UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

ANTHONY HENRIQUE DE ALMEIDA DUARTE

INTEGRAÇÃO DE IA GENERATIVA COM LANGCHAIN E RAG NA AUTOMAÇÃO DE PROPOSTAS TÉCNICO-COMERCIAL EM PROJETOS FOTOVOLTAICOS

Recife
2025

ANTHONY HENRIQUE DE ALMEIDA DUARTE

INTEGRAÇÃO DE IA GENERATIVA COM LANGCHAIN E RAG NA AUTOMAÇÃO DE PROPOSTAS TÉCNICO-COMERCIAL EM PROJETOS FOTOVOLTAICOS

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Engenheiro em Engenharia de Controle e Automação

Orientador(a): Prof. Dr. Jeydson Lopes da Silva

Recife
2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Duarte, Anthony Henrique de Almeida.
Integração de IA generativa com LangChain e RAG na automação de propostas técnico-comercial em projetos fotovoltaicos / Anthony Henrique de Almeida Duarte. - Recife, 2025.
69 : il., tab.

Orientador(a): Jeydson Lopes da Silva
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia de Controle e Automação - Bacharelado, 2025.
Inclui referências, apêndices, anexos.

1. Chatbot. 2. Modelos de linguagem. 3. Sistemas fotovoltaicos. 4. Inteligência artificial generativa. 5. RAG (Retrieval-Augmented Generation). 6. Open-source LLM. I. Silva, Jeydson Lopes da. (Orientação). II. Título.

620 CDD (22.ed.)

ANTHONY HENRIQUE DE ALMEIDA DUARTE

INTEGRAÇÃO DE IA GENERATIVA COM LANGCHAIN E RAG NA AUTOMAÇÃO DE PROPOSTAS TÉCNICO-COMERCIAL EM PROJETOS FOTOVOLTAICOS

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Engenheiro em Engenharia de Controle e Automação

Aprovado em: 14/08/2025

BANCA EXAMINADORA

_____

Prof. Dr. Jeydson Lopes da Silva
Universidade Federal de Pernambuco

_____

Prof. Dr. Alex Ferreira Falcão Moreira (Examinador Interno)
Universidade Federal de Pernambuco

_____

Eng. M.Sc. Antonio Gustavo Evangelista Muniz Santos (Examinador Interno)
Universidade Federal de Pernambuco

Àqueles que, de maneira direta ou indireta, tornaram possível a concretização deste trabalho. com palavras de incentivo, apoio técnico, paciência ou mesmo em silêncio nos momentos em que mais precisei de concentração. Cada gesto teve importância nesta caminhada.

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter guiado minha trajetória universitária, fortalecendo-me nos momentos difíceis, pelas amizades que construí e pelos mestres que me inspiraram. Em especial, sou grato ao meu orientador, Prof. Dr. Jeydson Lopes, por sua dedicação, paciência e apoio fundamentais para a conclusão deste trabalho.

Ao meu pai, Luiz Antonio, agradeço profundamente pela estrutura e suporte que me permitiram focar nos estudos com tranquilidade. Ele foi um verdadeiro alicerce durante toda essa caminhada. À minha mãe, Emanoela Santos, sou grato pelo carinho, incentivo constante, orações e gestos de apoio que fizeram grande diferença em minha formação. Sem meus pais eu não teria entrado nem saído da universidade.

Minha noiva, Maria Clara Alvino, foi peça essencial nessa conquista. Enfrentamos juntos um relacionamento à distância por grande parte da graduação, e seu apoio, paciência e amor foram combustíveis para que eu seguisse firme. Sua dedicação, inclusive assumindo sozinha diversas responsabilidades do dia a dia, foi crucial para que eu pudesse concluir esta pesquisa.

Também agradeço ao meu irmão, Alysson Muniz, por acompanhar minha trajetória de perto, sempre presente nos momentos importantes. Às minhas avós, Verônica Monteiro e Adriana Santos, minhas referências de força e sabedoria, agradeço por seus conselhos, orações e por sempre me ouvirem com carinho.

Aos colegas de turma, agradeço por todos os momentos de parceria, aprendizado e superação. Em especial: Gabriel Xavier, Ramatis Barros, Vinicius Barreto, Plácido Nilo, Bernardo Rehn, João Pedro Barreto, Júlio Cezar e Carlos César. Aos colegas do ABI, que marcaram o início da minha jornada acadêmica: Alencar, Caio Guilherme, Álvaro Nascimento, João Miranda, Álvaro Roberto, Heitor Melo, Luiz Costa e Ravi.

A todos que, direta ou indiretamente, fizeram parte dessa jornada, deixo meu sincero agradecimento.

O SENHOR é o meu pastor; nada me faltará. Ele me faz repousar em pastos verdejantes. Leva-me para junto das águas de descanso; refrigera-me a alma. Guia-me pelas veredas da justiça por amor do seu nome. Ainda que eu ande pelo vale da sombra da morte, não temerei mal nenhum, porque tu estás comigo. (SALMOS, 23)

RESUMO

Este projeto investigou o desenvolvimento e avaliação de *chatbots* baseados em modelos de linguagem para automatizar a análise técnico-comercial em sistemas fotovoltaicos. Quatro experimentos foram conduzidos, testando diferentes arquiteturas e modelos: o Ollama3.1 8B em versões com interface *desktop* e *web*, arquitetura modular com N8N, e a migração para o modelo GPT-4. Os resultados demonstraram que o desempenho do *chatbot* depende fundamentalmente do modelo de linguagem empregado, independente da arquitetura. Enquanto o modelo Ollama3.1 8B apresentou limitações significativas em cálculos técnicos e grande latência, o GPT-4 alcançou alta acurácia (96,7%), velocidade de resposta com média de 9 segundos e precisão em questões simples, intermediárias e complexas. Essas diferenças evidenciam que, apesar do custo de uso da API, modelos avançados como o GPT-4 são essenciais para aplicações que exigem rigor técnico e confiabilidade comercial. O projeto conclui destacando o potencial de modelos *open-source* para tarefas mais gerais e propõe caminhos futuros para aprimoramento via *fine-tuning* e expansão da base de conhecimento.

Palavras-chave: Chatbot; Modelos de linguagem; Sistemas fotovoltaicos; Inteligência artificial generativa; RAG (*Retrieval-Augmented Generation*); *Open-source* LLM.

ABSTRACT

This project explored the development and evaluation of language model-based chatbots for automating the technical-commercial analysis of photovoltaic systems. Four experiments tested different architectures and models: Ollama3.1 8B (with desktop and web interfaces), a modular architecture with N8N, and migration to the GPT-4 model. Results showed that chatbot performance depends primarily on the underlying language model. The Ollama3.1 8B model exhibited significant limitations in technical calculations and high latency, whereas GPT-4 achieved high accuracy (96.7%), fast response times (around 9 seconds), and precision across simple, intermediate, and complex queries. These findings demonstrate that despite API costs, advanced models like GPT-4 are crucial for tasks demanding technical rigor and commercial reliability. The project highlights the potential of open-source models for more general purposes and outlines future work including fine-tuning and knowledge base expansion.

Keywords: Chatbot; Language models; Photovoltaic systems; Generative artificial intelligence; RAG (Retrieval-Augmented Generation); Open-source LLM.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IA	Inteligência artificial
Gen AI	*Generative artificial intelligence*
RAG	*Retrieval-Augmented Generation*
LLM	*Large Language Model*
MPPT	*Maximum Power Point Tracking*

LISTA DE SÍMBOLOS

η	Eficiência média de painéis solares
C	Consumo diário de energia em kWh
I	Irradiação solar média
Pot	Potência dimensionada em kWp

SUMÁRIO

1 INTRODUÇÃO

A Inteligência Artificial (IA) é um ramo da ciência da computação que utiliza sistemas digitais e algoritmos para realizar diversas tarefas, possuindo uma extensa trajetória de desenvolvimento ao longo dos anos, a história inicia na Grécia antiga relacionada a máquinas inteligentes que operavam de forma independente do homem, ademais, após a segunda guerra mundial observou-se um interesse maior por programas de IA surgindo assim os primeiros protótipos, a exemplo jogos de dama e xadrez (SINGH, 2019).

O século XX foi um marco para IA ocasionado pela era digital, os sonhos de civilizações antigas começavam a se tornar realidade. Alan Turing, pai da computação, introduziu o Teste de Turing que estabeleceu a premissa de que uma máquina seria considerada inteligente se fosse capaz de reproduzir respostas humanas sob circunstâncias específicas e de forma tão semelhante que seria impossível distingui-la de um ser humano vivo. Um *workshop* realizado por estudantes e professores da *Darthmouth College* (Estados Unidos, 1956) ficou marcado pela divulgação do termo “Inteligência Artificial” e assim fincando na história o nascimento da IA como uma disciplina acadêmica a ser estudada e desenvolvida pela academia (QUANTUM GLOBAL GROUP, 2023).

Na década de 80, o algoritmo de retroprogramação possibilitou o treinamento eficaz das redes neurais de multicamadas. Nos anos 90, partir do aprendizado supervisionado, as redes neurais passaram a ser utilizadas em tarefas específicas como reconhecimento de padrões, caracteres e classificação de dados, porém o avanço era muito limitado devido a tecnologia e poder computacional de sua época. A partir do século XI, houve um crescimento acelerado da IA devido a criação do grande volume de dados (*Big data*), elevando a capacidade computacional e sofisticação dos algoritmos, dessa forma, permitiu o surgimento de redes neurais profundas com várias camadas e aplicações diversas como reconhecimento de imagem, linguagem e sons (KAUFMAN, 2018).

No cenário atual, a inteligência artificial vem contribuindo para a sustentabilidade ambiental tornando setores como agricultura, energia e transporte mais eficiente, reduzindo o desperdício de recursos e emissão de gases (DHIMA *et al.*, 2024). Ressaltando a importância da redução de gases poluentes na atmosfera, a energia

solar vem se destacando por ser uma fonte de energia renovável, que contribui para diminuição dos gases poluentes, mitigando os impactos ambientais gerados pelas fontes de energia convencionais. A IA se destaca por colaborar no desenvolvimento de novas tecnologias para sistemas fotovoltaicos com aplicação das redes neurais, lógicas *Fuzzy* e algotimos genéticos. O uso dessa ferramenta permite realizar previsão de irradiação solar, otmizando o funcionamento dos painéis solares em caso de usinas com *tracker*, mesmo sofrendo variações climáticas. Ademais, ajuda no diagnóstico e manuntenção dos equipamentos, por meio da análise dos relatórios de geração e dos dados do próprio sistema fotovoltaico (LE *et al.*, 2024). A inteligência artificial também aprimora as técnias de *Maximum Power Point Trancking* (MPPT), o que garante uma análise rápida e precisa da localização do ponto global de máxima potência, otimizando a geração elétrica da usina solar (YUNG *et al.*, 2020), além disso, a IA pode auxiliar engenheiros como ferramenta de auxílio para projetar e dimensionar sistemas fotovoltaicos.

Ainda que sistemas de IA possam ser surpreendentes e muito úteis em diversas aplicações, alguns pontos chamam a atenção dos especialistas para a segurança dos dados e das informações processadas por esses sistemas. A grande massa de dados podem acarretar em alucinações, vieses, erros de cálculo, risco de exposição de conteúdos sigilosos e vulnerabilidade em cibersegurança (SALAMI *et al*., 2025). Para reduzir os riscos, muitas organizações preferem desenvolver sua própria ferramenta de IA, treinadas com bases de dados internas (SJÖDIN *et al*., 2021). Uma das técnicas para treinar a IA com uma base de dados pré estabelecida é conhecida como *Retrival Augmented Generation* (RAG), traduzindo, a geração aumentada de recuperação, permitindo que os modelos de IA produzam respostas mais confiáveis, precisas e contextualizadas com base em dados resguardados (CHIDIPOTHU *et al*., 2025).

1.1 Objetivos

Desenvolver uma ferramenta capaz de ler, interpretar e discutir com usuário sobre documentos e normas para dimensionamento de usinas fotovoltaicas.

1.1.1 Objetivos Gerais

Desenvolver e avaliar um sistema automatizado para análise de documentos e normas aplicadas ao dimensionamento de projetos fotovoltaicos, por meio da utilização de inteligência artificial, do *framework LangChain* e da arquitetura RAG (Geração Aumentada por Recuperação). O objetivo central é otimizar o tempo de resposta e a precisão das informações, contribuindo para a elaboração mais eficiente de propostas técnico-comerciais no setor fotovoltaico.

1.1.2 Específicos

Para atingir o objetivo geral, foram definidos os seguintes objetivos específicos:

- Estruturar uma base de conhecimento especializada, composta por leis e normas técnicas relacionadas ao setor de geração fotovoltaica.
- Implementar e testar um protótipo inicial utilizando um modelo de linguagem de código aberto (Ollama 3.1 8B) operando em ambiente local.
- Avaliar e comparar o desempenho entre o modelo local (Ollama) e um modelo de alta performance via API (GPT-4 da OpenAI), considerando critérios como tempo de resposta e acurácia.
- Desenvolver a solução final com base na combinação de tecnologias que apresentarem os melhores resultados durante os testes experimentais. Organização do Trabalho

1.1.3 Organização do trabalho

A organização deste trabalho está delineada em cinco capítulos, detalhados a seguir:

- Capítulo 1 – Introdução: Apresenta o contexto do estudo, delineando os objetivos gerais e específicos, destacando a relevância do tema abordado, e descrevendo a estrutura do trabalho.

- Capítulo 2 – Fundamentação Teórica: Aborda os principais conceitos e fundamentos que sustentam o tema, incluindo explicações detalhadas sobre RAG (*Retrieval-Augmented Generation*), *frameworks*, sistemas fotovoltaicos e modelos de linguagem.
- Capítulo 3 – Metodologia: Descreve a abordagem metodológica adotada, incluindo o desenvolvimento de experimentos iterativos voltados à prototipagem de um modelo de Inteligência Artificial (IA) especializado no dimensionamento de usinas fotovoltaicas.
- Capítulo 4 – Resultados e Discussões: Apresenta a análise dos resultados obtidos em cada experimento, com uma discussão aprofundada sobre o desempenho dos diferentes modelos de linguagem utilizados, incluindo uma comparação entre suas performances.
- Capítulo 5 – Conclusões e Propostas de Continuidade: Sintetiza os principais achados do estudo, avaliando os protótipos a partir do atendimento aos requisitos estabelecidos, e propõe direções futuras para o desenvolvimento e aprimoramento da aplicação.

