

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO
CAMPUS ITAPETININGA

EDUARDO SANTOS DE ALMEIDA

**DESENVOLVIMENTO E TESTE DE INTELIGÊNCIA ARTIFICIAL PARA
CHATBOTS DE ATENDIMENTOS ESPECIALIZADOS**

ITAPETININGA

2025

EDUARDO SANTOS DE ALMEIDA

**DESENVOLVIMENTO E TESTE DE INTELIGÊNCIA ARTIFICIAL PARA
CHATBOTS DE ATENDIMENTOS ESPECIALIZADOS**

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, Campus Itapetininga, como requisito para a obtenção do título de Especialista em Desenvolvimento Web.

Orientador: Prof. Dr. Carlos Henrique da Silva Santos
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP) - Campus Itapetininga

ITAPETININGA
2025

Dedico esse trabalho, a Deus que é meu refúgio, a minha família que é a base de toda minha força de vontade e esforço que estiveram sempre ao meu lado me incentivando a continuar e a seguir os meus sonhos, e ao meu orientador que confiou e me acompanhou durante essa caminhada!

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus por me fornecer fé, força e coragem para trilhar o caminho do desenvolvimento pessoal e da educação para buscar uma melhor qualidade de vida, pois sem a força que vem Dele não sou nada.

Agradecer a minha família que em todos os momentos me motivaram e nunca me deixaram desanimar.

Agradeço também ao meu Orientador, Prof. Dr. Carlos Henrique da Silva Santos, pela paciência, cumplicidade e ajuda que me forneceu em todo o processo de elaboração deste trabalho de conclusão

Por fim agradeço ao Instituto Federal de Educação, Ciência e Tecnologia de São Paulo e todos os seus colaboradores pela oportunidade de realizar o curso de especialização e sempre garantir um excelente atendimento e suporte acadêmico

"Se você espera o melhor momento para criar algo, isso pode nunca ocorrer. ", Michiko Aoyama.

RESUMO

O presente trabalho teve como objetivo desenvolver um sistema de chatbot com inteligência artificial voltado ao atendimento de dúvidas sobre o processo seletivo do Instituto Federal de São Paulo – Campus Itapetininga. A proposta busca facilitar a obtenção de informações contidas nos editais, reduzindo o tempo de busca e promovendo maior clareza na comunicação com os candidatos. A metodologia adotada envolveu o uso do modelo GPT-4o, treinado com o edital oficial, integrado via LangChain e consumido por uma API desenvolvida em Python com FastAPI. Os dados foram persistidos em um banco Redis, permitindo a posterior exportação e reuso para novos treinamentos. A interface foi implementada em Vue.js visando acessibilidade e simplicidade. Para validação do sistema, foi realizado testes durante o desenvolvimento para verificar sua eficiência na busca das informações pelo edital.

Palavras-chave: Inteligência Artificial. Chatbot. LLM. Aplicação WEB.

ABSTRACT

This work aimed to develop an artificial intelligence chatbot system focused on answering questions about the selection process for the Federal Institute of São Paulo – Itapetininga Campus. The proposal seeks to facilitate the retrieval of information contained in the notices, reducing search time and promoting greater clarity in communication with candidates. The methodology adopted involved the use of the GPT-40 model, trained with the official notice, integrated via LangChain and consumed by an API developed in Python with FastAPI. The data was persisted in a Redis database, allowing for later export and reuse for new training. The interface was implemented in Vue.js aiming for accessibility and simplicity. For system validation, tests were carried out during development to verify its efficiency in searching for information from the notice.

Keywords: Artificial intelligence. Chatbot. LLM. WEB application.

LISTA DE ABREVIATURAS E SIGLAS

IA	Inteligência Artificial
IFSP	Instituto Federal de São Paulo
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
API	<i>Application Programming Interface</i>
JSON	<i>JavaScript Object Notation</i>
JSONL	<i>JavaScript Object Notation Lines</i>
HTTP	<i>Hypertext Transfer Protocol</i>
LLM	<i>Large Language Model</i>
ML	<i>Machine Learning</i>
NLP	<i>Natural Language Processing</i>
NLU	<i>Natural Language Understanding</i>
GPT	<i>Generative Pre-trained Transformer</i>
UI	<i>User Interface</i>
IP	<i>Internet Protocol</i>
ID	Identificador
Fine-Tuning	Ajuste fino de modelos de linguagem
Checkpoint	Ponto de salvamento intermediário do treinamento
Dataset	Conjunto de dados utilizado para treinamento
Loss Rate	Taxa de perda durante o treinamento do modelo
Token	Unidade básica de texto processada pelo modelo
Batch Size	Quantidade de dados processados por iteração
Learning Rate	Taxa de aprendizagem do modelo
Seed	Valor inicial para geração de números aleatórios

SUMÁRIO

LISTA DE FIGURAS	
1 – INTRODUÇÃO	11
2 – REVISÃO BIBLIOGRÁFICA	12
2.1 Aprendizado de Máquina (<i>Machine Learning</i>)	12
2.2 Chatbot: Conceitos e Tecnologias	14
2.3 Modelos de linguagem de grande escala (<i>large language model</i>)	15
2.4 Tecnologias LLM em Chatbots: LangChain e outros	16
3 – METODOLOGIA	18
4 – DESENVOLVIMENTO	20
5 – RESULTADOS	40
6 – CONCLUSÃO	42
6.1 TRABALHOS FUTUROS	42
Referências	44

LISTA DE FIGURAS

Figura 1 – <i>Stack</i> de Tecnologias para <i>Windows</i>	19
Figura 2 – <i>Stack</i> de Tecnologias <i>Linux</i>	19
Figura 3 – Diagrama de Tecnologias	22
Figura 4 – Diagrama de Caso de Uso	23
Figura 5 – Fluxograma de Resposta	24
Figura 6 – Fluxograma de <i>Download</i>	25
Figura 7 – Informações Básicas do <i>FineTuning</i>	26
Figura 8 – Informações Básicas do <i>FineTuning 2</i>	26
Figura 9 – Passos de Treinamento do <i>FineTuning</i>	27
Figura 10 – Checkpoints do <i>FineTuning</i>	28
Figura 11 – Métricas do <i>FineTuning</i>	28
Figura 12 – Formato do <i>FineTuning</i>	29
Figura 13 – Variáveis do Sistema	30
Figura 14 – Configuração Inicial	31
Figura 15 – Criação de Modelo LLM	32
Figura 16 – Configuração de Rota	33
Figura 17 – Configuração de Rota de Perguntas	34
Figura 18 – Configuração de Rota de <i>Download</i>	35
Figura 19 – Formato Teste	35
Figura 20 – Interface VueJS	36
Figura 21 – Teste de Consumo pelo <i>ThunderClient</i> na Rota de Perguntas	37
Figura 22 – Teste de Consumo pelo <i>ThunderClient</i> na Rota de <i>Download</i>	37
Figura 23 – Formato de <i>Download 2</i>	38
Figura 24 – Banco de Dados	39

1 INTRODUÇÃO

Buscou-se com o desenvolvimento deste trabalho disponibilizar um sistema de chatbot com inteligência artificial para responder questões sobre o processo seletivo no Instituto Federal de São Paulo para o Campus de Itapetininga. A ideia teve como intuito reduzir pela metade o tempo de procura pelas informações presentes nos editais.

Por meio de uma análise nos editais oficiais acessados na página do IFSP, verificou-se que o candidato dedicava um tempo significativo para buscar informações relacionadas a diversos tópicos envolvidos no processo seletivo, como pagamento, inscrição e matrícula. Esse tempo poderia ser melhor empregado na realização das etapas necessárias ao ingresso na instituição.

Observou-se que a linguagem formal aplicada na redação dos editais dificultava a compreensão por parte de alguns candidatos. Dessa forma, um sistema que fornecesse respostas de maneira mais clara, objetiva e simples reduziria o tempo necessário para compreender as informações contidas nos documentos oficiais.

Esse cenário ocorre em decorrência do analfabetismo funcional, condição na qual o indivíduo é capaz de realizar leituras curtas ou interpretar frases isoladas, mas apresenta dificuldades na compreensão de textos simples ou na resolução de operações mais complexas. De acordo com pesquisa apresentada por ([Oliveira 2025](#)), houve um crescimento desse índice de 27% em 2018 para 29% em 2024. Diante desse aumento no número de analfabetos funcionais, o desenvolvimento do sistema proposto surge como uma alternativa para auxiliar na leitura e compreensão de conteúdos mais complexos, promovendo maior acessibilidade à informação.

A escolha do tema ocorreu devido à popularização da inteligência artificial em diversas áreas, sendo uma delas a acadêmica como destaca a Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior [CAPES 2024](#) ao informar que o Brasil está entre as 20 nações que mais publicam trabalhos voltados ao tema entre os anos de 2019 e 2023, o que equivalem a 6,3 mil estudos que tem impulsionado o surgimento constante de novas ferramentas e tecnologias capazes de otimizar processos. A utilização de um sistema de inteligência artificial proporcionaria uma forma de consulta mais intuitiva, por meio de um layout baseado em perguntas e respostas. As informações seriam apresentadas de forma precisa, uma vez que o modelo empregado foi treinado com questões voltadas ao próprio edital, garantindo maior eficiência na busca por informações.

Além da otimização do tempo do candidato, também se buscou reduzir a demanda dos colaboradores nos canais de comunicação, como e-mail, telefone e WhatsApp, no esclarecimento de dúvidas sobre o processo seletivo. Dessa forma, a necessidade de contato direto com os atendentes ocorreria apenas nos casos em que o sistema não conseguisse fornecer a resposta desejada, direcionando o usuário a um suporte via e-mail.

2 REVISÃO BIBLIOGRÁFICA

Com o avanço acelerado das Inteligências Artificiais (IAs) nos últimos tempos, cada vez mais são desenvolvidas soluções, algoritmos e bibliotecas que visam sustentar a sua otimização. Desta forma, algumas ferramentas são destaque por estar presente na maioria dos projetos atuais como o ChatGPT, Copilot e demais IAs voltadas ao modelo de *chatbot*. A ideia da pesquisa é compreender quais tecnologias e bibliotecas são utilizadas atualmente.

Os resultados apresentados destacam que a principal tecnologia de desenvolvimento é o Python por meio de sua simplicidade, facilidade de uso, bibliotecas presentes no ambiente de *machine learning* e IA como é destacada em um artigo presente na escola Asimov Academy:

Python oferece uma grande variedade de bibliotecas e frameworks voltados para inteligência artificial e machine learning, como TensorFlow, Keras, Scikit-learn e PyTorch, que permitem o desenvolvimento ágil e eficiente de modelos de IA. Além disso, a ampla documentação disponível facilita o processo de aprendizado para desenvolvedores (Pereira 2023).

SECURITY FIRST 2024 salienta que o ChatGPT (*Generative Pre-Trained Transformer*) desenvolvido pela OpenAI é um dos principais exemplos de IA atualmente, seu desenvolvimento se baseia na linguagem Python que consome algumas bibliotecas, dentre elas o TensorFlow e PyTorch para instruir *transformers* pré-treinados dentro do modelo de LLM (*Large Language Model*).

Dentro de seu desenvolvimento há 4 principais bibliotecas que estão presentes nos projetos que envolvem inteligências artificiais: PyTorch, Scikit-Learn, TensorFlow e Keras. As bibliotecas são voltadas principalmente para *machine learning*, *deep learning* e redes neurais, como é destacado em DATA CAMP 2024 e CSP TECNOLOGIA 2024.

2.1 Aprendizado de Máquina (*Machine Learning*)

Segundo a ORACLE 2024 o *Machine Learning* é um subconjunto da inteligência artificial que melhora seu desempenho a partir de uma análise nos dados recebidos. Atualmente há três principais abordagens quando o assunto é aprendizado de máquina, a primeira delas é o **aprendizado supervisionado**, o **aprendizado não supervisionado** e **aprendizado semi-supervisionado**.

Ao realizar o aprendizado supervisionado é necessário criar uma base de dados rotulada que mapeia um atributo específico de um objeto fornecendo características conhecidas é possível treinar um modelo para retornar determinada saída, ou seja, fornecendo dados rotulados de um objeto 'maçã' para um modelo pela abordagem do aprendizado supervisionado é possível treiná-lo para reconhecer objetos que tenham as mesmas características do objeto "maçã" fornecido anteriormente como cita GOOGLE CLOUD 2024 e ainda completa com alguns exemplos de

algoritmos que realizam essa abordagem, como é o caso da **Regressão Linear, Regressão Polinomial, Vizinhos mais próximos exatos, Nayve Bayes e Arvore de Decisão**.

