Workstation
Número de CPUs	24	24
Memória RAM [GB]	96	3,75
Custo	2,316764	0,055

Fonte: Autor.

Para fazer o download e implantar o LLM, basta executar os seguintes comandos no terminal. Esse código utiliza o Docker para executar um contêiner com suporte a GPU, especificando o *token* Hugging Face, a configuração da porta e os detalhes do modelo para o modelo Mistral 7B Instruct v2, como mostrado na Seção de Código 4.2.

```
docker run --gpus all \
 -e HF_TOKEN={HUGGINFACE_TOKEN} -p 8000:8000 \
 ghcr.io/mistralai/mistral-src/vllm:latest \
 --host 0.0.0.0 \
 --model mistralai/Mistral-7B-Instruct-v0.2
```

Seção de Código 4.2 – Comando para fazer download e implantar o LLM

Ao executar, será liberado um IP na máquina virtual para chamadas de geração de texto, no seguinte formato de endereço

- http://VM_IP:8080/api/chat

Para usar o LLM independentemente da interface, é possível fazer solicitações HTTP para interagir com o modelo. A Seção de Código 4.3 e 4.4 trazem um exemplo de código em Python e seu comando curl correspondente.

```
import requests

url = 'http://<VM_IP>:8000/v1/completions'

headers = {

"Content-Type": "application/json",

"Authorization": "Bearer <HUGGINFACE_TOKEN>"

}

data = {

"model": "mistralai/Mistral-7B-Instruct-v0.2",  # Altere o nome do
   modelo para usar o Llama2

"prompt": "<role_user_content>",

"max_tokens": 150

}

response = requests.post(url, headers=headers, json=data)

resp = response.json()['choices'][0]['text']
```

```
26
27 print(resp)
```

Seção de Código 4.3 – Código em Python para fazer inferência usando a GCP

```
curl -X POST http://<VM_IP>:8000/v1/completions \
     -H "Content-Type: application/json" \
     -H "Authorization: Bearer <HUGGINFACE_TOKEN>" \
     -d '{
           "model": "mistralai/Mistral-7B-Instruct-v0.2",
           "prompt": "<role_user_content>",
           "max_tokens": 150
         }'
```

Seção de Código 4.4 – Código em curl para fazer inferência usando a GCP

O Código em curl realiza a mesma operação, enviando uma solicitação POST para o mesmo *endpoint* com parâmetros de cabeçalho e *payload* similares, utilizando JSON para especificar o modelo, o prompt e o número máximo de tokens. É crucial substituir <role_user_content>, <HUGGINGFACE_TOKEN>, e <VM_IP> pelos valores correspondentes: o conteúdo real das funções do sistema e do usuário, a credencial de autenticação da Hugging Face, e o endereço IP da máquina virtual na GCP. Os modelos mencionados nos exemplos podem ser alterados conforme necessário para utilizar outros modelos disponíveis na plataforma, facilitando a integração e utilização de modelos de linguagem poderosos de forma programática via solicitações HTTP.

Grandes modelos de linguagem, como o Mistral 7B Instruct e o LLama2 13B, são baseados em arquiteturas de transformadores. Eles são treinados com grandes quantidades de texto e possuem bilhões de parâmetros, o que lhes permite gerar textos coerentes e responder a consultas complexas. Por outro lado, o objetivo da solução é utilizar documentos específicos relacionados ao funcionamento dos equipamentos, dessa forma, apesar da grande quantidade de dados utilizados para treinamento desses grandes modelos, eles não possuem capacidade de responder de forma correta a respeito de novas informações. Por isso, os modelos implementados serão utilizados como um dos componentes da técnica chamada de Geração Aumentada de Recuperação.

4.3 CADEIA DE LLMS PARA SUMARIZAÇÃO DE LOGS

Os dados relacionados aos registros de eventos, do inglês logs, ao contrários dos dados dos manuais de usuário e PDFs, são dados estruturados ou, sendo mais especifico, dados tabulares, no qual existe uma relação entre as linhas e as colunas. Se tratarmos esses dados, da mesma forma como fizemos para os dados textuais, essas relações serão perdidas, fazendo com que as informações recuperadas e respostas geradas pelo nosso modelo de linguagem não sejam precisas.

Sendo assim, existe um desafio atrelado à utilização desse tipo de dado como contexto para nosso modelo de linguagem. A dificuldade está em como utilizar dados estruturados como contexto, mantendo a relação e as estruturas entre os dados, de forma que uma LLM possa usar essas informações para fazer inferências corretas.

Tradicionalmente, existem algumas formas para se lidar e manipular dados tabulares, sendo estas utilizando *Python* com auxílio da biblioteca pandas, linguagem R, consultas com SQL, entre outras. Basicamente, essas ferramentas possibilitam que o usuário realize diversos tipos de manipulação de dados, como por exemplo, limpeza, padronização, transformação e filtragem, permitindo diversos tipos de análises para entender as informações contidas nos dados.

RAG com dados tabulares pode não ser a técnica mais eficiente para extrair informação de conjunto de dados estruturados. Quando há a necessidade de interação com dados tabulares utilizando grandes modelos de linguagem, devemos levar em conta os desafios expostos. Sendo assim, existem duas diferentes formas para lidar com dados tabulares. A primeira é utilizar a técnica de RAG, e a segunda é implementar um cadeia de LLMs capaz de fazer consulta nos dados tabulares a partir das perguntas informadas pelo usuário.

Existe uma diferença fundamental na implementação desses dois métodos. Quando implementamos um projeto usando RAG, existe um modelo de *embeddings* no *pipeline* que irá transformar as perguntas do usuário em vetores, que são utilizados para fazer uma busca no banco de dados vetorizado e retornar o conteúdo mais relevante. Esse conteúdo, portanto, é passado para a LLM, juntamente com a pergunta, para gerar o resultado final.

Outra possibilidade, é o uso de uma cadeia de LLMs para reproduzir o que um humano faria para entender as informações contidas nesses dados tabulares, ou seja, manipulação de dados. Preferencialmente, essa cadeia deve ser capaz de receber a pergunta do usuário e, ao invés de transformar em um vetor, transformar a pergunta, utilizando uma LLM, em uma linguagem de consulta compatível com o dado tabular, por exemplo, SQL ou os métodos da biblioteca Pandas, na linguagem Python. Essa consulta é, então, aplicada aos dados e o resultado é usado de entrada para outra LLM, que irá, por fim, gerar a resposta.

Quando utilizamos a abordagem usando RAG, esperamos que o resultado nos forneça um relação semântica entre a pergunta e o conteúdo retornado do banco de dados vetorial, devido ao cálculo realizado para encontrar a similaridade. A cadeia de LLMs tentará traduzir o questionamento em uma consulta, da mesma forma que um profissional ou analista de dados realizaria para encontrar as informações desejadas.

A distinção entre o RAG e as cadeias de LLMs que consultam bancos de dados está no tipo de manipulação de dados e nos recursos de resolução de consultas de cada abordagem. O RAG se baseia na recuperação de documentos ou passagens

relevantes de um corpus pré-indexado e na geração de respostas com base nessas informações. Para responder perguntas que envolvem agregação de dados e cálculo, como por exemplo, a média dos dados, o RAG precisaria acessar registros individuais, o que normalmente não está disponível em uma única passagem ou documento. Isso requer a agregação e o cálculo de dados de vários registros, o que está além do escopo de uma simples recuperação.

Para responder perguntas desse tipo, o sistema precisa executar uma operação semelhante a um banco de dados: selecionar linhas e calcular a média de uma coluna específica. Isso requer a execução de uma consulta sobre dados estruturados, o que normalmente é feito por meio de consultas a um banco de dados.

Assim, as cadeias de LLMs podem ser vistas como um auxiliar ao RAG, oferecendo funcionalidades adicionais para lidar com dados estruturados e consultas a bancos de dados, permitindo a construção de soluções mais robustas a diferentes domínios e capazes de lidar com uma maior variedade de situações.

4.3.1 Carregamento dos *Logs*

Para carregar os dados dos registros não é possível realizar a mesma abordagem utilizada para carregamentos dos manuais dos equipamentos. O motivo está relacionado ao tamanho do contexto que podemos usar de entrada para os grandes modelos de linguagem, que atualmente ainda é limitado. Dessa forma, não é possível carregar toda a tabela para gerar uma resposta. Uma alternativa seria dividir essa tabela em *chunks*, porém isso afetaria o relacionamento existente em dados tabulares entre linhas e colunas, como mencionado anteriormente.

Na primeira abordagem, utilizaremos o método tradicional, empregando Python e a biblioteca pandas. Isso permite com que façamos limpeza e padronização dos dados, antes de utilizar esses dados como entrada para realizar perguntas.

4.3.2 Fluxo para Sumarização dos *Logs*

O fluxo de execução para trabalhar com o *log* dos equipamentos envolve a utilização de uma cadeia de LLMs, que realizam tarefas complexas, indo além de apenas responder perguntas do usuário. Esses modelos recebem um questionamento inicial, como "Quais eventos estão acontecendo e por que eles acontecem?", e, ao invés de consumir todos os dados tabulares na tentativa de gerar uma resposta válida, os LLMs traduzem a linguagem humana em uma linguagem consultiva. Essa linguagem consultiva é, então, aplicada aos dados tabulares e o resultado dessa consulta é utilizado para gerar a resposta desejada.

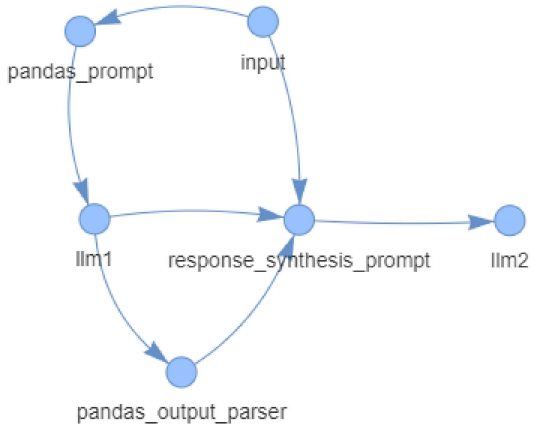
Nessa abordagem não é necessário realizar o *embedding* dos dados, já que o *embedding* é usado para encontrar os documentos mais relevantes através de uma busca semântica. Nesse caso, a pergunta do usuário é transformada em uma consulta

do pandas, trazendo a informação relevante para gerar a resposta da LLM. Assim, a tabela inicial é filtrada com a consulta de acordo com a pergunta e retorna apenas o que for relevante.

Com auxílio do LlamaIndex e sua documentação, é simples orquestrar um fluxo que realize o encadeamento de módulos com diferentes funcionalidades com o uso da abstração *QueryPipeline*. Para criar o fluxo desejado é possível codificar da forma como aconselhado na documentação da ferramenta, disponível em (LLAMAINDEX, 2024a). Primeiramente, com os dados já baixados, pode-se definir os módulos seguintes. São eles a geração de instruções, onde a pergunta é convertida em instruções Pandas usando LLMs, a execução de instruções no *DataFrame* do Pandas, a síntese de resposta e o retorno de resposta ao usuário final.

A Figura 8 apresenta o diagrama do fluxo construído usando o *QueryPipeline* para processar consultas. Ele começa com um nó de *"input"* que recebe a consulta, a qual é então processada pelo "llm1" para entendimento inicial da linguagem. Os dados passam para o nó "pandas_prompt" para operações específicas do pandas e filtrar os dados dos *logs* para a consulta desejada, a saída é então analisada pelo "pandas_output_parser". O nó "response_synthesis_prompt" adiciona mais intruções para geração da resposta final através do "llm2". Essa abordagem modular é ideal para processamento complexo de tarefas que podem ser resolvidas com LLM.

Figura 8 – Fluxo para sumarização dos *logs*.



Fonte: Autor.

Seguindo este fluxo, a partir de um *input* inicial do usuário, temos o processamento de cada um dos módulos, representados por nós na Figura 8. Tal processamento tem suas ações descritas em saídas internas do sistema, como apresentado em sequência:

```
> Running module input with input:
```

```
query_str: Can you explain which events are happening? For
the events that you found, can you explain what the causes are?

 > Running module pandas_prompt with input:
query_str: Can you explain which events are happening? For
the events that you found, can you explain what the causes are?

 > Running module llm1 with input:
messages: You are working with a pandas dataframe in Python.
The name of the dataframe is `df`.
This is the result of `print(df.head())`:
    FORMAT    SHORT_EVENT  Event Number    Date/Time ...

 > Running module pandas_output_parser with input:
input: assistant: df.columns

 > Running module response_synthesis_prompt with input:
query_str: Can you explain which events are happening? For
the events that you found, can you explain what the causes are?
pandas_instructions: assistant: df.columns
Type ,EVENT_CLASS,Event Number,Date/Time,Identification,Event

pandas_output: Index(['Type', 'EVENT_CLASS', 'Event Number',
'Date/Time', 'Identification', 'Event'], dtype='object')

 > Running module llm2 with input:
messages: Given an input question, synthesize a response from
the query results.
Query: Can you explain which events are happening? For the
events that you found, can you explain what the causes are?
```

A resposta de todo o processamento, a partir da pergunta *"Can you explain which events are happening? For the events that you found, can you explain what the causes are?"* é apresentado em sequência. A resposta fornece uma sumarização dos eventos e suas causas, juntamente com comentários relevantes.

```
Based on the query results, several events are happening,
and I'll break them down for you:
```

```
**Events:**

1. **Overcurrent**: This event is occurring in multiple phases,
specifically Phase A, Phase B, and Phase C.
2. **Underfrequency**: The generator output frequency is below
the threshold.
3. **Overvoltage**: This event is happening in multiple phases,
specifically Phase A, Phase B, and Phase C.
4. **Undervoltage**: This event is occurring in multiple phases,
specifically Phase A, Phase B, and Phase C.
5. **Overfrequency**: The system frequency is above the threshold.

**Causes:**

1. **Overcurrent**: The cause is attributed to overcurrent in
specific phases (A, B, or C).
2. **Underfrequency**: The cause is due to the generator output
frequency being below the threshold.
3. **Overvoltage**: The cause is attributed to overvoltage in
specific phases (A, B, or C).
4. **Undervoltage**: The cause is due to undervoltage in specific
phases (A, B, or C).
5. **Overfrequency**: The cause is due to the system frequency
being above the threshold.

It's worth noting that these events are recurring, with multiple
instances of each event occurring in the data.
```

Para que o cliente possa validar a solução é necessário desenvolver uma interface que realizará as funções necessárias para solicitar um questionamento e entregar a resposta. O resto será executado internamente à solução desenvolvida.

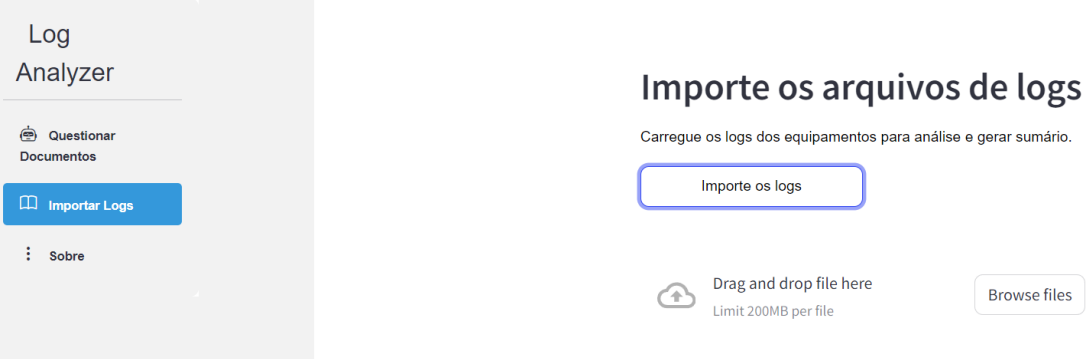
4.4 INTERFACE DE USUÁRIO

A interface da solução será desenvolvida usando o Streamlit, um *framework* em Python usado para desenvolver aplicativos interativos da Web de fácil implementação, indicado para validação de ideias e desenvolvimento de POCs.

