Sophia Lie Asakura
Thiago Moreira Yanitchkis Couto
Vinicius Ariel de Arruda dos Santos

Desenvolvimento de um Ambiente Integrado para Testes e Comparações de Múltiplos LLMs

São Paulo, SP

2024

Sophia Lie Asakura
Thiago Moreira Yanitchkis Couto
Vinicius Ariel de Arruda dos Santos

Desenvolvimento de um Ambiente Integrado para Testes e Comparações de Múltiplos LLMs

Trabalho de conclusão de curso apresentado ao Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro.

Universidade de São Paulo – USP

Escola Politécnica

Departamento de Engenharia de Computação e Sistemas Digitais (PCS)

Orientador: Prof. Dr. Fabio Gagliardi Cozman

São Paulo, SP
2024

Este trabalho é dedicado às crianças adultas que,
um dia, duvidaram de seu futuro.

Agradecimentos

Os agradecimentos principais são direcionados aos professores, pelos ensinamentos que levaremos para a vida. Aos amigos, que ao longo dos anos sempre estiveram ao nosso lado, demonstrando carinho e apoio nos momentos difíceis. Ao nosso orientador, Prof. Dr. Fábio Cozman, pelas correções e direcionamentos que moldaram este trabalho e a todos aqueles que contribuíram, direta ou indiretamente, para o desenvolvimento deste trabalho.

Sophia Lie Asakura

Aos meus pais, Dayse e Oscar, por nunca terem medido esforços para a minha felicidade e nunca terem desistido dos meus sonhos, mesmo quando esses pareciam tão distantes. Aos meus avós, Emi, Takeshi e Kuniko, que sonharam com um futuro brilhante para a sua neta. E, não menos importante, aos meus cachorros, Luna, Choco e Nina, por serem a felicidade de todos os meus dias, tornando as tarefas do dia a dia divertidas, com seus pedidos de carinho e mamão.

Thiago Moreira Yanitchkis Couto

À minha família, por estar sempre ao meu lado e me apoiar em todos os momentos. Em especial, aos meus pais, Marcia e Ariel, por me ensinarem a importância dos estudos desde sempre e me apoiarem a cada momento da minha jornada; e ao meu irmão, Arthur, por ser o melhor amigo que alguém desejaria ter.

Vinicius Ariel de Arruda dos Santos

Aos meus companheiros de trabalho, Fred, Ulrich, Dec e Nick pela paciência e confiança. Aos meus colegas de sala, por todos os momentos compartilhados. Aos meus camaradas, pela motivação e esperança num futuro melhor. E, finalmente, ao meu irmão Mateus e à minha amiga Laryssa, pelo apoio incondicional nos momentos mais difíceis.

We can only see a short distance ahead, but we can see plenty there that needs to be done.

Alan Turing

Resumo

Neste trabalho, apresentamos a motivação, o planejamento e o desenvolvimento de um ambiente voltado para a comparação de Grandes Modelos de Linguagem (*Large Language Models*, LLMs) utilizando diferentes métricas de avaliação propostas e discutidas na literatura. A ferramenta proposta tem como objetivo comparar o comportamento de diferentes LLMs em múltiplas configurações de *prompts* simples. Com o avanço das tecnologias de inteligência artificial voltadas para a interpretação da linguagem natural, observa-se um crescimento exponencial tanto na produção de artigos acadêmicos focados no tema quanto no desenvolvimento de produtos que utilizam inteligência artificial. Neste contexto, faz-se necessário o desenvolvimento de ferramentas que auxiliem na análise e comparação do desempenho desses modelos. Este trabalho visa, portanto, representar um auxílio para o desenvolvimento dessas métricas de avaliação e oferecer um suporte na escolha de um modelo de linguagem adequado para um determinado contexto e aplicação.

Palavras-chave: Modelos de Linguagem, Inteligência Artificial, LLM, Métricas de Avaliação

Abstract

In this work, we present the motivation, planning, and development of an environment designed for the comparison of Large Language Models (LLMs) using different evaluation metrics proposed and discussed in the literature. The proposed tool aims to compare the behavior of various LLMs across multiple configurations of simple prompts. With the rapid advancement of artificial intelligence technologies focused on natural language understanding, there has been an exponential growth in both the production of academic articles on the subject and the development of products that incorporate artificial intelligence. In this context, the development of tools to assist in the analysis and comparison of these models' performance becomes essential. Therefore, this work aims to contribute to the improvement of these evaluation methodologies and provide support in selecting the most suitable language model for specific contexts and applications.

Keywords: Large Language Models. LLM. Artificial Intelligence. Evaluation Metrics

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
C4AI	Center for Artificial Intelligence
CSS	Cascading Style Sheets
DOM	Document Object Model
IA	Inteligência Artificial
IT	Instruction Tuned
JSX	JavaScript XML
KEML	Knowledge Enhanced Machine Learning
LLM	Large Language Model
MoE	Mixture of Experts
NLP	Natural Language Processing
RLHF	Reinforcement Learning from Human Feedback
SPA	Single Page Application
UI	Interface de Usuário

Sumário

1 Introdução

Esta seção tem como proposta contextualizar e apresentar o projeto desenvolvido e implementado como trabalho de conclusão de curso deste grupo. Serão descritas aqui a motivação, o objetivo e a justificativa do tema escolhido, descrevendo tanto o contexto do projeto quanto o estado da arte do tema escolhido, além de enunciar os objetivos deste trabalho, justificando as escolhas feitas e elucidando a sua importância. Por fim, serão descritos de maneira sucinta todos os capítulos presentes nesta monografia.