Por outro lado o método de aprendizado não supervisionado não rotula os dados anteriormente, sendo assim o algoritmo que aprende por este método não tem a "supervisão" em si dos humanos para que sejam feitas correções e as pré-rotulações de saída. Sua lógica parte de uma categorização de grupos tomando por base os atributos. Dessa forma ao fornecer imagens diversas o algoritmo irá encontrar um padrão para o seu agrupamento, ou seja, tomando por base a cor, tamanho, estilo, será possível dividi-los e rotulá-los. Alguns exemplos segundo o [GOOGLE CLOUD 2024](#) são o **K-means Clustering, Clustering hierárquico e o Mínimo Quadrado Parcial**.

O aprendizado cujo mistura as técnicas das duas abordagens anteriores é denominado Aprendizado Semi-Supervisionado e é utilizado em sua maioria quando se há uma quantidade limitada de dados como aborda ([Almeida 2023](#)) e por isso em muitos casos é uma melhor opção pois não é tão custosa quanto o aprendizado supervisionado e ao mesmo tempo é mais precisa que o aprendizado não supervisionado, já que nela não é necessário a rotulação de todos os dados.

Dentro do aprendizado semi-supervisionado encontra-se premissas que visam garantir um resultado válido e concreto que partem de conceitos da **continuidade, cluster, baixa densidade e hipótese de manifold**.

A Premissa da Continuidade presente no aprendizado semi-supervisionado defende que dados que compartilham de mesmas características tendem a pertencer a uma mesma classe, sendo assim pode-se classificar amostras que ainda não tem uma rotulação previamente informada como informa [Almeida 2023](#). Sendo assim essa premissa além de realizar a classificação dos dados, ainda provém uma melhor generalização do modelo, já que os dados não rotulados possuem características que ainda não foram classificadas dentro do grupo em que foram adicionados.

O Conceito de Cluster surge como um aglomerado de dados que formam uma classe, ou seja, um grupo de dados. A premissa de cluster aprende como os dados rotulados estão agrupados e usa essa informação para identificar e prever onde cada dado não rotulado se encaixa, como é destacado no trecho a seguir:

"O aprendizado de máquina semi-supervisionado explora essa premissa ao usar dados rotulados para inferir a estrutura de agrupamento subjacente e, assim, atribuir rótulos a dados não rotulados próximos a um determinado grupo." ([Almeida 2023](#))

([Almeida 2023](#)) destaca ao falar da Premissa de Baixa Densidade que "Os algoritmos de aprendizado de máquina semi-supervisionado buscam aproveitar essas regiões, explorando os dados não rotulados para melhorar a capacidade de generalização em áreas menos densamente povoadas", ou seja, diferente da premissa de *cluster*, que foca nos grupos de dados semelhantes, a premissa de baixa densidade reconhece a importância das áreas entre esses *clusters* ou em regiões menos densamente povoadas e visa melhorar a sua acurácia ao explorar e entender

áreas com poucos dados rotulados.

Conhecido como hipótese de manifold ou premissa de variedade, sugere que ao analisar dados de alta dimensão é possível encontrar locais em que há dados rotulados que possuem características que podem ser agrupadas em clusters, para que assim consigam realizar previsões e definir novos padrões e categoriza-los.

"assume que os dados não rotulados próximos a dados rotulados provavelmente seguem a mesma estrutura subjacente da variedade. Algoritmos semi-supervisionados exploram essa premissa para prever rótulos em regiões pouco exploradas, melhorando a generalização do modelo em espaços de características complexas." (Almeida 2023)

2.2 Chatbot: Conceitos e Tecnologias

Os *Chatbots* são códigos que interpretam as entradas fornecidas pelos usuários e retornam uma saída, o processo consiste em simular conversas humanas permitindo que haja a interação de humanos com dispositivos digitais, como destaca (ORACLE 2024) e ainda completam que esse tipo de sistema é crescente, ou seja, podem se moldar a partir das informações coletadas durante a sua execução. (GOOGLE CLOUD 2024) informa que diferentemente dos *chatbots* convencionais que utilizam PLN e ML, os sistemas que trabalham com a tecnologia de IA utilizam LLM para comparar os fluxos de conversas geradas e respostas pré-treinadas.

Em sua execução o *chatbot* pode conter uma ou mais tecnologias, entre elas estão a utilização de inteligência artificial, regras automatizadas, NLP (Processamento de Linguagem Neural) e ML (Aprendizado de Máquina). Tais tecnologias geram dois principais tipos de chatbot: **Orientado a tarefas (declarativo)** e **Orientado por dados e preditivos (convencionais)**.

Bem populares no cotidiano, os *chatbots* declarativos ou orientados a tarefas desempenham uma função específica, muitas vezes focadas em situações simples e que não envolvem uma grande diversidade de variáveis. ORACLE 2024 salienta que mesmo utilizando regras, NLP e ML eles não possuem uma estrutura que permita ser utilizados em diversas situações, ou seja, são mais aplicáveis em situações de suporte e serviço.

Os *chatbots* convencionais, descritos como assistentes digitais ou virtuais pela empresa ORACLE 2024 apresentam uma interação mais sofisticada em comparação aos declarativos. Aplicam em sua construção a inteligência preditiva e a análise, o que permite que as respostas sejam baseadas no comportamento anterior. Para o seu funcionamento é necessário um contexto inicial, NLP, ML e também NLU (Entendimento de Linguagem Natural é um subconjunto do processamento de linguagem natural que ajuda a compreender a interpretar e compreender questões humanas como informa BOTPRESS 2024).

2.3 Modelos de linguagem de grande escala (*large language model*)

O LLM (*Large Language Model* ou Grande Modelo de Linguagens) é um modelo de Inteligência Artificial capaz de entender e gerar linguagem natural, onde seu treinamento é realizado por meio de um grande volume de dados, como afirma (IBM 2024). RED HAT 2024 complementa que ao usar as técnicas necessárias de machine learning uma LLM, ela se especializa em um determinado caso, eles podem ser variados entre geração de linguagem humana, imagem, textos, etc. Para o desenvolvimento deste modelo de base de IA é aplicado processamento de linguagem neural (NLP) para compreender de forma lógica a resposta, ou seja, por meio do NLP o computador aprende como interpretar perguntas, entende-las e por fim responder com precisão ao que foi solicitado.

O desenvolvimento de uma LLM como o BERT (*Bidirectional Encoder Representations from Transformers*) e o GPT-3 (*Generative Pre-Trained Transformer*) parte do princípio de um transformador generativo pré-treinado juntamente com mecanismo de atenção. Devido a isso o LLM é capaz de prever qual será a próxima palavra, levando em consideração o contexto fornecido anteriormente. Esse estilo de previsão só é possível por conta de uma probabilidade de recorrência das palavras que são *tokenizadas* para serem utilizadas no transformador, ou seja, elas são divididas em caracteres menores e disponibilizados de forma sequencial dentro de um vetor, onde por fim esses *tokens* viram *embeddings* para serem utilizadas pelo modelo.

Os *Embeddings* são uma forma de representação por meio de símbolos (palavras, caracteres, frase) em um vetor de valor contínuo, onde são correlacionadas uma com a outra, como destaca (DEEP LEARNING BOOK 2024).

Tais representações são geradas após passarem pelo Transformador dentro de um modelo, onde são um tipo de arquitetura de rede neural que transforma ou altera uma sequência de entrada em uma sequência de saída, ou seja, por meio do contexto fornecido a arquitetura do transformador rastreia entre os seus componentes (*tokens*) qual é a sequência por meio da relevância, explica (AWS 2024).

Na construção de uma LLM é necessário entender que a sua formação contém diversas camadas de componentes de uma arquitetura de transformação. É possível realizar a divisão em camadas para melhor entendimento e são elas: **Incorporações de Entrada, Codificação Posicional, Bloco de Transformador, e Blocos Lineares e Softmax.**

Na Incorporações de Entrada a entrada fornecida é convertida para um domínio matemático (função que fornece uma saída para cada valor fornecido como entrada) que por sua vez passa por um processamento, ou seja, acontece uma divisão do que foi informado em pequenas partes chamadas de "*tokens*" ou componentes de sequência individual de forma vetorizada, onde esses vetores contém informações semânticas e de sintaxe, representadas como números que permitem calcular relações entre as palavras.

"O *software* pode usar os números para calcular as relações entre palavras em termos matemáticos e entender o modelo da linguagem humana. As incorporações fornecem uma maneira de representar *tokens* discretos como

vetores contínuos, que o modelo pode processar e aprender.” (AWS 2024).

Após realizar a incorporação de entrada a próxima etapa é a codificação posicional que a AWS 2024 declara que por não processar dados sequenciais inerentes em ordem, um transformador necessita de uma codificação para ordenar os *tokens* em uma sequência de entrada. Desta forma a codificação posicional acopla ao *token* uma informação referente a sua posição na sequência. Realizando esse processo é possível entender o contexto da sequência.

O bloco de transformador tem uma arquitetura que contém uma composição no qual permite um *feedback* direto em relação à posição de cada elemento na sequência fornecida por meio da rede neural e uma avaliação precisa na importância dos símbolos existentes no que está sendo analisado por meio do mecanismo de autoatenção que permite o agrupamento das palavras *tokenizadas* para compreender melhor o sentido da frase, como salienta AWS 2024.

Finalizando o processo nos Blocos Lineares e *Softmax* é necessário fazer uma previsão precisa e concreta para determinar a próxima palavra da sequência. Dessa forma a camada densa ou bloco linear executa um mapeamento linear que percorre o espaço definido do vetor até o domínio de entrada inicial. Ao terminar esse procedimento, é gerado um *logit* (conjunto de pontuações) para cada *token*. A AWS 2024 informa que o *Softmax* converte essas pontuações em uma distribuição probabilística normalizada, onde cada resultado indica a probabilidade de um determinado *token*.

2.4 Tecnologias LLM em *Chatbots*: LangChain e outros

O langChain é uma estrutura de orquestração disponível em Python para criação de agentes virtuais, *chatbots* que trabalhem com LLM, segundo RED HAT 2024 o termo orquestração define um sistema que realiza a coordenação de tarefas e processos, ou seja, ela gerencia automações para que sejam possíveis operar de forma global todos os algoritmos que a aplicação está englobando. Desta forma o LangChain fornece de forma genérica um ambiente que pode ser modificado para suprir a demanda necessária, como destacado por IBM 2023.

Em sua página oficial, o LANGCHAIN 2025 informa a existência de muitos componentes para diversos casos, onde além de fornecer modelos e ferramentas, é possível encontrar conversores de saída, carregadores de documentos, entre outros módulos.

As 'cadeias' fortemente presentes na lógica do LangChain são fluxos de códigos que após a finalização de uma solicitação, faz com que a próxima desencadeada, dessa forma é possível criar um *prompt* funcional importando com poucas linhas de código.

Com a premissa de *Low-code*, essa alternativa apresenta a possibilidade de criação dos modelos partindo de uma interface gráfica, o que auxilia pessoas. Assim como o LangChain, é possível consumir um Modelo externo como o LLama, GPT, entre outros, como apresentado por FLOWISE AI 2025.

O TENSORFLOW 2025 é uma tecnologia que facilita a criação de LLMs com tecnologias voltadas ao *Machine Learning*, criada pela Google, essa tecnologia é compatível com

computação numérica geral, redes neurais e *deep learning*. Atualmente é uma das bibliotecas mais populares quando o assunto é IA, segundo [DATABRICKS 2025](#) o TensorFlow é a escolha de 64% das pessoas que utilizam alguma tecnologia de Inteligencia Artificial.

3 METODOLOGIA

O trabalho proposto tem o objetivo geral sendo o **desenvolver** um sistema web que auxilia os candidatos no processo seletivo do IFSP cujo o *backend* consome um **modelo de linguagem** (LLM) **treinado via FineTuning** pela plataforma da OpenAI, utilizando como conexão uma API construída em Python integrada a um banco de dados não relacional que armazena as interações.