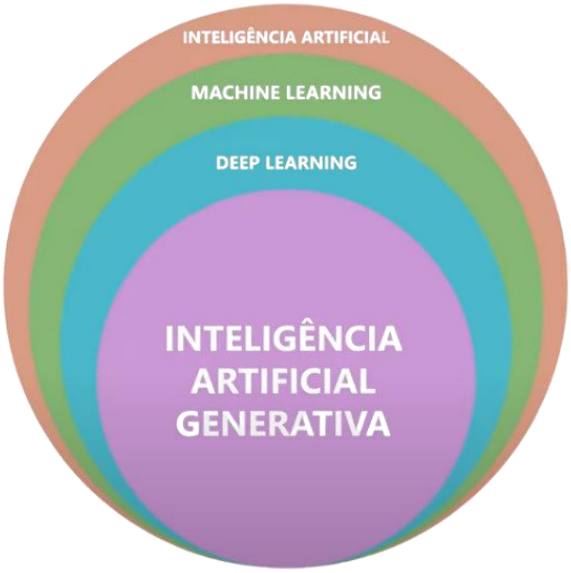
2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão apresentados os conceitos fundamentais que sustentam o desenvolvimento deste trabalho. A estrutura seguirá uma abordagem que parte das tecnologias que compõe a base da Inteligência Artificial (IA), em seguida aprofundamos o assunto das arquiteturas e ferramentas específicas que serão utilizadas neste projeto, resultando na apresentação do domínio de aplicação, uma ferramenta de auxílio a engenheiros para dimensionar usinas fotovoltaicas e elaboração de propostas comerciais referentes a essas usinas.

2.1 Inteligência Artificial Generativa e *Largue Language Models* (LLMs)

A inteligência artificial generativa também conhecida como "*Gen AI*" (do inglês *generative aritificial intelligence*) refere-se a um segmento específico da IA responsável pela criação de dados originais como textos, imagens e vídeos (SJÖDIN *et al*.,2021). Para entender melhor suas funcionalidades, é necessário voltarmos aos conceitos básicos de inteligência artificial, o qual é constituído por conjunto abrangente, incluindo as categorias como *Machine Learning*, *Deep Learning*, incluindo a IA generativa como ilustra Figura 1. Cada uma das categorias citadas anteriormente possui áreas de atuações distintas e específicas no mercado, abrangendo análise de dados, previsões, otimização de produtos, criação de conteúdo e assistentes virtuais, como *chatbots*.

Figura 1 - Camadas da IA, da teoria geral à tecnologia generativa.



Fonte: Microsoft Reactor (2025).

Diferentemente de outros modelos de IA, que são direcionados principalmente à analise, identificação e classificação de dados pré-existentes para melhorar conceitos ou extrair informações, a IA generativa possui a capacidade de criar dados semelhantes aos reais. Esses modelos se destacam na extração e análise de características, em tarefas de regressão, agrupamento, assimilação de padrões, visão computacional, previsão do tempo e criação de conteúdos multimídias (SJÖDIN *et al.*,2021). Ademais, essa tecnologia possui diversos outros tipos de aplicação além da área da engenharia e estatística, como por exemplo análises clínicas, diagnóstico médio, além de auxiliar cirurgias (SENGAR, 2024). O robô Da Vinci 5 conseguiu um feito inédito em 2024, realizando um procedimento cirúrgico em carne de aves, aplicando o aprendizado por demonstração após aprender com mais de 50 mil vídeos de cirurgia semelhante (AMINUDDIN, 2024), a imagem do robô está representada na Figura 2 .

Figura 2 - Robô Da Vinci 5.

Fonte: Scholl of Medicine Virginia News (2024).

Dentro desse cenário, os Modelos de Linguagem de Grande Escala (LLMS, do inglês, *Large Language Models*) representam um dos avanços mais importantes da IA generativa no campo de processamento de linguagem natural. Explicando de forma simplificada, LLMs são redes neurais profundas treinadas com *bigdatas*, utilizando arquiteturas como *transformers*, para assimilar e gerar as informações com graus elevado de coerência e fluidez (LIU *et al*., 2024). Podemos citar exemplos de ferramentas de LLMs bastante úteis como o GPT da OpenAI, Mistral, Ollama (Meta) e o *DeepSeek* que têm revolucionado diversas áreas, automatizando processos e tarefas dos setores como saúde, educação, finanças, segurança e algoritmos de recomendação de conteúdo (RAZA *et al*., 2025).

2.2 Fluxograma do LLM com arquitetura RAG

Nesta seção, serão detalhadas as etapas do fluxograma do LLM integrado com a arquitetura RAG. Para melhorar a abordagem do fluxo, serão apresentados os conceitos técnicos por trás do RAG.

2.2.1 Arquitetura RAG *(Retrieval-Augmented Generation)*

Os LLMs como GPT ainda enfrentam desafios para sua total eficácia, problemas como erros de operações matemáticas, alucinação (invenção de dados sem respaldo), além de limitações do acesso a base de dados mais recentes e privadas (SALAMI *et al*., 2025). Estes fatores prejudicam sua funcionalidade o que acaba culminando na desconfiança da ferramenta para aplicações específicas.

O *Retrival-Augmented Generation* (RAG) ou geração aumentada de recuperação, surge como uma solução para mitigar esses erros, melhorando a performance do LLM de gerar respostas mais precisas e confiáveis. Quando o usuário inserir a pergunta, a IA fará primeiramente uma consulta na base de dados ou repositório com informações confiáveis e atualizadas, buscando conteúdos relevantes que se assemelham a pergunta realizada pelo usuário. Por fim, ela utiliza os dados recuperados para gerar uma resposta fundamentada, reduzindo significativamente problemas como alucinação e falta de transparência (GAO *et al*., 2023).

Segundo estudos de (ASAI *et al*,. 2023), mostra que o RAG melhora a precisão e adaptabilidade das respostas, permitindo atualizações contínuas sem a necessidade de refazer treinamentos.

2.2.2 Chunking e Embeddings

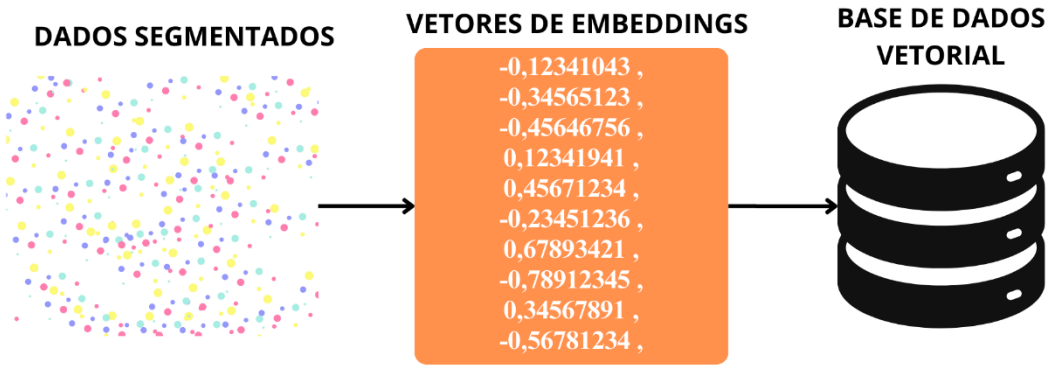
O termo *Chunking* (do inglês, pedaços) se refere ao processo de quebrar ou dividir documentos extensos em segmentos menores, com o intuito de facilitar sua análise. Essa técnica possui utilidades especiais quando trabalhamos com LLMs que possuem limitação na quantidade de *tokens* (menor unidade de um texto que um LLM consegue identificar e analisar) que podem ser processados de uma única vez, como os modelos baseados em arquitetura *transformers*. A relação entre *chunk* e *tokens* consiste no fato de que cada *chunk* é composto por vários tokens. Por exemplo, se um *chunk* contém 200 palavras pode conter de 400 a 500 tokens, a depender da estrutura do texto.

Os *embeddings* (do inglês, incorporação) é o processo de vetorização dos dados, consiste em transformar segmentos de textos, ou *chunks*, em representações

numéricas chamadas de vetor, com grandes dimensões que fazem a captura do seu aspecto semântico. Este processo é feito por IA, após a criação dos vetores, eles são armazenados em um banco de dados vetorial (*Vector Store*), onde são organizados em tabelas para facilitar a busca da LLM no momento das consultas (CHANG *et al*,. 2023).

Resumindo, o processo de *chunking* irá realizar a quebra de um documento inserido pelo usuário em segmentos menores, cada pedaço com um determinado número de *tokens* formam um *chunk*. Em seguida, cada *chunk* passará pelo processo de vetorização chamado *embedding*, onde um modelo de IA irá captura o significado semântico do texto e o representará numericamente por meio de um vetor. Por fim, o vetor numérico que represente algum dado específico será guardado no banco de dados vetorial, como ilustrado na Figura 3.

Figura 3 - Criação da base vetorial.



Fonte: O autor (2025).

2.2.3 Etapas do fluxograma

O fluxograma irá descrever o passo a passo de um sistema avançado de inteligência artificial integrado com RAG. Este sistema permite combinar a capacidade de raciocínio de um grande modelo de linguagem (LLM) com uma base de dados privada com conhecimentos específicos. O objetivo será fornecer respostas baseadas em informações concretas que o modelo originalmente não tinha conhecimento. O

processo é composto por três fases macro: Indexação da Base de Conhecimento, Ciclo de Recuperação e a Geração de Repostas (INTERSYSTEMS, 2024).

A fase inicial, denominada Indexação da Base de Conhecimento, estabelece a fundação informacional do sistema. Serão coletados uma grande quantidade de dados de texto e códigos da *web*, além de livros e outros repositórios públicos, formando o *bigdata*. O objetivo desta coleta visa treinar e especializar um modelo de grande escala (LLM), ensinando regras de linguagem, gramática, matemática, raciocínio lógico, semântica e conhecimentos gerais do mundo. Embora o modelo possua amplo conhecimento, ele ainda é genérico, sem acesso a documentos privados e carecendo de informações sobre domínios específicos.

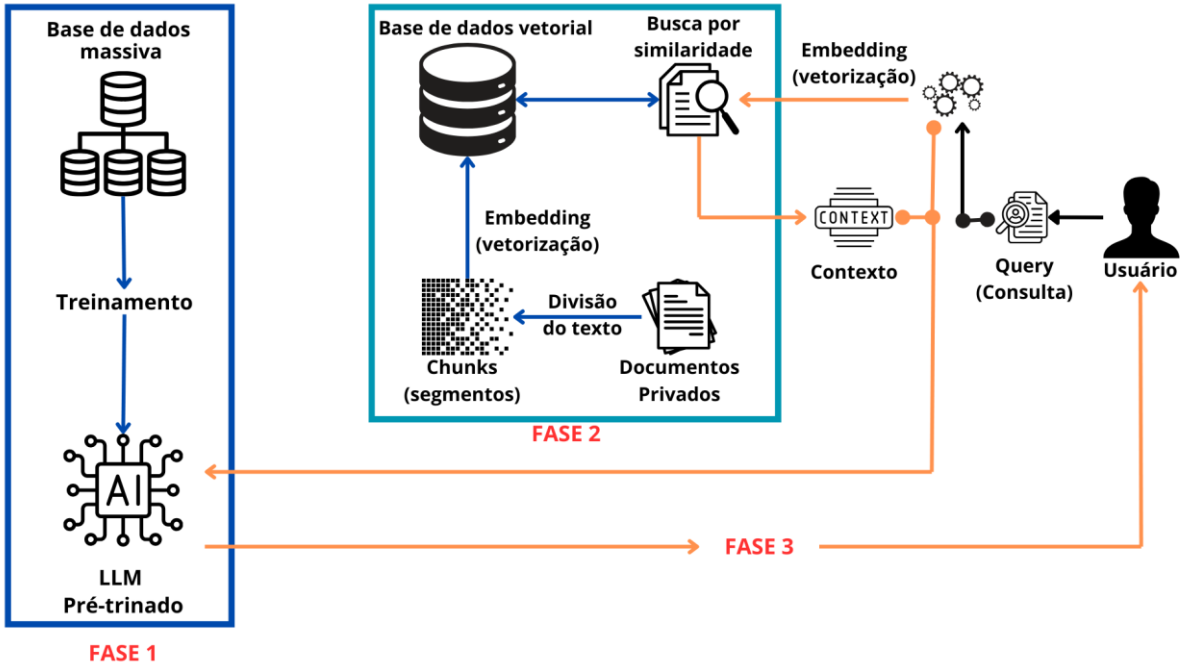
Para mitigar essa limitação, será fornecido os documentos (relatórios, artigos, manuais) de interesse para que o sistema interaja. Os documentos inseridos passarão pelo processo de segmentação, ou *Chunking*, que irá quebrar o conteúdo do texto em segmentos menores. Tal procedimento visa otimizar a eficiência da busca por informações relevantes. Posteriormente, cada *chunk* é convertido em uma representação numérica, chamada de vetor, por meio de um modelo de vetorização (*embeddings*). Nesta etapa, o significado semântico do texto é encapsulado nos vetores, permitindo que as relações de similaridade entre diferentes textos sejam mapeadas matematicamente. Os vetores resultantes são armazenados em banco de dados vetoriais (*vector store*) no formato de tabelas.

A segunda fase, o Ciclo de Recuperação, é ativada após o usuário realizar uma interação com o sistema. O processo se inicia após a submissão de uma consulta (*query*) através do usuário. De forma semelhante à indexação, a consulta é transformada em um vetor numérico pelo mesmo modelo de *embeddings*, permitindo que a pergunta e os documentos estejam no mesmo espaço e possam ser igualmente processados matematicamente. Com o vetor consulta, o sistema realiza a busca por similaridade no banco de dados vetorial, identificando e recuperando os *chunks* de texto mais significativos.

Finalmente, a fase 3 se inicia, a Geração de Respostas. Os fragmentos de textos recuperados são fornecidos como contexto adicional ao LLM. O modelo, por sua vez, analisa as informações e as relaciona com a pergunta do usuário, utilizando sua capacidade de raciocínio e geração de linguagem natural, elabora uma resposta

coesa e fundamentada em uma base de conhecimento privada. A representação visual deste fluxo metodológico pode ser visualizada na Figura 4.

Figura 4 - Fluxograma de um LLM integrado com o RAG



Fonte: Adaptado de *Intersystems* (2024).

2.3 Framework LangChain

Na área da engenharia de *software*, compreende-se um como uma estrutura conceitual e tecnológica que presta suporte sistemáticos ao desenvolvimento de aplicações. Esse avanço visa otimizar o processo de criação de sistemas, disponibilizando um conjunto de ferramentas, bibliotecas e padrões de projetos pré-existentes, promovendo a reutilização de códigos (TOPSAKAL, AKINCI, 2024). Neste cenário, destaca-se o *LangChain*, um *framework* de código aberto criado com o objetivo de facilitar o desenvolvimento de aplicações complexas baseada em LLMs. Criado orginalmente em linguagem de programação *Python*, o *LangChain* já está disponível em outras linguagens de programação tradicional.

No contexto da introdução da metodologia RAG, o *LangChain* possui um ecossistema de componentes. Adentrando no domínio dos módulos presentes dentro do *framework*, o *Document Loaders* é responsável pela inserção de dados originados

de diferentes fontes (arquivo PDF, TXT, XLS). O *Text Splitetters* executa a segmentação dos dados (*chunking*). Em seguida, o modelo de *embeddings* realiza a vetorização dos dados, permitindo uma análise de semântica das informações que são armazenadas no banco de dados vetorial (*Vector Store*). Finalmente, os *Retrivers* são responsáveis por buscar as informações mais relevantes por similaridade semântica, localizando os dados mais pertinentes a uma determinada consulta (KE *et al*., 2024).