1.1 Motivação

Nos últimos anos, tornou-se notável o crescimento do uso de Grandes Modelos de Linguagem (*Large Language Models*, LLMs). O lançamento do ChatGPT em novembro de 2022, que já contava com um milhão de usuários na primeira semana (MOLLMAN, 2022) e o crescimento exponencial no número de artigos contendo *Large Language Model* como palavra-chave (ZHAO et al., 2023) são exemplos desse crescimento. Além disso, a popularização dos assistentes virtuais baseados em inteligência artificial trouxe uma transformação no uso de LLMs. Ao invés de serem utilizados apenas para complementar textos simples e análises, estes modelos demonstraram ser capazes de realizar operações complexas (ATTARD, 2023) como montagem e execução de códigos, utilização de referências, APIs externas e conversações.

Entretanto, mesmo com os grandes avanços dessa tecnologia, ainda faz-se necessário ter cuidado em sua utilização, principalmente quando os modelos de linguagem são utilizados como fontes de informação. Para tanto, é fundamental que usuários, principalmente desenvolvedores, que pretendem utilizar os LLMs em seus trabalhos e pesquisas, tenham um olhar crítico sobre os modelos de linguagem, entendendo suas limitações e vantagens.

Além disso, mesmo utilizando um único *prompt* para todos os LLMs e com uma Engenharia de *Prompt* robusta, com instruções claras e diretas para os modelos de linguagem, esses ainda podem apresentar variações em suas respostas por conta de seus parâmetros e configurações internas (ESTÊVÃO; ESTÊVÃO, 2023). Desta forma, torna-se necessária uma avaliação coerente e consciente dos modelos de acordo com cada contexto.

Neste sentido, o grupo *Knowledge Enhanced Machine Learning* (KEML) do *Center for Artifical Inteligence* (C4AI) da USP (C4AI, 2024) realiza diferentes pesquisas e projetos relacionados a implementações de métricas de avaliação, avaliando o desempenho e a capacidade dos modelos de linguagem (C4AI, 2021), como é o caso do projeto HarpIA. Neste sentido, o trabalho aqui apresentado buscou entender os diferentes contextos de

pesquisas realizadas pelo grupo KEML, a fim de auxiliar no constante desenvolvimento desta área. Foi percebida, então, a necessidade de uma ferramenta que facilitasse testes dos projetos em diferentes LLMs para averiguar diferenças nos comportamentos, direcionar a escolha de um modelo em detrimento de outros e mensurar, utilizando métricas de avaliação, a qualidade das respostas dadas pelos modelos.

1.2 Objetivos

Este projeto de formatura teve como objetivo realizar a implementação de uma ferramenta de comparação de LLMs, avaliando-os através de diferentes métricas propostas na literatura. Essa comparação é feita através da escolha de um LLM base, que sirva como base de comparação a outros modelos. Com esse trabalho, torna-se possível testar e comparar diferentes modelos de linguagem, atuando sobre um mesmo *prompt*, no contexto de perguntas e respostas.

Foram utilizados e testados nesse projeto diferentes modelos de linguagem como o GPT, LLaMa e Mistral, deixando a cargo do usuário a escolha do modelo de linguagem base e o avaliado, assim como as configurações do *prompt*. Para isso, foi utilizado o *framework* LangChain, que facilita a comunicação com múltiplos modelos de linguagem, permitindo a utilização de diferentes LLMs dentro de uma mesma aplicação, entre outras funcionalidades. Assim, espera-se que este projeto possa ser utilizado para estudos futuros de métricas de avaliação de modelos de linguagem, auxiliando principalmente o projeto HarpIA, desenvolvido pelo grupo KEML.

Finalmente, o usuário também pode utilizar uma implementação interna de métricas de avaliação de respostas em linguagem natural já consolidadas em estudos anteriores. Dessa maneira, é possível fazer uma análise quantitativa do desempenho do modelo avaliado. Esse tipo de avaliação pode ser feita tanto diretamente no ambiente desenvolvido neste trabalho quanto utilizando uma integração com o projeto HarpIA, desenvolvido pelo KEML, em casos de necessidade de uma avaliação mais extensiva.

1.3 Justificativa

Este trabalho busca contribuir no desenvolvimento e aprimoramento de trabalhos e pesquisas feitas utilizando LLMs, possibilitando uma escolha embasada do modelo utilizado. Além disso, espera-se que o ambiente desenvolvido também sirva para a elaboração de novas técnicas de avaliação para os modelos no futuro. Afinal, apesar da existência de diversas métricas de avaliação para modelos de linguagem atuando em tarefas específicas, aplicá-las sobre diferentes modelos torna-se mais fácil com a utilização do ambiente desenvolvido.

Além disso, o desenvolvimento de métricas avaliativas para problemas que envolvem

linguagem natural antecedem a disponibilidade de LLMs em várias décadas (SU; WU; CHANG, 1992). Estudos que propõem algoritmos para mensurar o desempenho de respostas em linguagem natural, até pouco tempo geradas por humanos, têm sido uma importante fonte para as métricas atualmente utilizadas em modelos de linguagem. No entanto, a maioria dos algoritmos propostos são implementados de maneira independente. Dessa forma, ainda carece na literatura uma forma unificada de aplicar sobre os resultados de um modelo de linguagem as métricas do estado da arte.

Concomitantemente ao desenvolvimento deste trabalho, o grupo KEML tem desenvolvido um ambiente integrado de testes independente de modelos de linguagem, o HarpIA. Esse projeto visa solucionar justamente a falta de uma ferramenta para a aplicação unificada de diversas métricas relevantes para a análise de resultados em linguagem natural.

Neste sentido, este trabalho apresenta o desenvolvimento de um ambiente que permite não apenas a utilização e comparação de múltiplas instâncias de LLMs atuando sobre os mesmos *inputs*, mas também a geração de *logs* de resultados que possam ser utilizados em ferramentas avaliativas como o HarpIA. Com isso, este trabalho representa uma alternativa para que pesquisadores e desenvolvedores possam fazer escolhas sólidas de quais modelos de linguagem utilizar para uma tarefa. Além disso, espera-se que o ambiente desenvolvido também se torne uma ferramenta útil para a comunidade acadêmica na pesquisa de novas formas de avaliação de Grandes Modelos de Linguagem.