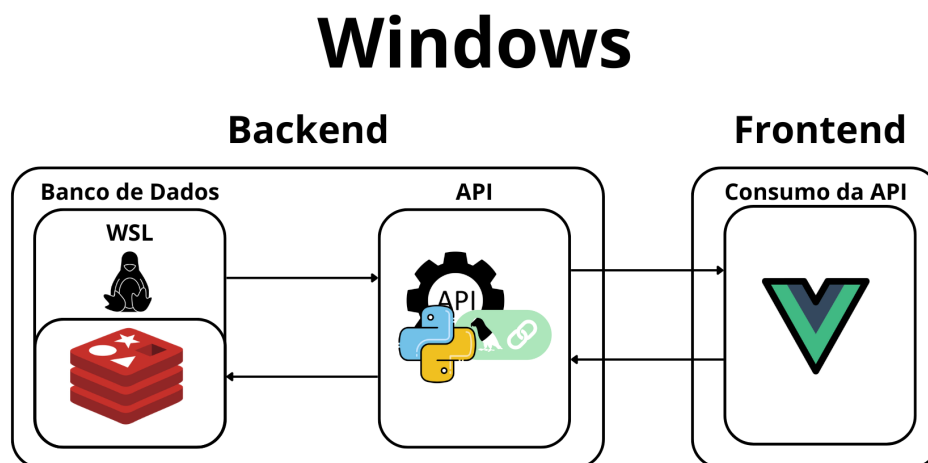
Um dos objetivos específicos são o **desenvolvimento** de uma interface *web* construída em VueJS que permite a interação dos candidatos com o sistema de forma acessível e intuitiva. Como segundo objetivo específico é a **avaliação do desempenho do sistema** por meio de testes durante o seu desenvolvimento, verificando assim a eficácia no auxílio aos candidatos durante a utilização no processo seletivo.

Durante a construção do trabalho houve a pesquisa em fóruns de comunidades, artigos acadêmicos e também em documentações oficiais das linguagens e *frameworks*.

Devido ao suporte fornecido pelos sistemas operacionais Linux e Windows para o banco de dados em questão, a *stack* de tecnologias tem uma pequena distinção, ao utilizar do Windows é necessário o recurso WSL (*Windows Subsystem for Linux* ou aportuguesando Subsistema Windows para Linux) para executar um ambiente Linux sem que seja necessário a utilização de uma máquina virtual ou uma dupla inicialização para se usar o banco de dados Redis. O *backend* fica composto pelo banco de dados instalado dentro do WSL que se comunica diretamente com a API desenvolvida em python com o *LangChain*, que por sua vez faz a comunicação com o *frontend* que é a interface construída em VueJS como aponta a [Figura 1](#).

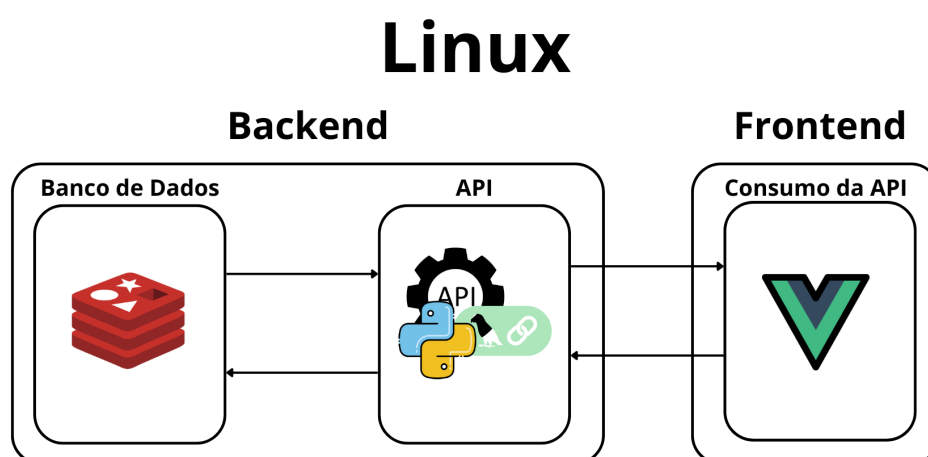
Pensando numa arquitetura em Linux, o *backend* é composto pelo banco de dados Redis que se comunica com a API *Python* com *LangChain*, e seu consumo é feito pelo *frontend* desenvolvido em VueJS, descrito assim pela [Figura 2](#). Dessa forma ao utilizar o Linux não se faz necessário a utilização do WSL, uma vez que o Linux dá suporte ao banco de dados utilizado no projeto.

Figura 1 – Stack de Tecnologias para Windows



Fonte: Elaborado pelo Autor

Figura 2 – Stack de Tecnologias Linux



Fonte: Elaborado pelo Autor

4 DESENVOLVIMENTO

As tecnologias e ferramentas bases para o desenvolvimento do projeto são conhecidas por sua presença e desempenho no desenvolvimento e refinamento de modelos pré-treinados. A conexão entre as tecnologias ocorreu via uma API desenvolvida em Python, que realiza a criação, chamada e utilização do modelo LLM, que salva tanto as perguntas como as respostas no banco de dados Redis. Dessa forma a utilização do sistema em *frontend* pode ser consumida por qualquer tecnologia, entretanto a utilização do VueJS se deu pela semântica simples, grande gama de conteúdo presente na internet e a limitação de *hardware* do desenvolvedor.

O ambiente de desenvolvimento é uma ferramenta muito importante no processo da criação de um software e diante disso foi escolhido o **Visual Studio Code** para codificar os arquivos e para a realizar a criação da lógica de uma inteligência artificial GPT, onde permite de forma gratuita a instalação de extensões, bibliotecas e *frameworks* utilizados no projeto. Este ambiente é utilizado para *machine learning*, codificação em colaboração, *data science*, como ferramenta de educação e para demais projetos que não envolvam inteligência artificial.

O **python** foi escolhido por ser uma linguagem simples e muito utilizada no desenvolvimento de *machine learning*, inteligências artificiais e demais algoritmos. Sua compatibilidade *Visual Studio Code* permite que sua utilização tenha uma melhor performance para o desenvolvimento do projeto por meio das extensões.

Todo esse gerenciamento de pacotes foi possível devido a utilização do **PIP** (*Pip Installs Packages*), que é o gerenciador de pacotes padrão para Python. Ele é uma ferramenta essencial que simplifica o processo de instalação, atualização e remoção de pacotes e bibliotecas Python. Com o PIP, é possível gerenciar as dependências do projetos, garantindo que todas as bibliotecas necessárias estejam instaladas e atualizadas.

Dessa forma o projeto faz uso do *framework* **LangChain**, utilizado na construção de LLMs, onde por meio do desenvolvimento de cadeias (blocos de construção que informa **o que** deve ser feito ao invés de **como** deve ser feito). Este *framework* utiliza em sua composição outras bibliotecas para seu funcionamento, como o exemplo da *Numpy*, *regex*, *langchain-text-splitters*, *langchain-community* entre outras. O modelo GPT utilizado é o GPT4-o desenvolvido pela OpenAI, onde passou pelo *fine tuning* e tendo como a base inicial o PDF do edital de processo seletivo do IFSP do ano de 2025, provendo assim um modelo único e treinado para o sistema em questão. Por ser um produto da OpenAI. Na leitura de arquivos do sistema é utilizada a biblioteca OS (*Operational System*, aportuguesando Sistema Operacional) para abrir o arquivo que contém a chave da API, o *python-dotenv* permite que um arquivo seja configurado com variáveis de ambientes locais para que o sistema as utilize para configurações necessárias, permitindo por exemplo que a chave da API da OpenAI seja consumida pelo sistema, fazendo assim com que o resto do processo aconteça. Para a disponibilização do sistema foi utilizado a biblioteca FastAPI que permite a criação de rotas por meio dos métodos

HTTPs, ou seja, ao realizar a criação de uma API e em conjunto da biblioteca *requests* é possível consumir de forma mais simples o sistema por meio de um JSON no *request*.

Por fim, para a persistência dos dados foi utilizado o Redis, um banco NoSQL que permite que dados não tabulares sejam inseridos sem nenhuma regra, sendo assim mais flexíveis para projetos em que os dados não apresentam um padrão, como afirma (GOOGLE CLOUD 2024). O Redis é um banco de dados orientado a documentos, onde seu armazenamento segue o mesmo padrão de objetos JSON (*JavaScript Object Notation*), mesmo estilo de dados que são enviados e consumidos por APIs. Para a utilização no sistema o Redis conta com uma biblioteca própria de mesmo nome, que permite utilizar de comandos do banco de dados.

Para a disponibilização de uma API *RestFull* foi utilizado do FastAPI, uma biblioteca que possibilita a criação de rotas e consumo dos métodos HTTPs, que ao coletar os dados, os processa para que o sistema seja alimentado. Para isso foi desenvolvido três rotas, sendo elas uma de teste para verificar a conexão da API com o banco de dados e com o *software* (ou interface) que irá consumi-la, onde é possível acessar pelo caminho "/" e outra que consome o sistema criado com o LangChain pelo caminho "/duvidas", além de uma última "/download" que possibilita salvar toda a interação do banco de dados no computador do usuário.

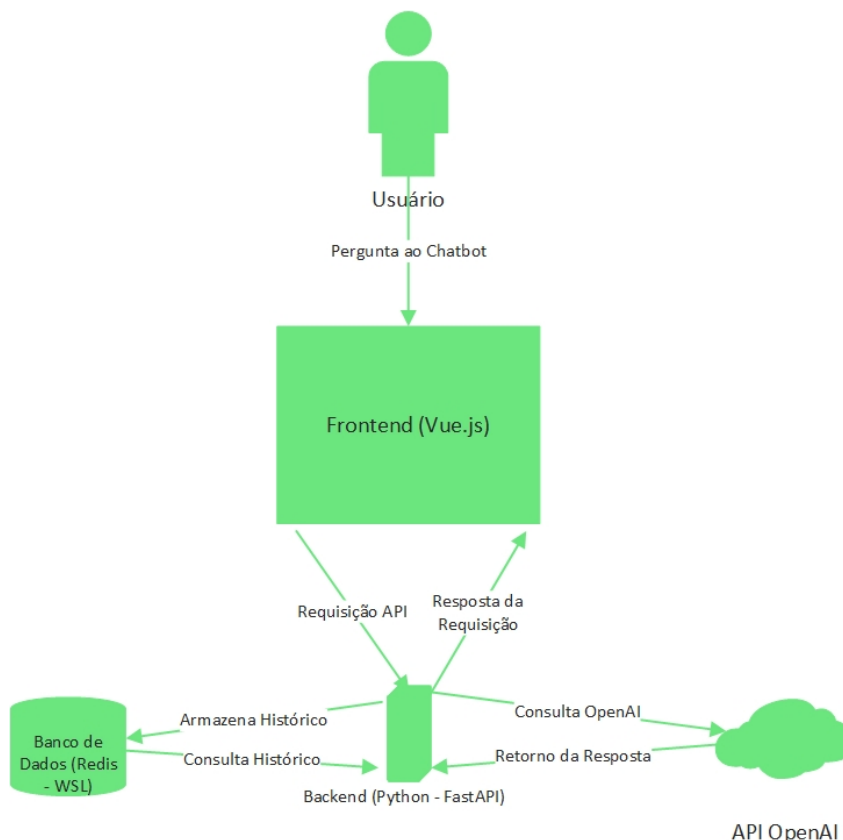
As informações necessárias são enviadas pelo corpo da requisição e seguem o seguinte padrão: `{"query": "Informe sua pergunta"}`, ao receber a informação ela será processada pelo sistema que previamente carregou o PDF para ser utilizado como base e dividiu texto em partes menores para uma análise mais rápida, e em seguida acontece o *embeddings* que nada mais é que a representação vetorial do texto que é armazenado de forma eficiente pelo FAISS (*Facebook AI Similarity Search*, aportuguesando Pesquisa de similaridade de IA do Facebook), partindo para a parte de perguntas, o sistema carrega uma funcionalidade de LLM que recupera os dados e realiza uma busca no texto que foi armazenado pela FAISS. Por fim o sistema utiliza do modelo *gpt-4o-2024-08-06:personal:fine-tuning2*, que foi treinado anteriormente no sistema da OpenAI. Caso haja algum problema nessa etapa o sistema irá retornar um código 500 (*Internal Server Error*, aportuguesando Erro do Servidor Interno) para informar que houve alguma questão na aplicação não foi concluída com sucesso.

O resultado desse processo é salvo no Redis que é um banco de chave-valor, perfeito para armazenar o tipo de dado retornado. Sua escolha se deu pelo desafio de integrar o sistema em um banco de dados não relacional.

O usuário final acessará o sistema por meio de uma interface *web* desenvolvida com VueJS, que se comunica via requisições HTTP com uma API construída em Python. Essa API, por sua vez, consome os serviços da OpenAI, armazena os dados da requisição no banco de dados e, em seguida, retorna a resposta ao usuário, e para representar a arquitetura geral do sistema desenvolvido, foi elaborado um diagrama tecnológico que resume os principais componentes e suas interações. A Figura 3 ilustra essa estrutura, destacando três camadas principais: a interface de usuário (*frontend*), a API do sistema (*backend*) e o banco de dados para armazenamento das interações. A interface foi desenvolvida com VueJS, permitindo que o

usuário interaja visualmente com o sistema, enviando perguntas e recebendo respostas. Essa interface consome os serviços da API construída em Python utilizando o *framework* FastAPI, que atua como intermediária entre o *frontend* e os módulos responsáveis pelo processamento, como o *LangChain* e a API da OpenAI. Após o processamento, as respostas são armazenadas em um banco de dados, permitindo que interações futuras possam ser reaproveitadas para afinação do modelo.