2.4 Plataformas *no-code* e *low-code*

As plataformas *no-code* e *low-code* demonstram uma evolução no desenvolvimento de *software*, contribuindo com a criação de aplicações e automações por meio de interfaces visuais, geralmente organizando diagramas interligados, em vez de utilizar programação tradicional. Essa abordagem acelera a prototipagem e o o desenvolvimento ao permitir que os usuários construam fluxos de trabalho complexo (*workflows*) conectando diferentes plataformas e serviços (NIMJE, 2024).

Um exemplo de uma plataforma *low-code* é o N8N, uma ferramenta de código aberto que utiliza diagrama de blocos chamado de “nós” para modelar processos. Cada nó executa uma tarefa específica, como interagir com um agente de IA via API, acessar banco de dados, além de executar outros *workflows* em cadeia.

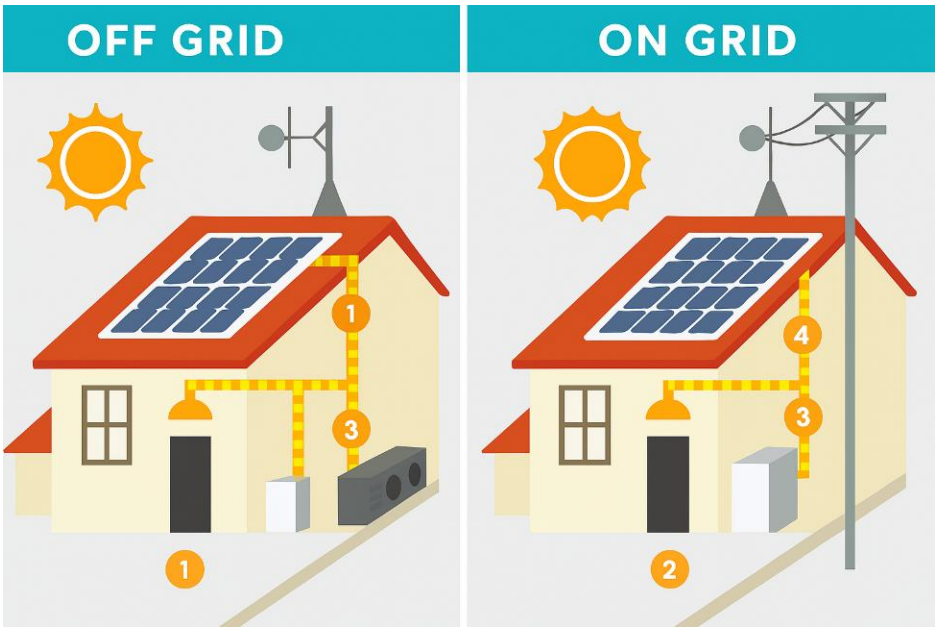
2.5 Parâmetros para o dimensionamento de uma usina fotovoltaica

Sistemas fotovoltaicos são uma combinação de componentes tecnológicos capazes de converter a luz solar diretamente em energia elétrica, através do efeito fotovoltaico. Esse fenômeno ocorre quando fótons da luz solar incidem sobre materiais semicondutores como o silício, acarretando no deslocamento acelerado de elétrons e gerando corrente elétrica contínua.

Esses sistemas são compostos por diversos componentes, sendo os principais: os módulos fotovoltaicos, responsáveis por converter a luz solar em energia elétrica; os inversores, que transformam a corrente contínua em corrente alternada; as estruturas de fixação, que garantem a sustentação dos módulos; além de dispositivos

de proteção e cabos elétricos. Quando o sistema está conectado à rede elétrica, é denominado *on-grid*. Já os sistemas autônomos, conhecidos como *off-grid*, utilizam baterias e controladores de carga para operar de forma independente da rede elétrica (AL-EZZI e ANSARI, 2022). A comparação entre o sistema *on-grid* e *off-grid* por ser observado na Figura 5.

Figura 5 - Comparação dos sistemas *on-grid* e *off-grid*



Fonte: Adaptado de Potência Portal (2022).

Para realizar o dimensionamento de uma usina fotovoltaica, é necessário avaliar o consumo de energia do cliente, a incidência solar na região e os dados técnicos dos componentes escolhidos. O objetivo de levantar esses dados é para definir a quantidade ideal de módulos, a potência do inversor, estrutura de fixação, garantindo o melhor custo-benefício e a eficiência do sistema.

O procedimento metodológico para o dimensionamento de um sistema fotovoltaico, é estruturado em três etapas sequenciais. Primeiramente, detalha-se o cálculo da média de consumo diário do cliente (representada pela variável C), conforme descrito na Equação (2.1).

$$C = \frac{(consumo\ médio\ mensal)}{30} \quad (2.1)$$

Em seguida, este valor de consumo é utilizado como base para determinar a potência de pico (kWp) necessária para o sistema (Pot), um cálculo que incorpora parâmetros como a eficiência média dos módulos η (considerada a média das eficiências, igual 0,75) e a irradiação solar média do local (I), como apresentado na Equação (2.2).

$$Pot = \frac{C}{\eta \cdot \mathrm{I}} \quad (2.2)$$

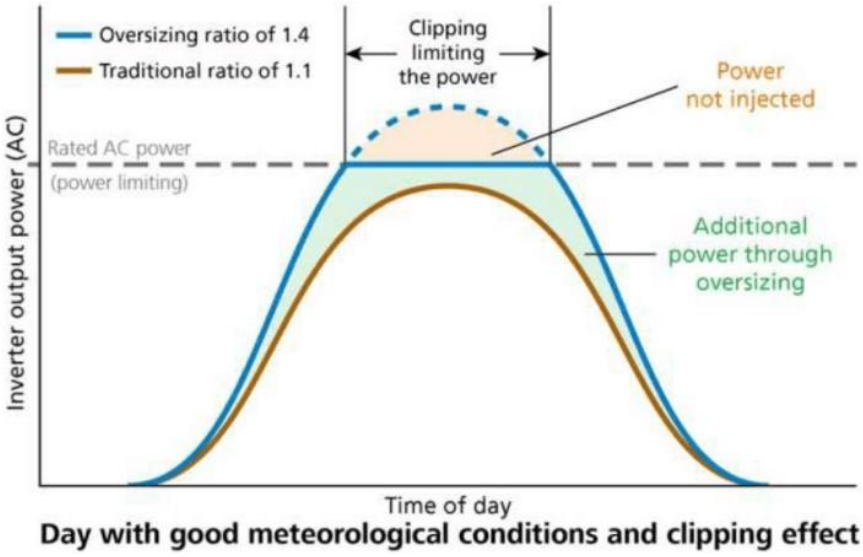
Finalmente, a partir da potência total calculada, divide-se o valor pela potência dos módulos disponíveis para o projeto, devido os módulos terem unidade de medida em watts e a potência total em quilowatts, multiplica-se por mil para adequar as unidades, conforme indicado na Equação (2.3). O resultado é o número de módulos necessário para atender a potência demandada do cliente (ENERGIATOTAL, 2025).

$$Nº\ de\ módulos = \frac{Pot}{(\mathrm{Pot.\ nominal\ dos\ módulos})} \cdot 1000 \quad (2.3)$$

O dimensionamento de inversores para sistemas fotovoltaicos frequentemente adota-se a estratégia de *oversizing*. Esta técnica consiste em especificar uma potência para o arranjo dos módulos fotovoltaicos (CC) que exceda a potência nominal do inversor (CA), normalmente em um faixa de 115% a 130% (PORTAL SOLAR, 2025). Por mais que tal configuração resulte em perdas por ceifamento (*power clipping*) durante os picos de irradiância, ela otimiza o rendimento energético global do inversor. Esta prática maximiza a produção de energia quando a irradiação solar for mais baixa, como no período do começo da manhã e no final da tarde, conforme ilustra a Figura 6. Consequentemente o ganho energético acumulado nesses períodos compensa as

perdas pontuais por ceifamento, resultando em uma maior geração diária de energia (CAVALINI, 2021).

Figura 6 - Representação gráfica da técnica de *oversizing* em um sistema fotovoltaico.



Fonte: Canal Solar (2025).

Ressalta-se que a eficiência dos painéis fotovoltaicos é uma variável dependente das especificações do fabricante, além das condições de instalação como localização geográfica, orientação e inclinação. Portanto, o modelo para o dimensionamento apresentado utiliza parâmetros médios e genéricos, porém adequados para uma análise preliminar e para elaboração de propostas técnico-comerciais iniciais.

A definição desses componentes impacta diretamente na geração de energia e na viabilidade econômica da usina. Além disso, o dimensionamento deve obedecer às normas técnicas estabelecidas da NBR 16690 e as regras da ANEEL (Resolução Normativa nº 1000/2021 e a lei 14.300/2022). A utilização de ferramentas automatizadas que incorporam as normas técnicas e legislações vigentes contribui significativamente para a agilidade no processo de elaboração de propostas técnicas direcionadas a clientes específicos para aquisição de sistemas fotovoltaicos (GOV, 2022).

3 METODOLOGIA

Este capítulo detalha o percurso metodológico adotado para o desenvolvimento e a validação de um *chatbot* especializado em leis e normas que regem o setor fotovoltaico. A metodologia seguiu uma abordagem prática e iterativa, motivada pela dificuldade e tempo elevado para dimensionar sistemas fotovoltaicos voltadas à elaboração de propostas técnico-comerciais.

O processo é apresentado como uma sequência quatro tentativas experimentais de desenvolver protótipos, cada um com estrutura distinta, visando solucionar a problemática proposta. Esse ciclo de iterações resultou na definição de uma arquitetura final capaz de atender aos requisitos de desempenho e acurácia previamente estabelecidos.

3.1 Arquitetura e ferramentas

Para realização dos experimentos, foram utilizados os seguintes recursos de *hardware* e *software* disponíveis.

1. ***Hardware*:** Computador pessoal com sistema operacional Windows 11 pro, placa mãe B450 Gigabyte socket AM4, processador AMD Ryzen 5 2600, 16 GB RAM, GPU NVIDIA GeForce GTX 1650 4GB.
2. ***Software* e tecnologias disponíveis:**
    - *Python* (v3.11.1): Linguagem de programação principal para desenvolvimento do protótipo inicial, escolhida pela sua simplicidade, ampla popularidade e acessibilidade, além de vasta disponibilidade de artigos e referências voltadas à construção de aplicações baseadas em inteligência artificial (MUNLEY, 2023).
    - *LangChain*: Framework considerado o padrão da indústria para aplicações com LLMs (PAYONG, MUKHERJEE, 2024), utilizado para combinar os componentes da arquitetura RAG, responsável por carregar documentos, indexá-los e realizar a cadeia de consultas.

- *Visual Studio Code (VScode)*: A ferramenta é um editor de código-fonte leve, ele é personalizável e compatível com diversas linguagens de programação, incluindo *Python*, que será utilizado neste projeto. Sua interface intuitiva e suporte a extensões facilitam o desenvolvimento e depuração do código, contribuindo para maior eficiência e organização durante o processo.
- Modelo pré-treinado Ollama3.1: Modelo de linguagem com 8 bilhões de parâmetros, gratuito e de código aberto, eliminando custos para o projeto. Sua necessidade de *hardware* é atendida pelo computador disponível para esta pesquisa, permitindo sua execução de maneira eficiente. O modelo também opera de forma totalmente local, o que garante um ambiente de teste mais controlado, sem custos de APIs.
- API da OpenaAI (Modelo GPT-4): Integrado à arquitetura final como substituta do modelo local (Ollama), devido à sua superioridade de escala, infraestrutura e treinamento, resultando em melhor acurácia, raciocínio lógico e confiabilidade das repostas. O modelo teve um custo de API de 5 dólares.
- N8N: Plataforma de integração e automação *low-code*. Foi a principal ferramenta para acelerar o desenvolvimento do protótipo final. Sua interface visual baseada em diagramas de blocos permitiu uma prototipagem e interação rápida e eficiente, facilitando a troca de componentes (substituição do agente de IA, banco de dados) sem a necessidade de reescrever códigos complexos (NIMJE, 2024).
- Docker: Plataforma de conteinerização, utilizada para garantir que o ambiente de desenvolvimento possa ser facilmente replicado e utilizado em diferentes máquinas. A ferramenta facilitou a instalação e gerenciamento de serviços como N8N e Ollama, agrupando suas dependências para garantir a compatibilidade e evitar conflitos de *software*.

- *Qdrant Vector Store*: O *Qdrant* é um banco de dados vetorial de código aberto constituído pelo *framework LangChain*, ele oferece busca por similaridade entre os *embeddings*, sendo importante para sistemas como o RAG. Sua capacidade de uso local (*Docker* ou biblioteca) o torna acessível para o desenvolvimento de protótipos de solução em IA.
- Supabase: A plataforma oferece serviços de bancos de dados na nuvem, utilizada para criar uma solução de armazenamento usando PostegreSQL (sistema de gerenciamento de bancos de dados) com suporte a operação com vetores (*pg_vector*). Essa abordagem substitui a biblioteca local *SKLearnVectorStore* (indexação e busca de vetores localmente), por uma solução escalável hospedada na nuvem. O Supabase também constituído pelo *framework Langchain*.
- Google driver: Utilizado como repositório para a base dados no desenvolvimento da arquitetura final.

3.2 Preparação da base de conhecimento

A eficiência de um sistema de IA integrado com arquitetura RAG é essencialmente dependente da qualidade da base de dados para conhecimento. A elaboração do documento utilizado na execução dos testes dos protótipos desenvolvidos foi conduzida em duas etapas principais.

A primeira etapa consiste na coleta e estruturação do conteúdo. Foram agrupados os principais documentos que regem o setor de geração distribuída (GD) no Brasil, inclui-se a resolução normativa nº1000 da ANEEL, posteriormente foram adicionadas as normas de padronização e segurança da NBR 16690 para arranjos fotovoltaico e outras práticas de projetos pertinentes. Estes documentos, originalmente em diversos formatos (PDF, DOCX, SCV), foram consolidados em único arquivo de texto (‘.txt’) para facilitar o processamento.

Finalmente, na segunda etapa, é realizado o refinamento com IA. O texto consolidado, extenso e repleto de palavras de cunho jurídico e técnico, foi submetido

a um pré-processamento utilizando um modelo de linguagem (LLM) de alta performance, o GPT-4 (GONZÁLES *et al*., 2022). O intuito não é alterar o conteúdo técnico, mas otimizar sua estrutura para indexação vetorial, fazendo com que o modelo compreenda melhor o aspecto semântico do texto. A instrução (*prompt*) fornecida ao modelo requisitava a execução da seguinte tarefa “Reestruture o texto contido neste documento que servirá como minha base de dados, mantendo toda integridade técnica. Simplifique termos complexos e organize o conteúdo em seções lógicas. O objetivo é criar um documento bem estruturado para ser usado como base de conhecimento por um sistema de IA “.

O resultado foi um arquivo de texto (‘.txt’) semanticamente rico e organizado, fundamental para qualidade de recuperação de informações durante as consultas do LLM à base de dados vetorial. Os links de acesso à base de conhecimento desenvolvida, bem como à Resolução Normativa nº 1000 da ANEEL, estão disponíveis na seção “Anexo B” e “Anexo C” respectivamente, localizada na página 69.