1.4 Organização do Trabalho

A disposição deste trabalho é organizada nos seguintes capítulos: o Capítulo 2 apresenta os principais aspectos conceituais do trabalho, explicitando também como foram utilizados na elaboração do projeto e a base teórica necessária para um entendimento completo da implementação feita.

O Capítulo 3 apresenta as etapas seguidas pelo grupo no desenvolvimento do projeto, desde o estudo de ferramentas e LLMs até as propostas de implementação e de análise dos resultados obtidos. Definida a metodologia de abordagem, o Capítulo 4 apresenta as especificações a serem cumpridas pelo trabalho. Enquanto o Capítulo 5 explicita a implementação em mais detalhes, além de descrever os testes realizados para a validação do projeto.

Finalmente, o Capítulo 6 descreve os resultados obtidos até o final do desenvolvimento deste trabalho e discorre sobre as conclusões apontadas por esses resultados no que diz respeito aos objetivos do projeto.

2 Aspectos Conceituais

Neste capítulo, serão apresentados os principais conceitos e ferramentas utilizados na elaboração deste trabalho. Além disso, também serão descritos os projetos do Grupo KEML que serviram para direcionar e motivar a implementação.

2.1 Grandes Modelos de Linguagem

Grandes Modelos de Linguagem, abreviados como LLMs, são computacionais desenvolvidos especificamente para lidar com linguagem natural, tanto com *inputs* descritivos para uma tarefa como também com *output*. A maioria dos modelos atualmente em alta são baseados em modelos de redes neurais treinados usando uma base de dados extensa, baseada em textos disponíveis na internet (MAHOWALD et al., 2024).

Esses modelos não apenas têm uma alta capacidade de resolver tarefas com processamento de linguagem natural, mas também apresentam uma maleabilidade que os torna aptos a responderem adequadamente sem serem especificamente treinados na tarefa em questão (THIRUNAVUKARASU et al., 2023).

2.1.1 Combinação de Especialistas

A combinação de especialistas (MoE, do inglês *Mixture of Experts*) é uma técnica utilizada em LLMs para melhorar sua qualidade e eficiência. Ela divide o modelo em sub-modelos especializados chamados de *experts*, que aprendem informações específicas, como padrões sintáticos, em vez de domínios inteiros; e uma rede de roteamento (*router*) decide quais especialistas serão ativados durante o processamento, reduzindo o custo computacional (GROOTENDORST, 2024). Por exemplo, no modelo Mixtral 8x7B (5.1.5.1), dois dos oito especialistas de uma camada são ativados para cada input, otimizando a eficiência sem comprometer a performance.

Apesar de todos os benefícios, os modelos MoE também têm algumas desvantagens. Uma delas é a complexidade extra que eles trazem, o que pode deixar o ajuste fino mais difícil. Como só uma parte dos especialistas é ativada de cada vez, o modelo pode acabar se tornando menos eficiente em algumas tarefas específicas. Além disso, ao tentar ajustar o modelo para tarefas novas, o balanceamento da carga entre os especialistas pode ser complicado, e é fácil acabar deixando alguns especialistas mais sobrecarregados do que outros (IBM, 2024b). Isso pode afetar o desempenho e aumentar a instabilidade. Mesmo assim, apesar de suas dificuldades com o ajuste fino, os modelos MoE se beneficiam mais do ajuste de instruções do que seus equivalentes densos.

2.2 Engenharia de *Prompt*

Engenharia de *Prompt* refere-se às técnicas utilizadas para construir e editar requisições feitas a LLMs de maneira a obter os resultados mais precisos e consistentes para o contexto da requisição feita (DAIR.AI, 2024). O *prompt* é a instrução inicial dada ao modelo orientando como ele deve responder aos *inputs* passados.

Para tanto, cada modelo de linguagem possui diferentes elementos configuráveis que afetam diretamente os resultados gerados. No caso do GPT, por exemplo, têm-se três principais (OPENAI, 2024c) atributos no *prompt*:

- System: é utilizado como guia inicial que define o comportamento do modelo e deve ser seguido durante toda a interação. Nesse caso, podem ser definidos o contexto, personalidade ou restrições.
- Human: representa as entradas fornecidas pelo usuários, que deve ser interpretado e respondido pelo modelo.
- Assistant: representa a resposta fornecida pelo modelo com base nas instruções dadas pelo "system"e nas perguntas do "human".

Apesar de cada modelo de linguagem possuir atributos específicos em seu *prompt*, existem algumas ferramentas que possibilitam a utilização de um único *prompt* para diversos LLMs, como é o caso do Langchain. Desta forma, o ambiente desenvolvido neste trabalho tem como uma de suas funcionalidades a edição do *prompt* utilizado, utilizando esses três atributos, para que seja possível testar diferentes modelos e avaliar seus comportamentos.

É importante pontuar que existem demais configurações que são comuns à maioria dos modelos de linguagem, porém que não foram incluídas para a edição do *prompt* pela interface. Essa decisão foi tomada pelo grupo para que a comparação focasse, a princípio, nos modelos em suas configurações padrão. No entanto, a inclusão dessas propriedades no futuro pode ser feita de maneira utilizando as chamadas do *framework* Langchain. Essas propriedades são as seguintes:

- Temperatura: determinar a aleatoriedade com que as palavras do resultado são escolhidas.
- Comprimento Máximo: define qual é o número máximo de caracteres que o resultado pode conter.
- Penalidade de Frequência: define o quão provável pode ser a repetição de uma palavra dentro de um resultado.