Nessa versão inicial, é possível interagir com uma página chamada “Importar *Logs*”, como apresentado na Figura 9. A partir da interface, é possível inserir um

arquivo com registro de eventos e uma pergunta e, então, receber a resposta de forma automatizada, ou seja, a sumarização dos eventos.

Figura 9 – Interface para envio de arquivos de *Logs*.



Fonte: Autor

Ao selecionar um arquivo de *log* no formato CSV, é possível visualizar os dados diretamente na interface do usuário. Os códigos implementados internamente são ativados para gerar a sumarização automática do *log*. conforme mostrado na Figura 10. Podemos ver que o sistema carregou um arquivo de *log* com colunas que incluem "*Event Number*", "*Date/Time*", "*Identification*"e "*Event*". Após o carregamento, o sistema inicia a geração da sumarização automaticamente.

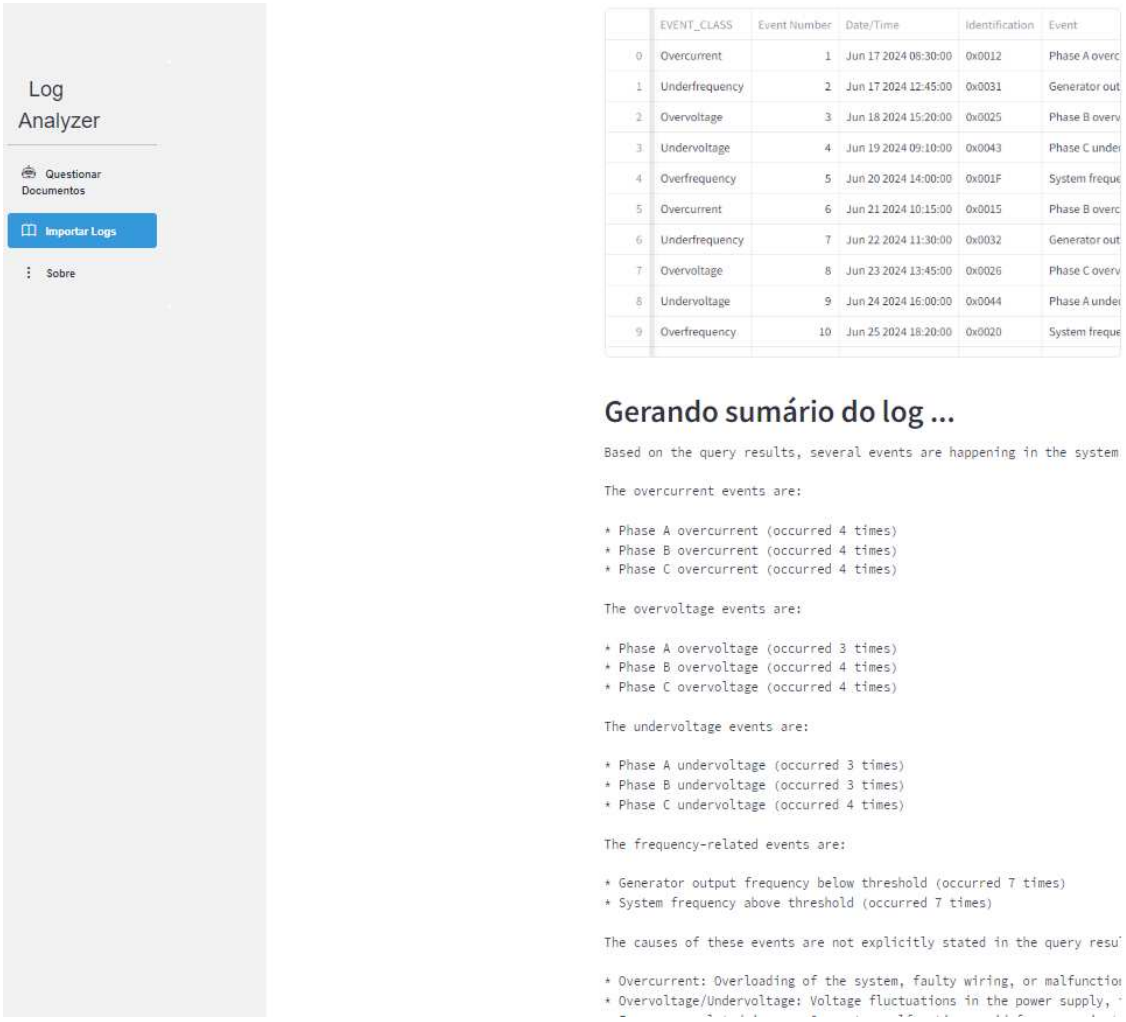
A sumarização inclui a análise dos eventos por data e tipo de causa, como ilustrado na imagem, onde é apresentado um resumo das ocorrências de diferentes eventos e causas ao longo do tempo. Por exemplo, podemos ver que eventos como "*Overcurrent*"e "*Underfrequency*"são listados com suas respectivas datas e identificação. Isso permite uma análise rápida dos padrões de eventos registrados no *log*.

4.5 RAG COM MANUAIS DE EQUIPAMENTOS

Esta seção descreve detalhadamente a implementação da arquitetura do RAG para manuais de equipamentos. Para que um grande modelo de linguagem possa responder questionamentos relacionados aos manuais dos equipamentos, os documentos servem como fonte de conhecimento, ou contexto, para a geração de respostas. Essa técnica complementa os grandes modelos de linguagem com o uso de mecanismos de recuperação para encontrar informações relevantes de fontes de dados adicionais, permitindo, assim, que modelos pré-treinados possam ser adequados para utilização em domínios específicos, até mesmo podendo utilizar informações sensíveis e dados corporativos sem que haja comprometimento e exposição dos dados.

A arquitetura por trás do funcionamento do RAG envolve alguns processos, sendo eles o entendimento do questionamento, a recuperação dos documentos e a geração da resposta. Para que todas as etapas do RAG sejam executadas, foi utilizado

Figura 10 – Geração automática do sumário dos *logs.*



Fonte: Autor.

uma solução de código aberto chamada LlamaIndex. Essa ferramenta fornece métodos e abstrações que permitem orquestrar e integrar diferentes módulos, como LLMs, modelos de incorporação, banco de dados vetoriais, entre outros. É possível, assim, que apenas uma única ferramenta seja utilizada para construir aplicações completas com LLM de ponta a ponta, incluindo RAG.

Ao utilizar RAG introduzimos ao LLM a possibilidade de interagir com uma base de de conhecimento. Assim, toda vez que a LLM recebe uma entrada (*input*), a base de dados é consultada para retornar informações relevantes e, então, combinada juntamente com a entrada do usuário para, então, entrar no LLM e gerar uma resposta, com muito mais contexto e reduzindo as chances de alucinação.

A Figura 11 ilustra cada etapa da técnica do RAG, desde o carregamento de documentos até a geração de respostas. A primeira etapa envolve o carregamento dos documentos utilizando um “Carregador de documentos”, especificamente um *Simple-*

DirectoryReader, que facilita a leitura dos arquivos de documentos armazenados. Esta etapa prepara os documentos para o processamento subsequente e está diretamente ligada à próxima fase, que é a segmentação do texto.

Na fase de “*Chunking*”, os textos são divididos em pedaços menores utilizando um “*SentenceSplitter*”. Isso é crucial porque os modelos de linguagem têm limitações quanto ao número de tokens que podem ser processados de uma vez. Segmentar os textos em *chunks* gerenciáveis permite um processamento mais eficiente nas etapas subsequentes.

Os *chunks* são então transformados em representações vetoriais através do processo de “*Embeddings*” utilizando "*HuggingFaceEmbeddings*". Esta transformação é fundamental para capturar a semântica do texto de forma que possa ser comparável numericamente. Os vetores resultantes são enviados para a próxima etapa de "Indexação".

Durante a "Indexação", esses vetores são armazenados em um "Banco de dados vetorial", aqui identificado como Chroma, que organiza e permite a recuperação eficiente desses dados. Este banco de dados forma a espinha dorsal do sistema de recuperação de informações, permitindo que respostas relevantes sejam geradas rapidamente baseadas em consultas de usuários.

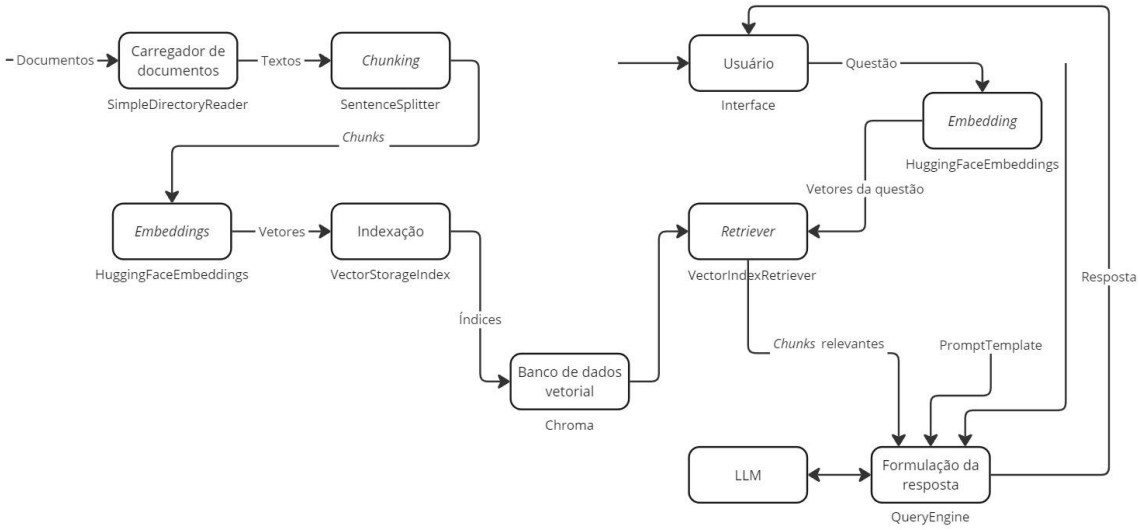
Quando um usuário submete uma pergunta através da "Interface de usuário", a consulta é transformada em vetores pelo mesmo processo de "*Embedding*"que foi usado para os documentos. Esta etapa garante que a consulta do usuário possa ser comparada diretamente com os dados indexados no banco de dados vetorial.

Os vetores de consulta são então utilizados para recuperar os *chunks* relevantes através de um "*Retriever*", que busca correspondências no banco de dados vetorial. Este é um ponto crítico no processo, pois determina quais informações dos documentos são mais relevantes para a consulta em questão.

Os *chunks* recuperados, juntamente com a pergunta inicial do usuário, são então processados pelo "*Prompt Template*". Este template organiza os dados de forma que possam ser eficazmente utilizados pelo LLM. Essencialmente, o "*Prompt Template*"integra as instruções do *prompt* com os *chunks* de contexto recuperados e a pergunta inicial do usuário, formando uma entrada coerente que é então fornecida ao LLM.

Finalmente, o LLM, empregando a entrada organizada pelo "*Prompt Template*", formula uma resposta através de um "*Query Engine*". Este motor utiliza o modelo de linguagem com os dados preparados para produzir uma resposta que não apenas é relevante, mas também informativa, baseando-se nas informações extraídas e no conhecimento embutido no modelo.

Figura 11 – Fluxograma do RAG com documentos.



Fonte: Autor.

4.5.1 Processamento dos Documentos

No RAG, tudo se inicia com os documentos externos que desejamos utilizar no fluxo. Para isso, alguns passos devem ser executados para carregar e adequar os documentos para uso. Existem diversas maneiras de executar esses passos, permitindo com que possamos testar diferentes maneiras e avaliar como elas se comportam ao realizar a recuperação de documentos. Esses passos são carregar os documentos e converter para texto, dividir textos em blocos, incorporar os textos em vetores e criar uma banco de dados vetorial. Esse processo de adequação e conversão dos documentos pode ser chamado de indexação.

4.5.1.1 Extração dos Textos dos Documentos

A primeira etapa do processo de *Retrieval Augmented Generation* inicia com o carregamento e extração dos textos que serão utilizados como contexto para resposta da LLM. Como já mencionado, para o RAG serão utilizados apenas conteúdos relacionados ao funcionamento do equipamento. No site do cliente são disponibilizados diversos recursos relacionados a diferentes tipos de equipamentos.

Para a POC, como relatado no Capítulo 3, foi determinado a utilização de apenas uma família de equipamentos, os relês digitais. Como a solução deve ser escalável para outros tipos de equipamentos, apenas com a adição de novos conteúdos como contexto do RAG, não é relevante se aprofundar no funcionamento desse equipamento, pois nesse caso estaríamos desenvolvendo uma solução especialista. Os únicos requisitos necessários são que o equipamento possua uma ampla documentação descrevendo seu funcionamento e seja capaz de gerar *logs* de eventos para as próximas

etapas de desenvolvimento da solução.

Foram coletados do site do cliente catálogos, manuais, instruções de configuração, notas de atualização, comunicados, alertas de segurança e casos de suporte resolvidos no passado. Todos esses documentos são arquivos do tipo PDF, porém, caso não fossem, o LlamaIndex possui um centro de integrações com métodos para carregamento de arquivos de mais diversos formatos e conexões.

Esses arquivos foram armazenados em pastas no computador local para cada uma das categorias de documentos. Para a leitura e conversão dos documentos em texto, foi utilizado um conector do LlamaIndex chamado *SimpleDirectoryReader*. Ao informar o caminho do diretório onde os arquivos estão armazenados, esse método percorre todas as pastas e subpastas, fazendo a leitura do conteúdo de todos os documentos encontrados.

Esses conteúdos, então, são retornados como um objeto *Document* para cada página do arquivo original, ou arquivo pai. Esse objeto inclui diversas informações sobre o documento carregado, os metadados, além do texto extraído. O metadado inclui o nome do arquivo original, a página, o caminho do diretório, a data de criação, entre outras informações que são extraídas por padrão, como mostrado na Seção de Código 4.5 Também é possível adicionar outros tipos de metadados de forma customizada.

```
{'page_label': '1',
 'file_name': 'support_cases.pdf',
 'file_path': 'c:\\Users\\..\\document_folder\\support_cases.pdf',
 'file_type': 'application/pdf',
 'file_size': 2494386,
 'creation_date': '2024-04-09',
 'last_modified_date': '2024-05-10'}
```

Seção de Código 4.5 – Metadados de um *Document*

Com o carregamento de 46 documentos, nossa coleção resultou em um total de 673 objetos do tipo *Document*. Isso se dá por conta da quantidade de páginas dentro de cada documento.

A etapa de ingestão de dados é importante para garantir que todos os documentos sejam agregados em um único local. Para tornar os textos aptos para serem utilizados nos processamentos posteriores, ainda é preciso decidir entre usar o texto inteiro de cada página ou dividir-lo em blocos segmentados com tamanho pré definido. Essa decisão leva em conta o tamanho dos documentos usados. Quando os documentos são curtos e possuem apenas alguns trechos, podemos seguir utilizando os textos originais. Caso contrário, se temos textos longos, é desejável quebrar esses textos em pedaços menores. É importante mencionar que o tamanho de um texto, quando lidamos com modelos de linguagem, é determinado pelo número de *tokens* contidos no texto e não pelo número de palavras.