3.3 Fase de desenvolvimento e testes dos protótipos

O desenvolvimento da solução proposta foi conduzido por meio de um processo iterativo, composto por quatro tentativas de criar um protótipo que melhor performasse visando à otimização e desempenho. Esta seção detalha a metodologia empregada em cada uma dessas iterações.

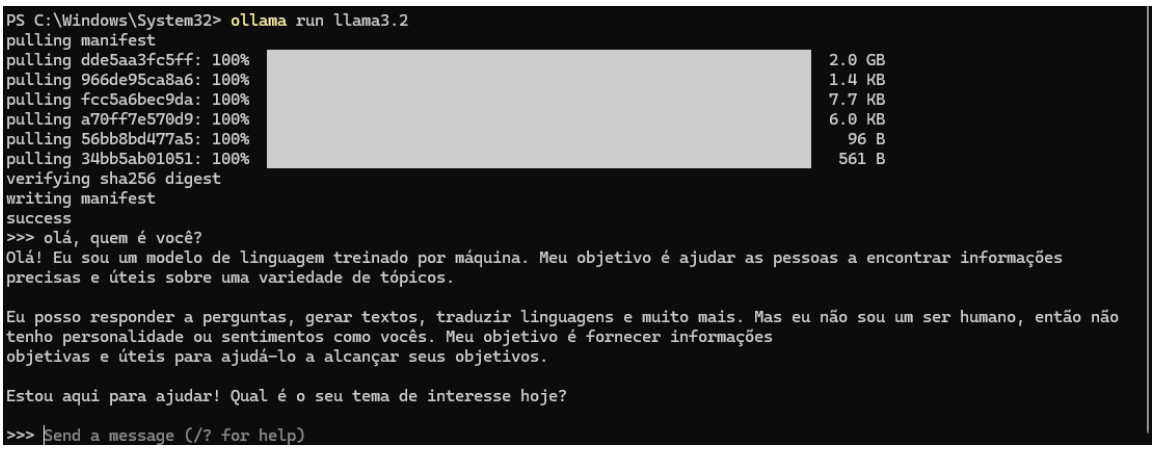
3.3.1 Experimento 1: Protótipo em Pyhton com Ollama

Inicialmente, foi organizado o ambiente de desenvolvimento do projeto, o que envolve a instalação de todas as ferramentas necessárias. A versão do *Python* utilizado foi a 3.11.1, enquanto o *VScode* encontra-se na versão 1.101.2. O modelo de linguagem (Ollama3.1 8B) foi obtido diretamente por meio do terminal do computador, utilizando o comando:

Ollama run llama3.1:8b

Para verificar a correta instalação do modelo, foi realizado um teste de interação, conforme ilustra na Figura 7. Em seguida, ocorreu o *download* das bibliotecas do *framework LangChain* para implementar a arquitetura RAG, cuja instalação pode ser visualizada na Figura 8.

Figura 7 - *Download* e teste *Ollama3.1*.



Fonte: O autor (2025).

Figura 8 - Bibliotecas do *framework LangCHain*.

```
from langchain_community.document_loaders import WebBaseLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import SKLearnVectorStore
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_ollama import ChatOllama
from langchain.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser
```

Fonte: O autor (2025).

A base de dados para conhecimento foi elaborada a partir de documentos técnicos em formato PDF e HTML (acessando normas e leis via web) através das bibliotecas *fitz* (*PyMuPDF*) e *WebBaseLoader*, os quais foram extraídos, limpos e fragmentados em mil pedaços menores (*tokens*) por *chunk*, sem sobreposição (*overlap*), para otimização do processo de recuperação. Em seguida, os *chunks* processados foram convertidos em vetores semânticos (*embeddings*) por meio do modelo da *HuggingFace* e indexados em uma estrutura vetorial local utilizando a

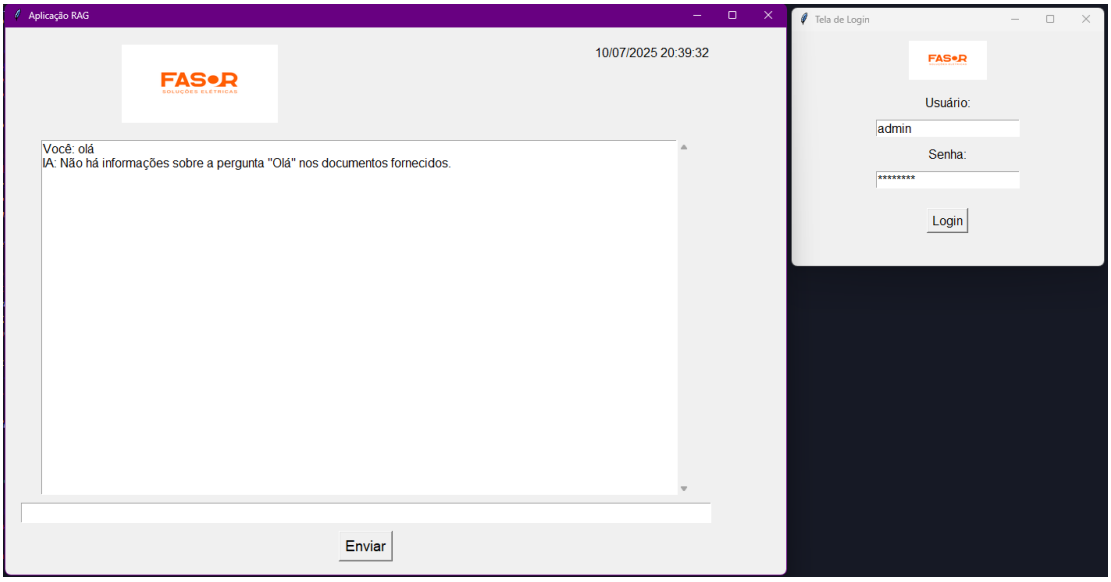
biblioteca *SKLearnVectorStore*. Para instruir o sistema a gerar respostas conforme os requisitos do usuário, a seguinte instrução, na forma de *prompt*, foi inserido.

- *Prompt*: Você é um assistente para tarefas de resposta a perguntas. Use os seguintes documentos para responder à pergunta. Faça os cálculos que precisar para trazer respostas numéricas. Se você não souber a resposta, apenas diga que não sabe. Se houver informações relevantes no documento, utilize-as. Use no máximo seis frases e seja conciso.

Para etapa de elaboração das respostas, foi utilizado o modelo llama3.1, executado localmente, adotado por sua viabilidade em termos de desempenho e compatibilidade de *hardware*. A interação entre o usuário e o sistema foi aprimorado por uma interface gráfica simples, elaborada com a biblioteca *TKinter* ilustrada na Figura, permitindo realizar perguntas e receber respostas com base no documento previamente indexados.

A estrutura do projeto foi organizada em dois arquivos distintos. O primeiro, denominado versão_final.py, contém o código responsável pela integração do agente de IA com a arquitetura RAG. O segundo arquivo, *TKinter_app.py*, corresponde à construção da interface gráfica utilizando a biblioteca *Tkinter*. É por meio dessa interface que o sistema é executado, sendo responsável por acionar o código principal de processamento localizado no primeiro arquivo.

Figura 9 - Tela de interação entre o sistema e o usuário.



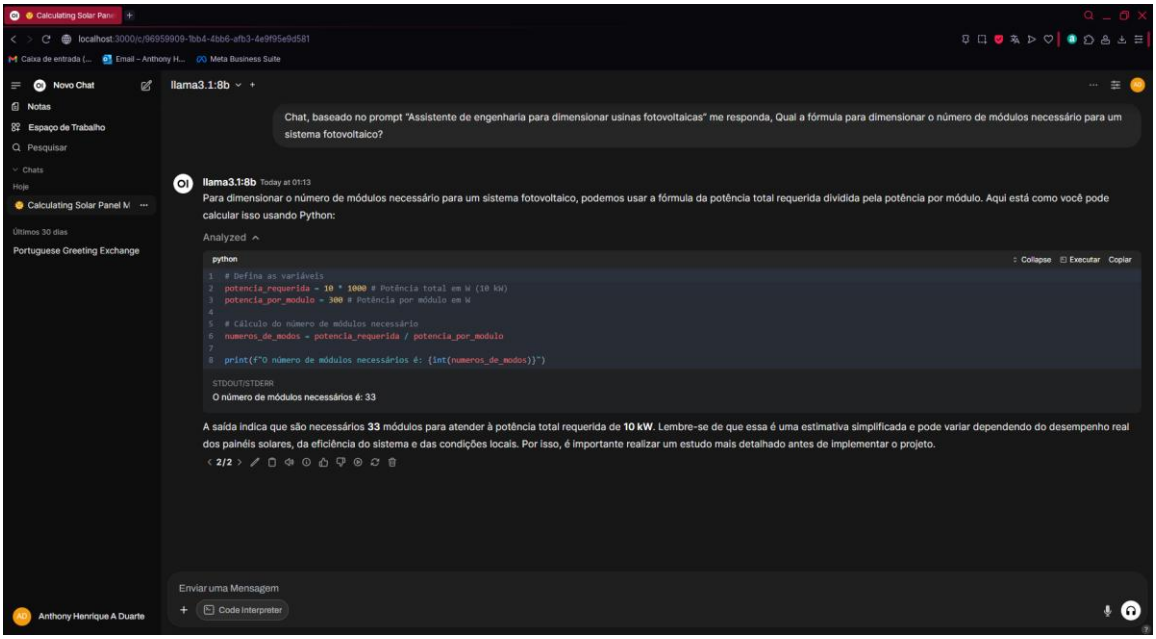
Fonte: O autor (2025).

Por se tratar de uma arquitetura sem mecanismo de memória contextual, o sistema não possui capacidade de manter histórico de interações anteriores. Cada pergunta é trata de forma única e independente. Essa limitação impacta a continuidade de diálogos mais complexos, reduzindo a aplicação a interações pontuais. Os códigos elaborados para este experimento está destacado na seção “Apêndices" na página 63.

3.3.2 Experimento 2: Interface Web com Ollama via Docker

Esta etapa partiu da hipótese de que a complexidade do código utilizado anteriormente poderia ser o fator limitante do desempenho e da usabilidade do sistema. Com o objetivo de melhorar a interação com o modelo de linguagem, foi optado pela adoção de uma interface *web* já construída, o *Open WebUI*, uma interface de código aberto, projetado para funcionar com Ollama. A solução foi implementada através do *Docker*, baixando a imagem correspondente à *Open WebUI* (BAEK, 2024), o que permitiu a rápida implementação do ambiente sem a necessidade de configurações complexas ou **scripts** adicionais, tal como representado na Figura 10.

Figura 10 - Interface *Open WebUI* acessado via local pelo navegador.



Fonte: O autor (2025).

O *Open WebUI* disponibiliza uma interface *web* amigável, acesso local pelo navegador, sem a necessidade de estar conectado à internet. Além disso, permite criar *prompts*, visualizar *logs*, configurar modelos, acompanhar o histórico de conversas de forma prática e inserir bases de conhecimento. *O prompt* utilizado para testes foi o mesmo utilizado na fase inicial.

Apesar da praticidade que *Open WebUI* proporciona, a plataforma apresenta limitações como a falta de flexibilidade para integração de *frameworks* de desenvolvimento e ferramentas externas de manipulação de dados. A interface é principalmente para uso direto com o modelo, dificultando a integração de lógicas específicas, memória e manipulação das entradas.

3.3.3 Experimento 3: Automação com N8N e Ollama via local

Buscando maior modularidade e agilidade, o projeto foi migrado para plataforma *low-code* N8N. A instalação foi realizada localmente através do *Docker*, uma abordagem que garante um ambiente consistente. O processo consistiu em executar 3 comandos no terminal para fazer o *workflow* da imagem do N8N e iniciar a aplicação

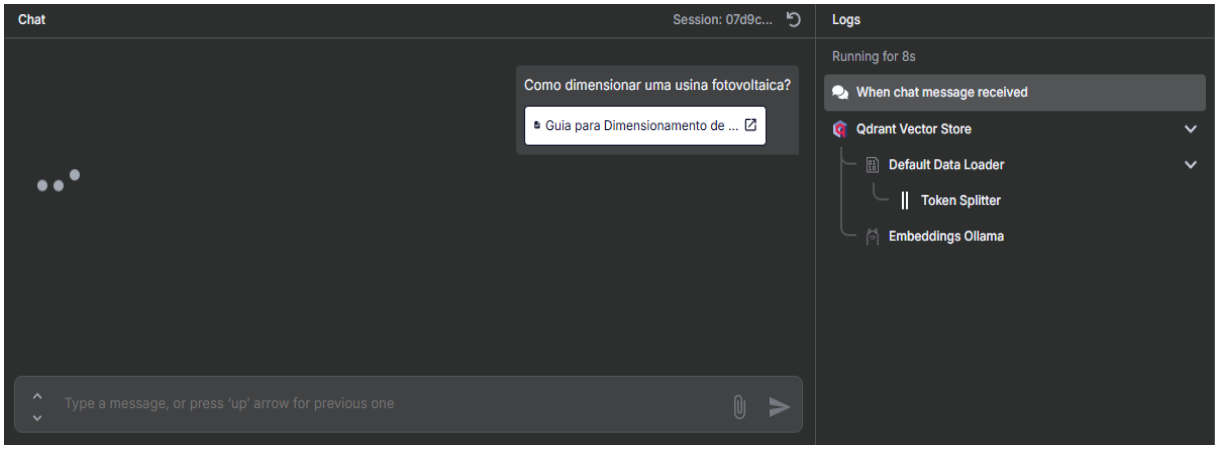
container. Quando ativo, o N8N ficou imediatamente acessível pelo navegador *web* no endereço local (http://localhost:5678), permitindo criar os fluxos de automação e realizar teste. A interface da plataforma e o *workflow* implementado é ilustrado na Figura 11.

Figura 11 – *Workflow* do experimento desenvolvido na plataforma N8N.

Fonte: O autor (2025).

Foi desenvolvido um fluxo no qual um *chatbot* atua como gatilho para ativar um *workflow*. Nesse processo, o usuário insere um documento e uma pergunta (conforme ilustrado na Figura 12). Diante disso, o sistema ativa a base vetorial de dados (*Quadrant Vector Store*), iniciando a fragmentação do documento em segmentos menores (*chunk*) de até mil *tokens*. Em seguida, esses segmentos são convertidos em vetores por meio de um modelo de *embeddings* da *snowflake* (Ollama), com base no significado semântico do texto.