Figura 3 – Diagrama de Tecnologias

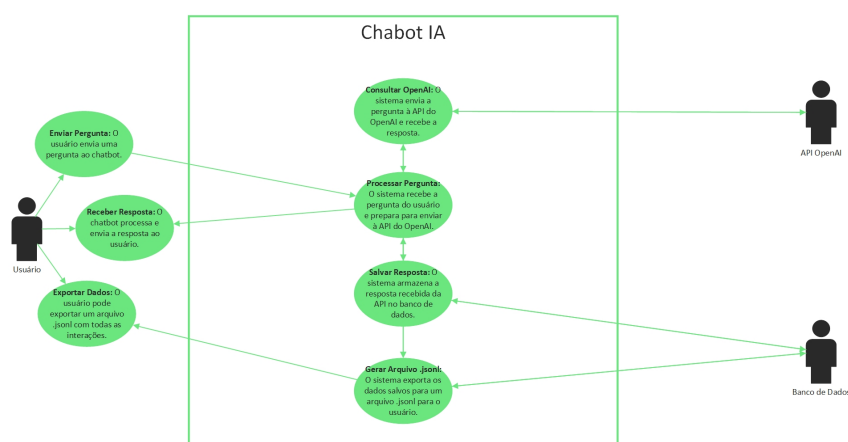


Fonte: Elaborado pelo Autor

O usuário pode realizar três ações ao utilizar o sistema, enviar perguntas, receber respostas e exportar os dados por meio de um *download*. Ao enviar e receber perguntas o sistema processa a informação e consulta o sistema externo da OpenAI, após receber uma resposta ela é processada novamente pelo sistema *Chatbot* IA e é salvo no banco de dados. Quando o usuário solicita uma exportação por meio da rota de *download*, é enviada uma solicitação ao sistema que consulta o banco de dados e gera um arquivo com a extensão *.jsonl*. Para uma melhor compreensão da interação entre os usuários e o sistema, foi elaborado um diagrama de caso de uso. Esse diagrama está apresentado na [Figura 4](#) e mostra os três atores principais: o usuário, a API da OpenAI e o banco de dados. O usuário envia perguntas ao chatbot, recebe respostas e pode exportar os dados armazenados. A API da OpenAI processa as perguntas e retorna as respostas, enquanto o banco de dados armazena as interações para

consultas futuras.

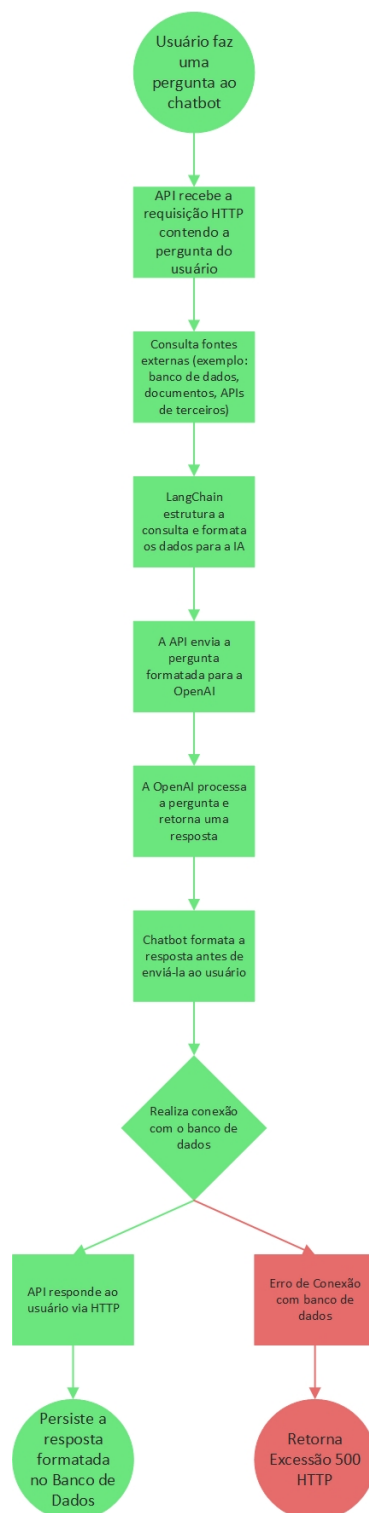
Figura 4 – Diagrama de Caso de Uso



Fonte: Elaborado pelo Autor

A resposta é gerada quando o usuário realiza uma pergunta por meio do sistema. Essa pergunta é recebida por uma requisição HTTP, e o sistema carrega as informações dos PDFs previamente configurados. Em seguida, a consulta é formatada e estruturada antes de ser enviada à API da OpenAI, que processa e retorna uma resposta. Ao receber essa resposta, o sistema a formata novamente para armazená-la no banco de dados. Somente após esse processo, a resposta é enviada ao usuário. Caso a conexão com o banco de dados falhe, é retornada uma exceção HTTP 500, indicando um erro interno no servidor. O processo de resposta automatizada do sistema foi representado em um fluxograma, conforme ilustrado na [Figura 5](#). O sistema recebe uma pergunta via protocolo HTTP, realiza a formatação e validação da entrada e, depois, envia para o modelo GPT-4o configurado com o LangChain. Após o processamento, a resposta é retornada ao usuário e armazenada no banco de dados.

Figura 5 – Fluxograma de Resposta



Fonte: Elaborado pelo Autor

Ao solicitar a exportação dos registros existentes no banco de dados, o usuário acessa uma rota específica chamada "download". Quando a requisição é feita, o sistema se conecta ao banco de dados, recupera os dados solicitados, formata essas informações e gera um arquivo .jsonl (JavaScript Object Notation Lines), que é então enviado como resposta ao usuário. Esse

processo está representado na [Figura 6](#).

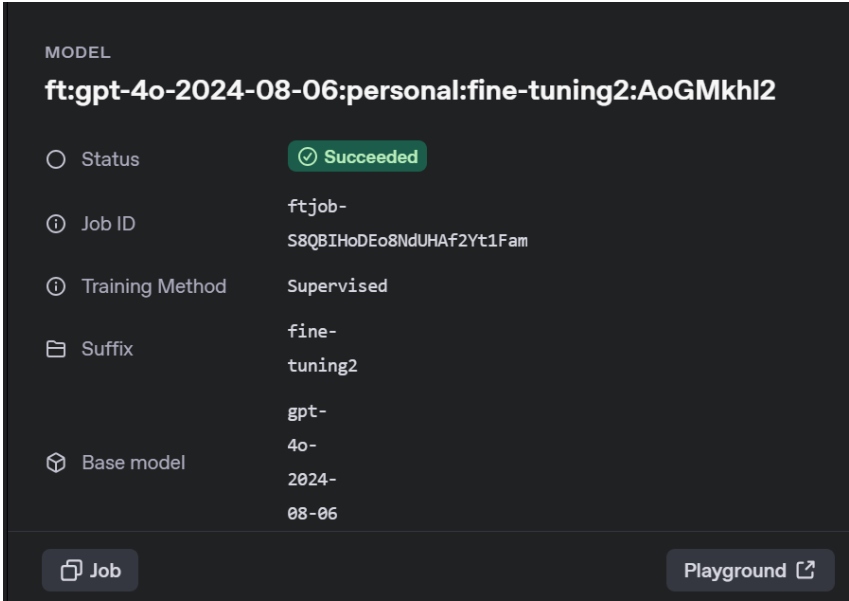
Figura 6 – Fluxograma de *Download*



Fonte: Elaborado pelo Autor

Durante o treinamento de um modelo na plataforma da OpenAI, algumas informações são geradas automaticamente. O nome do modelo é composto por uma combinação da versão do GPT, data de criação, um prefixo e o nome definido pelo usuário no momento da criação. Em seguida, é exibido o *status* do treinamento, indicando se foi concluído com sucesso ou não. Também é gerado um ID de trabalho, que identifica o processo de treinamento, seguido pelo tipo de modelo (neste trabalho, supervisionado). Após isso, aparece o sufixo, que corresponde ao nome fornecido na criação, e, por fim, o modelo base utilizado como referência. O identificador único (ID) foi incorporado diretamente no código da API em Python para garantir que todas as requisições utilizem o modelo personalizado. No caso deste sistema, o ID gerado foi `ft:gpt-4o-2024-08-06:personal:fine-tuning2:AoGMkhI2`, como pode ser observado na [Figura 7](#).

Figura 7 – Informações Básicas do *FineTuning*



Fonte: Interface OpenAI

No modelo exibido na saída, é possível observar que sua criação ocorreu em 10 de janeiro de 2025, às 17h45. O treinamento utilizou 17.010 *tokens* ao longo de três épocas, com um tamanho de lote correspondente a um dado existente no arquivo de treinamento fornecido. A taxa de aprendizagem utilizada foi igual a 2, e a semente aleatória gerada automaticamente pela plataforma foi 42. Essas informações do ID, a semente aleatória (*seed*), o número de épocas, o tamanho do lote (*batch size*) e a taxa de aprendizagem (*learning rate*) estão destacadas na Figura 8.

Figura 8 – Informações Básicas do *FineTuning 2*

Output model	08-06:personal:fine-tuning2:AoGMkhI2
Created at	10 de jan. de 2025, 17:45
Trained tokens	17.010
Epochs	3
Batch size	1
LR multiplier	2
Seed	42

Fonte: Interface OpenAI

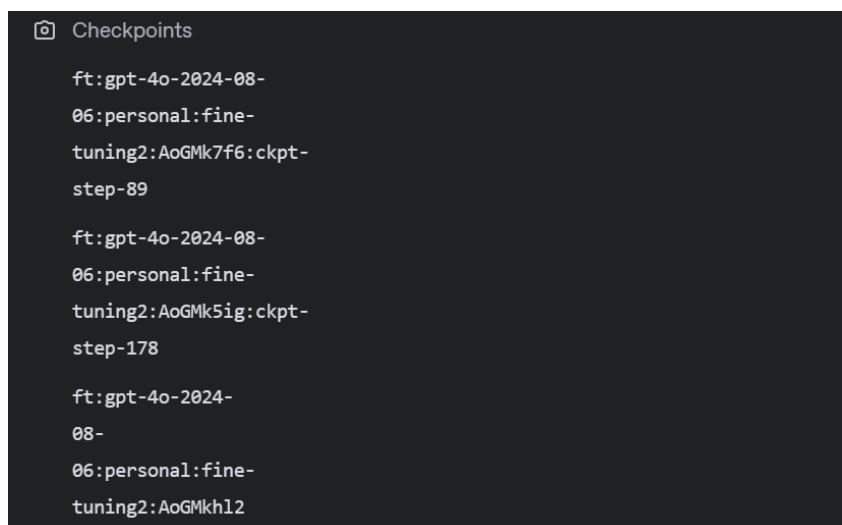
O processo é acompanhado por mensagens de status, que documentam cronologicamente cada etapa. Após a criação do modelo, ele passa por uma fase de validação do arquivo fornecido. Uma vez validado, o *status* é atualizado e inicia-se o processo de *fine-tuning*. Durante esse processo, são gerados três *checkpoints* utilizáveis, correspondentes à quantidade de épocas informadas no momento da configuração. No exemplo analisado, os *checkpoints* ocorrem nos passos 89, 178 e, por fim, na conclusão do treinamento, quando o modelo recebe o *status* de "treinamento concluído com sucesso". O processo completo de *fine-tuning* levou aproximadamente doze minutos e foi documentado passo a passo na [Figura 9](#), onde são mostradas as fases desde o envio do *dataset* até a conclusão do treinamento.