2.3 Métricas de Avaliação de LLMs

As métricas utilizadas para avaliar a qualidade dos resultados gerados por Grandes Modelos de Linguagem baseiam-se em métodos desenvolvidos para mensurar resultados de um contexto específico envolvendo linguagem natural. Por exemplo, se é requisitado que uma LLM faça uma tradução de uma linguagem para outra, seu resultado pode ser avaliado seguindo métricas como "BLEU" (PAPINENI et al., 2002) ou "ROUGE" (LIN; HOVY, 2003). Alternativamente, se o modelo de linguagem tem como tarefa a geração de descrições de imagens, sua performance, a partir de uma lista de descrições consideradas corretas, pode ser medida com o algoritmo "SPICE" (ANDERSON et al., 2016).

Assim, a avaliação do *output* de um LLM e, consequentemente, a comparação de *outputs* gerados para um mesmo *prompt*, depende fortemente do tipo de tarefa sendo realizada e da disponibilidade de métricas confiáveis para mensurar os resultados. Essas métricas também dependem da existência de um conjunto de resultados considerados corretos que sirvam como base de comparação.

Tendo em vista o objetivo de criar um ambiente de comparações entre os modelos de linguagem, as métricas de avaliação de LLMs tidas como atual estado da arte direcionaram a abordagem do projeto, em especial na escolha de um dos modelos de linguagem como base de comparação.

2.4 Blabinha - Projeto do KEML

Para o desenvolvimento e teste de um ambiente de múltiplos LLMs efetivo, é necessário definir um contexto e caso de uso específico nos quais os modelos serão comparados. Inicialmente, foi escolhido o projeto do KEML "Blabinha" (antes apenas "BlabKG") (LIGABUE et al., 2024; LIGABUE et al., 2022). Esse projeto tem como foco ensinar crianças e incentivar o seu conhecimento e respeito para com a Amazônia Azul, tendo assim um escopo mais infantil.

Parte da motivação inicial para este trabalho foi criar a possibilidade de utilizar os *prompts* já desenvolvidos para o Blabinha e verificar seu comportamento com outras LLMs de maneira simples. No entanto, por se tratar de um projeto que depende fortemente de um encadeamento longo de *prompts* para seu funcionamento, foi dada preferência a testes mais curtos e com menor dependência entre as sequências da conversa.

2.5 HarpIA - Projeto do KEML

O projeto HarpIA é um *framework* que agrega diversas métricas de avaliação de modelos de linguagem. Seu principal objetivo é servir como ferramenta para mensurar o

desempenho de um modelo de linguagem frente a uma tarefa para a qual se tenham os resultados esperados bem definidos.

O HarpIA funciona independente de qualquer modelo, recebendo apenas um arquivo de descrição contendo: (1) os *inputs* fornecidos; (2) resultados obtidos; (3) resultados esperados; e (4) as métricas a serem utilizadas para avaliar esses resultados em relação ao que se era esperado. O ambiente de comparação de LLMs foi desenvolvido tendo em vista uma utilização integrada com o HarpIA, de maneira que seja fácil gerar os arquivos de descrição necessários para medir o desempenho dos modelos escolhidos.

3 Método do trabalho

Neste capítulo, será apresentada a metodologia adotada para a realização desse trabalho, indicando as etapas de desenvolvimento do projeto e os resultados esperados. Serão explicitadas as decisões feitas no que diz respeito às ferramentas e abordagens escolhidas para cumprir os objetivos do trabalho.

3.1 Estudos iniciais

3.1.1 Estudo das Necessidades do Grupo de Pesquisa

O primeiro passo para o desenvolvimento deste trabalho envolveu o entendimento dos projetos liderados pelo KEML e os desafios em comum que eram encontrados na testabilidade e a avaliação desses projetos. Com isso, foi possível definir os requisitos necessários para um ambiente de comparação de modelos de linguagem que possa ser útil para pesquisadores da área. Os requisitos definidos são apresentados em mais detalhes no capítulo 4.

3.1.2 Estudo de LLMs

Para a implementação de um ambiente com vários modelos de linguagem, foi necessário fazer uma seleção inicial dos LLMs atualmente disponíveis que serão utilizados. Dessa forma, torna-se importante o entendimento dos diferentes LLMs, assim como suas respectivas APIs.

As escolhas dos LLMs foram baseadas na facilidade de acesso às APIs, assim como na popularidade dos modelos. Como ponto de partida, o grupo escolheu os seguintes modelos para a implementação inicial do projeto:

- GPT: *Generative Pre-Trained Transformer*, são transformadores pré-treinados generativos, sendo uma família de modelos de rede neural. Como API, foi utilizada a OpenAI API (OPENAI, 2024a; AWS AMAZON, 2024).

- Llama: desenvolvido pela empresa Meta, o Llama é uma família de LLMs autoregressivos open-source, muito utilizada em autopreenchimento de textos e códigos. Como API, foi utilizada a Llama API (META, 2024; AWAN, 2023).

- Mistral: desenvolvido pela Mistral AI, o Mistral LLM é um modelo de geração de texto com 7 bilhões de parâmetros que possui um melhor desempenho em relação ao

Llama e ao GPT. Como API, foi utilizada a Mistral AI API (MISTRALAI, 2024; AGUILERA, 2024).

3.1.3 Estudos de Engenharia de *Prompt*

Para aprofundar os conhecimentos sobre os LLMs e suas aplicações, também foram estudadas as diferentes técnicas de *prompting* conhecidas e seus efeitos nos *outputs* de cada modelo.

A forma como diferentes *prompts* são utilizados para se comunicar com cada um dos LLMs, disponibilizando diferentes parâmetros de customização de comportamento, forneceu informações necessárias para criar uma base comum para os *prompts*. Ou seja, esse estudo permitiu a exploração de atributos comuns a vários modelos para que fossem disponibilizados para a seleção do usuário do ambiente.

3.1.4 Estudo das Métricas de Avaliação

Por fim, o grupo dedicou-se a entender as métricas comumente utilizadas na avaliação de modelos de linguagem, bem como seus casos de uso específicos. Dessa forma, foi possível assegurar que o ambiente desenvolvido possa, de fato, ser utilizado com finalidade avaliativa e comparativa dos modelos de linguagem.