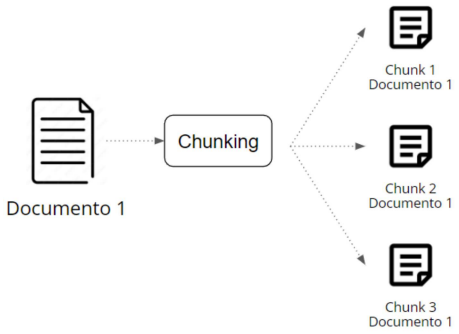
Os documentos usados são documentos longos e com grande quantidade de páginas. Como exemplo, um documento de 9 páginas possui um objeto associado a cada página. Além disso, dada a natureza dos documentos de expor informações, configurações e instruções dos equipamentos, diversas informações diferentes podem estar presentes em um curto trecho. Sendo assim, dividir os textos em blocos menores pode ser benéfico para quando a etapa de recuperação de documentos for realizada.

4.5.1.2 Divisão dos Documentos - *Chunking*

No *framework* do LlamaIndex, assim como temos o objeto *Documents* para representar as informações de um documento, temos o objeto *Node* para representar os *chunks*, ou pedaços de texto de um documento. Ao realizar a a divisão do texto dos documentos, todos os*Nodes* derivados de um *Document* irão herdar os metadados associados ao *Document* utilizado.

A segmentação do conteúdo de um documento em pedaços menores, como mostrado na Figura 12, faz com que o documento seja transformado em pedaços mais controláveis, que possam ser processados e recuperados de uma forma mais eficiente pelos modelos que serão utilizados.

Figura 12 – Representação da segmentação do texto de documentos em trechos menores.



Fonte: Autor.

Antes de realizar esse processo, existem alguns aspectos que devem ser considerados. O *chunking* do texto é uma etapa crucial no *pipeline* do RAG, já que pode influenciar se a LLM conseguirá encontrar ou não a informação correta dada uma pergunta do usuário final. Se o tamanho do *chunk* utilizado para segmentar o texto for muito grande, é possível que informações sejam diluídas e perca a especificidade semântica do trecho. Por outro lado, caso o tamanho escolhido seja pequeno, a divisão pode acabar quebrando uma informação em diferentes *chunks*, o que fará com que o trecho relevante seja encontrado com facilidade, porém com profundidade de informação limitada.

A escolha do tamanho dos *chunks* dos *Nodes* afeta diretamente a qualidade da resposta. Um documento que foi dividido de forma eficaz, ao ser recuperado, pode conter as informações necessárias para a geração de uma resposta completa e coerente, que contém as informações corretas para o usuário final. Isso acontece sem que nenhum contexto tenha ficado perdido no processo de recuperação.

Em outras palavras, ao realizar essa técnica buscamos evitar, posteriormente, na etapa de recuperação de informações, um fenômeno chamado de *"Lost in the middle"*, ocasionado ao utilizar grandes contextos. Asim como humanos, os grandes modelos de linguagem tendem a focar mais no início e no fim de sentenças, possivelmente perdendo informações inseridas no meio dos textos (GAO *et al.*, 2024).

Além das implicações semânticas que podem ser causadas ao selecionar um tamanho adequado, os grandes modelos de linguagem e os modelos de *embeddings* possuem limitações no tamanho dos *tokens* que podem ser usados como entrada. Por isso, é necessário realizar as adequações no tamanho do texto dos documentos.

Para implementar o processo de *chunking* em nossos documentos, continuamos a utilizar o LlamaIndex, através da classe *SentenceSplitter*. Esse separador busca, prioritariamente, manter as sentenças juntas, de forma a evitar que frases sejam divididas no meio, ficando isoladas do contexto original. Ainda, é definido o parâmetro *"chunk_size"*, limitando o tamanho máximo de cada *chunk*. Para testes iniciais o tamanho máximo definido foi de 512 *tokens*.

Definidas a estratégias de divisão dos textos em *chunks*, o método *get_ nodes_ from_ documents* é utilizado, passando nossa coleção de *Documents* como argumento. Para cada documento o texto será dividido em *chunks* usando a estratégia do *SentenceSplitter* e para cada *chunks* resultante dessa segmentação um nodo é gerado. Além desse novo objeto do tipo *TextNode* herdar os metadados dos *Documents*, ele mantém informações do relacionamento com o *chunk* anterior, posterior e o documento parente. Na Seção de Código 4.6 podemos observar o resultado desse processamento. Dos 673 *Documents* que tínhamos inicialmente, foram criados 1542 objetos do tipo *TextNode*. Na Seção de Código 4.7 é possível verificar os metadados de um exemplo de nodo, ao fim dessa etapa.

```
1 [TextNode(id_='e2ec3928-a9fa-466c-8eba-f1adc8508004', embedding=None,
    metadata={'page_label': '1', 'file_name': 'support_cases.pdf', '
    file_path':...
2  TextNode(id_='6804116f-140b-4df4-a282-e5535aeacc16', embedding=None,
    metadata={'page_label': '1', 'file_name': 'support_cases.pdf', '
    file_path':...
3  TextNode(id_='e52ab061-19c7-4e48-bbae-02bad30ee70d', embedding=None,
    metadata={'page_label': '2', 'file_name': 'support_cases.pdf', '
    file_path':...
4  TextNode(id_='a47037b6-66be-4458-a8a2-f93ecce40f44', embedding=None,
    metadata={'page_label': '2', 'file_name': 'support_cases.pdf', '
```

```
    file_path':...
 TextNode(id_='b1b65ddb-d007-478c-832b-00897738a0e4', embedding=None,
    metadata={'page_label': '3', 'file_name': 'support_cases.pdf', '
    file_path':...

...
 TextNode(id_='3212c935-0693-441b-873c-b8b2284107c2', embedding=None,
    metadata={'page_label': '275', 'file_name': 'manuals.pdf', 'file_path
    ':...
]
```

Seção de Código 4.6 – Resultado do processamento dos nodos

```
{'id_': '63ac0ad7-09fb-4c59-8d57-737321160786',
 'embedding': None,
 'metadata': {'page_label': '2',
  'file_name': 'support_cases.pdf',
  'file_path': '..\\document_folder\\support_cases.pdf',
  'file_type': 'application/pdf',
  'file_size': 2494386,
  'creation_date': '2024-04-09',
  'last_modified_date': '2024-05-10'},
 'excluded_embed_metadata_keys': ['file_name',
  'file_type',
  'file_size',
  'creation_date',
  'last_modified_date',
  'last_accessed_date'],
 'excluded_llm_metadata_keys': ['file_name',
  'file_type',
  'file_size',
  'creation_date',
  'last_modified_date',
  'last_accessed_date'],
 'relationships': {<NodeRelationship.SOURCE: '1'>: RelatedNodeInfo(
    node_id='b8f057a5-4f91-4e19-bd76-9168357fbadd', node_type=<ObjectType
    .DOCUMENT: '4'>, metadata={'page_label': '2', 'file_name': '
    support_cases.pdf', 'file_path': '..\\document_folder\\support_cases.
    pdf', 'file_type': 'application/pdf', 'file_size': 2494386, '
    creation_date': '2024-04-09', 'last_modified_date': '2024-05-10'},
    hash='
    f8cb7c9338c93dc8c2a546bea943dca76627c432722fb40694070d9a9d876cdb'),
  <NodeRelationship.PREVIOUS: '2'>: RelatedNodeInfo(node_id='3ba4f799-7
    e29-4e1c-b572-2c52c2e71146', node_type=<ObjectType.TEXT: '1'>,
    metadata={'page_label': '2', 'file_name': 'support_cases.pdf', '
    file_path': '..\\brochures\\document_folder\\support_cases.pdf', '
    file_type': 'application/pdf', 'file_size': 2494386, 'creation_date':
     '2024-04-09', 'last_modified_date': '2024-05-10'}, hash='5
```

```
    b954fd71e712eba7f844354f2cac0fa07dcbdca1fef212102924f90c7baf8aa'),
24  <NodeRelationship.NEXT: '3'>: RelatedNodeInfo(node_id='dfa5b7c0-f563
    -4005-83dc-904a9583ec5d', node_type=<ObjectType.TEXT: '1'>, metadata
    ={}, hash='5
    fb33b68592babfceb606218e8b515f5db2c3a191128c500f636986ee9e1dc41')},
25 'text': 'Protection procedure...',
26 'start_char_idx': 1490,
27 'end_char_idx': 3467,
28 'text_template': '{metadata_str}\n\n{content}',
29 'metadata_template': '{key}: {value}',
30 'metadata_seperator': '\n'}
```

Seção de Código 4.7 – Metadados de um exemplo de nodo

4.5.1.3 **Incorporação de Textos - *Embeddings***

Após a segmentação dos textos em *chunks*, a próxima etapa de processamento dos documentos é a incorporação das sentenças em uma representação numérica, como explicado na Seção 2.4 .

Para escolha de qual modelo de *embedding* de texto utilizar no desenvolvimento, foi consultado a plataforma *Massive Text Embedding Benchmark (MTEB) Leaderboard*, que consiste em um ranqueamento dos modelos com resultado de 8 tarefas realizadas em 58 conjunto de dados e 112 idiomas (MUENNIGHOFF *et al.*, 2022).

Na data de desenvolvimento do projeto, optou-se por utilizar o modelo de *embedding* “BAAI/bge-base-en-v1.5”, cujo desempenho apresentava um bom equilíbrio entre tamanho do modelo e posição no ranqueamento. Pelo fato de ser leve, esse modelo pode ser utilizado com boa performance em uma máquina local. O modelo é de código aberto e pode ser acessado através da plataforma do Hugging Face.

Para utilização do modelo é possível utilizar a biblioteca da própria Hugging Face ou os métodos de integração do LlamaIndex, como mostrado na Seção de Código 4.8.

```
1 from llama_index.embeddings.huggingface import HuggingFaceEmbedding
2
3 embed_model_name = "BAAI/bge-base-en-v1.5"
4
5 embed_model = HuggingFaceEmbedding(model_name=embed_model_name)
```

Seção de Código 4.8 – Código para importar o modelo de *embedding*

Essa incorporação é responsável pela conversão dos *chunks* em vetores numéricos de alta dimensionalidade. Tal representação é importante para que a recuperação de documentos e busca semântica seja realizada com sucesso.

4.5.1.4 **Indexação**

A indexação é uma técnica fundamental na gestão de dados, projetada para otimizar a recuperação de informações em grandes conjuntos de dados. Um índice em sistemas de dados permite localizar dados específicos rapidamente, sem a necessidade de escanear todo o conjunto de dados. Bancos de dados tradicionais utilizam índices escalares que dependem de correspondências exatas das entradas de dados. No entanto, com o aumento da complexidade dos tipos de dados e a necessidade de buscas mais sutis, a indexação vetorial surgiu como uma alternativa poderosa.

A indexação vetorial é um método usado para manipular e recuperar dados vetoriais de alta dimensão. Nesta abordagem, os dados são representados como vetores em um espaço multidimensional. Esses vetores encapsulam várias características dos dados, permitindo realizar buscas por similaridade com base nessas características. Diferente dos dados escalares, que possuem um único valor, os dados vetoriais compreendem múltiplos valores, permitindo buscas complexas e levando em conta o valor semântico e o contexto.

Com a preparação dos dados e a geração de *embeddings* já concluídas, resta apenas a etapa de construção dos índices no processo de indexação vetorial. As representações vetoriais, ou *embeddings*, são organizadas em um índice. Esse índice é estruturado para facilitar a recuperação rápida de vetores similares, permitindo operações de busca eficientes. Durante uma consulta de busca, o índice é utilizado para recuperar os vetores que são mais similares ao vetor de consulta. A similaridade é determinada usando a métrica de similaridade predefinida.

O LlamaIndex aproveita o poder da indexação vetorial para fornecer capacidades avançadas de busca dentro de grandes textos. O *VectorStoreIndex* é um tipo específico de índice usado pelo LlamaIndex, projetado para lidar com representações vetoriais de texto para recuperação eficiente por busca semântica. Esse método armazena cada *Node* e seu *embedding* correspondente no índice.

Para buscar e recuperar dados de um conjunto de *embeddings*, é necessário definir uma medida de similaridade. Uma métrica de similaridade comum em buscas vetoriais é a similaridade de cossenos, que mede o ângulo entre dois vetores. Os valores variam de -1 a 1, sendo 1 para os vetores que apontam na mesma direção, -1 para vetores que apontam em direções opostas e 0 para vetores ortogonais.

O objetivo principal da indexação vetorial é possibilitar buscas rápidas por similaridade em grandes conjuntos de dados. No entanto, o padrão do *VectorStorageIndex* é bastante simples. Com ele, todo índice é mantido na memória e a recuperação é feita comparando o vetor de consulta com cada vetor de nodo e calculando a similaridade cosseno. Esse método pode se tornar ineficiente à medida que o tamanho do índice aumenta. Para utilizar índices maiores de forma eficiente, iremos utilizar bancos de dados especializados em vetores que funcionam utilizando algoritmos mais otimizados,

capazes de realizar buscas mais rápidas e com menor uso de memória.

A indexação vetorial representa um avanço significativo na recuperação de informações, oferecendo uma ferramenta poderosa para gerenciar e buscar grandes conjuntos de dados. O LlamaIndex exemplifica a aplicação prática desta tecnologia, proporcionando capacidades de busca eficientes e ricas em contexto. À medida que os dados continuam a crescer em complexidade e volume, a importância de técnicas avançadas de indexação, como a indexação vetorial, só aumentará, tornando-se um componente essencial dos sistemas modernos de gestão de dados. Para gerenciar índices em grande escala, é essencial utilizar bancos de dados especializados em vetores, como o ChromaDB, que oferecem soluções robustas para armazenar, indexar e consultar grandes volumes de dados vetoriais.

4.5.2 Banco de Dados de Vetores

Como mencionado, banco de dados tradicionais não são eficientes nem possuem propriedades adequadas para lidar com vetores de alta dimensionalidade. Os banco de dados especializados para armazenar vetores extraídos dos *embeddings* são conhecidos como banco de dados de vetores.

O LlamaIndex possui uma vasta prateleira com diferentes opções de banco de dados de vetores com suporte para integração. Como a opção oferecida como método padrão do *framework* não oferece desempenho que permita a escalabilidade futura da solução, foi optado por utilizar a ferramenta ChromaDB para realizar o armazenamento e indexação dos documentos. Essa ferramenta permite que o banco de dados seja armazenado localmente, de forma que não haja necessidade de enviar informações para servidores terceirizados, além de ser uma ferramenta de código aberto.

Os bancos de dados vetoriais, como o ChromaDB, oferecem uma solução robusta para armazenar, indexar e consultar grandes volumes de dados vetoriais, potencializando ainda mais a eficiência e a capacidade de resposta dos sistemas de busca semântica.

O ChromaDB utiliza um algoritmo chamado de *Hierarchical Navigable Small World* (HNSW). De forma geral, esse algoritmo utiliza conceitos de estrutura de grafos e camadas para entender e organizar as conexões entre nodos de camadas superiores e inferiores. Ao manter esses relacionamentos quando for realizada uma busca, esse algoritmo consegue, de forma eficaz, encontrar os vetores mais próximos do vetor utilizado como entrada.

Dessa forma, banco de dados vetoriais facilitam o desenvolvimento de soluções que utilizam *embeddings* extraídos de uma grande coleção de documentos. Ao realizar uma busca, a consulta também será vetorizada utilizando o mesmo modelo de *embedding*, de forma que a similaridade semântica entre entrada e vetores no banco de dados seja reconhecida, para então retornar o vetor e, portanto, o documento mais

similar.