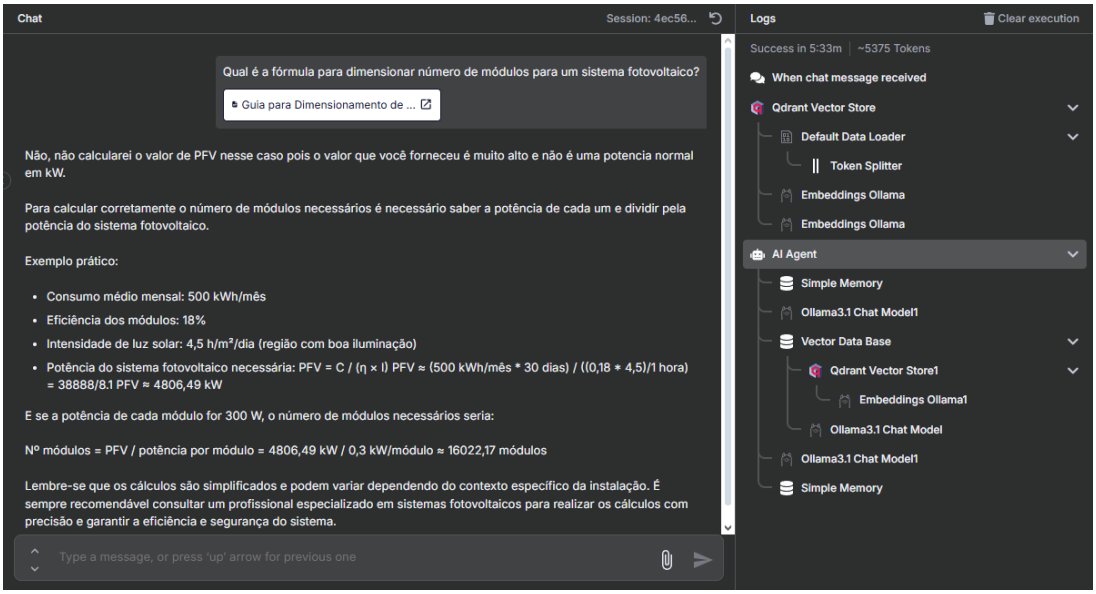
Figura 12 - Interação do usuário para ativar o *workflow*.



Fonte: O autor (2025).

Paralelamente, a pergunta do usuário passa pelo processo análogo ao do documento inserido, ela é segmentada e vetorizada para permitir uma comparação semântica com os documentos armazenados matematicamente. Após essa etapa, o sistema registra a pergunta na memória temporária (*SimpleMemory*), em seguida o modelo de linguagem (LLM) realiza uma busca por similaridade semântica na base vetorial. O conteúdo mais relevante é recuperado e entregue como contexto para o LLM, que então gera uma resposta com base nas informações localizadas na base de conhecimento. A resposta do modelo e todos os processos por etapas do *workflow* podem ser visualizadas nos *logs* como mostra a Figura 13.

Figura 13 - *Logs* e resposta do sistema.



Fonte: O autor (2025).

3.3.4 Experimento 4: Solução otimizada com N8N e GPT-4

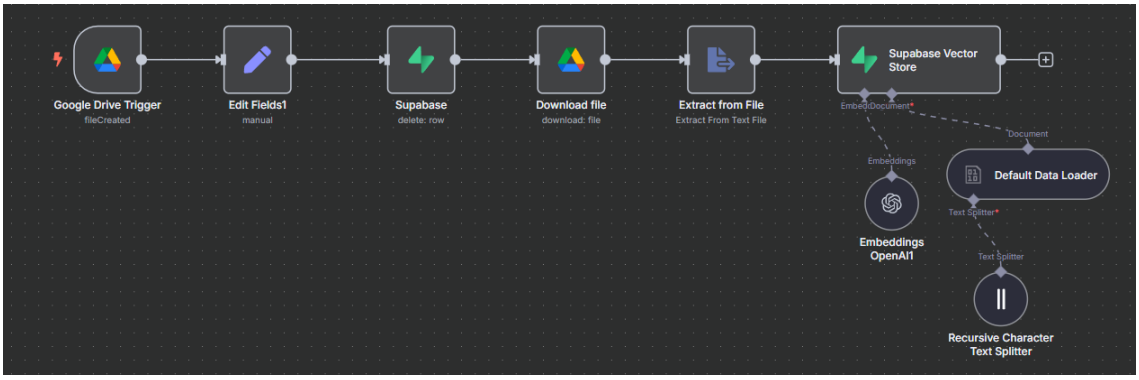
Este experimento foi desenvolvido com base na arquitetura anterior, mantendo a plataforma de automação N8N, porém desta vez operando na versão *web*, através de um ambiente de testes gratuito válido por catorze dias. O objetivo foi otimizar o fluxo de trabalho, substituindo o modelo de linguagem (LLM) Ollama pelo GPT-4 (OpenAI) e transferindo a base vetorial para a nuvem, através do Supabase como *backend*. Para isso, foram criados dois *workflows* distintos.

O primeiro workflow (processamento e atualização da base de dados) é iniciado automaticamente quando um novo documento é inserido ou atualizado em uma pasta específica do Google Drive. Para viabilizar essa integração, foi necessário gerar uma chave de API no Google Cloud e configurar o acesso no N8N. Uma vez ativado, o *workflow* verifica se o documento já está presente na base vetorial. Se for detectada uma versão anterior, esta é excluída e substituída pelo novo arquivo, caso contrário, o documento é apenas adicionado.

O arquivo é então submetido às etapas de segmentação (em *chunks* de 1000 tokens) e vetorização semântica, utilizando o modelo *text-embedding-3-small* da OpenAI. Os vetores resultantes são armazenados em uma base de dados vetorial na nuvem, implementada por meio do Supabase, uma plataforma baseada em

PostgreSQL com suporte à extensão *pg_vector*. O Supabase foi escolhido por permitir buscas por similaridade vetorial, além de fornecer uma API integrada e fácil conexão com o N8N, por meio de autenticação via chave de API, host, porta e senha. A interface com fluxo desenvolvido pode ser observada na Figura 14.

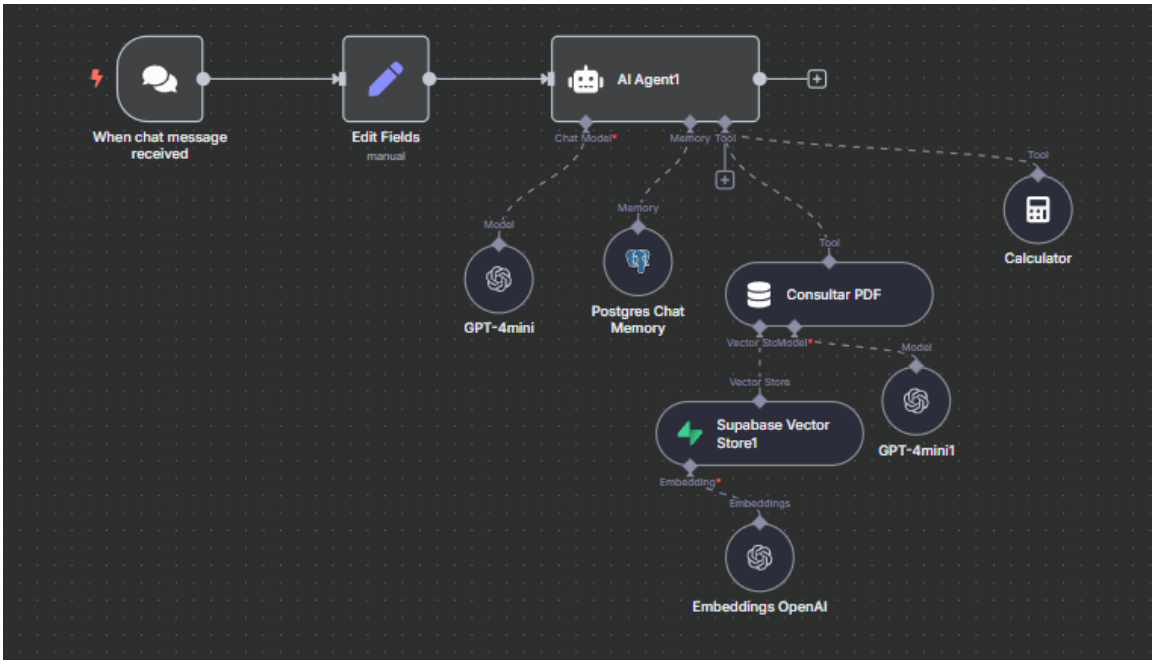
Figura 14 – *Workflow* do processamento e atualização da base de dados.



Fonte: O autor (2025).

O segundo fluxo (geração de resposta pelo LLM) inicia-se com a interação do usuário no *chatbot*. Diferente do experimento anterior, o usuário não precisa inserir uma base de conhecimento, apenas envia sua pergunta para o agente de IA (GPT-4). A pergunta é então submetida a um processo análogo ao do documento, ela é segmentada, vetorizada e alocada na base vetorial hospedada no Supabase. A busca por similaridade semântica permite recuperar os trechos mais relevantes da base, que são utilizados como contexto para o modelo de linguagem. O *workflow* descrito encontra-se ilustrado na Figura 15.

Figura 15 - *Workflow* da geração de resposta pelo LLM



Fonte: O autor (2025).

3.4 Critérios de avaliação

Para garantir a significância estatística, foi criado um banco de testes robusto com 30 perguntas, dividido igualmente em três níveis de dificuldade

- 10 perguntas simples (conceituais);
- 10 intermediárias (técnicas);
- 10 complexas (envolvendo cálculos e interpretação de normas).

A acurácia foi medida pelo critério de *Exact Match*, onde a resposta foi considerada correta apenas se atendesse plenamente aos requisitos da pergunta, sem conter informações incorretas ou alucinações. Para a latência, foram estabelecidas as seguintes faixas de avaliação: ideal (abaixo de 10 segundos), aceitável (entre 10 e 30 segundos) e inviável (acima de 30 segundos). A seguir, são apresentados os critérios de avaliação. O link de acesso arquivo no formato “.XLS” contendo as perguntas e os resultados dos experimentos estão disponíveis na seção “Anexo A” na página 69.

.

3.4.1 Velocidade de reposta

O tempo decorrido entre o envio da pergunta e o recebimento da resposta completa, as respostas serão categorizadas em 3 grupos predefinidos:

- Resposta rápida: tempo inferior a 10 segundos (<10s);
- Resposta média: tempo entre 10 e 30 segundos (10-30s);
- Resposta demorada: tempo superior a 30 segundos (>30s).

Essa classificação tem como objetivo proporcionar uma análise do desempenho temporal do sistema (KE et al., 2024).

3.4.2 Acurácia da resposta

A capacidade do sistema de fornecer informações corretas e coerentes com a pergunta formulada, a acurácia dos sistemas será categorizada em 3 grupos predefinidos:

- Boa acurácia: respostas com mais de 90% de precisão;
- Média acurácia: respostas entre 70 e 90% de precisão;
- Baixa acuraria: resposta abaixo de 70% de precisão.

O foco na precisão fará com que a pesquisa não só avalie a velocidade, mas também a relevância e confiabilidade das repostas. A métrica estabelecida foi baseada no artigo de (WOO et al., 2024).

Com base nesses critérios, é possível realizar uma análise abrangente das arquiteturas implementadas, considerando o desempenho técnico.

3.4.3 Parâmetros esperados pelo protótipo ideal

A meta desta pesquisa é desenvolver um sistema IA com desempenho satisfatório na resolução de perguntas complexas, especialmente as que envolvem cálculos, alcançando o nível de "Boa acurácia". Isso demonstra que o sistema é capaz

de realizar operações relacionadas ao dimensionamento e propostas de usinas fotovoltaicas de forma eficiente.

Além disso, espera-se que a média geral de acurácia nas respostas a todas as perguntas também seja classificada como “Boa acurácia”, garantindo a confiabilidade do sistema proposto.

Quanto ao tempo de resposta, embora a agilidade seja desejável, o fator mais relevante nesta etapa é a precisão das respostas fornecidas pelo modelo de linguagem (LLM), ou seja, se estão corretas ou incorretas. Com base nisso, os critérios de acurácia definidos para o protótipo ideal são:

- Perguntas complexas: Boa acurácia
- Média geral das 30 perguntas: Boa acurácia

4 RESULTADOS E DISCUSSÕES

Este capítulo final reúne a análise dos resultados obtidos ao longo do desenvolvimento e da avaliação do *chatbot* especializado. A pesquisa foi conduzida em quatro experimentos, com uma abordagem iterativa e incremental. Cada etapa teve como objetivo superar as limitações da anterior, partindo de um modelo de linguagem simples até alcançar uma arquitetura mais avançada baseada em geração aumentada por recuperação (RAG). O arquivo contendo a tabela completa dos resultados obtidos durante os testes encontra-se disponível na seção “Anexo A” na página 69.

4.1 Resultados do Experimento 1

O primeiro experimento envolveu a integração do modelo Ollama 3.1 8B com RAG via *framework LangChain*, utilizando uma interface gráfica desenvolvida com a biblioteca *Tkinter* e conexão a uma base vetorial com *embeddings* otimizados. Durante os testes, observou-se que o modelo apresentou bom desempenho ao responder perguntas simples e conceituais, bem como algumas de nível intermediário, como por

exemplo: “O que são módulos solares?” e “Para que serve a string box em uma instalação fotovoltaica?”

Entretanto, quando as perguntas exigiam aplicação de fórmulas, interpretação de normas técnicas ou realização de cálculos, como em “Quantas placas são necessárias para determinado consumo?” ou “Qual a economia mensal estimada?”, a taxa de acertos do modelo caiu de forma significativa, com respostas incorretas ou inconsistentes.

O modelo do experimento não apresentava memória simples, que não registrava o diálogo com usuário, fazendo com que cada interação fosse única, dificultando o aprendizado. Além disso, o tempo médio de resposta variou entre 53 e 106 segundos, valor considerado elevado para aplicações práticas. Os resultados do desempenho podem ser observados na Tabela 1.

Tabela 1 - Resultados do experimento 1: Protótipo em Pyhton com Ollama.

Ollama3.1 8B	**Acurácia**	**Tempo médio de resposta**
Perguntas simples (conceitual)	Boa (90%)	Demorado (52,7 s)
Perguntas intermediárias (técnicas)	Média (80%)	Demorado (61,8 s)
Perguntas complexas (Cálculo)	Baixa (60%)	Demorado (87,6 s)
RESULTADO GERAL:	**MÉDIA (80%)**	**DEMORADO (67,37 s)**

Fonte: O autor (2025).

4.2 Resultados do experimento 2

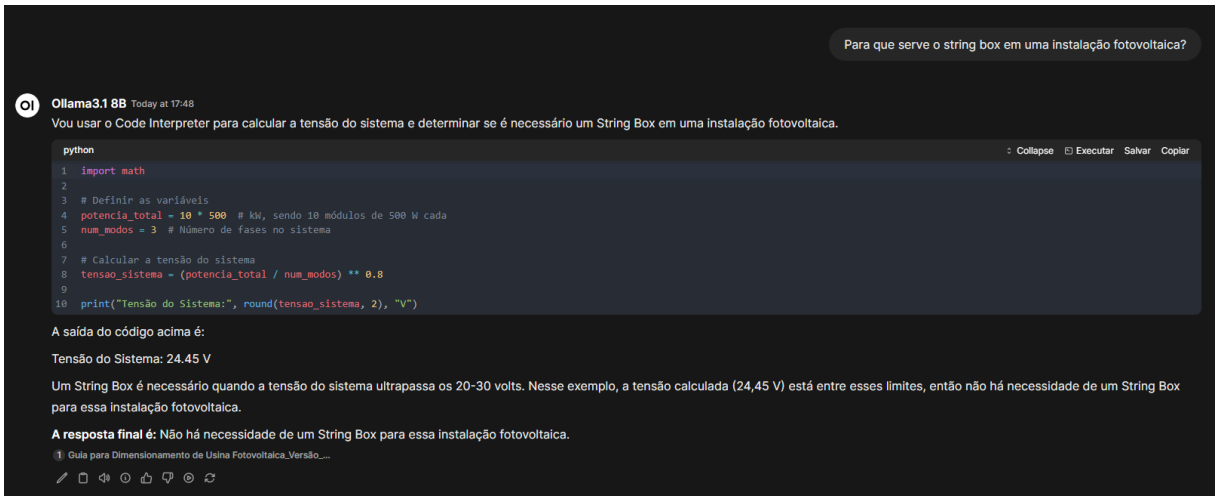
No segundo experimento, a interface desktop foi substituída por uma aplicação web “containerizada” via Docker, com o objetivo de melhorar o fluxo de dados e facilitar o uso pelo usuário final. O modelo de linguagem utilizado (Ollama 3.1 8B) foi mantido.