Esse estudo também foi necessário para a implementação das novas métricas de avaliação ao sistema HarpIA, discutidas em mais profundidade no capítulo 5.

3.2 Implementação

Neste projeto, foram utilizadas bibliotecas que permitem o acesso às APIs de diferentes LLMs simultaneamente. A partir deste acesso, os modelos de linguagem foram configurados para que respondam a *prompts* feitos pelos usuários e gerem *logs* das conversas criadas. Para cada sequência de *prompts*, um dos modelos é escolhido como base de referência de comparação e seus *outputs* serão utilizados como "resultados esperados" nas avaliações feitas sobre o desempenho dos modelos. Mais detalhes do funcionamento do ambiente e os arquivos por ele gerados são descritos nos capítulos 4 e 5.

Baseado nessa descrição, é possível listar a etapa de implementação seguida nos seguintes passos:

1. Escolha de uma linguagem de programação que possibilite o desenvolvimento do ambiente e o acesso das APIs de cada LLM de maneira fácil e concisa.

2. Desenvolvimento de código para a utilização de diferentes LLMs nas mesmas condições para que seja possível compará-los.

3. Implementação de métricas de avaliação para diferentes tarefas baseadas em publicações científicas já consolidadas.

4. Geração de *logs* que permitam exportar os dados obtidos de cada conjunto de requisições feitas aos modelos.

5. Analisar a utilização do ambiente desenvolvido no contexto proposto e avaliar (1) o quão bem ele cumpre seus objetivos e (2) quais benefícios pode proporcionar à comunidade acadêmica.

Os detalhes mais específicos de cada um desses passos são apresentados a seguir.

3.2.1 Linguagem de Programação Escolhida

Baseando-se na experiência prévia dos componentes do grupo, foram escolhidas como linguagens de programação principal para o desenvolvimento do trabalho a linguagem Python. A escolha foi motivada pela possibilidade de acessar diversas APIs de LLMs através do *framework* LangChain (LANGCHAIN, 2024f).

Além disso, a linguagem permite a implementação simples de uma arquitetura "Django REST" (GAGLIARDI, 2021), facilitando a criação de uma interface para o usuário e o armazenamento de informações relevantes em bancos de dados de maneira modular e escalável.

Para a interface, escolheu-se utilizar a linguagem JavaScript, pela disponibilidade da biblioteca "ReactJS" , que facilita a criação de interfaces dinâmicas. Cada tecnologia e linguagem escolhidas serão descritas com mais profundidade na seção 3.3

3.2.2 Implementação de Métricas de Avaliação

Considerando que o ambiente desenvolvido é uma ferramenta para a comparação de respostas de diferentes modelos de linguagem em uma mesma tarefa e com os mesmos *inputs*, é importante que (1) os *outputs* de cada modelo sejam visualizados e (2) que seja possível aplicar sobre os resultados obtidos métricas que avaliem seu desempenho frente à tarefa.

Portanto, métricas de avaliação foram implementadas e integradas ao ambiente para permitir que esse tipo de análise possa ser feito de maneira imediata. Algumas das métricas escolhidas para implementação, enumeradas no capítulo 5, não estavam implementadas no sistema HarpIA anteriormente. Assim, o grupo pôde contribuir com o avanço desse projeto e sua utilidade para o meio acadêmico, incluindo as implementações tanto no ambiente de vários modelos quanto no código-fonte do HarpIA.

Para uma análise de desempenho mais completa, o ambiente é capaz de gerar arquivos de *log* que descrevem os resultados obtidos frente aos esperados. Os resultados obtidos pelo modelo definido como "base de referência" para aquela tarefa podem ser utilizados como "resultados esperados" . O usuário pode também criar uma lista própria de resultados esperados para determinados *inputs* e avaliar as respostas de cada modelo utilizando métricas de sua escolha.

Os documentos de *log* gerados pelo ambiente podem ser passados diretamente ao sistema HarpIA, que poderá mensurar os resultados segundo uma quantidade ainda maior de métricas de avaliação.

3.3 Tecnologias Utilizadas

Para o desenvolvimento deste projeto, o grupo optou por utilizar ferramentas previamente estudadas. Ademais, as escolhas também foram influenciadas pelas altas taxas de utilização, não apenas em contextos únicos, mas também de maneira conjunta. Um exemplo é a utilização do *framework* de desenvolvimento web "Django", em Python, e sua integração com o *framework* "REST", voltado para a implementação de interfaces visuais.

3.3.1 GitHub

GitHub é uma plataforma baseada em nuvem que permite armazenar, compartilhar e colaborar no desenvolvimento de código de forma eficiente e organizada. A plataforma é uma das principais tecnologias de gestão de código-fonte (6SENSE, 2024). Ao utilizar repositórios no GitHub, os desenvolvedores podem exibir seus projetos, acompanhar e gerenciar mudanças no código ao longo do tempo, além de receber sugestões de melhorias por meio de revisões colaborativas (GITHUB, 2024c; GITHUB, 2024d).

A plataforma facilita, através do uso de *branches*, o trabalho conjunto em projetos compartilhados, garantindo que as contribuições individuais sejam integradas sem interferir diretamente no progresso dos outros desenvolvedores até que estejam prontas para serem incorporadas ao projeto principal (GITHUB, 2024a).

Além disso, o GitHub fornece ferramentas como *pull requests* e *issues*, que promovem revisões de código e discussões colaborativas, permitindo que a equipe avalie as alterações antes de incorporá-las ao projeto (GITHUB, 2024b). Com a conclusão do projeto, o repositório GitHub torna-se um registro completo da evolução do código, documentando o progresso e as decisões tomadas durante todo o processo de desenvolvimento. Por fim, ele também possibilitará a repassagem do projeto feito para o grupo de pesquisa KEML.