Simplificando o funcionando do ChromaDB, para cada node será criado um índice com HNSW. Esse índice contém os metadados do nodo, incluindo o texto, e o *embedding* respectivo. O conjunto de todos os nodos armazenados em um banco de dados vetorial configura uma coleção, que pode ser salva localmente como um arquivo de banco de dados do tipo SQLite.

O LlamaIndex permite que todo o processamento dos documentos sejam realizados em poucas linhas de código, como vemos na Seção de Código 4.9. No código, os documentos são convertidos em vetores e armazenados. Ao definir um cliente persistente, fazemos com que a coleção seja salva para uso futuro. O modelo de *embedding* traduz os documentos para um formato adequado para armazenamento e busca de vetores, como explicado nas Seções 2.4 e 4.5.1.3.

```
1
2 # Cria um cliente Chroma, que sera salvo no caminho definido
3 db = chromadb.PersistentClient(path="./chroma_db")
4 # Checa se uma colecao ja existe, caso contrario cria uma nova para
    armazenar os vetores
5 chroma_collection = db.get_or_create_collection("colecao_manuais")
6 # Cria a vector store, utilizando a colecao criada, para lidar com o
    armazenamento e requisi oes de consultas
7 vector_store = ChromaVectorStore(chroma_collection=chroma_collection)
8
9 # Gerenciador das configura  es para interagir com o vector store
    definido
10 storage_context = StorageContext.from_defaults(vector_store=vector_store
    )
11
12 # Carrega o modelo de embedding usado para vetorizar os chunks de
    documentos
13 embed_model = HuggingFaceEmbedding(model_name="BAAI/bge-base-en-v1.5")
14
15 # Metodo para criar o indice usando os
16 index = VectorStoreIndex(
17         base_nodes, storage_context=storage_context, embed_model=
    embed_model, show_progress=True
18     )
19 )
```

Seção de Código 4.9 – Código para executar o processamento de documento e criar um banco de dados vetorial

O código configura um fluxo para armazenar e gerenciar vetores de alta dimensão usando o Chroma. É inicializado um cliente persistente que salva os dados no disco, garantindo a persistência. O código cria um armazenamento de vetores vinculado à coleção "colecao_manuais"e configura o gerenciamento das interações com

esse armazenamento.

Além disso, o código configura um modelo de *embedding* do Hugging Face chamado "BAAI/bge-base-en-v1.5", que converte nodos em vetores. Finalmente, como apresentado na Seção de Código 4.10, o código cria um índice dos documentos, convertendo-os em vetores usando o modelo de *embedding* e armazenando-os no banco de dados de vetores.

```
INFO - Load pretrained SentenceTransformer: BAAI/bge-base-en-v1.5
INFO - Loaded '673' documents
INFO - 673' documents turned into '1542'
INFO - index creating with `1542` chunks                https://docs.
   trychroma.com/telemetry for more information.
Batches: 100% |||||||||| 1/1 [00:16<00:00, 16.05s/it]?, ?it/s]
...
...
Batches: 100% |||||||||| 1/1 [00:14<00:00, 14.33s/it]57<00:17,  1.47s/it
   ]
Batches: 100% |||||||||| 1/1 [00:02<00:00,  2.84s/it]11<00:02,  1.46s/it
   ]
Generating embeddings: 100% |||||||||| 1542/1542 [38:14<00:00,  1.49s/it
   ]
INFO - Vector Store created
```

Seção de Código 4.10 – Código para executar o processamento de documento e criar um banco de dados vetorial

O tempo de criação de dois bancos vetoriais em máquina local foi comparado, sendo um utilizando *chunks* de tamanho de 512 *tokens* e outro sem a segmentação dos textos do documentos, ou seja, utilizando o texto completo das páginas. O primeiro banco de dados vetorial foi criado em 39 minutos e 30 segundos, já o segundo, apesar de ter textos maiores, foi criado em 18 minutos e 31 segundos. Isso acontece pois com menores *chunks* são criados mais índices de nodos que devem ser indexados.

4.5.3 RECUPERAÇÃO DE DOCUMENTOS

Com o índices criados, é possível aplicar consultas para recuperar documentos relevantes para a entrada executada. Quando o usuário executa a consulta usando um índice criado com LlamaIndex, o mesmo modelo de *embedding* é aplicado para converter o texto em um vetor numérico, para então calcular a similaridade entre a consulta e os nodos armazenados no banco de dados vetorial. O índice, então, recupera os “K” nodos com maior pontuação de similaridade, sendo “K” definido de acordo com o propósito da recuperação. Caso esses nodos sejam posteriormente utilizados para a síntese de respostas com uma LLM, a soma do tamanho de *tokens* dos “K” nodos não podem ultrapassar o tamanho limite de *tokens* de entrada do grande modelo de linguagem utilizado, portanto “K” deve ser adequado para essa utilização.

Cada modelo de linguagem possui um tamanho de contexto de entrada definido em seu treinamento.

O método de recuperação dos “K” documentos mais relevantes é comumente conhecido como *Top K Retrieval*. Podemos definir um método para recuperar os 3 primeiros documentos com o índice criado anteriormente usando a classe do LlamaIndex *VectorIndexRetriever*, definindo da forma como mostrado na Seção de Código 4.11

```
retriever = VectorIndexRetriever(
    index=index,
    similarity_top_k=3,
)
```

Seção de Código 4.11 – Código para executar o processamento de documento e criar um banco de dados vetorial

No exemplo de Seção de Código 4.12, executamos uma consulta para retornar os *chunks* que possuam informações sobre “Como identificar um problema em um relê?”. Como definimos nosso método para retornar apenas os 3 primeiros documentos, nosso método *retriever* deve retornar os 3 documentos com mais pontuação de relevância. Podemos ver ainda o nome dos arquivos dos quais os *chunks* retornados estão associados e também a pontuação de relevância com a consulta.

```
nodes_retrieved = retriever.retrieve("How to identify a problem in
    equipment?")

# Resposta
score:  0.495609155112431  file_name:  manual.pdf
score:  0.469781987654433  file_name:  trigger.pdf
score:  0.47050362795575174  file_name:  support_cases.pdf
```

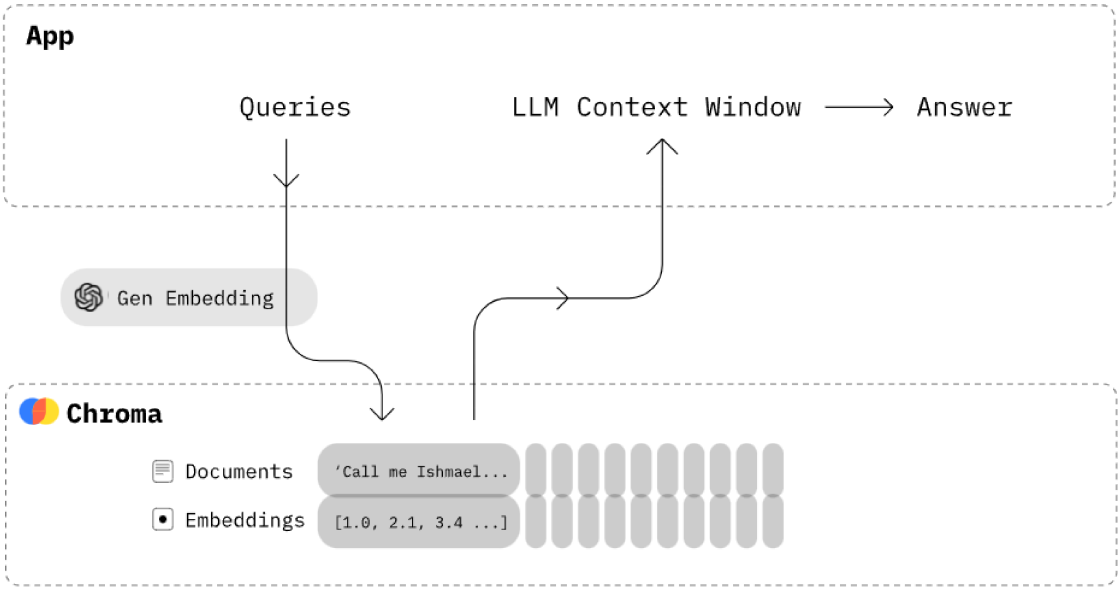
Seção de Código 4.12 – Exemplo de recuperação

4.5.4 Síntese de Respostas

Após o processo de recuperação de documentos, os primeiros *K chunks* dos documentos mais relevantes são retornados juntamente com seus metadados associados. Com esses metadados, conseguimos acessar o texto utilizado para gerar os *embeddings* desse *chunk*. Podemos então utilizar esse texto para gerar uma resposta semântica à consulta inicial, um processo chamado de Síntese de Resposta. Para isso, a consulta inicial, juntamente com os *K* documentos, são passados como entrada para um grande modelo de linguagem (LLM) para a geração de uma resposta final que seja contextualmente relevante e compreensível para o usuário final, em vez de apenas apresentar trechos de textos. Esse fluxo pode ser visto na Figura 13.

Para executar essa tarefa utilizamos LlamaIndex com o método *get_ response_ synthesizer* . Além disso precisamos utilizar a classe *RetrieverQueryEngine* para criar

Figura 13 – Processo de recuperação com banco de dados vetorial Chroma.



Fonte: CHROMA... (s.d.)

um mecanismo que realiza o fluxo de integração do *retriever* criado anteriormente com o *response synthesizer*. O grande modelo de linguagem LLM utilizado para gerar respostas foi o *"meta-llama/Llama-2-70b-chat-hf"*.

A configuração para criar um sintetizador de respostas pode ser visto na Seção de Código 4.13.

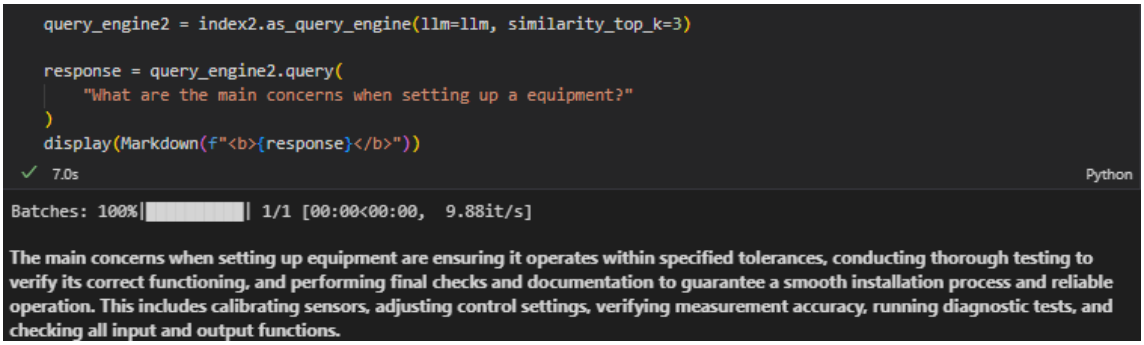
```
# Configurar o recuperador
retriever = VectorIndexRetriever(
    index=index,
    similarity_top_k=3,
)

# Configurar o sintetizador de respostas
response_synthesizer = get_response_synthesizer(llm=llm)

# Definir o mecanismo de consulta
query_engine = RetrieverQueryEngine(
    retriever=retriever,
    response_synthesizer=response_synthesizer
    )

# Consulta com a entrada do usuario
response = query_engine.query("How to identify a problem in equipment?")
```

Seção de Código 4.13 – Código para configurar o sintetizador de respostas

Com o fluxo de geração de respostas definido, é possível realizar um teste de

como as respostas serão geradas para uma mesma consulta, alterando apenas a forma de processamento dos documentos. Na Figura 14 temos a resposta gerada para a indexação dos documentos sem a realização do método de *Chunking*, usando o textos completos de cada página dos documentos.

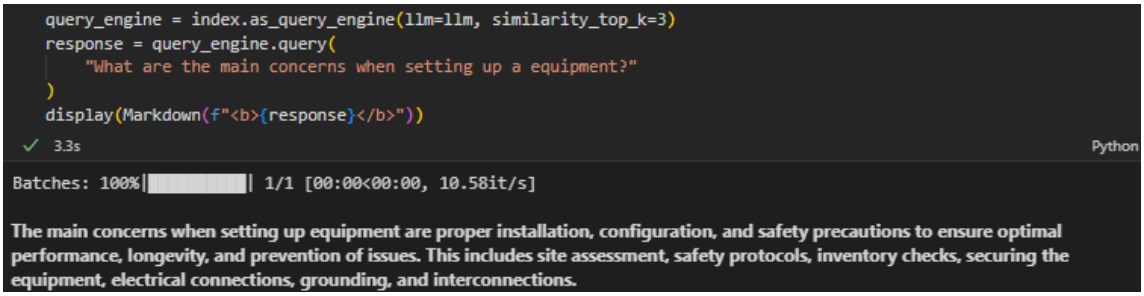
Figura 14 – Resposta gerada usando contexto dos documentos completos.



Fonte: Autor.

Já a Figura 15 mostra a resposta do processo quando realizado o *Chunking* dos documentos com uma configuração de *chunk_size* de 512 *tokens*.

Figura 15 – Resposta gerada usando contexto com trecho dos documentos.



Fonte: Autor.

Comparando as duas respostas é possível observar que ao realizar a segmentação dos textos, as respostas geradas se tornam mais específicas, trazendo informações com maiores detalhes. Já no caso onde os textos são usados completos, a resposta é mais generalista, dando uma informação superficial que aparentemente não indica nenhuma solução potencial.

4.5.5 Instrução Textual para o Modelo - *Prompt*

Como explicado na Seção 2.7, o *prompt* pode impactar muito em como a LLM se comporta ao gerar respostas. Um *prompt* bem formulado funciona como uma instrução de como o modelo deve pensar para gerar a resposta final, entendendo o contexto desejado e o formato esperado de como a resposta deve ser gerada.

Dessa forma, ao se pensar em uma solução de Inteligência artificial generativa utilizando a abordagem com RAG, deve se entender muito bem o contexto das informações nas quais os documentos estão inseridas, e também qual o propósito final do seu sistema, para então gerar uma resposta que faça sentido com o que era esperado no começo.