Figura 9 – Passos de Treinamento do *FineTuning*

Messages	Metrics
17:57:59	The job has successfully completed
17:57:55	New fine-tuned model created
17:57:55	Checkpoint created at step 178
17:57:55	Checkpoint created at step 89
17:46:23	Fine-tuning job started
17:46:20	Files validated, moving job to queued state
17:45:13	Validating training file: file-AhZF6fLhSPdA1uKGf6onqi
17:45:13	Created fine-tuning job: ftjob-S8QBIHoDEo8NdUHAf2Yt1Fam

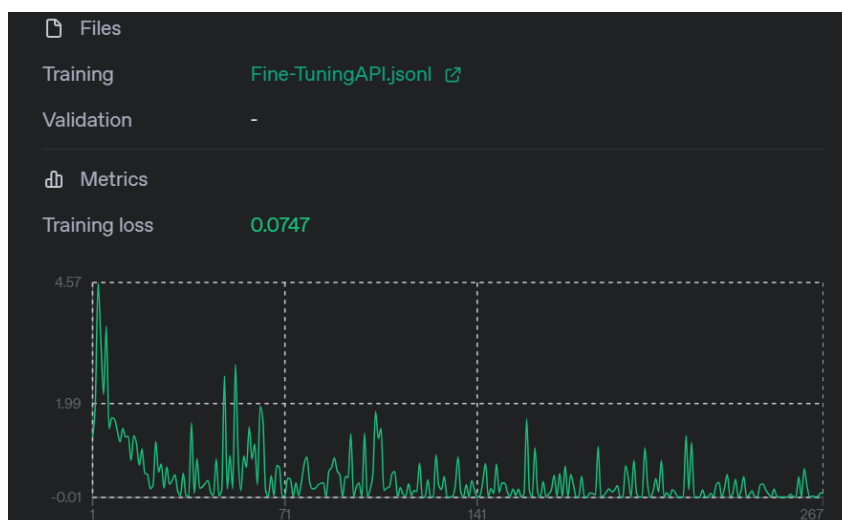
Fonte: Interface OpenAI

Ao final do processo, é fornecida a ordem cronológica das épocas juntamente com os identificadores únicos de cada uma, que podem ser utilizados posteriormente, se necessário. A OpenAI também disponibiliza os *checkpoints* correspondentes a cada época do modelo treinado, permitindo a análise do desempenho do modelo em diferentes estágios do aprendizado. Essa funcionalidade está ilustrada na [Figura 10](#).

Figura 10 – Checkpoints do *FineTuning*

Fonte: Interface OpenAI

O arquivo utilizado no processo de treinamento permanece disponível para consulta, juntamente com uma métrica que indica a taxa de perda (*loss rate*) durante o treinamento, cujo valor final foi de 0,0747. Essa métrica representa a quantidade de informações não assimiladas pelo modelo (quanto mais próxima de 1, maior a perda). Todo o processo é representado graficamente, mostrando que a maior taxa de perda ocorreu durante a primeira época. A partir da segunda, a perda cai pela metade e, ao final do treinamento, a taxa já se apresenta consideravelmente baixa. A Figura 11 exibe esse gráfico, demonstrando a evolução da taxa de perda ao longo do processo e destacando a boa consistência e aprendizado do modelo com os dados fornecidos.

Figura 11 – Métricas do *FineTuning*

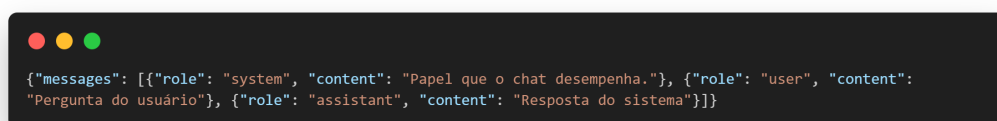
Fonte: Interface OpenAI

De acordo com a (OPEN AI 2024), o processo de *fine-tuning* permite treinar modelos personalizados a partir de exemplos fornecidos pelo próprio desenvolvedor. Para isso, é necessário utilizar um arquivo no formato **JSONL** (*JavaScript Object Notation Lines*), no qual cada linha representa um objeto estruturado em pares chave-valor.

Esses objetos contêm mensagens que definem os papéis (*roles*) no contexto da conversa, sendo eles: o papel *system*, responsável por fornecer instruções gerais sobre o comportamento do modelo; o papel *user*, que representa a pergunta ou solicitação do usuário; e o papel *assistant*, que corresponde à resposta esperada do modelo.

Conforme ilustrado na Figura 12, cada linha do arquivo JSONL inclui a pergunta do usuário e a resposta esperada do assistente, podendo também conter instruções adicionais no papel *system*, como tom de voz, nível de formalidade ou estilo da resposta, que orientam o comportamento do modelo durante o treinamento.

Figura 12 – Formato do *FineTuning*

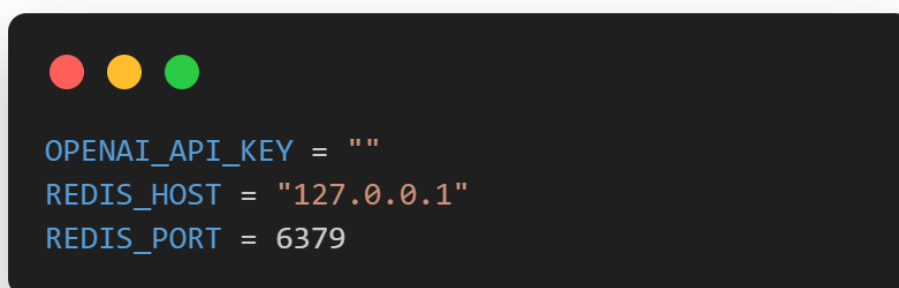


```
{ "messages": [ { "role": "system", "content": "Papel que o chat desempenha." }, { "role": "user", "content": "Pergunta do usuário" }, { "role": "assistant", "content": "Resposta do sistema" } ] }
```

Fonte: Elaborado pelo Autor

Durante o desenvolvimento do sistema, algumas informações sensíveis tornaram-se essenciais para seu funcionamento. Por esse motivo, foi necessária a criação de um arquivo de variáveis de ambiente para evitar o vazamento desses dados no código-fonte. A variável **OPENAI_API_KEY** armazena a chave de acesso fornecida pela OpenAI. As variáveis **REDIS_HOST** e **REDIS_PORT** armazenam, respectivamente, o endereço IP de conexão com o banco de dados (neste caso, 127.0.0.1) e a porta utilizada para essa conexão (porta 6379). Com o arquivo de variáveis de sistema e o modelo treinado e disponível para uso, foi desenvolvida uma API utilizando FastAPI em Python para disponibilizá-lo aos usuários. Todas as informações descritas na Figura 13 são fundamentais para o desenvolvimento e a execução do sistema, detalhando as variáveis e seus valores que possibilitam a conexão com o banco de dados e o acesso à API da OpenAI. A criação do arquivo .env permitiu construir uma estrutura de código mais limpa, segura e organizada.

Figura 13 – Variáveis do Sistema



Fonte: Elaborado pelo Autor

Para o desenvolvimento da conexão com o banco de dados e com a API da OpenAI, são utilizadas as bibliotecas *dotenv*, *os* e *redis*. Essas bibliotecas têm funções específicas: carregar as variáveis de ambiente do arquivo `.env`, acessar essas variáveis por meio do sistema operacional e instanciar a conexão com o Redis, respectivamente. A configuração tem início com o carregamento do arquivo `.env`, utilizando o comando **`load_dotenv()`**. Em seguida, a variável **`api_key`** é instanciada com o valor da chave configurada no arquivo de variáveis do sistema. Caso o carregamento dessa variável falhe, é exibida uma mensagem de erro indicando a ausência da chave de API. Após isso, a conexão com o banco de dados Redis é estabelecida, carregando os valores de **`REDIS_HOST`** e **`REDIS_PORT`** do arquivo `.env`. A conexão é instanciada utilizando o parâmetro **`decode_responses=True`**, que garante que os dados retornados pelo Redis sejam tratados como *strings* nativas do Python, em vez de *bytes*. Dessa forma, a configuração inicial da API é realizada com a leitura das variáveis do sistema, organizadas de maneira segura no arquivo `.env`, permitindo o uso de dados sensíveis como chaves de acesso e parâmetros de conexão. Esse processo está representado na [Figura 14](#).

Figura 14 – Configuração Inicial

A terminal window with a dark background and light-colored text. The code is written in Python and includes imports for dotenv, os, and Redis. It defines environment variables for API keys and Redis, and initializes a Redis client. The code is as follows:

```
from dotenv import load_dotenv
import os
from redis import Redis

load_dotenv()

api_key = os.getenv("OPENAI_API_KEY")
if not api_key:
    raise ValueError("A chave OPENAI_API_KEY não foi encontrada. Verifique o arquivo .env.")

redis_host = os.getenv("REDIS_HOST", "localhost")
redis_port = os.getenv("REDIS_PORT", 6379)
redis_db = Redis(host=redis_host, port=redis_port, decode_responses=True)
```

Fonte: Elaborado pelo Autor

O processo de carregamento do modelo tem início com a importação das bibliotecas e módulos da biblioteca **LangChain**, além do módulo **os**, responsável por interações com o sistema operacional. Em seguida, a variável `base_dir` define o diretório onde o *script* está localizado, enquanto `pdf_path` monta o caminho completo até o arquivo PDF, localizado dentro da pasta `src`. Com o caminho completo definido, é criado um carregador para o PDF por meio da variável `loader`, que realiza a leitura do arquivo e o transforma em uma lista de documentos, armazenada na variável `documents`. Para que o texto possa ser processado de forma eficiente, é necessário dividi-lo em partes menores. Esse processo é realizado pelas variáveis `text_splitter` e `docs`, que configuram e executam a divisão do conteúdo em blocos de até 1000 caracteres, com uma sobreposição de 100 caracteres entre eles, garantindo melhor continuidade sem perda de contexto. A etapa seguinte consiste na geração dos *embeddings*, que são representações vetoriais dos trechos de texto. A variável `embeddings` inicializa o gerador de vetores utilizando a API da OpenAI, enquanto `vectorstore` constrói um banco vetorial com base nos documentos divididos, permitindo buscas eficientes por similaridade textual por meio da biblioteca FAISS. Por fim, a criação da cadeia de Perguntas e Respostas é realizada pela variável `qa_chain`, que instancia o modelo de linguagem previamente treinado e ajustado (llm), neste caso, uma versão personalizada do **GPT-4o**. Além disso, configura-se o `retriever`, que utiliza o banco vetorial FAISS para localizar os trechos mais relevantes ao responder às perguntas. Essa etapa está representada na [Figura 15](#), ilustrando o momento em que o modelo treinado é integrado ao sistema, possibilitando sua aplicação prática.

Figura 15 – Criação de Modelo LLM

```
import os
from langchain_community.document_loaders import PyPDFLoader
from langchain.text_splitter import CharacterTextSplitter
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.chains import RetrievalQA
from langchain.chat_models import ChatOpenAI

base_dir = os.path.dirname(os.path.abspath(__file__))
pdf_path = os.path.join(base_dir, "src", "edital_ifsp_itapetininga.pdf")

loader = PyPDFLoader(pdf_path)
documents = loader.load()

text_splitter = CharacterTextSplitter(chunk_size=1000, chunk_overlap=100)
docs = text_splitter.split_documents(documents)

embeddings = OpenAIEmbeddings()
vectorstore = FAISS.from_documents(docs, embeddings)

qa_chain = RetrievalQA.from_chain_type(
    llm=ChatOpenAI(model="ft:gpt-4o-2024-08-06:personal:fine-tuning2:AoGMkh12"),
    retriever=vectorstore.as_retriever()
)
```

Fonte: Elaborado pelo Autor

A construção da API tem início com a utilização do *framework* **FastAPI**. A classe **FastAPI** é responsável por instanciar a aplicação, enquanto a **HTTPException** permite o tratamento de erros personalizados. São importados também o **CORSMiddleware**, necessário para liberar o acesso à API por diferentes origens (CORS), e o **BaseModel**, do *Pydantic*, utilizado para validar os dados recebidos nas requisições. Além dessas bibliotecas, dois módulos internos também são importados: **redis_db**, que gerencia a conexão com o banco de dados Redis, e **qa_chain**, que representa a cadeia de Perguntas e Respostas utilizada no sistema. As bibliotecas nativas **json** e os completam a configuração, auxiliando na manipulação de arquivos e caminhos. A aplicação é iniciada com a criação da instância `app = FastAPI()`. Em seguida, é configurado o *middleware* CORS por meio do método `add_middleware()`, permitindo o acesso à API a partir de qualquer origem, com qualquer método HTTP e cabeçalho. Essa configuração é essencial para garantir a integração com aplicações *frontend*. A classe **QueryRequest** é definida com base na estrutura do **BaseModel**, contendo um único atributo chamado `query`, do tipo *string*, que representa a pergunta enviada pelo usuário. Por fim, é criada uma rota do tipo GET, mapeada no caminho `"/"`, que retorna uma mensagem simples indicando que a API está ativa e funcionando corretamente. Essa configuração é apresentada na [Figura 16](#).