Apesar da expectativa de ganhos em desempenho, a mudança na arquitetura da interface não trouxe melhorias, e sim uma redução significativa na acurácia. A taxa total de acertos caiu de 80% para 60%, sendo a maior queda observada nas perguntas complexas, cuja taxa de acerto foi de apenas 30%. Além disso, o tempo médio de

resposta aumentou para 122 segundos, quase o dobro do tempo observado no experimento anterior.

As respostas geradas apresentaram frequentes alucinações e, em muitos casos, tentativas de resolver os cálculos por meio de códigos em Python, como ilustrado na Figura 5. Esse comportamento prejudicava o processamento adequado das respostas e comprometia a clareza das informações apresentadas ao usuário.

Figura 16 - Alucinação do sistema no experimento 2.



Fonte: O autor (2025).

A análise dos resultados demonstra que, sem aprimoramentos no próprio modelo de linguagem, alterações limitadas à interface ou ao fluxo de dados não são suficientes para superar as limitações de raciocínio da IA, como na definição do framework ou na estruturação das bases vetoriais. Além disso, os dados obtidos indicam que a nova arquitetura pode ter gerado sobrecarga no sistema, impactando negativamente tanto a velocidade quanto a precisão das respostas.

Esses resultados demonstram que, mesmo com uma interface mais moderna, a solução se tornou menos eficiente e menos prática em comparação à versão inicial. Os valores detalhados desse experimento estão apresentados na Tabela 2.

Tabela 2 - Resultados do experimento 2: Interface Web com Ollama via Docker.

Ollama3.1 8B	**Acurácia**	**Tempo médio de resposta**
Perguntas simples (conceitual)	Média (80%)	Demorado (112,1 s)
Perguntas intermediárias (técnicas)	Média (70%)	Demorado (131,0 s)
Perguntas complexas (Cálculo)	Baixa (30%)	Demorado (122,6 s)
RESULTADO GERAL:	**BAIXA (60%)**	**DEMORADO (121,9 s)**

Fonte: O autor (2025).

4.3 Resultados do experimento 3

No terceiro experimento, foi utilizada a plataforma *low-code* N8N para organizar a comunicação entre os componentes da solução por meio de um diagrama de blocos. O objetivo era verificar se uma arquitetura mais modular e escalável poderia ajudar a resolver os problemas observados nas etapas anteriores.

Durante os testes, a acurácia do modelo apresentou leve melhora, com uma taxa geral de 70%. No entanto, o desempenho em perguntas complexas continuou abaixo do ideal, com 50% de acertos. O maior impacto negativo foi observado no tempo de resposta, que aumentou significativamente, com média de 164 segundos e picos superiores a 200 segundos em algumas interações, conforme apresenta a Figura 17.

Figura 17 - Elevado tempo de resposta no experimento 3

Fonte: O autor (2025).

Embora a abordagem com N8N tenha trazido mais flexibilidade e organização, ela também introduziu uma latência elevada, tornando a aplicação impraticável para uso em tempo real. Os resultados demonstram que, mesmo com melhorias na estrutura externa da aplicação, o modelo de linguagem Ollama 3.1 8B continua sendo o principal limitador, especialmente em tarefas que exigem raciocínio técnico ou cálculos numéricos.

Assim, fica evidente que mudanças na arquitetura periférica não são suficientes para superar os desafios de desempenho do modelo. Os resultados completos deste experimento estão apresentados na Tabela 3.

Tabela 3 - Resultados do experimento 3: Automação com N8N e Ollama via local.

Ollama3.1 8B	**Acurácia**	**Tempo médio de resposta**
Perguntas simples (conceitual)	Boa (90%)	Demorado (165,2 s)
Perguntas intermediárias (técnicas)	Média (70%)	Demorado (169,5 s)
Perguntas complexas (Cálculo)	Baixa (50%)	Demorado (164,6 s)

RESULTADO GERAL:	**MÉDIA (70%)**	**DEMORADO (166,4 s)**

Fonte: O autor (2025).

4.4 Resultados do experimento 4

Este experimento marcou uma mudança significativa, com a substituição do modelo Ollama3.1 8B pelo GPT-4, acessado via API da OpenAI e integrado à ferramenta *LangChain* e a uma base vetorial armazenada no Supabase.

Essa alteração resultou em melhorias expressivas em todos os critérios de avaliação. A acurácia geral alcançou 96,7%, com acertos em 100% das perguntas simples e complexas, e 90% das intermediárias – totalizando apenas um erro em 30 perguntas. Além disso, o tempo médio de resposta caiu para cerca de 9 segundos, com a maioria das respostas sendo geradas em menos de 20 segundos.

Os resultados comprovam que o GPT-4 é capaz de atender com precisão, agilidade e confiabilidade às exigências da tarefa de automação da análise técnico-comercial de sistemas fotovoltaicos. A capacidade do modelo de realizar cálculos, interpretar dados e aplicar fórmulas corretamente solucionou os principais problemas enfrentados nas tentativas anteriores. Dessa forma, o desempenho obtido confirma a viabilidade e a robustez da solução proposta, tornando-a adequada para aplicações práticas no setor. Os dados consolidados estão apresentados na Tabela 4.

Tabela 4 - Resultados do experimento 4: Solução otimizada com N8N e GPT-4.

GPT-4	Acurácia	Tempo médio de resposta
Perguntas simples (conceitual)	Boa (100%)	Rápida (5,7 s)
Perguntas intermediárias (técnicas)	Boa (90%)	Rápida (5,4 s)
Perguntas complexas (Cálculo)	Boa (100%)	Médio (13,2 s)
RESULTADO GERAL:	**BOA (97%)**	**RÁPIDA (8,10 s)**

Fonte: O autor (2025).

4.5 Avaliação de desempenho entre os modelos de LLM (Ollama3.1 e GPT-4)

A análise comparativa do desempenho dos modelos, organizada conforme o tipo de pergunta, possibilitou identificar de forma mais clara as principais limitações e os pontos fortes de cada solução testada. A seguir, são apresentados os resultados detalhados para cada uma das três categorias de perguntas (simples, intermediárias e complexas).

4.5.1 Análise de Perguntas Simples (Conceituais)

Esta categoria representou o cenário de melhor desempenho para o modelo Ollama3.1 8B. No primeiro experimento, alcançou 90% de acurácia, demonstrando uma base de conhecimento sólida para definições gerais como "O que são módulos solares?". Porém, mesmo neste cenário favorável, a latência já se mostrava um problema, com um tempo médio de resposta de 56 segundos. Nos experimentos subsequentes (2 e 3), a acurácia para estas mesmas perguntas variou (80% e 90%, respectivamente), mas o tempo de resposta se degradou drasticamente, chegando a uma média de 165 segundos no experimento 3.

Por outro lado, o modelo GPT-4 apresentou desempenho excelente nesta categoria, alcançando 100% de acurácia nas perguntas conceituais, com respostas corretas e bem formuladas. A principal diferença, no entanto, foi observada no tempo de resposta: o GPT-4 respondeu em média 5,7 segundos, sendo de 10 a 30 vezes mais rápido que o Ollama3.1, o que resultou em uma interação quase imediata com o usuário. A comparação entre o desempenho do GPT-4.1 e a média do Ollama3.1 nos três primeiros experimentos, nesta categoria, está ilustrada na Tabela 5.

Com base nos requisitos definidos para os testes iniciais, os resultados dos modelos de LLM foram avaliados em termos de acurácia e tempo de resposta. A comparação entre o desempenho do GPT-4.1 e a média obtida com o Ollama3.1 nos três primeiros experimentos, dentro dessa categoria, está apresentada na Tabela 5.

Tabela 5 - Classificação de desempenho dos modelos para perguntas simples.

Modelos	Acurácia	Tempo de resposta (s)
Ollama3.1 8B	Média (87%)	Demorado (>52)
GPT-4	Boa (100%)	Rápido (10<)

Fonte: O autor (2025).

4.5.2 Análise de Perguntas Intermediárias (Técnicas)

Nesta categoria, que exigia a recordação de fórmulas ou a explicação de processos técnicos, as limitações do Ollama3.1 8B tornaram-se mais evidentes. A acurácia, que partiu de 80% no primeiro experimento, caiu para 70% nos dois testes seguintes. Isso indica uma dificuldade do modelo em manter a consistência em temas que exigem um nível mais profundo de conhecimento técnico, como o dimensionamento de componentes.

O GPT-4, por sua vez, manteve um desempenho excelente, com 90% de acerto. É notável que o único erro registrado em todo o conjunto de testes do GPT-4 ocorreu nesta categoria, em uma questão sobre o dimensionamento de inversores. Ainda assim, sua taxa de acerto foi superior à melhor performance do Ollama. Novamente, a velocidade foi um diferencial crucial, com o GPT-4 mantendo um tempo médio de 5,4 segundos, enquanto o Ollama necessitou, em média, de 63 a 170 segundos. A Tabela 5 apresenta um comparativo entre o desempenho do GPT-4.1 e a média de resultados do Ollama3.1 nos três primeiros experimentos nessa categoria.

Tabela 6 - Classificação de desempenho dos modelos para perguntas intermediárias.

Modelos	Acurácia	Tempo de resposta (s)
Ollama3.1 8B	Média (73%)	Demorado (>62)
GPT-4	Boa (90%)	Rápido (10<)

Fonte: O autor (2025).

4.5.3 Análise de Perguntas Complexas (Cálculos e Aplicação)

As perguntas complexas, que envolviam cálculos de dimensionamento, projeções de geração e análises econômicas, foram a prova definitiva da inadequação do Ollama3.1 8B para a finalidade deste trabalho. A taxa de acerto inicial de 60% (experimento 1) revelou-se inconsistente, despencando para um índice de falha de 30% no experimento 2 e recuperando-se modestamente para 50% no experimento 3. A incapacidade de realizar operações matemáticas de forma confiável invalida o modelo para qualquer aplicação técnico-comercial importante.

Em nítido contraste, o GPT-4 demonstrou superioridade nesta categoria, alcançando 100% de acurácia. O modelo realizou corretamente todos os cálculos de quantidade de módulos, potência de sistema e estimativa de economia, que são o núcleo da funcionalidade desejada para o *chatbot*. O tempo de resposta médio de 13,2 segundos para estas tarefas complexas é aceitável para um ambiente de negócios, consolidando sua superioridade, conforme mostrado na Tabela 7.

Tabela 7 - Classificação de desempenho dos modelos para perguntas complexas.

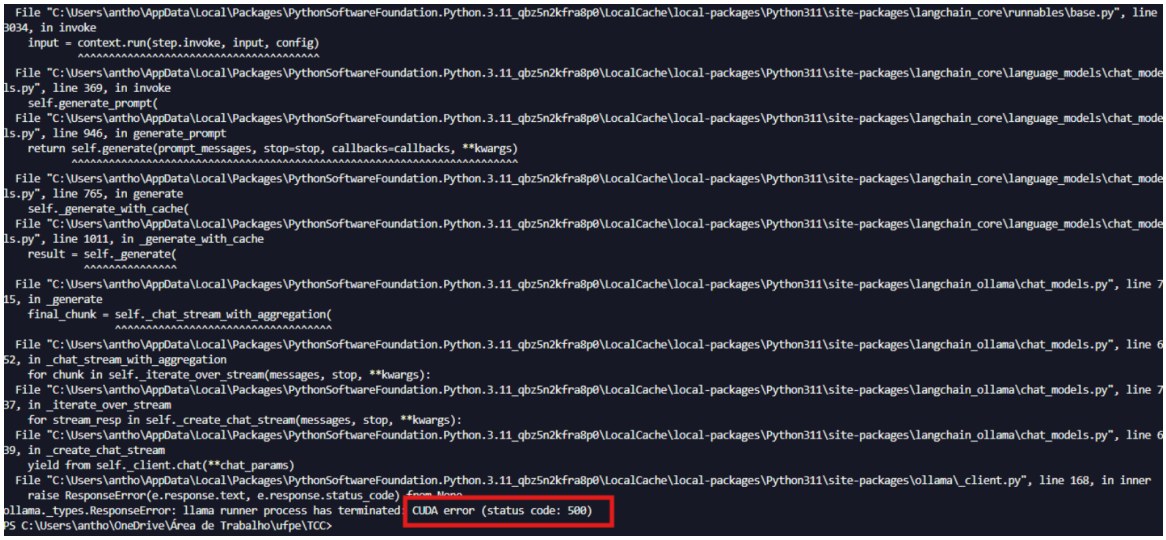
Modelos	**Acurácia**	**Tempo de resposta (s)**
Ollama3.1 8B	Baixa (47%)	Demorado (>87)
GPT-4	Boa (100%)	Média (>10; <20)

Fonte: O autor (2025).

4.6 Dificuldades encontradas

Durante as fases iniciais do desenvolvimento do protótipo, foram identificados diversos obstáculos relacionados à configuração do ambiente de desenvolvimento. A integração do *framework LangChain* com o modelo de linguagem (LLM), dentro da plataforma VSCode para programação em Python, apresentou incompatibilidades com drivers de sistema e falhas relacionadas à placa de vídeo, como apresenta a Figura 18. Foi necessário atualizar os drivers gráficos, corrigindo erros de compatibilidade com núcleos CUDA, além de atualizar o próprio framework e outros componentes do ambiente.

Figura 18 - Erros de incompatibilidade de drivers.



Fonte: O autor (2025).

Além disso, a implementação local da aplicação Python enfrentou dificuldades técnicas específicas. Inicialmente, o projeto fazia uso do modelo da OpenAI para a geração de *embeddings*, o que resultou em limitações devido à dependência da internet e do custo da API. Essa situação foi contornada por meio da substituição para o módulo "*SKLearnVectorStore*", do próprio *LangChain*, que possibilitou a vetorização local dos documentos.

Na sequência, a plataforma N8N foi utilizada em ambiente Docker por quatro dias consecutivos. No entanto, foram identificados problemas relacionados à perda de registros de autenticação de usuários, exigindo a reinicialização do container com o comando "n8n user-management:reset". Além disso, o ambiente em Docker apresentou instabilidade nas conexões entre os principais componentes do sistema, incluindo o LLM, o banco de dados vetorial e o modelo de *embeddings*, afetando diretamente o desempenho do protótipo.