O LlamaIndex permite a customização do *prompt* utilizado para realizar buscas e gerar respostas. Na solução que utiliza documentos de manuais, foi empregado um modelo que fornece instruções para todas as consultas realizadas com o índice criado. O *prompt* criado pode ser visto na Seção de Código 4.14.

```
    template = (
    "Voce e um especialista em equipamentos eletricos, com documentos
   tecnicos dos equipamentos mais recentes e relevantes. "
    "Sua tarefa e fornecer respostas claras, precisas e informativas
   para perguntas dentro deste dominio.\n\n"
    "Abaixo esta algum contexto relacionado a pergunta:\n"
    "---------------------------------------\n"
    "{context_str}\n"
    "---------------------------------------\n"
    "Usando as informacoes acima, por favor responda a seguinte pergunta
    com referencia a documenta  o relevante, "
    "casos historicos ou principios fundamentais, quando necessario:\n\n
   "
    "Pergunta: {query_str}\n\n"
    "Certifique-se de que sua explicacao seja compreensivel para uma
   pessoa sem conhecimento em equipamentos eletricos."
)
```

Seção de Código 4.14 – Modelo de prompt customizado traduzido para o português

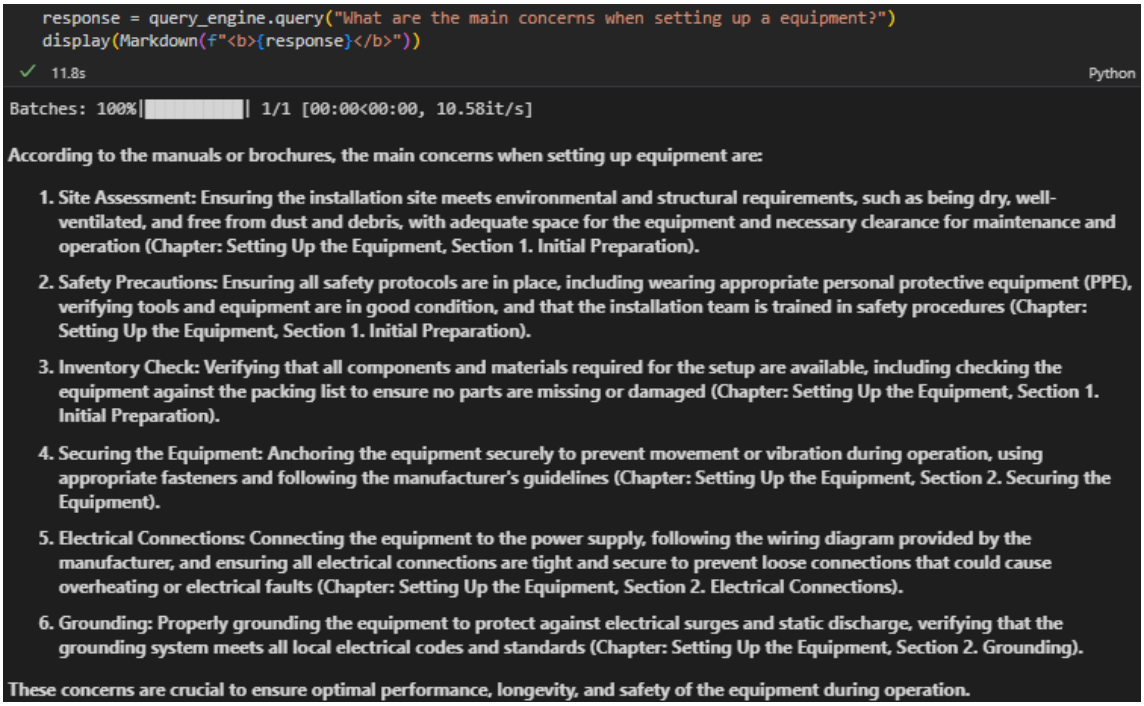
Na seção anterior foi mostrado uma das formas de criar um mecanismo de consulta, que pode ser abstraída ainda mais com o método *"as_retriever"* da classe *VectorStoreIndex*. Dessa forma, podemos criar um mecanismo de consulta como mostrado na Seção de Código 4.15, passando como argumento o LLM usado, o modelo de *prompt* definido e o valor de "K".

```
query_engine = index.as_retriever(llm=llm,text_qa_template=qa_template,
   similarity_top_k=3)
```

Seção de Código 4.15 – Código para criar um mecanismo de consulta de forma mais direta

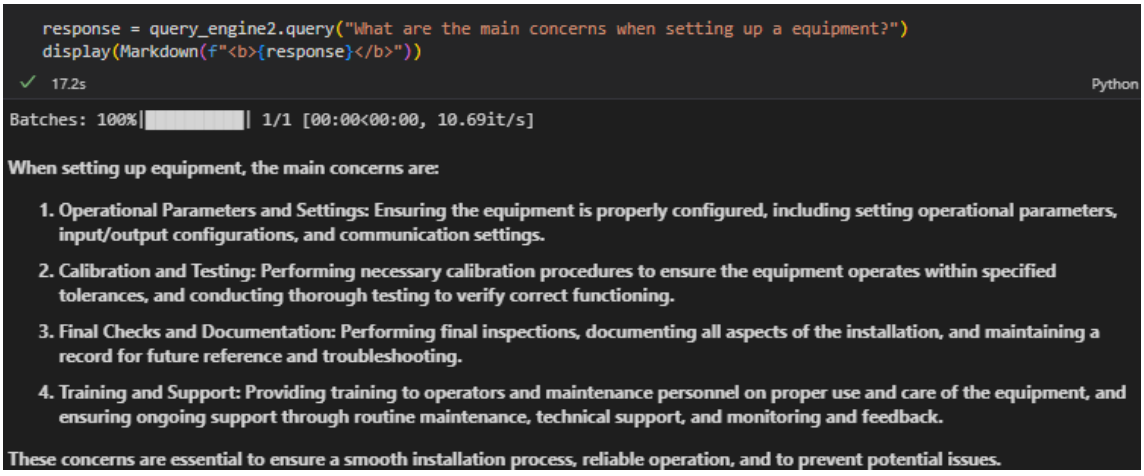
Ao utilizar esse modelo de *prompt*, temos um grande aprimoramento na qualidade das respostas geradas através do nosso RAG. Na Figura 16 temos a resposta gerada com o método de *chunking* e na Figura 17 a resposta equivalente sem utilizar o método de *chunking*.

Figura 16 – Resposta usando *prompt* customizado com *chunks*.



Fonte: Autor.

Figura 17 – Resposta usando *prompt* customizado sem *chunks*.



Fonte: Autor.

4.5.6 Interface de usuário com RAG

Esta funcionalidade também foi inserida na interface do usuário, é possível interagir com uma página chamada "Questionar Documentos", responsável por permitir que o usuário realize consultas aos documentos utilizando a abordagem de RAG.

Nessa página, três códigos são utilizados para execução das funcionalidades. O primeiro é responsável pela lógica principal da aplicação, definindo a navegação e

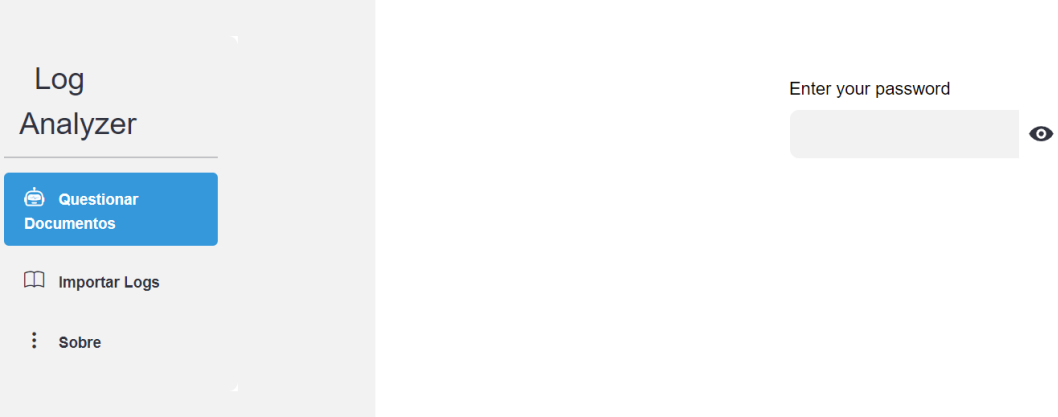
organizando como e quais páginas são exibidas.

Posteriormente temos um código definindo a lógica da página “Questionar Documentos”, permitindo com que o usuário tenha acesso a solução de *Retrieval Augmented Generation*, desenvolvida e explicada nas Seções 4.5.1 e 4.5.4, interagindo com a LLM através de consultas e recebendo respostas. Também é realizado a lógica de conversação e a chamada dos modelos utilizados.

Por último, temos o código responsável por lidar com as diferentes formas de fazer a chamada das LLMs, estas podendo estar em execução local em um contêiner ou em alguma plataforma na nuvem, sendo chamadas através de uma API.

A interface tela de login é apresentada na Figura 18.

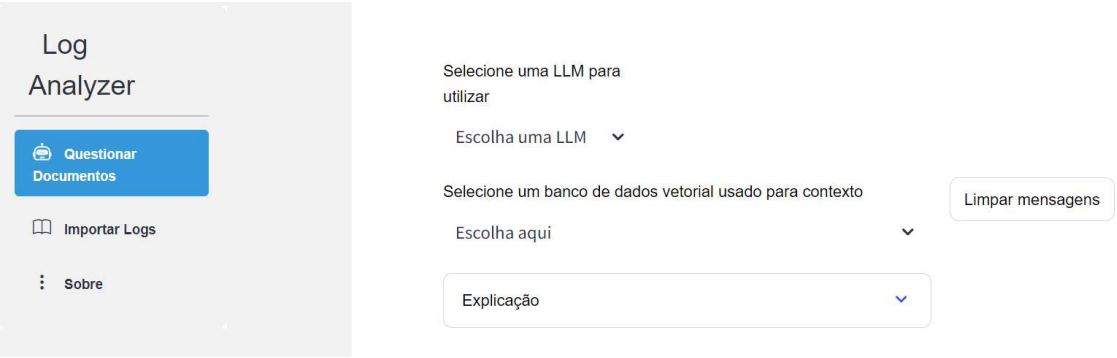
Figura 18 – Página de *login* da interface.



Fonte: Autor.

A interface da aba “Questionar Documentos” permite o usuário interagir com a solução de RAG com os documentos é apresentada na Figura 19.

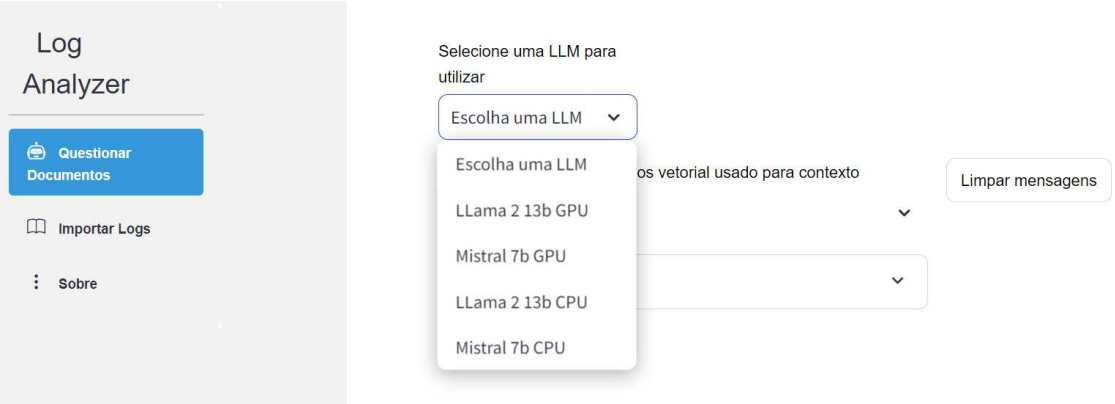
Figura 19 – Página para interação com módulo de RAG.



Fonte: Autor.

A seleção de modelos disponíveis para a geração de respostas pode ser vista na Figura 20.

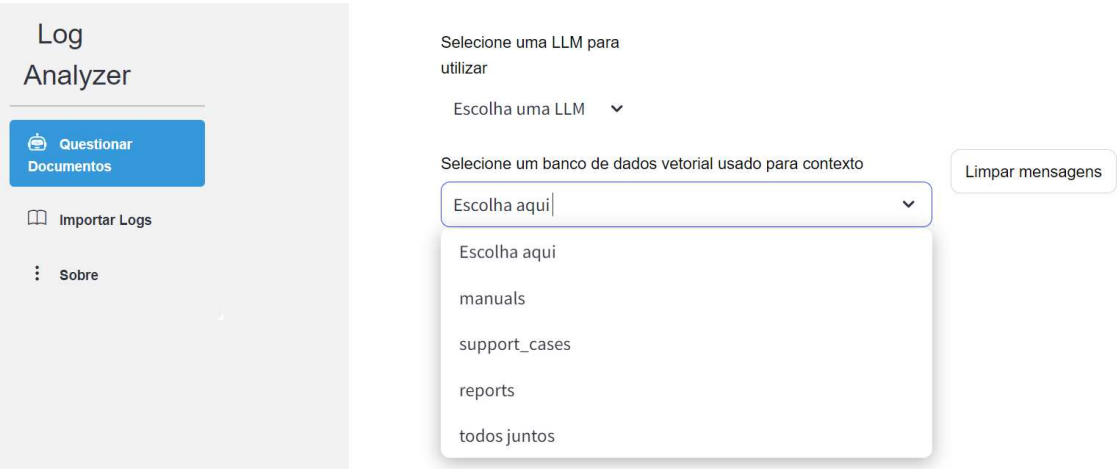
Figura 20 – Seletor de LLM.



Fonte: Autor.

A seleção de qual banco de dados vetorial utilizar de fonte de conhecimento para o RAG é apresentada na Figura 21.

Figura 21 – Seletor de banco de dados para usar de contexto.

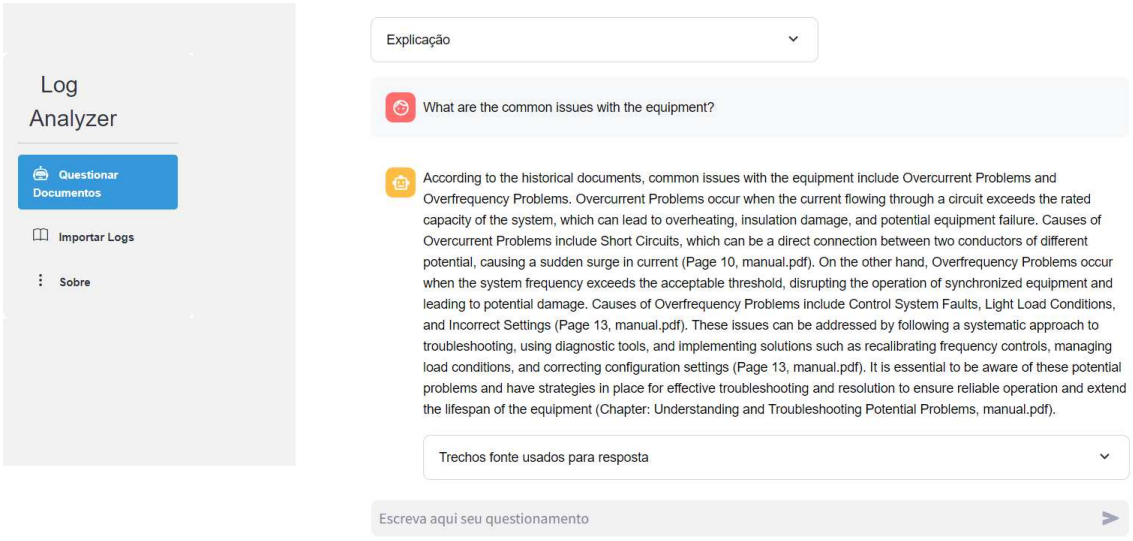


Fonte: Autor.

Ao selecionar qual banco de dados vetorial utilizar como contexto, uma nova seção se abre na aplicação, permitindo que o usuário execute uma consulta, como pode ser visto na Figura 22.