Figura 16 – Configuração de Rota

A screenshot of a code editor with a dark background and light-colored text. The code is written in Python and defines a FastAPI application. It includes imports for FastAPI, CORS, Pydantic, Redis, and other utilities. The application is configured with CORS middleware and a root endpoint that returns a message.

```
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from config import redis_db
from model import qa_chain
import json
import os

app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

class QueryRequest(BaseModel):
    query: str

@app.get("/")
async def root():
    return {"message": "API Funcionando!"}
```

Fonte: Elaborado pelo Autor

A aplicação conta com uma rota principal do tipo POST, no caminho /duvidas, que tem como objetivo receber a pergunta enviada pelo usuário e retornar uma resposta gerada pelo modelo. Essa funcionalidade é definida pela função `answer_query()`, que recebe como parâmetro um objeto do tipo `QueryRequest`, contendo a *string* `query` com a dúvida a ser processada. A partir disso, a variável `response` armazena o resultado da função `invoke()` da cadeia `qa_chain`, responsável por gerar a resposta com base na pergunta recebida. Em seguida, é construída a estrutura `data`, que organiza as mensagens trocadas entre sistema, usuário e assistente. O campo `system` define o comportamento do modelo, informando que ele deve agir como um atendente do Instituto Federal de São Paulo - Campus de Itapetininga. O campo `user` registra a pergunta enviada pelo usuário e o campo `assistant` armazena a resposta gerada pelo modelo. A estrutura `data` é então armazenada no banco de dados Redis por meio do método `rpush()`, que insere o conteúdo serializado com `json.dumps()` na lista `queries_responses`. Por fim, os dados são retornados como resposta da API. Caso ocorra alguma exceção durante a execução, é lançado um erro do tipo `HTTPException`, com o status 500, indicando uma falha no *backend*. Essa etapa está representada na [Figura 17](#).

Figura 17 – Configuração de Rota de Perguntas

```
@app.post("/duvidas")
async def answer_query(request: QueryRequest):
    try:
        response = qa_chain.invoke(request.query)

        data = {
            "messages": [
                {
                    "role": "system",
                    "content":
"Você é um atendente do Instituto Federal de São Paulo do Campus de Itapetininga."
                },
                {
                    "role": "user",
                    "content": request.query
                },
                {
                    "role": "assistant",
                    "content": response["result"]
                }
            ]
        }
        redis_db.rpush("queries_responses", json.dumps(data))

        return data
    except Exception as e:
        print("Erro no Backend:", str(e))
        raise HTTPException(status_code=500, detail=str(e))
```

Fonte: Elaborado pelo Autor

Para realizar o *download* da base de dados, foi criado uma rota `/download` que contém uma função responsável por exportar os dados salvos no Redis no formato JSON Lines, que é muito usado quando queremos treinar modelos de linguagem com exemplos personalizados. Ela começa buscando todos os registros da lista `queries_responses` no Redis usando o comando `lrange`. Depois disso, cada item da lista (que está como uma string JSON) é convertido em um dicionário Python usando o `json.loads()`. Na sequência, o código abre (ou cria) o arquivo `ArquivoFineTuning.jsonl` e começa a escrever os dados nele. Cada item é escrito como um objeto JSON em uma linha do arquivo, e o parâmetro `ensure_ascii=False` é usado para manter os acentos e caracteres especiais corretamente. Por fim, uma quebra de linha é adicionada entre os registros, exceto no último, para manter o padrão do formato `.jsonl` corretamente. Toda essa lógica pode ser vista na [Figura 18](#).

Figura 18 – Configuração de Rota de *Download*

```
@app.get("/download")
async def download():
    data = redis_db.lrange("queries_responses", 0, -1)

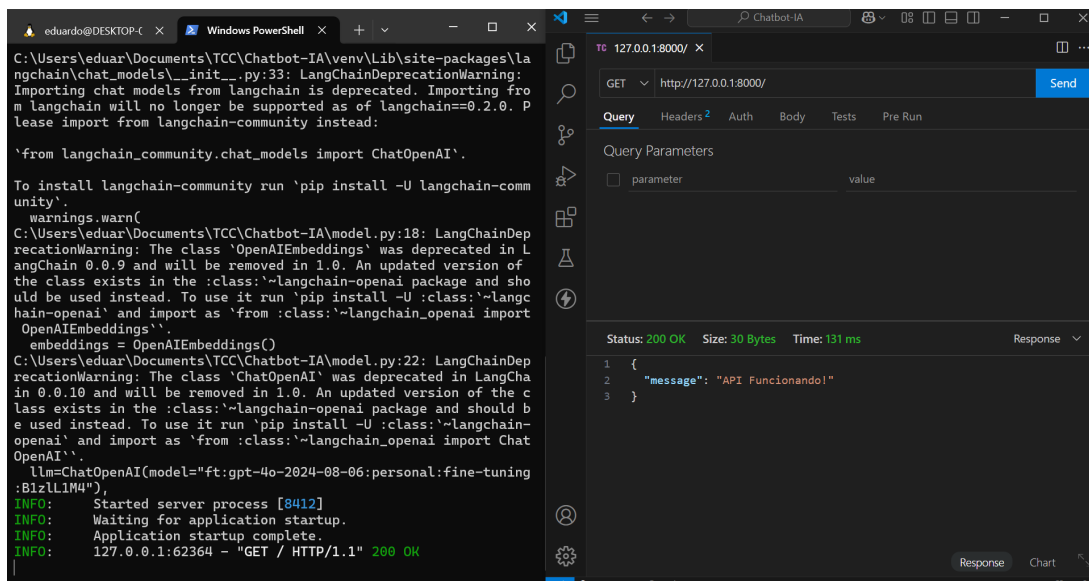
    if data:
        decoded_data = [json.loads(item) for item in data]

        with open("ArquivoFineTuning.jsonl", "w", encoding="utf-8") as file:
            for index, item in enumerate(decoded_data):
                json.dump(item, file, ensure_ascii=False)
                if index < len(decoded_data) - 1:
                    file.write("\n")
```

Fonte: Elaborado pelo Autor

Ao acessar a rota de teste "/", é possível verificar, na tela à direita, um dicionário contendo a chave `message` com o valor "API funcionando", o que confirma que a API está ativa e operando corretamente. Simultaneamente, a tela à esquerda exibe no terminal os *logs* de inicialização do sistema, iniciados pelo processo de ID 8412, além das informações da requisição recebida. Como saída, observam-se o IP e a porta utilizados, o método HTTP empregado e o código de *status* 200, indicando sucesso na operação. Dessa forma, a API está pronta para ser consumida por diferentes meios. Uma das formas de interação é por meio de ferramentas de linha de comando, como o cURL, que permite o envio de requisições e a visualização das respostas retornadas pelo modelo treinado, conforme ilustrado na [Figura 19](#).

Figura 19 – Formato Teste

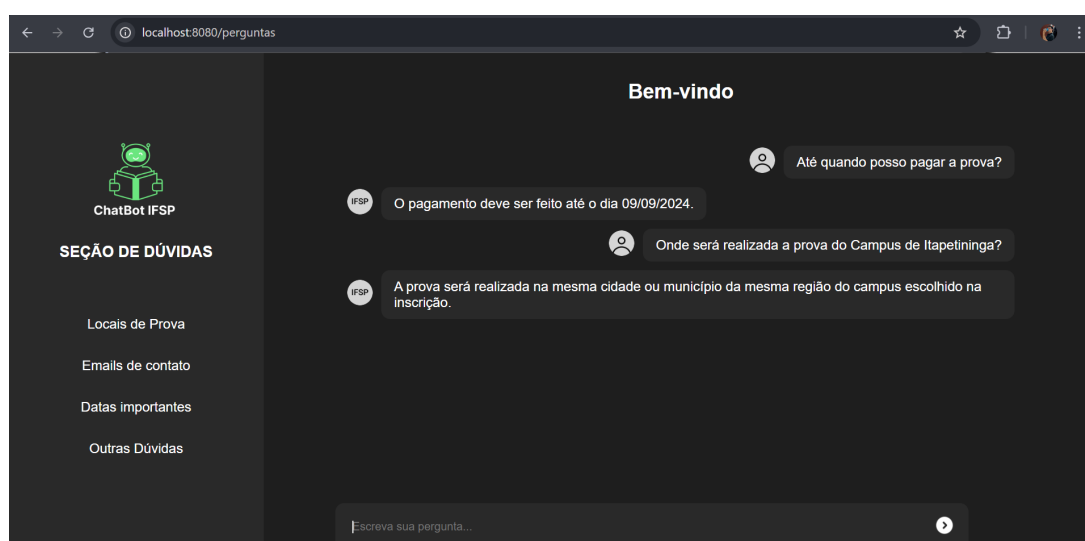


Fonte: Elaborado pelo Autor

Com o objetivo de aprimorar a experiência do usuário final, foi desenvolvida uma

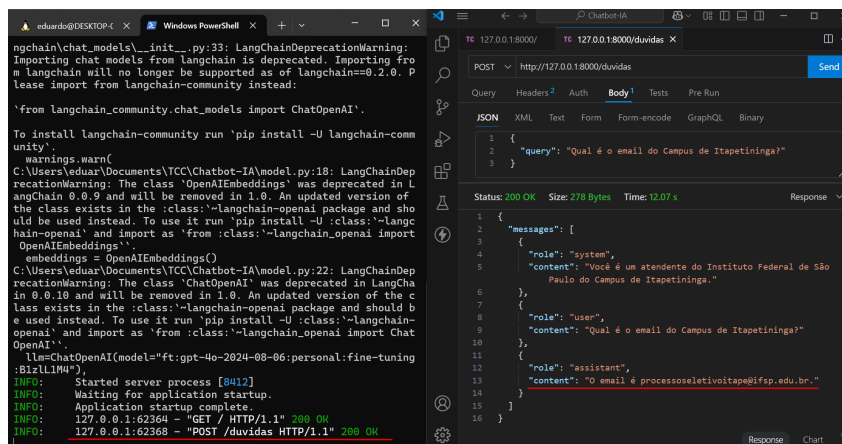
interface web intuitiva, composta por um menu lateral contendo a logo do sistema e filtros de pesquisa organizados por seções específicas do edital, como Locais de Prova, E-mails de Contato, Datas Importantes e Outras Dúvidas. Na página principal, as interações são apresentadas de forma clara, com as perguntas do usuário exibidas à direita e as respostas geradas pelo sistema à esquerda. Ao final da interface, há um campo de entrada para digitação da dúvida e um botão de envio. Essa abordagem torna a comunicação com o sistema mais acessível e amigável, conforme representado na [Figura 20](#).

Figura 20 – Interface VueJS



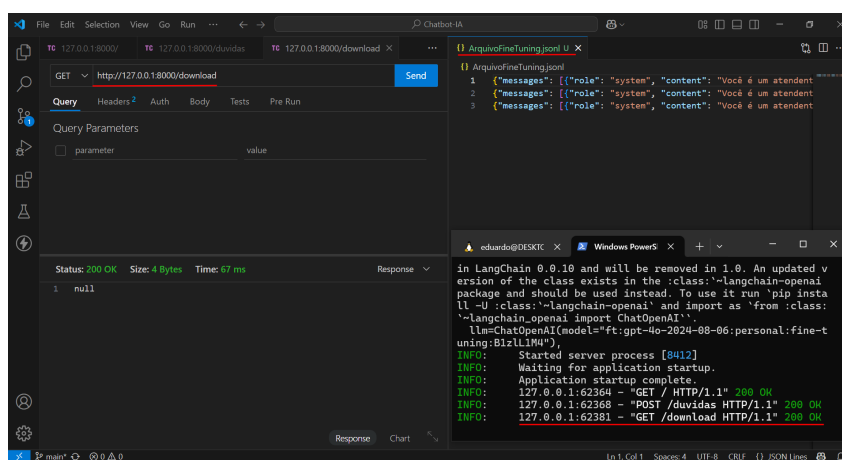
Fonte: Elaborado pelo Autor

Ao testar a rota principal da aplicação, `"/duvidas"`, é exibido, à direita da interface, o ID do sistema, a rota acessada e o método HTTP utilizado, neste caso o `POST`. O corpo da requisição é um JSON contendo a chave `query` com o valor `"Qual é o email do Campus de Itapetininga?"`. Como resposta, o sistema retorna um código de *status* 200, um corpo de 278 *bytes* e um tempo de resposta de 12,07 segundos. A resposta inclui ainda uma estrutura no formato JSON, na qual a chave `messages` armazena três dicionários, representando as interações entre o sistema, o usuário e o assistente, por meio das chaves `role` e `content`. Nelas, o sistema se comporta como um atendente do Instituto Federal de São Paulo (Campus Itapetininga), recebendo a dúvida do usuário e fornecendo a resposta correspondente. À esquerda, é possível visualizar os *logs* da aplicação, incluindo o momento da inicialização, os dados da requisição recebida, como IP, porta, método utilizado e o código de status gerado. Esses resultados podem ser observados na [Figura 21](#).