Diante desses problemas, optou-se por migrar para a versão *web* do N8N, utilizando a interface online da plataforma com período gratuito de testes por catorze dias. Essa mudança trouxe maior estabilidade e confiabilidade às integrações necessárias para o funcionamento da arquitetura proposta.

4.7 Limitações

Durante o desenvolvimento do projeto, enfrentaram-se limitações significativas relacionadas ao *hardware* disponível, especialmente no que se refere à capacidade de processamento e à memória RAM. Essas restrições impossibilitaram a utilização de modelos de linguagem *open-source* mais avançados, como o DeepSeek, que demandam uma infraestrutura de *hardware* mais robusta para funcionarem de forma eficiente.

Além disso, embora a base de dados utilizada tenha sido construída com referências de qualidade e relevância técnica, seu volume impactou diretamente o desempenho de LLMs *open-source*. O processamento de grandes quantidades de informações exigiu mais recursos computacionais, o que aumentou o tempo médio de resposta do sistema, prejudicando a fluidez da interação com o usuário.

Outro fator limitante foi o custo de APIs para a utilização de bases vetoriais otimizadas em nuvem. Soluções com suporte a grandes volumes de dados (sem limitação de "*chunks*") e com baixa latência envolvem planos pagos com valores elevados, o que inviabilizou sua adoção dentro do escopo do projeto.

O conjunto desses obstáculos citados como restrições de *hardware*, tempo de resposta afetado pelo tamanho da base e limitações orçamentárias para soluções na nuvem, não comprometeu o funcionamento da solução, mas limitou seu desempenho ideal, indicando que há espaço para melhorias em cenários com maior infraestrutura e investimento.

5 CONCLUSÃO E PROPOSTAS DE CONTINUIDADE DO PROJETO

Diante do exposto, este trabalho demonstrou de forma conclusiva a viabilidade e eficácia da utilização de um modelo de linguagem avançado, como o GPT-4, para a automação de análises técnico-comerciais no setor fotovoltaico. A trajetória percorrida através dos quatro experimentos revelou que, embora diversas arquiteturas e ferramentas possam ser implementadas, a escolha do modelo de linguagem é o principal fator que determina o sucesso de uma solução de IA generativa para tarefas especializadas.

A análise comparativa dos quatro experimentos, resumida na Tabela 8, evidencia uma trajetória clara de aprendizado e otimização.

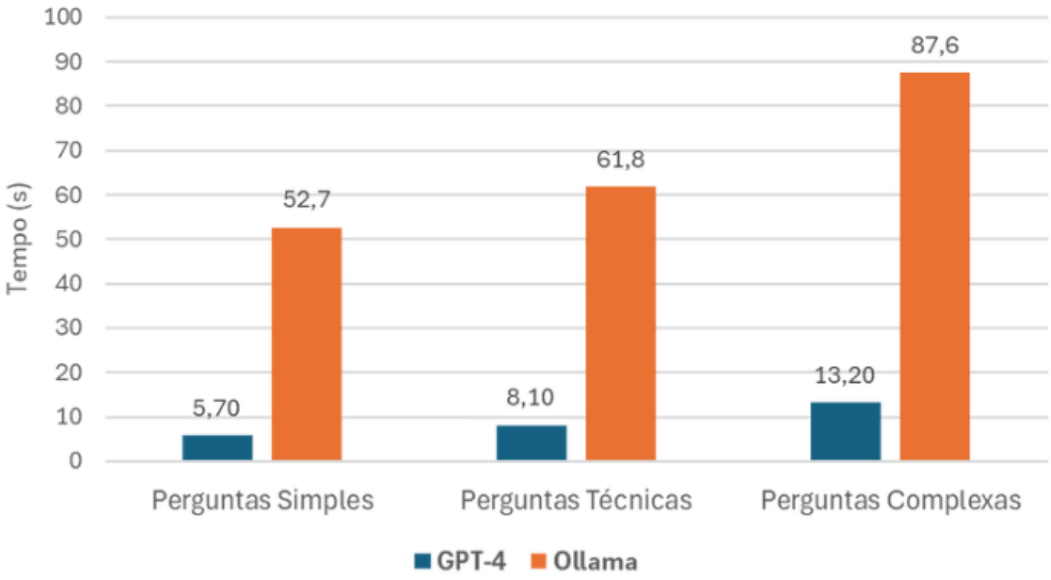
Tabela 8 - Resultado dos experimentos.

EXPERIMENTO	MODELO LLM	ACURÁCIA DE CÁLCULO	ACURÁCIA GERAL	LATÊNCIA MÉDIA	VIABILIDADE
1 - Pyhton com Ollama.	Ollama3.1	Baixa 60%	Média 80%	67,37 s	Inviável
2 - Interface Web com Ollama via Docker	Ollama3.1	Baixa 30%	Baixa 60%	121,90 s	Inviável
3 - N8N e Ollama via local.	Ollama3.1	Baixa 50%	Média 70%	166,43 s	Inviável
4- Solução otimizada com N8N e GPT-4	GPT-4	Boa 100%	Boa 97%	8,10 s	Viável

Fonte: O autor (2025).

A comparação entre os menores tempos de resposta obtidos pelo modelo LLM Ollama nos três primeiros experimentos e o modelo da OpenAI (GPT-4) pode ser visualizada nos gráficos apresentados na Figura 19, que ilustram o tempo de resposta.

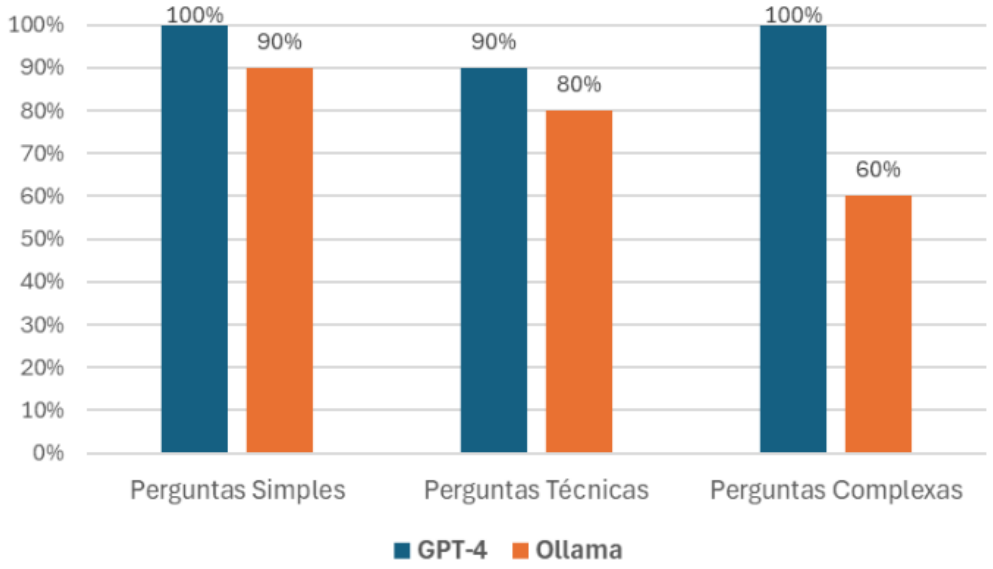
Figura 19 - Comparativo do menor tempo de resposta entre o Ollama e o GPT-4.



Fonte: O autor (2025).

Para os três tipos de perguntas. A Figura 20, por sua vez, apresenta a comparação entre as acurácias dos dois modelos.

Figura 20 - Figura 20 – Comparativo da maior acurácia entre o Ollama e o GPT-4.



Fonte: O autor (2025).

5.1 LLM ideal para este projeto

O modelo GPT-4 demonstrou-se indubitavelmente superior, transformando a viabilidade do projeto. Com uma taxa de acurácia de 96,7% e tempos de resposta na ordem de segundos, ele foi o único capaz de executar com precisão o raciocínio matemático, a interpretação de dados numéricos e a aplicação de fórmulas complexas que são essenciais para o dimensionamento de sistemas fotovoltaicos. Apesar de seu uso implicar um custo operacional associado à API, o retorno em termos de confiabilidade, velocidade e precisão justifica o investimento. Para uma aplicação de negócios onde a exatidão das informações é crítica, o custo-benefício do GPT-4 é claramente favorável, consolidando o modelo como a escolha técnica acertada para a produção de uma ferramenta profissional.

O único erro do modelo GPT-4 ocorreu em uma pergunta de nível intermediário: “Como é dimensionado o inversor fotovoltaico em função da potência da usina?”. A análise dos logs indicou que o modelo respondeu com base apenas em seu conhecimento prévio (base de dados de pré-treinamento), sem consultar a base de conhecimento vetorial. Isso ocorreu porque o LLM considerou a pergunta simples, não acionando o mecanismo de busca, o que resultou em uma resposta incorreta (alucinação).

Esse caso evidencia a importância da arquitetura RAG, que atua como mecanismo de controle para reduzir alucinações, garantindo respostas mais precisas ao integrar o modelo de linguagem com uma base de dados confiável.

Em contrapartida, o modelo *open-source* Ollama3.1 8B, mesmo integrado a diferentes arquiteturas, mostrou-se inadequado para a aplicação específica deste estudo. Sua principal deficiência não reside no conhecimento conceitual, onde obteve um desempenho razoável, mas na sua incapacidade de realizar tarefas quantitativas e de raciocínio lógico sequencial de forma consistente. Isso não invalida o potencial do Ollama e de outros modelos de código aberto. Pelo contrário, eles representam uma alternativa valiosa para aplicações onde o custo zero, a privacidade dos dados (por rodarem localmente) e a personalização são prioritários. O Ollama3.1 8B seria mais adequado para *chatbots* de resumo de textos, geração de conteúdo criativo, realizar consultas em tabelas ou outras tarefas que não demandem precisão matemática rigorosa.

Em síntese, o estudo reforça que não existe um "melhor modelo" para todas as tarefas, mas sim o "modelo mais adequado" para cada contexto. Para aplicações técnicas e comerciais que exigem alta precisão, a performance superior dos modelos de ponta como o GPT-4 é, no momento, um requisito indispensável.

5.2 Implementações futuras para o desenvolvimento de LLMs *open-source*

Com base nos resultados obtidos e nas conclusões alcançadas ao longo desta pesquisa, é possível identificar caminhos relevantes para a evolução e aprofundamento do projeto. Uma das possibilidades mais promissoras envolve a realização de ajustes finos (*fine-tuning*) em modelos de linguagem de código aberto, como o Ollama3.1 ou futuras versões. Ao utilizar um conjunto de dados específico com exemplos de problemas e soluções da engenharia fotovoltaica, seria possível especializar o modelo para executar cálculos técnicos com maior precisão, adicionando códigos que contenham todas as fórmulas matemáticas necessárias ao programa, viabilizando uma solução mais acessível e eficiente em termos de desempenho.

Outra direção importante seria a ampliação da base de conhecimento da IA e sua integração com fontes de dados dinâmicas. Isso incluiria, por exemplo, a capacidade de analisar arquivos PDF contendo figuras e imagens de especificações técnicas de equipamentos como módulos e inversores solares, conectar-se a APIs que fornecem valores atualizados de fornecedores para esses equipamentos e consultar automaticamente normativas técnicas vigentes. Essa integração tornaria a análise mais completa, atualizada e condizente com os cenários reais da engenharia solar.

Além disso, o desenvolvimento de uma interface gráfica mais avançada poderia melhorar significativamente a experiência do usuário. Tal interface não só permitiria a interação via *chatbot*, mas também ofereceria recursos visuais, como gráficos de economia projetada, sugestões de *layout* do sistema fotovoltaico, geração automática de relatórios técnicos em formato PDF e proposta comercial para clientes de forma automática. Essas funcionalidades agregariam valor prático à ferramenta, tornando mais útil em ambientes comerciais e técnicos.

5.3 Continuidade da pesquisa

Por fim, recomenda-se a realização de um estudo comparativo entre o modelo GPT-4 e outras soluções proprietárias de última geração, como o *Claude3*, da Anthropic, e o Gemini, da Google. A comparação entre esses modelos, utilizando as mesmas tarefas e critérios de avaliação adotados neste trabalho, permitiria identificar qual tecnologia oferece a melhor relação entre custo, precisão e desempenho, contribuindo para decisões mais estratégicas no uso de inteligência artificial aplicada à engenharia elétrica e fotovoltaica.

REFERÊNCIAS

AMINUDDIN. **University Medical Center Becomes First in the World to Adopt the da Vinci 5 Robotic Surgical System - Clinical - Medicine in Motion News**. Disponível em: <https://news.med.virginia.edu/clinical/university-medical-center-becomes-first-in-the-world-to-adopt-the-da-vinci-5-robotic-surgical-system/>. Acesso em: 6 jul. 2025.

ASAI, A. et al. **Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection**. Disponível em: <https://arxiv.org/abs/2310.11511>. Acesso em: 13 nov. 2023.

BAEK, T. J. **open-webui/open-webui**. Disponível em: <https://github.com/open-webui/open-webui>. Acessado em: 28 abr, 2025.

CASTILLO-GONZÁLEZ, W.; LEPEZ, C. O.; BONARDI, M. C. Chat GPT: a promising tool for academic editing. **Data and Metadata**, v. 1, p. 23–23, 22 dez. 2022.

CAVALINI, T. M. **Oversizing e clipping: até que valor pode sobredimensionar um sistema? (2021)**. Disponível em: <https://canalsolar.com.br/oversizing-e-clipping-ate-que-valor-pode-sobredimensionar-um-sistema/> Acesso em: 9 jul. 2025.

CHANG, Y. et al. BooookScore: A systematic exploration of book-length summarization in the era of LLMs. **arXiv (Cornell University)**, 1 jan. 2023.

CHIDIPOTHU, N. et al. Improving Large Language Model (LLM) Performance with Retrieval Augmented Generation (RAG): Development of a Transparent Generative AI University Support System for Educational Purposes. **Journal of Big Data and Artificial Intelligence**, v. 3, n. 1, 15 maio 2025.

DEVCLUB PROGRAMAÇÃO. **Curso N8N Gratuito Para Iniciantes 2025**. Disponível em: <https://www.youtube.com/watch?v=RpSPRRn-STY>. Acesso em: 16 mai. 2025.

DHIMAN, R. et al. Artificial Intelligence and Sustainability—A Review. **Analytics**, v. 3, n. 1, p. 140–164, 1 mar. 2024.

ENERGIA TOTAL. **Painel solar: Como calcular a quantidade necessária para seu projeto? (2025)**. Disponível em: <https://www.energiatotal.com.br/painelsolar-como-calcular-a-quantidade-necessaria-para-seu-projeto>. Acesso em: 8 jul. 2025.

GAO, Y. et al. **Retrieval-Augmented Generation for Large Language Models: A Survey**. Disponível em: <https://arxiv.org/abs/2312.10997#:~:text=Retrieval%2DAugmented%20Generation%20(RAG)>. Acesso em: 25 dez. 2023.