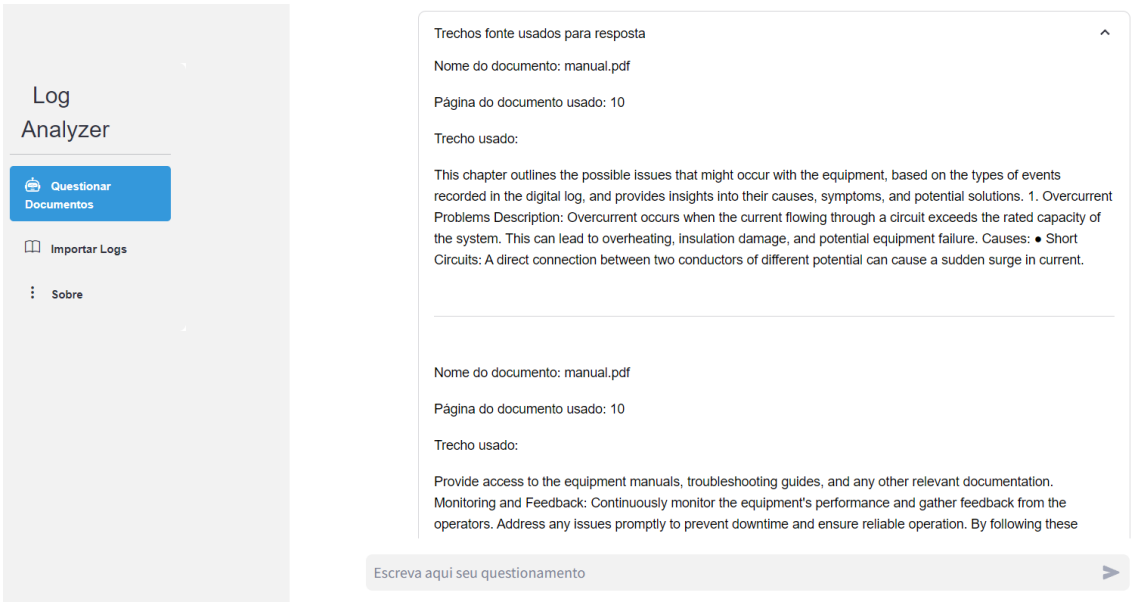
Na Figura 23 é possível observar que a seção também permite consultar qual fonte de informação foi utilizada para gerar a resposta final para o usuário. Como explicado anteriormente, foi utilizado um “K” de 3 documentos, portanto serão apresentados o nomes, páginas e trechos dos documentos mais relevantes para a consulta.

Figura 22 – Exemplo interação pergunta-resposta.



Fonte: Autor.

Figura 23 – Fontes utilizadas para gerar a resposta do RAG.



Fonte: Autor.

4.6 LLM PARA CRIAÇÃO DE CONJUNTO DE DADOS QUESTÃO-RESPOSTA

Um dos objetivos do projeto consiste na avaliação do desempenho dos modelos utilizados na tarefa de recuperação de informações. Isso é importante porque, como visto na Seção 4.5.1, diversas são as formas e estratégias que podem ser adotadas para configuração do fluxo de processamento dos documentos.

É de conhecimento comum que para realizar a avaliação de alguma tarefa é

necessário ter conhecimento da resposta correta. Isso também vale para avaliação da tarefa de recuperação de informações. Entretanto, esta avaliação não é uma tarefa trivial.

Quando lidamos com consultas sobre assuntos relacionados aos documentos do equipamento, é necessário acessar os documentos e achar exatamente os trechos que contém as informações que podem ser usadas como resposta a uma consulta. Além disso, a relevância de um trecho pra uma consulta pode ser muito subjetiva, dependendo do leitor. Esse desafio aumenta de escala, visto que, para avaliar um modelo de recuperação é necessário uma grande quantidade de exemplos de consultas associadas com o trecho de resposta.

Como esse conjunto de perguntas e respostas não existe, seria necessário a criação dessa base de dados e, para isso, contar com o apoio ativo de um especialista do cliente com vasto conhecimento do equipamento e também da documentação. Essa pessoa deve vasculhar a documentação, criar questionamentos e mapear os trechos que contém as respostas. Essa poderia ser uma solução a ser seguida, porém custosa para o cliente e sujeita a erros no processo.

Uma alternativa para criação desse conjunto de dados de questão-resposta é a utilização dos próprios LLMs para automatizar essa tarefa, realizando o papel inverso dos que estávamos fazendo previamente. Ao invés de realizarmos perguntas e recebermos uma resposta, podemos usar como entrada os trechos dos documentos e instruir o LLM para criar uma questão, com base no conteúdo semântico presente.

Novamente, foi utilizado um método do LlamaIndex para auxiliar na resolução desse desafio. O método *generate_question_context_pairs*, como mostrado na Seção de Código 4.16, é uma abstração que utiliza os nodos criados a partir dos documentos e a LLM de preferência para criar um arquivo do formato JSON contendo o número desejado de perguntas para cada nodo de entrada e associando a questão com o *id* do nodo utilizado.

```
qa_dataset = generate_question_context_pairs(
    nodes,
    llm=llm, num_questions_per_chunk=1, qa_generate_prompt_tmpl=
   question_gen_query
)
```

Seção de Código 4.16 – Código para executar a criação de conjunto Questão-Resposta

Neste trecho de código, existe um parâmetro *“qa_generate_prompt_tmpl”*, assim como no RAG, onde utilizamos um modelo de *prompt* para dar instruções para cada geração de resposta. Aqui podemos escrever um *prompt* para que as respostas sejam mais especialistas para nosso domínio de aplicação. Na Seção de Código 4.17 temos o modelo *prompt* utilizado para a criação de questões para cada nodo percorrido.

```
question_gen_query = """\
O contexto da questao encontra-se a seguir.

---------------------
{context_str}
---------------------

Dada o contexto informado e sem usar informacoes anteriores.
gere perguntas baseadas na instrucao abaixo.

Voce e um especialista em eletrica e hardware. A sua tarefa e configurar \
{num_questions_per_chunk} perguntas para um proximo \
 um teste/exame tecnico. As perguntas devem ser altamente especializadas \
em todo o documento. Restringir as perguntas as \
informacoes de contexto fornecidas.
Nao responda a nada alem da questao fornecida.
Nao crie a resposta para a pergunta, apenas a pergunta.
Evite anunciar o que vai dizer, por exemplo: "Claro, aqui estao algumas
   perguntas baseadas nas informacoes de contexto fornecidas:
"
"""
```

Seção de Código 4.17 – Modelo de prompt usado para gerar perguntas traduzido para português

Na Seção de Código 4.6.1 temos o conjunto de dados Questão-Resposta gerado com LLM para os *chunks* de textos criados na etapa de processamento dos documentos. Podemos ver que para cada questão existe um *id*, que por sua vez está associado ao *id* do nodo utilizado na geração da questão, assim como é mostrado na Figura 4.6.2.

4.7 INTEGRAÇÃO DA SOLUÇÃO DE DOCUMENTOS COM A DE *LOGS*

Como explicado anteriormente, o objetivo da interface é integrar os módulos desenvolvidos, os quais, individualmente, já representam funcionalidades relevantes. Atuando em conjunto, o módulo de RAG com documentos pode enriquecer o resultado do módulo de sumarização de *logs*, trazendo informações sólidas e procedimentos validados que ajudam a identificar e interpretar os eventos registrados nos *logs*. O fluxo resultante da integração dos dois módulos pode ser visto na Figura 24, no qual a integração é realizada com a saída do *log* sumarizado, que serve de entrada para o RAG com manuais.

Para realizar essa integração, uma série de modificações tiveram de ser feitas nos códigos originais. Inicialmente desenvolvidos em um compilador de códigos onde

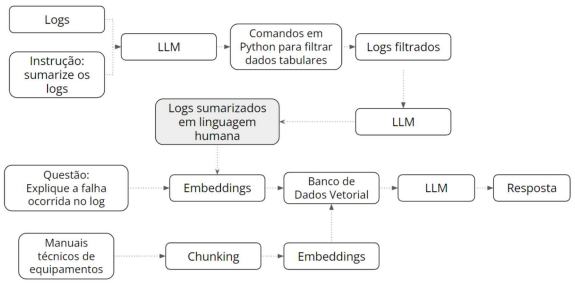
```
{
"queries": {
    "8e33739a-a50b-4592-95ef-07c922382ff7": "What is the primary
        cause of overfrequency problems in a distribution system,
        and how can it be mitigated by adjusting the load
        conditions?",
    "483ddd8e-111f-4751-b65c-672b64a7e4f9": "What are the essential
        environmental and structural requirements that the
        installation site must meet, as specified in the Site
        Assessment step of the initial preparation process?",
    "cbe55658-00e2-4236-bdbd-8abefa28b55c": "What is the primary
        purpose of recalibrating the frequency regulators in response
        to system overfrequency events, as described in the potential
        solutions?",
    "8c7c6fde-c64e-4f1d-b2fd-ffbf7ce11a9c": "What is the primary
        purpose of SNMP protocol in the context of network management
        and monitoring?",
    "affa9755-1db4-4181-bea4-0c1186c48254": "What is the primary
        consequence of faults or misconfigurations in the generator
        control system on the electrical system's performance?",
    "a3115c8f-92d6-40e9-906a-99589cf73399": "What type of assessments
        can be used to evaluate the competency of personnel in
        performing their tasks safely, according to the provided
        context information?",
    "8b16f648-879d-47f0-a6e5-c6a3097ef4ff": "What is the primary
        purpose of adjusting load conditions in a frequency
        regulation system?",
    ...
    "cc691989-e1fe-480c-8eb0-0b761ba129b9": "What is the primary
        purpose of implementing voltage regulators or stabilizers
        in the distribution network to address Phase C imbalances
        or faults?",
}
}
```

Seção de Código 4.6.1 – Conjunto de dados Pergunta-Resposta gerado com LLM.

```
{
    "relevant_docs": {
        "8e33739a-a50b-4592-95ef-07c922382ff7": [
            "3435173c-d018-4981-b83d-92eb87bd7b37"
        ],
        "483ddd8e-111f-4751-b65c-672b64a7e4f9": [
            "aee70083-37a5-4df2-a58b-aa32c84a09b7"
        ],
        "cbe55658-00e2-4236-bdbd-8abefa28b55c": [
            "0676098f-647d-4ae3-b70e-5b237dd32485"
        ],
        "8c7c6fde-c64e-4f1d-b2fd-ffbf7ce11a9c": [
            "49096651-8861-4260-a1e4-3f01b5a034af"
        ],
        "affa9755-1db4-4181-bea4-0c1186c48254": [
            "a554f98c-148b-440a-8e49-ddd2cfd08d0c"
        ],
        "a3115c8f-92d6-40e9-906a-99589cf73399": [
            "c179a118-4418-4781-80e1-73c777715542"
        ],
        "8b16f648-879d-47f0-a6e5-c6a3097ef4ff": [
            "6551b974-bc33-4a7d-91c8-9919adb5b64d"
        ],
        ...
        "cc691989-e1fe-480c-8eb0-0b761ba129b9": [
            "56710707-0d96-46c3-9ee4-f13ebb91d795"
        ]
}
```

Seção de Código 4.6.2 – Associação dos *ids* dos nodos e perguntas.

Figura 24 – Fluxo resultante da integração entre módulos desenvolvidos.



Fonte: Autor.

blocos de código pode ser executados individualmente, o *Jupyter Notebook*, por conta disso os códigos precisaram ser modularizados em classes para que pudessem ser reutilizados e chamados em diferentes partes da solução. Essa modularização facilita a manutenção e a escalabilidade do sistema.

A lógica de *upload* de *logs* também foi aprimorada. Agora, quando um arquivo CSV de *log* é carregado, ele é salvo em uma pasta específica, e o sumário extraído é salvo em um arquivo JSON, juntamente com outros metadados. Esses metadados podem ser utilizados como entrada para o módulo de RAG, juntamente com um *prompt* customizado para a tarefa. Essa estrutura permite que, ao acessar a página "RAG com Sumário dos *Logs*", o usuário possa selecionar o CSV importado e visualizar o sumário do *log* processado.

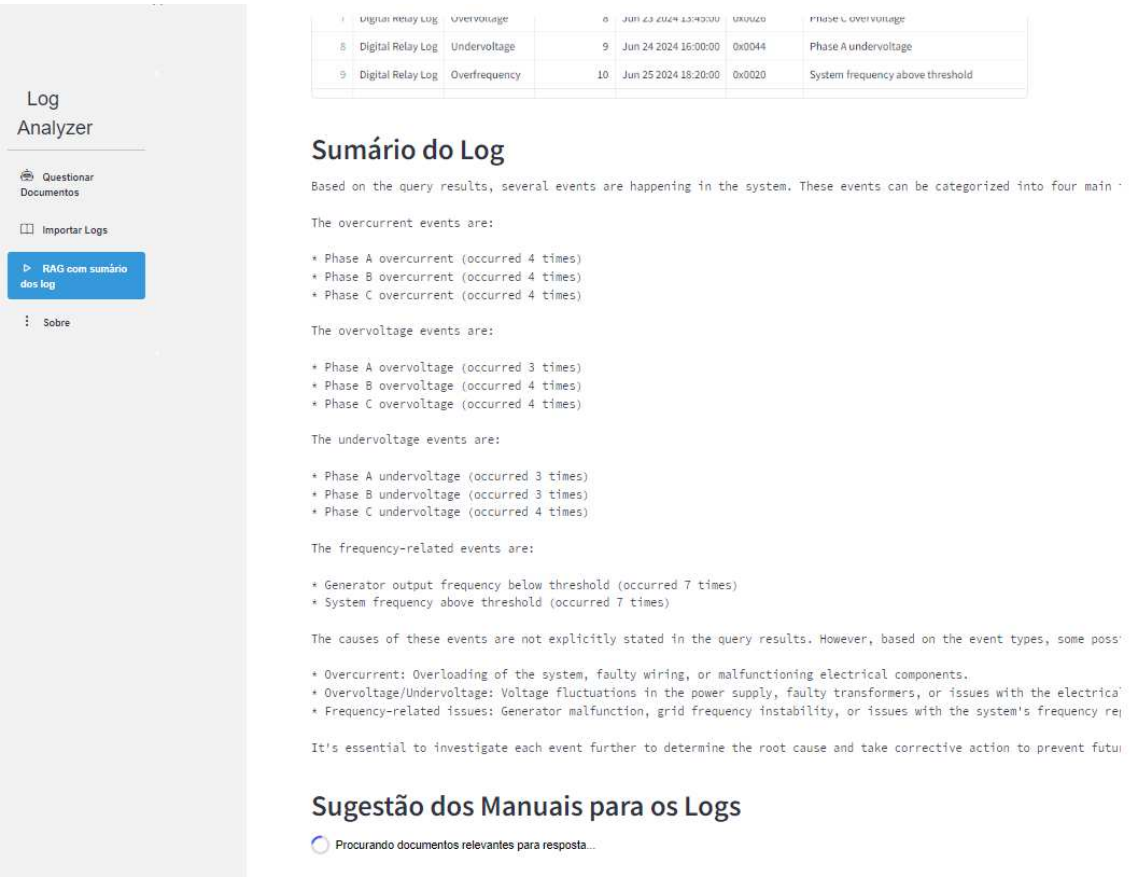
Ao selecionar um *log* na página "RAG com Sumário dos *Logs*", o arquivo JSON correspondente é acessado e o sumário do *log* é exibido logo abaixo da tabela de *log*, Isso proporciona uma visualização prévia das informações mais importantes extraídas dos *logs*.

Além disso, a interface foi ajustada para oferecer ao usuário a opção de selecionar o modelo de LLM (Modelo de Linguagem de Grande Escala) de sua preferência, assim como o banco de dados vetorial que será usado como contexto após a recuperação dos documentos. Ao fazer essa seleção, outro código é acionado para executar os módulos de RAG. Nessa versão da interface, as possibilidades de questionamento pelo usuário foram limitadas, assim, existe um *prompt* que instrui o RAG a fornecer sugestões ou ideias sobre o desempenho do equipamento e potenciais problemas evidenciados nos *logs* sumarizados. Como pode ser observado na Figura 25 ao selecionar o *log* desejado, é exibido tanto os dados da tabela original, quanto o sumário do *log* gerado automaticamente, é possível ver também, que logo abaixo existe uma seção "Sugestão dos Manuais para os Logs", onde após a finalização da execução do módulo de RAG a resposta gerada será mostrada ao usuário, como pode ser visto na Figura 26, assim como os trechos dos documentos relevantes recuperados para a geração da resposta.