Figura 21 – Teste de Consumo pelo *ThunderClient* na Rota de Perguntas

Fonte: Elaborado pelo Autor

Por meio da extensão *ThunderClient*, integrada ao *Visual Studio Code*, foi possível consumir a rota `"/download"`, cuja função é gerar um arquivo contendo todas as interações registradas no banco de dados. À esquerda da imagem, visualizam-se as informações referentes ao IP, à rota acessada e ao método HTTP utilizado — neste caso, o GET. Logo abaixo, são apresentados o código de *status* 200 (OK), o tamanho da resposta (4 bytes) e o tempo de resposta do sistema, que foi de 67 milissegundos. À direita, há duas seções: a primeira exibe o arquivo gerado pelo sistema, que reúne todos os registros persistidos no banco de dados no formato JSONL; a segunda apresenta os *logs* de inicialização do sistema, além dos testes realizados nas três rotas disponíveis (`"/"`, `"/duvidas"` e `"/download"`). Todos os testes retornaram com sucesso, conforme indica o código de *status* 200 ilustrado na Figura 22.

Figura 22 – Teste de Consumo pelo *ThunderClient* na Rota de Download

Fonte: Elaborado pelo Autor

O arquivo gerado pela rota `"/download"` apresenta as interações no formato adequado para fins de treinamento na plataforma da OpenAI, podendo ser manipulado para que o sistema aprenda a responder as perguntas que ele não sabe. Todas as perguntas realizadas durante os testes, especialmente aquelas baseadas no conteúdo do edital, estão registradas no arquivo, acompanhadas das respectivas respostas fornecidas pelo sistema. Alguns questionamentos de conhecimento geral o sistema é capaz de responder, visto que utiliza o modelo GPT-4o como base. No entanto, ao receber perguntas muito específicas e fora do escopo previsto, o sistema responde com a mensagem: "Não sei, essa informação não está nas referências". A Figura 23 ilustra o resultado desse processo de persistência.

Figura 23 – Formato de *Download* 2

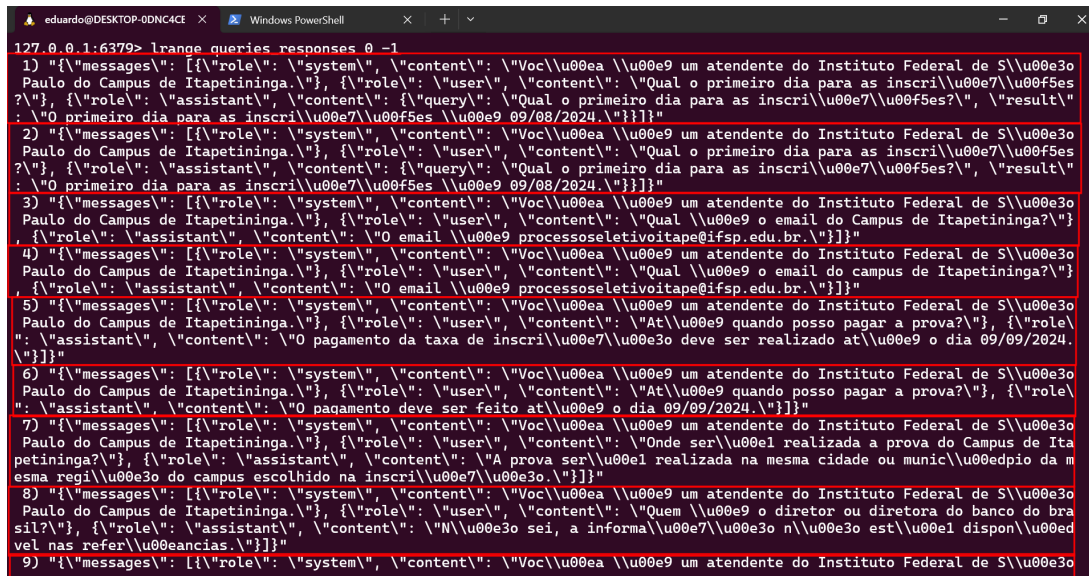
```
{
  "messages": [
    {
      "role": "system",
      "content": "Você é um atendente do Instituto Federal de São Paulo do Campus de Itapetininga."
    },
    {
      "role": "user",
      "content": "Qual o primeiro dia para as inscrições?"
    },
    {
      "role": "assistant",
      "content": {
        "query": "Qual o primeiro dia para as inscrições?",
        "result": "O primeiro dia para as inscrições é 09/08/2024."
      }
    },
    {
      "role": "user",
      "content": "Qual o primeiro dia para as inscrições?"
    },
    {
      "role": "assistant",
      "content": {
        "query": "Qual o primeiro dia para as inscrições?",
        "result": "O primeiro dia para as inscrições é 09/08/2024."
      }
    },
    {
      "role": "user",
      "content": "Qual é o email do Campus de Itapetininga?"
    },
    {
      "role": "assistant",
      "content": "O email é processosseletivoitape@ifsp.edu.br."
    },
    {
      "role": "user",
      "content": "Qual é o email do campus de Itapetininga?"
    },
    {
      "role": "assistant",
      "content": "O email é processosseletivoitape@ifsp.edu.br."
    },
    {
      "role": "user",
      "content": "Até quando posso pagar a prova?"
    },
    {
      "role": "assistant",
      "content": "O pagamento da taxa de inscrição deve ser realizado até o dia 09/09/2024."
    },
    {
      "role": "user",
      "content": "Até quando posso pagar a prova?"
    },
    {
      "role": "assistant",
      "content": "O pagamento deve ser feito até o dia 09/09/2024."
    },
    {
      "role": "user",
      "content": "Onde será realizada a prova do Campus de Itapetininga?"
    },
    {
      "role": "assistant",
      "content": "A prova será realizada na mesma cidade ou município da mesma região do campus escolhido na inscrição."
    },
    {
      "role": "user",
      "content": "Quem é o diretor ou diretora do banco do Brasil?"
    },
    {
      "role": "assistant",
      "content": "Não sei, a informação não está disponível nas referências."
    },
    {
      "role": "user",
      "content": "Quem é o desenvolvedor deste projeto?"
    },
    {
      "role": "assistant",
      "content": "O desenvolvedor é o Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP)."
    },
    {
      "role": "user",
      "content": "Quando foi o lançamento do primeiro foguete ao espaço?"
    },
    {
      "role": "assistant",
      "content": "O primeiro foguete foi lançado ao espaço em 3 de outubro de 1942."
    },
    {
      "role": "user",
      "content": "Não foi em 4 de outubro de 1957?"
    },
    {
      "role": "assistant",
      "content": "Não sei. Poderia dar mais contexto para que eu possa ajudar?"
    },
    {
      "role": "user",
      "content": "Em 4 de outubro de 1957, a União Soviética colocou em órbita o Sputnik 1, primeiro foguete espacial. Estou errado?"
    },
    {
      "role": "assistant",
      "content": "Sim, está um pouco errado. O Sputnik 1 foi o primeiro satélite artificial colocado em órbita, não um foguete."
    },
    {
      "role": "user",
      "content": "O Sputnik 1 ainda existe?"
    },
    {
      "role": "assistant",
      "content": "Não, o Sputnik 1 não existe mais. Após completar sua missão, ele reentrou na atmosfera da Terra e queimou."
    },
    {
      "role": "user",
      "content": "Foi em 6 de agosto de 2024?"
    },
    {
      "role": "assistant",
      "content": "Não foi No dia 4 de janeiro de 1958?"
    },
    {
      "role": "assistant",
      "content": "Não."
    },
    {
      "role": "user",
      "content": "Você é um atendente do Instituto Federal de São Paulo do Campus de Itapetininga."
    },
    {
      "role": "assistant",
      "content": "me passa o link de onde tirou essa informação"
    },
    {
      "role": "assistant",
      "content": "A informação foi retirada de https://processosseletivo.ifsp.edu.br/"
    }
  ]
}
```

Fonte: Elaborado pelo Autor

Por fim, ao acessar o banco de dados via WSL por um terminal no *Windows* e fornecer o comando do Redis `lrange queries_responses 0 -1` é obtido como respostas todas as interações realizadas pelo sistema que estão são armazenadas no banco de dados. Nesse processo os caracteres especiais não funcionam corretamente, então em alguns casos trechos com caracteres unicode como `"u00e9"` são comuns por representarem a letra *"é"*. Cada entrada corresponde a um objeto JSON com os dados completos da pergunta, resposta

e metadados adicionais. Essa estrutura está descrita na [Figura 24](#).

Figura 24 – Banco de Dados



```

127.0.0.1:6379> lrange queries responses 0 -1
1) {"messages": [{"role": "system", "content": "Voc\u00ea \u00e9 um atendente do Instituto Federal de S\u00e3o Paulo do Campus de Itapetininga."}, {"role": "user", "content": "Qual o primeiro dia para as inscri\u00e7\u00f5es?"}, {"role": "assistant", "content": "O primeiro dia para as inscri\u00e7\u00f5es \u00e9 09/08/2024."}]}
2) {"messages": [{"role": "system", "content": "Voc\u00ea \u00e9 um atendente do Instituto Federal de S\u00e3o Paulo do Campus de Itapetininga."}, {"role": "user", "content": "Qual o primeiro dia para as inscri\u00e7\u00f5es?"}, {"role": "assistant", "content": "O primeiro dia para as inscri\u00e7\u00f5es \u00e9 09/08/2024."}]}
3) {"messages": [{"role": "system", "content": "Voc\u00ea \u00e9 um atendente do Instituto Federal de S\u00e3o Paulo do Campus de Itapetininga."}, {"role": "user", "content": "Qual \u00e9 o email do Campus de Itapetininga?"}, {"role": "assistant", "content": "O email \u00e9 processosseletivoitape@ifsp.edu.br."}]}
4) {"messages": [{"role": "system", "content": "Voc\u00ea \u00e9 um atendente do Instituto Federal de S\u00e3o Paulo do Campus de Itapetininga."}, {"role": "user", "content": "Qual \u00e9 o email do campus de Itapetininga?"}, {"role": "assistant", "content": "O email \u00e9 processosseletivoitape@ifsp.edu.br."}]}
5) {"messages": [{"role": "system", "content": "Voc\u00ea \u00e9 um atendente do Instituto Federal de S\u00e3o Paulo do Campus de Itapetininga."}, {"role": "user", "content": "At\u00e9 quando posso pagar a prova?"}, {"role": "assistant", "content": "O pagamento da taxa de inscri\u00e7\u00e3o deve ser realizado at\u00e9 o dia 09/09/2024."}]}
6) {"messages": [{"role": "system", "content": "Voc\u00ea \u00e9 um atendente do Instituto Federal de S\u00e3o Paulo do Campus de Itapetininga."}, {"role": "user", "content": "At\u00e9 quando posso pagar a prova?"}, {"role": "assistant", "content": "O pagamento deve ser feito at\u00e9 o dia 09/09/2024."}]}
7) {"messages": [{"role": "system", "content": "Voc\u00ea \u00e9 um atendente do Instituto Federal de S\u00e3o Paulo do Campus de Itapetininga."}, {"role": "user", "content": "Onde ser\u00e1 realizada a prova do Campus de Itapetininga?"}, {"role": "assistant", "content": "A prova ser\u00e1 realizada na mesma cidade ou munic\u00edpio da mesma regi\u00e3o do campus escolhido na inscri\u00e7\u00e3o."}]}
8) {"messages": [{"role": "system", "content": "Voc\u00ea \u00e9 um atendente do Instituto Federal de S\u00e3o Paulo do Campus de Itapetininga."}, {"role": "user", "content": "Quem \u00e9 o diretor ou diretora do banco de dados?"}, {"role": "assistant", "content": "N\u00e3o sei, a informa\u00e7\u00e3o n\u00e3o est\u00e1 dispon\u00edvel nas refer\u00eancias."}]}
9) {"messages": [{"role": "system", "content": "Voc\u00ea \u00e9 um atendente do Instituto Federal de S\u00e3o Paulo do Campus de Itapetininga."}]}

```

Fonte: Elaborado pelo Autor

5 RESULTADOS

O presente trabalho caracteriza-se como uma pesquisa de natureza aplicada, uma vez que buscou, por meio do desenvolvimento de um sistema utilizando métodos e tecnologias atuais, propor uma solução prática para o problema identificado ao longo deste estudo.