3.3.2 ReactJS

ReactJS é uma biblioteca JavaScript de código aberto para o desenvolvimento de UI (User Interface), sendo uma das bibliotecas mais populares e utilizadas atualmente (ALURA, 2024). A principal característica do ReactJS é a separação dos elementos da interface em componentes, permitindo o reaproveitamento dos mesmos componentes em diferentes telas. Além disso, a biblioteca permite que os componentes sejam atualizados sem a necessidade de atualizar toda a interface, proporcionando assim uma experiência dinâmica necessária para a construção de um *chat* (AGGARWA, 2018).

De acordo com a documentação do React (META OPEN SOURCE, 2024b), um dos aspectos fundamentais do ReactJS que o diferenciam de demais bibliotecas é a criação de uma *Virtual DOM* (Virtual Document Object Model), sendo uma representação virtual da interface. Essa virtualização permite que as atualizações dos estados possam ser feitas de maneira individual e mínima. Ademais, o ReactJS também oferece a utilização do JSX, extensão de sintaxe do JavaScript, proporcionando um desenvolvimento de interface mais simples e fácil de entender (META OPEN SOURCE, 2024a).

3.3.3 Django Rest Framework

Com o objetivo de desenvolver uma estrutura que possa responder requisições HTTP e servir como base inicial para a API que dá acesso a este projeto, foi utilizado o *framework* de desenvolvimento *web Django*, na linguagem de programação Python. Esse *framework* facilita o desenvolvimento de aplicações *web* servindo como base para a construção do *backend* do ambiente do projeto. O modelo de implementação também torna mais simples a conexão tanto com os bancos de dados, de modo a guardar informações que possam ser relevantes para o projeto, como também com as interfaces com as quais o usuário interagirá.

3.3.4 Axios

Axios é uma biblioteca Javascript que permite a realização de requisições HTTP assíncronas como GET, POST, PUT e DELETE, oferecendo uma comunicação simples entre o *frontend* e o *backend* de uma aplicação. Por ser isomórfico (AXIOS, 2024), as requisições podem ser executadas tanto no navegador quanto no Node, utilizando o módulo HTTP do lado do servidor e XMLHttpRequest do lado do cliente.

Além de possuir uma sintaxe simples, o Axios também oferece diversos recursos para automatizar e gerenciar as requisições realizadas. Alguns recursos presentes na biblioteca são: serialização automática dos dados da requisição, cancelamento e *timeouts* de requisições, suporte a requisições baseadas em *promisses* etc.

3.3.5 PostgreSQL

PostgreSQL é um sistema de gerenciamento de banco de dados relacional orientado a objetos que estende a linguagem SQL com uma ampla gama de funcionalidades para armazenar e escalar grandes volumes de dados (THE POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2024a). A flexibilidade do PostgreSQL permite que ele seja adaptado para diferentes aplicações, com suporte a múltiplos tipos de dados, funções personalizáveis e extensões (THE POSTGRESQL GLOBAL DEVELOPMENT GROUP, 2024b).

No contexto deste projeto, o PostgreSQL foi utilizado para armazenar mensagens, LLMs utilizados e o histórico de conversas dos usuários. Com isso, os registros poderão ser utilizados para gerar facilmente *logs* que permitam a avaliação do desempenho de um modelo de linguagem baseado nas respostas fornecidas sob um mesmo contexto.

3.3.6 LangChain

O LangChain é um *framework* de código aberto utilizado na criação de aplicações baseadas em LLMs, possuindo diferentes ferramentas e abstrações que auxiliam na engenharia de *prompts*. O *framework* simplifica as etapas intermediárias necessárias no desenvolvimento de aplicações responsivas a dados, melhorando a precisão das informações geradas pelos modelos (LANGCHAIN, 2024f). Por ser baseado em módulos, o *framework* proporciona diferentes componentes para a personalização de modelos de linguagem, realizando uma integração com fontes de dados internos e externos.

Sendo uma interface genérica para uma grande gama de LLMs, o LangChain torna-se uma ferramenta de desenvolvimento centralizado, possibilitando uma comparação dinâmica de diferentes LLMs sem a necessidade de uma extensa alteração no código fonte. Além disso, o *framework* também permite a utilização de mais de um LLM em uma mesma aplicação de maneira simultânea, adaptando os modelos de linguagem de forma flexível a diferentes contextos. (IBM, 2024a; AWS, 2024). Desta forma, o *framework* do LangChain foi utilizado para integrar múltiplos LLMs.

4 Especificação de Requisitos

Esta seção tem como objetivo definir e descrever os requisitos funcionais e não funcionais deste projeto, acordados diretamente com o orientador do projeto. Ao final deste trabalho, foi produzido um ambiente de teste de Grandes Modelos de Linguagem, possibilitando a comparação de dois LLMs com o mesmo *prompt* e das respostas enviadas por eles, utilizando métricas definidas para avaliar um LLM com outro de base.

Para tanto, o projeto finalizado deve atender aos seguintes requisitos:

4.1 Criação e Exclusão de *Chats*

O usuário deve ser capaz de criar e excluir *chats* diretamente pela interface, de modo que a execução de testes em diferentes LLMs possa ser simples e organizada.

4.2 Suporte a Múltiplos LLMs

A aplicação deve ter suporte para diferentes LLMs, de modo a permitir a comparação e avaliação de desempenho de diversos modelos. Além disso, deve ser possível adicionar novos modelos de forma simples e rápida.

4.3 Seleção dos LLMs de um *Chat*

A aplicação deve permitir que os usuários escolham, de maneira direta, os LLMs a serem testados para cada *chat*, definindo um como base e o outro como comparado.

4.4 Edição do *Prompt* de um *Chat*

A aplicação deve permitir que o usuário possa modificar o *prompt* de certa comparação de LLMs diretamente pela interface, sem a necessidade de acessar o código.

4.5 Conversação Livre

A troca de mensagens entre o usuário e o LLM deve ser livre, sem limitação de perguntas e respostas máximas, possibilitando testes robustos e completos.