Para melhorar a usabilidade e a funcionalidade da solução, foram implementadas as seguintes melhorias:

- Adição de um seletor para alterar o modelo LLM utilizado na geração de sumários.
- Implementação de um caminho para que a saída do módulo de sumarização sirva como entrada para o módulo de RAG com os manuais.
- Adequação da interface para não permitir que o usuário execute consultas diretamente. As respostas são geradas internamente a solução, proporcionando ao usuário acesso apenas à resposta final.

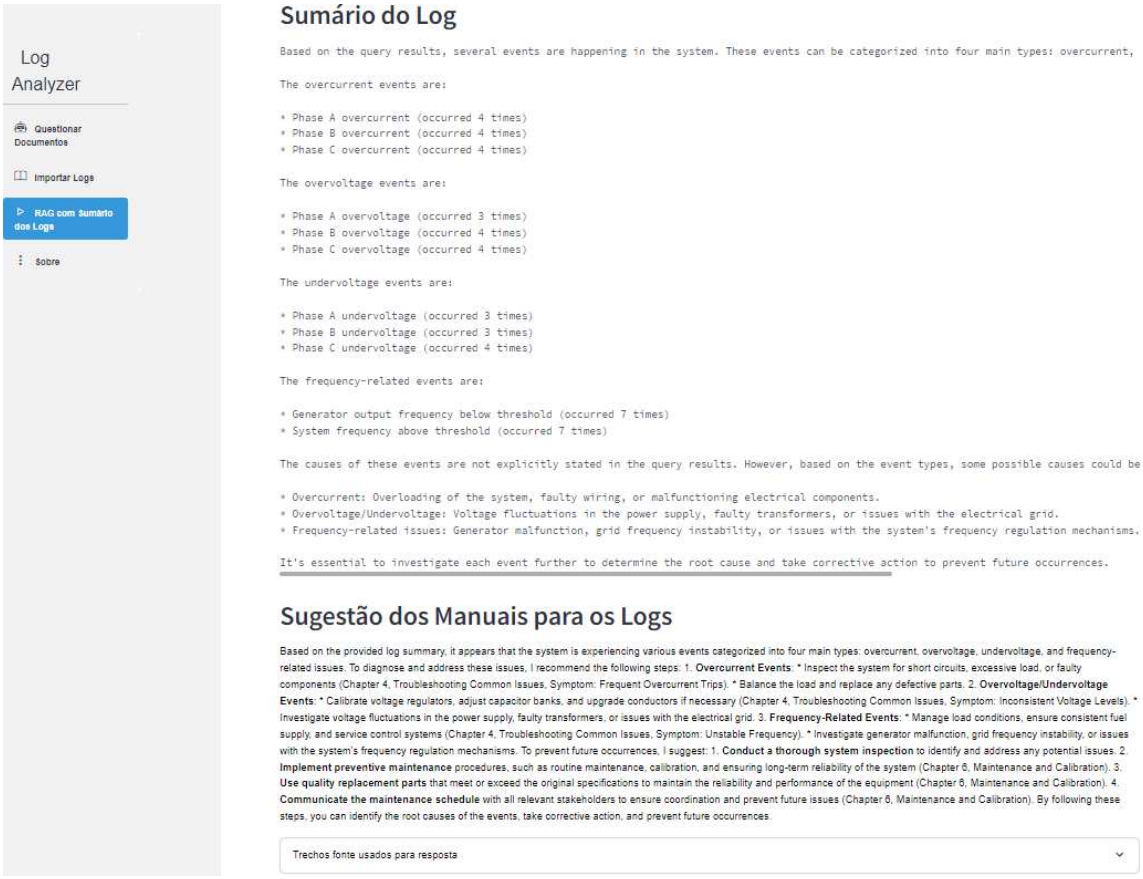
Figura 25 – Página “Sugestão dos Manuais para os Logs” integrando as soluções de RAG com documentos e sumarização dos *logs*.



Fonte: Autor.

Essas modificações e integrações garantem que a solução final seja robusta fornecendo uma ferramenta mais completa, de forma que várias LLMs foram utilizadas com diferentes propósitos para atingir o objetivo desejado. Oferecendo uma experiência de usuário fluida e intuitiva, ao mesmo tempo que fornece informações mais precisar e úteis derivadas tanto dos documentos técnicos quanto dos *logs* de eventos dos equipamentos.

Figura 26 – Resposta gerada ao fim da execução dos dois módulos integrados.



Fonte: Autor.

5 ANÁLISE DOS RESULTADOS OBTIDOS

Neste capítulo, será realizada a análise dos resultados obtidos com base na avaliação do desempenho dos modelos utilizados para recuperação de informações. Essa análise é crucial, conforme discutido na Seção 4.6, devido às diversas estratégias e configurações que podem ser adotadas para o processamento de documentos.

5.1 METODOLOGIA PARA ANÁLISE DE RESULTADOS

Para avaliar o desempenho das diferentes estratégias de *chunk size* e modelos de *embeddings*, foi realizado uma análise abrangente utilizando todo o conjunto de documentos. A seguir, é detalhado os passos adotados nessa metodologia:

- **Processamento Inicial**: Todos os documentos foram processados em conjunto para garantir uma avaliação global do desempenho dos modelos.
- **Geração de Perguntas**: Foi utilizado o método *generate_question_context_pairs* para criar perguntas baseadas nos trechos dos documentos. Esse método foi configurado para gerar uma quantidade específica de perguntas por *chunk*, utilizando o modelo de *prompt* customizado.
- **Avaliação do Desempenho**: Foram utilizadas as métricas de Taxa de Acerto (*Hit Rate*) e Classificação Recíproca Média (MRR) para avaliar o desempenho dos modelos.

Essa metodologia permitiu uma avaliação inicial do desempenho dos modelos de recuperação de informações em um cenário geral, considerando a variação e a diversidade dos documentos como um todo.

5.2 MÉTRICAS

Para avaliar os experimentos foram utilizadas duas principais métricas: Taxa de Acerto (*Hit Rate*) e Classificação Recíproca Média (MRR)[1].

- Taxa de Acerto (*Hit Rate*): Calcula a fração de consultas em que a resposta correta é encontrada nos principais documentos recuperados. Em termos simples, essa métrica avalia a frequência com que nosso sistema acerta entre as primeiras tentativas.
- Classificação Recíproca Média (MRR): Avalia a precisão do sistema examinando a classificação do documento relevante mais bem colocado para cada consulta.

[1] Essas métricas foram explicadas em maiores detalhes na Seção 2.8

Calcula a média dos recíprocos dessas classificações em todas as consultas. Por exemplo, se o primeiro documento relevante tiver a classificação mais alta, a classificação recíproca será 1; se for o segundo, a classificação recíproca será 1/2, e assim por diante.

5.3 EXPERIMENTOS

Foi realizado diversos experimentos para testar diferentes estratégias de *chunk size* e *embeddings*:

- Estratégias de *Chunk Size*: Diferentes tamanhos de *chunks* foram testados para verificar como isso afetaria o desempenho dos modelos.
- Modelos de *Embeddings*: Foram avaliados vários modelos de *embeddings* para identificar qual proporcionava os melhores resultados.

Os experimentos demonstraram pequenas melhorias entre os diferentes modelos de *embeddings*. Quando o tamanho dos *chunks* aumenta, o número de nodos de documentos diminui, o que inicialmente pareceu melhorar os resultados. No entanto, ao aumentar o número de perguntas, foi observado uma piora nas métricas. A adequação na quantidade de perguntas foi realizada, pois uma questão é criada para cada *chunk*. Com *chunks* maiores há menos questões para realizar a avaliação, portanto uma menor amostra para avaliação.

5.4 RESULTADOS OBTIDOS

Os resultados obtidos estão apresentados na Tabela 3. Esta tabela mostra a comparação entre diferentes configurações de *chunk size* e modelos de *embeddings*.

Tabela 3 – Comparação de estratégias de *chunk size* e modelos de *embeddings*.

hit_rate	mrr	chunk_size	chunk_strategy	embedding_model	retriver k
0.7536	0.6062	512	SenteceSplitter	mixedbread-ai/mxbai-embed-large-v1	3
0.6521	0.5458	512	SenteceSplitter	BAAI/bge-base-en-v1.5	3
0.7317	0.6097	2048	SenteceSplitter	mixedbread-ai/mxbai-embed-large-v1	3
0.7345	0.5936	2048	SenteceSplitter	BAAI/bge-base-en-v1.5	3
0.7463	0.6123	512	SenteceSplitter	Alibaba-NLP/gte-large-en-v1.5	3

Fonte: Autor.

Os resultados mostram que o modelo de *embedding mixedbread-ai/mxbai-embed-large-v1* com um *chunk size* de 512 proporcionou a melhor taxa de acerto

(*Hit Rate*) e a segunda maior MRR. No entanto, é possível observar que ao aumentar o *chunk size* para 2048, os resultados se mantiveram competitivos, indicando que *chunk sizes* maiores podem ser viáveis dependendo do contexto e da necessidade de balancear a quantidade de nodos e a precisão das respostas.

5.5 ANÁLISE POR CATEGORIA DE DOCUMENTOS

Além dos experimentos gerais descritos anteriormente, também foi realizado uma avaliação específica para cada categoria de documento. Essa abordagem permitiu identificar como diferentes tipos de documentos influenciam o desempenho dos modelos de recuperação de informações. Os documentos foram separados em categorias distintas, utilizando metadados para organizar os nodos.

5.5.1 Metodologia de Avaliação

Para cada categoria de documento, os seguintes passos foram seguidos:

- Os documentos foram separados em pastas diferentes de acordo com suas categorias específicas, utilizando os metadados disponíveis no banco de dados vetorial.
- As perguntas foram geradas novamente, pois o JSON existente não continha os metadados do caminho dos arquivos, impossibilitando a separação dos documentos em suas categorias.
- Foi criado novos arquivos JSON de perguntas para cada categoria de documento.
- A função de avaliação foi executada novamente para cada categoria, calculando as métricas de *Hit Rate* e MRR.

5.5.2 Resultados por Categoria de Documento

Na análise dos dados apresentados na Tabela 4, observa-se uma variação no desempenho dos diferentes tipos de recuperação de documentos, medidos através da Taxa de Acerto (Hit Rate) e da Classificação Recíproca Média (MRR). Estes resultados fornecem indícios sobre a eficiência dos métodos de recuperação implementados em categorias específicas de documentos.

Primeiramente, observa-se que o conjunto de casos de suporte apresentou o melhor desempenho, com uma Taxa de Acerto de 0.7931 e MRR de 0.6704. Este resultado pode ser atribuído à natureza específica e bem definida das perguntas frequentes em suporte técnico, que permite uma recuperação mais eficiente através de métodos de busca direcionados e otimizados para este tipo de conteúdo.

Tabela 4 – Performance de diferentes recuperações.

retrievers	hit rate	mrr
catalogos_qa_dataset top-3	0.6744	0.5658
updates_qa_dataset top-3	0.7272	0.5984
support_cases_qa_dataset top-3	0.7931	0.6704
manuals_qa_dataset top-3	0.5869	0.5048
reports_qa_dataset top-3	0.6800	0.5733

Fonte: Autor.

Por outro lado, o conjunto de manuais teve o desempenho mais baixo, com Taxa de Acerto de 0.5869 e MRR de 0.5048. A complexidade e o volume substancial de informações contidas nos manuais podem dificultar a precisão dos métodos de recuperação, resultando em uma eficácia inferior. A natureza técnica e detalhada dos manuais pode exigir técnicas de recuperação mais avançadas ou específicas para melhorar a precisão.

Os conjuntos de catálogos, atualizações e relatórios mostraram desempenhos intermediários, com Taxas de Acerto e MRR variando de 0.6744 a 0.7272 e 0.5658 a 0.5984, respectivamente. Esses resultados sugerem que, embora essas categorias de documentos possam ser menos desafiadoras que os manuais, ainda existem oportunidades para otimizar os métodos de recuperação, possivelmente através da personalização das técnicas de busca para melhor atender às características específicas de cada tipo de documento.

Os resultados dos experimentos evidenciam a importância de adaptar e refinar as técnicas de recuperação de documentos conforme as características específicas de cada categoria. A incorporação de tecnologias de inteligência artificial e aprendizado de máquina personalizadas para cada tipo de documento pode ser uma abordagem eficaz para melhorar significativamente a precisão e eficiência dos sistemas de recuperação de informações. Esta personalização das técnicas de recuperação é essencial no contexto de grandes volumes de dados.

A importância dos resultados obtidos estende-se para além das métricas, impactando diretamente a eficiência operacional da empresa cliente em uma aplicação implantada em um sistema em produção. A otimização do sistema de recuperação de informações aprimora significativamente o suporte ao diagnóstico de falhas e a tomada de decisões, reduzindo o tempo necessário para acessar informações e aumentando a qualidade das respostas fornecidas. Para a BIX, essa preocupação na melhoria nas capacidades de recuperação fortalece o valor percebido pelo cliente, afetando diretamente sua reputação no mercado. Para os times de suporte, tem potencial de facilitar o acesso rápido a informações, melhorando a qualidade e a velocidade de realização das atividades.

6 CONCLUSÃO

No cenário industrial atual, é fundamental aumentar a automação e diminuir a dependência de trabalho manual para se manter competitivo e enfrentar os desafios do setor. Minimizar erros humanos é crucial para a eficiência e segurança operacional. Diante da complexidade em treinar colaboradores para múltiplas ferramentas e atividades, bem como o alto custo operacional, surge a necessidade de automatizar o processo de diagnóstico de problemas em equipamentos eletroeletrônicos. A análise de *logs*, que registram o funcionamento dos equipamentos, é uma tarefa complexa que exige conhecimento especializado e treinamento intensivo. A automatização desse processo por meio de grandes modelos de linguagem pode simplificar significativamente o diagnóstico, reduzindo a necessidade de treinamento específico e trazendo benefícios tanto para as equipes de suporte quanto para os clientes.

A empresa cliente, assim como todo o mercado, focada em otimizar suas operações, tem explorado e validado a adoção de grandes modelos de linguagem, para simplificar significativamente processos e reduzir a dependência de especialistas em tarefas altamente especializadas e que necessitam grande curva de aprendizado.

Durante o desenvolvimento deste trabalho, foram elaborados dois módulos que visam facilitar a interação do usuário do time de suporte com grandes volumes de dados e manuais técnicos. O primeiro módulo integra o uso do RAG com manuais especializados, permitindo que os usuários do suporte técnico realizem perguntas específicas e obtenham respostas precisas, extraídas diretamente das fontes técnicas. O segundo módulo possui capacidade de sumarizar registro de eventos de equipamentos, convertendo perguntas em linguagem de consulta utilizando uma LLM. Essa consulta filtra os dados tabulares, que são enviados a outra LLM que responde apenas com informações pertinentes à questão levantada.

Além disso, foi desenvolvida uma interface que permite a interação com os módulos mencionados, oferecendo ao usuário um meio prático para carregar *logs* e interagir com os manuais. Posteriormente a interface foi atualizada para permitir a integração completa entre os módulos de sumarização dos *logs* e do RAG. Este avanço permite a realização de uma tarefa mais complexa com base nos manuais com e nos sumários dos *logs*, resultando, de maneira mais direta, em sugestões aplicáveis às situações evidenciadas pelos dados sumarizados.