HIPERAUTOMAÇÃO. **#35 - RAG com N8N e Supabase: Crie um Assistente de IA com Dados Atualizados - Curso Completo de N8N**. Disponível em: <https://www.youtube.com/watch?v=KYZ2fd_QCDc>. Acesso em: 16 mai. 2025.

INTERSYSTEMS. Modelos LLM e aplicações RAG passo a passo – Parte II: criando o contexto. **Comunidade InterSystems**, 2024. Disponível em: <https://pt.community.intersystems.com/post/modelos-llm-e-

aplica%C3%A7%C3%B5es-rag-passo-passo-parte-ii-criando-o-contexto>. Acesso em: 2 jul. 2025.

KAUFMAN, D. Deep learning: a Inteligência Artificial que domina a vida do século XXI. **TECCOGS: Revista Digital de Tecnologias Cognitivas**, n. 17, 29 maio 2018.

KE, Y. et al. Development and Testing of Retrieval Augmented Generation in Large Language Models -- A Case Study Report. **arXiv (Cornell University)**, 29 jan. 2024.

LE, T. T. et al. Artificial intelligence applications in solar energy. **JOIV International Journal on Informatics Visualization**, v. 8, n. 2, p. 826–826, 31 maio 2024.

LIU, Y. et al. Understanding LLMs: A Comprehensive Overview from Training to Inference. **arXiv (Cornell University)**, 3 jan. 2024.

MUNLEY, C.; JARMUSCH, A.; CHANDRASEKARAN, S. LLM4VV: Developing LLM-Driven Testsuite for Compiler Validation. **arXiv (Cornell University)**, 1 jan. 2023.

NOCODE STARTUP. **Curso N8N Gratuito Para Iniciantes 2025 | Crie Automações com IA**. Disponível em: <https://www.youtube.com/watch?v=-Ka4YKW7RwM>. Acesso em: 16 mai. 2025.

OGUZHAN TOPSAKAL; TAHIR CETIN AKINCI. Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast. **International Conference on Applied Engineering and Natural Sciences**, v. 1, n. 1, p. 1050–1056, 22 jul. 2023.

PAYONG, A.; MUKHERJEE, S. **LangChain Explained: The Ultimate Framework for Building LLM Applications**. Disponível em: <https://www.digitalocean.com/community/conceptual-articles/langchain-framework-explained>. Acesso em: 10 jul. 2025.

PIRES, W. **Masterclass N8n: Domine Agentes IA do Básico ao Avançado**. Disponível em: <https://www.youtube.com/watch?v=8czBd4isHSg>. Acesso em: 16 mai. 2025.

PORTAL DO GOV. **Resolução 1000 da ANEEL, seus direitos sobre energia elétrica, agora num só lugar! (2022)**. Disponível em: <https://www.gov.br/aneel/pt-br/assuntos/campanhas/resolucao-1000-da-aneel-seus-direitos-sobre-energia-eletrica-agora-num-so-lugar-2022> Acesso em: 8 jul. 2025.

PORTAL SOLAR. **Inversor Solar: O que é, Como Funciona e Como Escolher? (2023)**. Disponível em: <https://www.portalsolar.com.br/inversor-solar-o-que-e> Acesso em: 9 jul. 2025.

PRAJWAL NIMJE. The Rise of Low-Code/No-Code Development Platforms. **International Journal of Advanced Research in Science, Communication and Technology**, p. 650–653, 26 jun. 2024.

QUANTUMGLOBALGROUP. **The Evolution of Artificial Intelligence: A Comprehensive Historical Overview**. Disponível em: <https://quantumglobalgroup.com/evolution-of-artificial-intelligence-history/?utm> Acesso em: 2 jun. 2025.

RAZA, M. et al. Industrial applications of large language models. **Scientific Reports**, v. 15, n. 1, 21 abr. 2025.

RENATO. **Curso Gratuito N8N para Iniciantes - Crie um Agente IA do ZERO**. Disponível em: <https://www.youtube.com/watch?v=g3sXsi_OYqQ>. Acesso em: 16 mai. 2025.

REVISTA POTÊNCIA. **Energia solar: sistemas on-grid e off-grid - Portal Potência**. Disponível em: <https://revistapotencia.com.br/portal-potencia/mercado/energia-solar-sistemas-on-grid-e-off-grid/>. Acesso em: 8 jul. 2025.

SALAMI, I. A. et al. Addressing Bias and Data Privacy Concerns in AI-Driven Credit Scoring Systems Through Cybersecurity Risk Assessment. **Asian Journal of Research in Computer Science**, v. 18, n. 4, p. 59–82, 11 mar. 2025.

SENGAR, SANDEEP SINGH et al. Generative Artificial Intelligence: A Systematic Review and Applications. **arXiv (Cornell University)**, 17 maio 2024.

SINGH, A. The concept of artificial intelligence. **Jetir**, v.6, 2019.

SJÖDIN, D. et al. How AI capabilities enable business model innovation: Scaling AI through co-evolutionary processes and feedback loops. **Journal of Business Research**, v. 134, n. 1, p. 574–587, set. 2021.

SOUZA, D. **Tenha sua IA local de graça com OLLAMA e N8N!**. Disponível em: <https://www.youtube.com/watch?v=CFBV7gnW0BY>. Acesso em: 15 mai. 2025.

SOUZA, D. **Tenha sua VECTOR DATABASE com Qdrant + n8n**. Disponível em: <https://www.youtube.com/watch?v=SqQe5jgdg8s>. Acesso em: 15 mai. 2025.

WOO, J. J. et al. Custom Large Language Models Improve Accuracy: Comparing Retrieval Augmented Generation and Artificial Intelligence Agents to Non-Custom Models for Evidence-Based Medicine. **Arthroscopy: The Journal of Arthroscopic & Related Surgery**, 7 nov. 2024.

YUNG YAP, K.; R. SARIMUTHU, C.; MUN-YEE LIM, J. Artificial Intelligence Based MPPT Techniques for Solar Power System: A review. **Journal of Modern Power Systems and Clean Energy**, v. 8, n. 6, p. 1043–1059, 2020.

APÊNDICES

APÊNDICE A – CÓDIGO-FONTE DA INTERFACE GRÁFICA EM TKINTER E INTEGRAÇÃO DO SISTEMA DE IA COM GERAÇÃO AUMENTADA POR RECUPERAÇÃO (RAG) VIA FRAMEWORK LANGCHAIN

-INTERFACE TKINTER

```
import tkinter as tk
from tkinter import messagebox, scrolledtext, ttk
from PIL import Image, ImageTk
from versao_final import carregar_documentos, RAGApplication
import datetime
import os

LOGO_PATH = r"C:\Users\antho\OneDrive\Documentos\Fasor\logo\Logo
Vetorizada.png"

def carregar_logo(path, tamanho):
    try:
        img = Image.open(path)
        img = img.resize(tamanho, Image.Resampling.LANCZOS)
        return ImageTk.PhotoImage(img)
    except Exception as e:
        print(f"Erro ao carregar logo: {e}")
        return None

def atualizar_data_hora(label, janela):
    agora = datetime.datetime.now().strftime("%d/%m/%Y %H:%M:%S")
    label.config(text=agora)
    janela.after(1000, lambda: atualizar_data_hora(label, janela))

def enviar_pergunta():
    pergunta = entrada_pergunta.get()
    if pergunta.strip():
        resposta = rag_application.run(pergunta)
        area_resposta.insert(tk.END, f"Você: {pergunta}\nIA: {resposta}\n\n")
        area_resposta.see(tk.END)
        entrada_pergunta.delete(0, tk.END)

def iniciar_janela_principal():
    janela = tk.Toplevel(root)
    janela.title("Aplicação RAG")
    janela.geometry("1000x700")

    frame = tk.Frame(janela)
    frame.pack(expand=True, fill='both', padx=10, pady=10)
```

```
    # Logo
    logo = carregar_logo(LOGO_PATH, (200, 100))
    if logo:
        tk.Label(frame, image=logo).grid(row=0, column=0, padx=10, pady=10)
        janela.logo = logo  # manter referência

    # Relógio
    label_data_hora = tk.Label(frame, font=("Arial", 12))
    label_data_hora.grid(row=0, column=1, sticky='ne', padx=10, pady=10)
    atualizar_data_hora(label_data_hora, janela)

    # Chat
    aba_chat = ttk.Frame(frame)
    aba_chat.grid(row=1, column=0, columnspan=2, padx=10, pady=(5, 10),
sticky='nsew')

    global entrada_pergunta, area_resposta
    area_resposta = scrolledtext.ScrolledText(aba_chat, width=90, height=25,
font=("Arial", 12))
    area_resposta.pack(pady=5)

    entrada_pergunta = tk.Entry(aba_chat, width=80, font=("Arial", 14))
    entrada_pergunta.pack(pady=5)

    tk.Button(aba_chat, text="Enviar", command=enviar_pergunta, font=("Arial",
14)).pack(pady=5)

def verificar_login():
    if entrada_usuario.get() == "admin" and entrada_senha.get() == "senha123":
        messagebox.showinfo("Login bem-sucedido", "Bem-vindo!")
        iniciar_janela_principal()
    else:
        messagebox.showerror("Erro", "Usuário ou senha incorretos")

# Janela de login
root = tk.Tk()
root.title("Tela de Login")
root.geometry("400x300")
root.config(bg="#f0f0f0")

# Logo login
logo_login = carregar_logo(LOGO_PATH, (100, 50))
if logo_login:
    tk.Label(root, image=logo_login, bg="#f0f0f0").pack(pady=10)

# Formulário
for texto in [("Usuário:", False), ("Senha:", True)]:
```

```
    tk.Label(root, text=texto[0], font=("Arial", 12),
bg="#f0f0f0").pack(pady=5)
    entrada = tk.Entry(root, font=("Arial", 12), show="*" if texto[1] else "")
    entrada.pack(pady=5)
    if texto[1]:
        entrada_senha = entrada
    else:
        entrada_usuario = entrada

tk.Button(root, text="Login", command=verificar_login, font=("Arial",
12)).pack(pady=20)

# Carrega documentos
rag_application = carregar_documentos()

root.mainloop()
```

-SISTEMA DE IA COM RAG

```
import os
import requests
import fitz  # PyMuPDF
from bs4 import BeautifulSoup  # Importando BeautifulSoup para limpeza de HTML

# Definindo a variável USER_AGENT
os.environ['USER_AGENT'] = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3'

# Importando componentes necessários da biblioteca LangChain
from langchain_community.document_loaders import WebBaseLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import SKLearnVectorStore
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_ollama import ChatOllama
from langchain.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser

# Função para ler o conteúdo de um arquivo PDF
def read_pdf(file_path):
    document = fitz.open(file_path)  # Abre o documento PDF
    text = ""
    # Loop para extrair texto de cada página do PDF
    for page in document:
        text += page.get_text()  # Extrai texto da página atual
    document.close()  # Fecha o documento após a leitura
    return text  # Retorna todo o texto extraído
```

```
# Função para limpar o HTML usando BeautifulSoup
def clean_html(html_content):
    soup = BeautifulSoup(html_content, 'html.parser')
    # Remover scripts e estilos
    for script in soup(["script", "style"]):
        script.decompose()
    # Extrair texto limpo
    clean_text = soup.get_text()
    # Remover múltiplos espaços em branco e quebras de linha desnecessárias
    return ' '.join(clean_text.split())

# Classe Document para armazenar conteúdo e metadados
class Document:
    def __init__(self, page_content, metadata=None):
        self.page_content = page_content
        self.metadata = metadata if metadata is not None else {}

# Função para carregar documentos de URLs e PDF
def carregar_documentos():
    urls = [
        "https://www.planalto.gov.br/ccivil_03/_ato2019-
2022/2022/lei/l14300.htm",
    ]
    headers = {'User-Agent': os.environ['USER_AGENT']}
    docs = []
    # Carregar documentos das URLs
    for url in urls:
        try:
            loader = WebBaseLoader(url, requests_kwargs={'headers': headers})
            loaded_docs = loader.load()
            for doc in loaded_docs:
                clean_content = clean_html(doc.page_content)
                if clean_content:
                    docs.append(Document(clean_content))
        except Exception as e:
            print(f"Erro ao carregar {url}: {e}")

    # Corrigir descompactação: não usar para documento único
    # Remover a linha: docs_list = [item for sublist in docs for item in
sublist]
    # Pois 'docs' já é uma lista de Document

    # Carregar conteúdo do PDF
    pdf_path = r"C:\Users\antho\OneDrive\Área de Trabalho\Guia para
Dimensionamento de Usina Fotovoltaica_Versão_Final.pdf"
    pdf_content = read_pdf(pdf_path)
    if pdf_content:
```

```
        docs.append(Document(pdf_content))

    # Dividir documentos em partes menores
    text_splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
        chunk_size=1000,
        chunk_overlap=0,
    )
    doc_splits = text_splitter.split_documents(docs)

    # Criar embeddings e loja vetorial
    embedding_model = HuggingFaceEmbeddings(model_name="sentence-
transformers/all-MiniLM-L6-v2")
    vectorstore = SKLearnVectorStore.from_documents(
        documents=doc_splits,
        embedding=embedding_model,
    )
    retriever = vectorstore.as_retriever(k=5)

    # Template de prompt
    prompt = PromptTemplate(
        template="""Você é um assistente para tarefas de resposta a perguntas.
Use os seguintes documentos para responder à pergunta. Faça os cálculos que
precisar para trazer respostas numéricas.
Se você não souber a resposta, apenas diga que não sabe.
Se houver informações relevantes no documento, utilize-as.
Use no máximo seis frases e seja conciso.
Pergunta: {question}
Documentos: {documents}
Resposta:""",
        input_variables=["question", "documents"],
    )

    # LLM
    llm = ChatOllama(model="llama3.1", temperature=0)

    # Chain de recuperação e resposta
    rag_chain = prompt | llm | StrOutputParser()

    return RAGApplication(retriever, rag_chain)

# Classe da aplicação RAG
class RAGApplication:
    def __init__(self, retriever, rag_chain):
        self.retriever = retriever
        self.rag_chain = rag_chain

    def run(self, question):
        docs = self.retriever.invoke(question)
```

```
        doc_texts = "\n".join([doc.page_content for doc in docs])
        answer = self.rag_chain.invoke({"question": question, "documents":
doc_texts})
        return answer
```

ANEXOS

ANEXO A – TABELA DE TESTES DE DESEMPENHO DE LLMS EM PERGUNTAS TÉCNICAS DE SISTEMAS SOLARES

https://1drv.ms/x/c/5b2e8de022fd4034/EeHsN_Ne80VCnIs0fKUoJ2ABMsAeb1fFEQRbVzjNIFrTFQ?e=hxcaYX

ANEXO B – BASE DE CONHECIMENTO “GUIA PARA DIMENSIONAMENTO DE USINA FOTOVOLTAICA”

Guia para Dimensionamento de Usina Fotovoltaica.docx

ANEXO C – RESOLUÇÃO NORMATIVA Nº1000 ANEEL

https://www.gov.br/aneel/pt-br/assuntos/campanhas/resolucao-1000-da-aneel-seus-direitos-sobre-energia-eletrica-agora-num-so-lugar-2022