O sistema foi estruturado em uma arquitetura composta por três principais camadas: uma interface *web* desenvolvida em Vue.js, responsável pela interação com o usuário; uma API em Python utilizando o *framework* FastAPI, responsável pelo processamento das requisições e pela comunicação com o modelo de linguagem; e um banco de dados Redis, utilizado para armazenar as interações realizadas. Essa divisão possibilitou uma comunicação eficiente entre os componentes, maior organização do código e flexibilidade para futuras melhorias.

Durante a construção do código e a configuração do modelo, foram encontrados alguns desafios, especialmente relacionados ao consumo de APIs externas, que exigem o envio de *tokens* e, conseqüentemente, geram custos por utilização. O desenvolvimento das *chains* (cadeias de execução) mostrou-se um processo interessante e didático, permitindo estruturar uma lógica simples e eficiente para o fluxo do sistema.

Um dos principais diferenciais deste projeto foi a utilização do processo de *FineTuning* disponibilizado pela OpenAI, que permitiu o treinamento de um modelo ajustado às necessidades do contexto escolar. Esse processo exigiu a preparação de um conjunto de dados no formato JSONL, contendo exemplos de perguntas e respostas que orientaram o comportamento do modelo. Além disso, foi necessário configurar parâmetros de treinamento, como taxa de aprendizado, número de épocas e tamanho do lote, de forma a otimizar os resultados obtidos.

No uso de um banco de dados não relacional, um dos desafios mais relevantes foi a formatação adequada dos arquivos JSON. Durante a implementação da lógica em Python responsável por persistir as informações, observou-se que a codificação dos caracteres não era mantida corretamente, resultando em inconsistências, principalmente com caracteres especiais. Assim, as informações retornavam com símbolos incorretos. Com o apoio de fóruns e da comunidade de desenvolvedores, foi possível identificar a causa do problema e implementar uma solução eficiente utilizando codificação no padrão UTF-8, garantindo a integridade dos dados armazenados e a correta leitura das informações posteriormente.

Após as configurações e ajustes, o sistema demonstrou respostas mais coerentes e contextualizadas às perguntas dos usuários, especialmente nas consultas relacionadas ao ambiente escolar, documentos e procedimentos institucionais. O desempenho geral manteve-se satisfatório, apresentando baixo tempo de resposta mesmo em situações de múltiplas requisições simultâneas.

O desenvolvimento do sistema proporcionou uma compreensão aprofundada sobre o funcionamento de APIs, bancos de dados não relacionais e integração com modelos de linguagem. Além disso, reforçou a importância do tratamento adequado de dados, da organização modular do

código e da adoção de boas práticas de codificação para garantir a confiabilidade, escalabilidade e manutenção da aplicação.

6 CONCLUSÃO

O presente trabalho teve como objetivo desenvolver um sistema de] *chatbot* utilizando inteligência artificial para auxiliar candidatos no processo seletivo do IFSP – Campus Itapetininga, tornando-se uma alternativa mais acessível e rápida para a consulta de informações presentes nos editais oficiais.

Por meio da aplicação de tecnologias atuais como *LangChain*, *FastAPI*, *GPT-4o* e banco de dados *Redis*, foi possível construir uma solução funcional, testada durante o desenvolvimento. Os resultados demonstraram que, para perguntas de maior complexidade e interpretação, o *chatbot* apresentou desempenho melhor do que diante da leitura direta dos editais, reduzindo significativamente o tempo de obtenção das respostas.

O desenvolvimento deste projeto também reforça a importância do uso de modelos de linguagem de grande escala e técnicas de *FineTuning* no contexto educacional, ampliando as possibilidades de aplicação da IA em ambientes acadêmicos.

Conclui-se, portanto, que o sistema atendeu parcialmente os objetivos propostos, demonstrando viabilidade técnica e impacto positivo na experiência do usuário. Ainda assim, há espaço para melhorias, como refinamentos na interface e mecanismos de tolerância a erros de digitação, que podem ser abordados em trabalhos futuros.

6.1 TRABALHOS FUTUROS

O presente trabalho poderá originar dois artigos científicos complementares. O primeiro poderá abordar, de forma detalhada, todo o processo de criação, configuração e ajustes realizados durante o desenvolvimento do modelo de linguagem LLM. Já o segundo terá como foco a validação dos resultados obtidos, por meio de uma pesquisa aplicada com usuários, a fim de avaliar a efetividade e usabilidade do sistema proposto. Para a realização dessa etapa, será necessária a submissão de um projeto de pesquisa ao Comitê de Ética em Pesquisa (CEP), considerando o que haverá a participação de seres humanos. Parte da documentação exigida para esse processo já se encontra parcialmente preparada, facilitando assim o envio por meio da Plataforma Brasil.

Além disso, como proposta de aprimoramento no sistema, pretende-se melhorar o processamento textual, tornando-o capaz de reconhecer e tratar automaticamente erros de digitação ou de escrita. Essa melhoria visa permitir que o modelo identifique falhas na entrada do usuário e solicite a correção ou reescrita da mensagem, aumentando a eficiência e acessibilidade da interação com o *chatbot*.

Outro avanço planejado é a implementação de modelos locais e *offline*, utilizando a ferramenta *Ollama*, que possibilita a criação e execução de instâncias de modelos de linguagem diretamente no ambiente local, sem a necessidade de conexão constante com servidores externos.

Entre os modelos disponíveis estão o LLaMA 3.1, GPT-OSS, DeepSeek-R1, entre outros, cada um projetado para diferentes finalidades e contextos de aplicação. Essa abordagem busca reduzir custos de uso de APIs, aumentar a privacidade dos dados e permitir testes comparativos de desempenho entre modelos locais e em nuvem.

Referências

- Almeida 2023 ALMEIDA, M. **Machine Learning: Aprendizado Semi-Supervisionado**. 2023. Acesso em: 08 nov. 2024. Disponível em: <https://www.alura.com.br/artigos/machine-learning-aprendizado-semi-supervisionado?srsId=AfmBOoqwFTjWIJF3LKPxV0yzLk5IsifJ8ksB_olvl0QzkbeGGD26N51r>. Citado 2 vezes nas páginas 13 e 14.
- AWS 2024 AWS. **O que são Transformadores em Inteligência Artificial?** 2024. Acessado em: 03 out. 2024. Disponível em: <<https://aws.amazon.com/pt/what-is/transformers-in-artificial-intelligence/>>. Citado 2 vezes nas páginas 15 e 16.
- BOTPRESS 2024 BOTPRESS. **O que é Compreensão de Linguagem Natural (NLU)?** 2024. Acessado em: 06 dez. 2024. Disponível em: <<https://botpress.com/pt/blog/what-is-natural-language-understanding-nlu>>. Citado na página 14.
- CAPES 2024 CAPES. **Brasil produz 6,3 mil estudos sobre inteligência artificial**. 2024. Acessado em: 20 jan. 2025. Disponível em: <<https://www.gov.br/capes/pt-br/assuntos/noticias/brasil-produz-6-3-mil-estudos-sobre-inteligencia-artificial>>. Citado na página 11.
- CSP TECNOLOGIA 2024 CSP TECNOLOGIA. **Conheça 10 ferramentas de IA mais usadas**. 2024. Acesso em: 30 set. 2024. Disponível em: <<https://blog.csptecnologia.com/conheca-10-ferramentas-de-ia-mais-usadas/>>. Citado na página 12.
- DATABRICKS 2025 DATABRICKS. **TensorFlow Guide**. 2025. Acesso em: 20 jan. 2025. Disponível em: <<https://www.databricks.com/br/glossary/tensorflow-guide#:~:text=O%20TensorFlow%20%C3%A9%20uma%20biblioteca,maneira%20mais%20r%C3%A1pida%20e%20f%C3%A1cil>>. Citado na página 17.
- DATA CAMP 2024 DATA CAMP. **Top AI frameworks and libraries**. 2024. Acesso em: 30 set. 2024. Disponível em: <<https://www.datacamp.com/pt/blog/top-ai-frameworks-and-libraries>>. Citado na página 12.
- DEEP LEARNING BOOK 2024 DEEP LEARNING BOOK. **Como funcionam os transformadores em Processamento de Linguagem Natural? - Parte 1**. 2024. Acessado em: 03 out. 2024. Disponível em: <<https://www.deeplearningbook.com.br/como-funcionam-os-transformadores-em-processamento-de-linguagem-natural-parte-1/>>. Citado na página 15.
- FLOWISE AI 2025 FLOWISE AI. **FlowiseAI - Open Source AI Tool**. 2025. Acessado em: 20 jan. 2025. Disponível em: <<https://flowiseai.com/>>. Citado na página 16.
- GOOGLE CLOUD 2024 GOOGLE CLOUD. **Casos de uso: Chatbot de IA**. 2024. Acessado em: 06 dez. 2024. Disponível em: <<https://cloud.google.com/use-cases/ai-chatbot?hl=pt-BR>>. Citado na página 14.
- GOOGLE CLOUD 2024 GOOGLE CLOUD. **O que é Machine Learning? | Google Cloud**. 2024. Acesso em: 08 nov. 2024. Disponível em: <<https://cloud.google.com/learn/what-is-machine-learning?hl=pt-BR>>. Citado 2 vezes nas páginas 12 e 13.

GOOGLE CLOUD 2024 GOOGLE CLOUD. **O que é NoSQL?** 2024. Acessado em: 06 dez. 2024. Disponível em: <<https://cloud.google.com/discover/what-is-nosql?hl=pt-BR>>. Citado na página 21.

IBM 2023 IBM. **O que é LangChain?** 2023. Acessado em: 20 jan. 2025. Disponível em: <<https://www.ibm.com/br-pt/topics/langchain>>. Citado na página 16.

IBM 2024 IBM. **Modelos de Linguagem de Grande Escala (Large Language Models)**. 2024. Acessado em: 03 out. 2024. Disponível em: <<https://www.ibm.com/br-pt/topics/large-language-models>>. Citado na página 15.

LANGCHAIN 2025 LANGCHAIN. **LangChain - Official Website**. 2025. Acessado em: 20 jan. 2025. Disponível em: <<https://www.langchain.com/langchain>>. Citado na página 16.

Oliveira 2025 OLIVEIRA, A. **Quase 30% dos brasileiros são analfabetos funcionais, revela pesquisa**. 2025. Acesso em: 26 dez. 2025. Disponível em: <<https://www.educamaisbrasil.com.br/educacao/noticias/quase-30-dos-brasileiros-sao-analfabetos-funcionais-revela-pesquisa>>. Citado na página 11.

OPEN AI 2024 OPEN AI. **Fine-Tuning Guide**. 2024. Acesso em: 3 abr. 2025. Disponível em: <<https://platform.openai.com/docs/guides/fine-tuning>>. Citado na página 29.

ORACLE 2024 ORACLE. **O que é um chatbot?** 2024. Acessado em: 06 dez. 2024. Disponível em: <<https://www.oracle.com/br/chatbots/what-is-a-chatbot/>>. Citado na página 14.

ORACLE 2024 ORACLE. **What is Machine Learning?** 2024. Acesso em: 08 nov. 2024. Disponível em: <<https://www.oracle.com/br/artificial-intelligence/machine-learning/what-is-machine-learning/>>. Citado na página 12.

Pereira 2023 PEREIRA, L. C. **Inteligência artificial com Python: por que usar Python para trabalhar com IA?** 2023. Acesso em: 30 set. 2024. Disponível em: <<https://hub.asimov.academy/tutorial/inteligencia-artificial-com-python-por-que-usar-python-para-trabalhar-com-ia/#:~:text=Conclus%C3%A3o,para%20desenvolver%20aplica%C3%A7%C3%B5es%20de%20IA>>. Citado na página 12.

RED HAT 2024 RED HAT. **O que são Modelos de Linguagem de Grande Escala (Large Language Models)?** 2024. Acessado em: 03 out. 2024. Disponível em: <<https://www.redhat.com/pt-br/topics/ai/what-are-large-language-models>>. Citado na página 15.

RED HAT 2024 RED HAT. **O que é orquestração?** 2024. Acessado em: 20 jan. 2025. Disponível em: <<https://www.redhat.com/pt-br/topics/automation/what-is-orchestration>>. Citado na página 16.

SECURITY FIRST 2024 SECURITY FIRST. **Qual linguagem de programação foi usada para criar o ChatGPT?** 2024. Acesso em: 30 set. 2024. Disponível em: <<https://securityfirst.com.br/qual-linguagem-de-programacao-foi-usada-para-criar-o-chatgpt/>>. Citado na página 12.

TENSORFLOW 2025 TENSORFLOW. **TensorFlow - Uma Biblioteca de Aprendizado de Máquina de Código Aberto**. 2025. Acessado em: 20 jan. 2025. Disponível em: <<https://www.tensorflow.org/?hl=pt-br>>. Citado na página 16.