O projeto pode ser considerado um sucesso. Durante conversas com o cliente, foi relatado que a empresa possui diversas iniciativas para a inclusão de IA generativa nos seus processos internos. No entanto, um diferencial significativo deste trabalho foi o olhar atento para as regras de negócio e os casos de uso específicos da solução. Essa abordagem permitiu que a solução estivesse alinhada às necessidades reais do processo a ser implementado, mesmo que as regras de avaliação sejam subjetivas e

não mensuráveis.

6.1 VISÃO GERAL DO DESEMPENHO OBTIDO

Os resultados obtidos demonstram que o desempenho do sistema de recuperação de informações pode variar consideravelmente dependendo da categoria de documento. A análise detalhada por categoria é crucial para entender melhor onde o sistema se destaca e onde pode ser necessário realizar melhorias. Para aplicações práticas, é importante considerar essas variações e possivelmente ajustar as estratégias de *chunk size* e modelos de *embeddings* conforme o tipo de documento processado.

Os experimentos mostraram que o modelo de *embedding mixedbread-ai/mxbai-embed-large-v1* com um *chunk size* de 512 é uma boa configuração geral, mas ajustes específicos podem ser necessários para otimizar a recuperação de informações em categorias de documentos mais desafiadoras, como manuais técnicos.

Essa análise detalhada fornece uma base sólida para futuras melhorias e adaptações do sistema, garantindo que ele continue a atender às necessidades dos usuários de forma eficaz e eficiente.

6.2 VISÃO GERAL DA AVALIAÇÃO DOS USUÁRIOS

Além das métricas quantitativas, realizamos uma avaliação qualitativa com usuários que testaram o sistema de recuperação de informações. Essa avaliação ajudou a entender melhor como as diferentes estratégias de *chunk size* e modelos de *embeddings* afetavam a usabilidade e a eficácia do sistema na prática. *Feedbacks* foram coletados para refinar ainda mais os parâmetros e melhorar a experiência do usuário final.

Com base nessa análise, podemos concluir que a escolha do modelo de *embedding* e a estratégia de *chunk size* são cruciais para otimizar o desempenho de sistemas de recuperação de informações. A combinação de métricas quantitativas e avaliações qualitativas fornece uma visão abrangente para guiar futuras melhorias no sistema.

6.3 SUGESTÕES PARA TRABALHOS FUTUROS

Ao concluir o projeto, algumas atividades e testes planejados não puderam ser realizados no escopo desta escrita. Com base nos resultados obtidos e nas avaliações realizadas, são sugeridas algumas direções para trabalhos futuros:

- Explorar a utilização de diferentes bancos de dados vetoriais para melhorar a performance e escalabilidade do sistema;

- Implementar e testar a inclusão de metadados customizados para melhorar a precisão das respostas;
- Investigar novas estratégias de segmentação de documentos (*chunks*, *overlap*, *semantic splitters*) para melhorar a recuperação de informações;
- Realizar comparações detalhadas entre diferentes modelos de LLMs para identificar quais oferecem melhores resultados para diferentes tipos de documentos.

Em conclusão, o projeto não apenas atingiu seus objetivos iniciais, mas também abriu caminho para melhorias e inovações futuras. As análises realizadas e as sugestões propostas fornecem um direcionamento para o desenvolvimento contínuo e a otimização do sistema de recuperação de informações, permitindo que ele possa atender de maneira eficiente às demandas do cliente.

Foram identificadas áreas críticas onde a automação e a inteligência artificial podem transformar consideravelmente na tomada de decisão e a avaliação de problemas de equipamentos eletroeletrônicos. O trabalho desenvolvido proporcionou uma boa compreensão sobre a aplicabilidade prática da IA generativa em ambientes industriais.

Avaliações dos usuários sugerem que, enquanto o sistema oferece uma base robusta para a tomada de decisões rápida e informada, melhorias na interface e na precisão dos dados são necessárias para otimizar a usabilidade e eficácia. Futuros trabalhos devem, portanto, focar em refinar essas tecnologias, explorando novas abordagens de segmentação de documentos e modelos de LLM mais avançados para melhorar a qualidade da solução.

Este projeto estabeleceu uma forte introdução para a inclusão de soluções de IA nas operações da empresa cliente, promovendo uma base sólida para futuros avanços tecnológicos que possam beneficiar a indústria como um todo. Além disso, o desenvolvimento desse projeto trouxe um ótimo relacionamento entre a BIX Tecnologia e a empresa cliente, o resultado gerou oportunidades de novo projetos.

REFERÊNCIAS

AMAZON WEB SERVICES. **O que são arquivos de log? – Explicação sobre arquivos de log**. [*S.l.*: *s.n.*], 2024. https://aws.amazon.com/pt/what-is/log-files/. Acessado em 14 de junho de 2024.

BROWN, Tom B. *et al.* Language Models are Few-Shot Learners. **CoRR**, abs/2005.14165, 2020. Disponível em: https://arxiv.org/abs/2005.14165.

CHIANG, Wei-Lin *et al.* **Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference**. [*S.l.*: *s.n.*], 2024. arXiv: 2403.04132 [cs.AI].

CHROMA. **Chroma: Vector database**. Disponível em: https://docs.trychroma.com/. Acesso em: 2024.

CHROMA Documentation. [*S.l.*: *s.n.*]. https://docs.trychroma.com/. Acessado em: 18 de junho de 2024.

CLOUDFLARE. **Processamento de Linguagem Natural (PLN)**. [*S.l.*: *s.n.*], 2024. https://www.cloudflare.com/pt-br/learning/ai/natural-language-processing-nlp/. Acessado em 15 de Junho de 2024.

CONTRIBUTORS, LlamaIndex. **HitRate - LlamaIndex Documentation**. [*S.l.*: *s.n.*], 2024a. https://docs.llamaindex.ai/en/stable/api_reference/evaluation/metrics/#llama_index.core.evaluation.HitRate. Acessado em 16 de junho de 2024.

CONTRIBUTORS, LlamaIndex. **MRR - LlamaIndex API Documentation**. [*S.l.*: *s.n.*], 2024b. https://docs.llamaindex.ai/en/stable/api_reference/evaluation/metrics/#llama_index.core.evaluation.MRR. Acessado em 16 de junho de 2024.

CROFT, W Bruce; METZLER, Donald; STROHMAN, Trevor. **Search engines: Information retrieval in practice**. [*S.l.*]: Addison-Wesley Reading, 2010. v. 520.

DATABRICKS. **O que são grandes modelos de linguagem (LLMs)?** [*S.l.*: *s.n.*], 2024. https://www.databricks.com/br/glossary/large-language-models-llm. Acessado em 15 de junho de 2024.

DEEPLEARNING.AI. **Google Cloud Vertex AI Course**. [*S.l.*: *s.n.*]. Curso online disponível em DeepLearning.AI. Acesso em: 15 de junho de 2024. Disponível em: https://www.deeplearning.ai/short-courses/google-cloud-vertex-ai/.

DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. **CoRR**, abs/1810.04805, 2018. Disponível em: http://arxiv.org/abs/1810.04805.

DOCKER. **Docker: Accelerated Container Application Development**. Disponível em: https://www.docker.com/. Acesso em: 2024.

ELASTIC. **O que é Monitoramento de Logs?** [*S.l.*: *s.n.*], 2024. https://www.elastic.co/pt/what-is/log-monitoring. Acessado em 14 de junho de 2024.

GAO, Yunfan *et al.* **Retrieval-Augmented Generation for Large Language Models: A Survey**. [*S.l.*: *s.n.*], 2024. arXiv: 2312.10997 [cs.CL].

GÉRON, Aurélien. **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition**. Sebastopol, CA: O'Reilly Media, 2019. ISBN 9781492032649.

GOOGLE. **Google Cloud Platform, Serviços de computação em nuvem**. Disponível em: https://cloud.google.com/?hl=pt-BR. Acesso em: 2024.

HOTZ, Nick. **What is CRISP DM? - Data Science Process Alliance**. [*S.l.*: *s.n.*], 2024. https://www.datascience-pm.com/crisp-dm-2/. Última atualização em 28 de abril de 2024.

HUGGINFACE. **Hugging Face Hub documentation**. Disponível em: https://www.prnewswire.com/news-releases/dell-technologies-and-hugging-face-to-simplify-generative-ai-with-on-premises-it-301986352.html. Acesso em: 2024.

IBM. **Engenharia de Prompts: O que é e por que é importante**. [*S.l.*: *s.n.*], 2024a. Acessado em: 16 de junho de 2024. Disponível em: https://www.ibm.com/br-pt/topics/prompt-engineering.

IBM. **Large Language Models - Uma Visão Geral**. [*S.l.*: *s.n.*], 2024b. https://www.ibm.com/br-pt/topics/large-language-models. Acessado em 15 de Junho de 2024.

LASSILA, Juuso; LEPPÄNEN, Leo. Sentence Embeddings via Token Inference. Helsingin yliopisto, 2024.

LEWIS, Patrick *et al.* Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *In*: LAROCHELLE, H.; RANZATO, M.; HADSELL, R.; BALCAN, M.F.; LIN, H. (Ed.). **Advances in Neural Information Processing Systems**. [*S.l.*]: Curran Associates, Inc., 2020. P. 9459–9474. Disponível em: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

LIN, Tianyang; WANG, Yuxin; LIU, Xiangyang; QIU, Xipeng. A survey of transformers. **AI Open**, v. 3, p. 111–132, 2022. ISSN 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2022.10.001. Disponível em: https://www.sciencedirect.com/science/article/pii/S2666651022000146.

LIU, Qi; KUSNER, Matt J.; BLUNSOM, Phil. A Survey on Contextual Embeddings. **CoRR**, abs/2003.07278, 2020. arXiv: 2003.07278. Disponível em: https://arxiv.org/abs/2003.07278.

LLAMAINDEX. **LlamaIndex Query Pipeline over Pandas DataFrames**. Disponível em: https://docs.llamaindex.ai/en/stable/examples/pipeline/query_pipeline_pandas/. Acesso em: 2024.

LLAMAINDEX. **LlamaIndex, Data Framework for LLM Applications**. Disponível em: https://docs.llamaindex.ai/en/stable/. Acesso em: 2024.

MANNING, Christopher; SCHUTZE, Hinrich. **Foundations of statistical natural language processing**. [*S.l.*]: MIT press, 1999.

META AI. **Meta Llama: Democratizando o acesso através de uma plataforma aberta com modelos de IA, ferramentas e recursos**. [*S.l.*: *s.n.*], 2024. https://llama.meta.com/. Acesso em 15 de junho de 2024.

MIKOLOV, Tomas; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey. **Efficient Estimation of Word Representations in Vector Space**. [*S.l.*: *s.n.*], 2013. DOI: https://doi.org/10.48550/arXiv.1301.3781. arXiv: 1301.3781.

MISHRA, Mridul; VIRADIYA, Jaydeep. Survey of Sentence Embedding Methods, abr. 2019. DOI: 10.13140/RG.2.2.21861.45289.

MUENNIGHOFF, Niklas; TAZI, Nouamane; MAGNE, Loıc; REIMERS, Nils. MTEB: Massive Text Embedding Benchmark. **arXiv preprint arXiv:2210.07316**, arXiv, 2022. DOI: 10.48550/ARXIV.2210.07316. Disponível em: https://arxiv.org/abs/2210.07316.

NEWSWIRE, PR. **Dell Technologies and Hugging Face to Simplify Generative AI with On-Premises IT**. 2023. Disponível em: https://www.prnewswire.com/news-releases/dell-technologies-and-hugging-face-to-simplify-generative-ai-with-on-premises-it-301986352.html. Acesso em: 2024.

NVIDIA. **What is Retrieval-Augmented Generation?** [*S.l.*: *s.n.*], 2024. https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/. Acessado em 15 de Junho de 2024.

OPENAI. **ChatGPT: Optimizing Language Models for Dialogue**. [*S.l.*: *s.n.*], 2024. https://openai.com/index/chatgpt/. Acessado em 15 de Junho de 2024.

PROVOST, Foster; FAWCETT, Tom. Data Science and Its Relationship to Big Data and Data-Driven Decision Making. **Big Data**, v. 1, mar. 2013. DOI: 10.1089/big.2013.1508.

REIMERS, Nils; GUREVYCH, Iryna. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. **CoRR**, abs/1908.10084, 2019. arXiv: 1908.10084. Disponível em: http://arxiv.org/abs/1908.10084.

RIEDEL, Sebastian; KIELA, Douwe; LEWIS, Patrick; PIKTUS, Aleksandra. **Retrieval Augmented Generation: Streamlining the creation of intelligent natural language processing models**. [*S.l.*: *s.n.*], 2020. https://ai.meta.com/blog/retrieval-augmented-generation-streamlining-the-creation-of-intelligent-natural-language-processing-models/. Acessado em 15 de junho de 2024.

SCHRÖER, Christoph; KRUSE, Felix; GÓMEZ, Jorge Marx. A systematic literature review on applying CRISP-DM process model. **Procedia Computer Science**, Elsevier, v. 181, p. 526–534, 2021.

SERVICES, Amazon Web. **O que é a Engenharia de Prompt?** [*S.l.*: *s.n.*], 2024a. Acessado em 15 de junho de 2024. Disponível em: https://aws.amazon.com/pt/what-is/prompt-engineering/.

SERVICES, Amazon Web. **O que são grandes modelos de linguagem?** [*S.l.*: *s.n.*], 2024b. https://aws.amazon.com/pt/what-is/large-language-model/. Acessado em 15 de junho de 2024.

TECHCRUNCH. **Hugging Face raises 235M from investors, including Salesforce and Nvidia**. 2023. Disponível em: https://techcrunch.com/2023/08/24/hugging-face-raises-235m-from-investors-including-salesforce-and-nvidia/. Acesso em: 2024.

TECHNOLOGIES, SeMI. **Vector Index - Weaviate vector database documentation**. [*S.l.*: *s.n.*], 2021. https://weaviate.io/developers/weaviate/concepts/vector-index. Acessado em 16 de junho de 2024.

TUNSTALL, Lewis; VON WERRA, Leandro; WOLF, Thomas. **Natural language processing with transformers**. [*S.l.*]: "O'Reilly Media, Inc.", 2022.

VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. Attention Is All You Need. **CoRR**, abs/1706.03762, 2017. arXiv: 1706.03762. Disponível em: http://arxiv.org/abs/1706.03762.

VIEIRA, Renata; LOPES, Lucelene. Processamento de linguagem natural e o tratamento computacional de linguagens científicas. **Em corpora**, p. 183, 2010.

WANG, Bin; WANG, Angela; CHEN, Fenxiao; WANG, Yuncheng; KUO, C.-C. Jay. Evaluating word embedding models: methods and experimental results. **APSIPA Transactions on Signal and Information Processing**, v. 8, e19, 2019. DOI: 10.1017/ATSIP.2019.12.

WANG, Jiaqi *et al.* Review of large vision models and visual prompt engineering. **Meta-Radiology**, v. 1, n. 3, p. 100047, 2023. ISSN 2950-1628. DOI: https://doi.org/10.1016/j.metrad.2023.100047. Disponível em: https://www.sciencedirect.com/science/article/pii/S2950162823000474.

WIRTH, Rüdiger; HIPP, Jochen. CRISP-DM: Towards a standard process model for data mining. *In*: MANCHESTER. PROCEEDINGS of the 4th international conference on the practical applications of knowledge discovery and data mining. [*S.l.*: *s.n.*], 2000. P. 29–39.

WOLFF, Hayden. RAG 101: Demystifying Retrieval-Augmented Generation Pipelines. **NVIDIA Technical Blog**, dez. 2023. Disponível em: https://developer.nvidia.com/blog/rag-101-demystifying-retrieval-augmented-generation-pipelines/.