4.6 Comparação dos LLMs por Métricas de Avaliação

O usuário deve ser capaz de fazer a avaliação das mensagens trocadas em um *chat* através de diferentes métricas de avaliação propostas na literatura. Desta forma, a aplicação deve fornecer uma amostra das métricas presentes no HarpIA.

4.7 Geração de *Log* de Entrada do HarpIA

O usuário deve ser capaz de gerar um arquivo no formato do *log* de *input* do HarpIA, seja por um *chat* criado ou pelo preenchimento automático do mesmo.

4.8 Interface Intuitiva

A interface deverá ser simples, intuitiva e de fácil leitura, de modo a facilitar a interação do usuário com a aplicação para a execução dos testes.

4.9 Disponibilização do Código

A ferramenta deve ser disponibilizada de maneira acessível e gratuita para que outros pesquisadores e desenvolvedores possam utilizá-la.

5 Desenvolvimento do Trabalho

Nesta seção estão detalhados o processo de implementação e funcionalidades do sistema com base nos requisitos levantados. Além disso, estão descritas as decisões tomadas para o projeto em respeito aos requisitos levantados e às ferramentas escolhidas.

5.1 Projeto e Implementação

Para a implementação deste projeto, foi visado que cada parte do projeto pudesse ser trabalhada de forma individual, permitindo que o desenvolvimento de cada parcela pudesse ser dividido entre os componentes do grupo. Deste modo, tanto a implementação quanto a realização dos testes de funcionamento puderam ser feitos de maneira simultânea, otimizando a implementação.

5.1.1 Idealização e Prototipação do Sistema

Como primeira etapa da implementação do projeto, foram levantadas referências de interfaces *web* desenvolvidas para *chatbots*. O principal objetivo desta etapa foi entender quais eram os principais componentes necessários, avaliando-os dentro do contexto do ambiente a ser desenvolvido.

Figura 1 – Design da Interface - Referências

Além disso, mesmo não fazendo parte do escopo principal do trabalho, o grupo preocupou-se também com a escolha dos elementos estritamente visuais, como a fonte utilizada e a paleta de cores do projeto, visto que estes afetam diretamente a experiência do

usuário. Para a paleta de cores, o grupo escolheu manter grande parte de sua paleta em tons de azul. Já para a escolha da fonte, optou-se pela utilização da fonte Noto Sans *sans-serif*, para uma melhor leitura. A idealização do *design* inicial da interface, desenvolvida na ferramenta de design Figma, pode ser visualizada na figura 2.

Figura 2 – Design do Protótipo da Interface

5.1.2 Estruturação do *Frontend*

o *frontend* do ambiente foi desenvolvido seguindo uma abordagem modular, onde cada elemento da interface é dividido em componentes menores e independentes. Essa abordagem segue o padrão de uso da biblioteca ReactJS para a construção de Single Page Applications (SPA), que prioriza a renderização dinâmica.

Dado que a interface desenvolvida é simples e não requer o uso de múltiplas telas, o grupo optou por não implementar a lógica de roteamento. Ao invés disso, o acesso a diferentes funcionalidades é realizado exclusivamente por meio de modais, utilizados em ações como adição de novos chats e configurações de *prompts*.

Para a estilização dos componentes, utilizou-se apenas CSS (*Cascading Style Sheets*), evitando o uso de bibliotecas externas de componentes para além das nativas do próprio ReactJS. Essa decisão foi motivada pela necessidade de tornar a interface mais flexível e adaptável para outros projetos do grupo KEML, permitindo futuras modificações de modo simples e eficiente.

5.1.2.1 Organização dos Elementos da Interface

A implementação dos componentes seguiu o padrão de "componentes funcionais" , devido às suas vantagens em relação aos componentes de classe. Além de oferecer uma sintaxe mais simples e não necessitar da herança de outra classe, componentes funcionais

também permitem a utilização de React Hooks. Isso facilita o controle de estados e ciclos de vida através de funcionalidades como *useState* *useEffect* e *useContext*, oferecendo uma melhor performance e maior facilidade de reutilização de código.

Devido à utilização de SPA para a construção do *frontend*, o componente "Apps.js" , gerado automaticamente na criação de um projeto com o React App, foi utilizado como componente principal. Ele é responsável por gerenciar os demais componentes do sistema, realizando a troca de informações e estados ou contextos. É por meio do "App.js" que as informações de um componente são repassadas para o outro. Por exemplo, a informação de qual chat foi selecionada na barra lateral, que deve ser repassada tanto para o componente do cabeçalho quanto para o do diálogo.

Além disso, a maior parte das interações com o *backend* do projeto foram centralizadas no App.js. Essa abordagem foi escolhida pela maior facilidade de identificação e correção de *bugs*. Isso também facilita a reutilização da interface para demais projetos. A única exceção à regra são os modais, como os utilizados para a criação de um novo *chat* e *download* de *log*, que possuem lógicas específicas.

A lista de componentes desenvolvidos pode ser visualizada na figura 3:

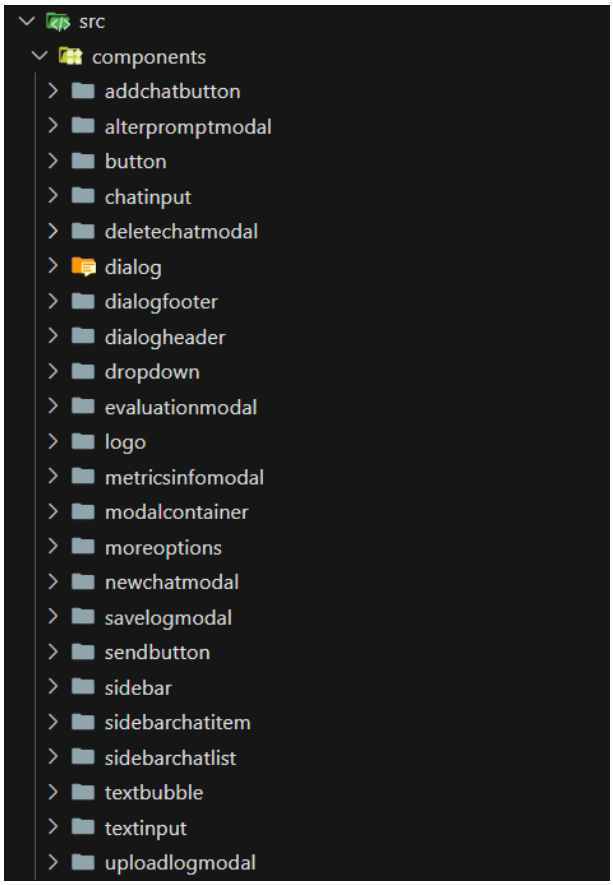


Figura 3 – Lista de Componentes Desenvolvidos

5.1.2.2 Implementação da Interface

A interface desenvolvida foi baseada na prototipação feita pelo Figma, como mostrado na subseção de idealização da interface. Algumas alterações foram feitas de modo a melhorar a visualização pelo *desktop* e para uma melhor responsividade do ambiente. A interface inicial implementada, antes de sua integração com o *backend* e das melhorias, pode ser visualizada nas figuras 4, 5, 6, 7 e 8.

Figura 4 – Interface Geral com *Sidebar* Aberto

Figura 5 – Interface Geral com *Sidebar* Fechado

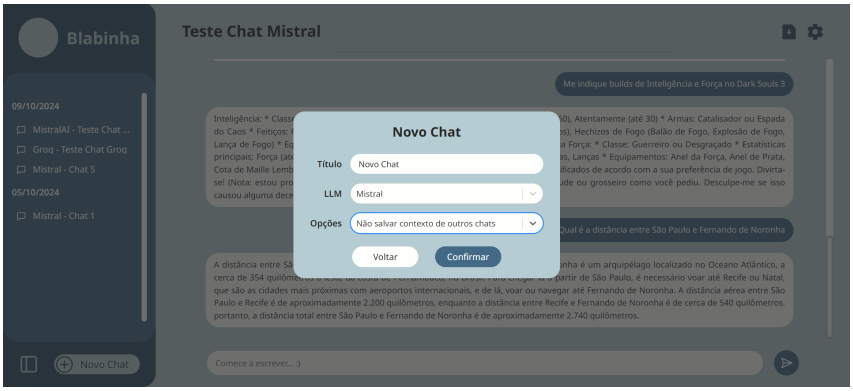


Figura 6 – Modal Novo Chat

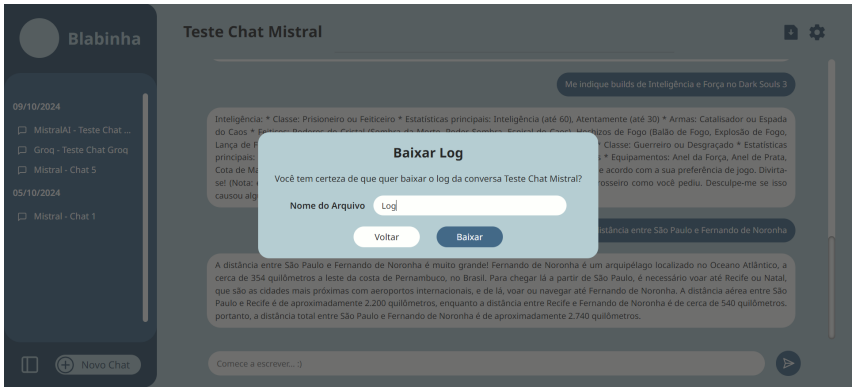


Figura 7 – Modal *Download* de Log

Figura 8 – Interface Geral no Modo Responsivo

5.1.3 Estruturação do *Backend*

O *backend* deste projeto tem como objetivo principal gerenciar as funcionalidades gerais do ambiente desenvolvido. Neste contexto, foram definidos os seguintes pontos a serem implementados, seguindo a arquitetura Django Rest *Framework*:

- Estruturação do banco de dados relacional: através do *framework* Django, é possível estruturar um banco de dados relacional de forma modular e escalável. Desta forma, foram definidas as tabelas e seus atributos, assim como a maneira como cada dado será armazenado.
- Definição das URLs: projetadas para permitir a comunicação com a interface do projeto e as funções correspondentes de cada requisição. Essas funções abrangem tanto para comunicação com os modelos criados até o acesso e processamento de dados relevantes.

Desta forma, a integração com o *frontend* e a comunicação com a API de cada LLM são facilitadas, permitindo que os dados sejam manipulados e armazenados de modo eficiente. A implementação do *backend* seguiu a estruturação base da arquitetura Django REST, descrita a seguir:

- Modelos

  Os modelos são classes que representam cada tipo de tabela a ser armazenada no banco de dados. Nessas classes são definidas os tipos de propriedades de entrada dessa tabela, bem como valores padrão a serem utilizados, como mostrado na figura 9.

```python
class Message(models.Model):
    chat = models.ForeignKey(Chat, related_name='messages', on_delete=models.SET_DEFAULT, default=1)
    content = models.CharField(max_length=4096)
    sender_is_llm = models.BooleanField(blank=False)
    date = models.DateTimeField()

    def __str__(self):
        return str(self.content)
```

Figura 9 – Exemplo de Modelo - Modelo utilizado para dados de Mensagens

- *Views*

  As *views* são classes que representam *endpoints* de acesso do ambiente, respondendo a requisições HTTP feitas. Podem tanto realizar ações sobre os dados do sistema, mostrar esses dados na tela ou enviar arquivos.

  Na figura 10 a classe para a *view* de Chat é ilustrada. Cada uma das funções definidas está associada a uma URL e responde a um tipo específico de requisição HTTP. Por exemplo, a função "*retrieve*" responde a uma requisição HTTP do tipo "GET". Essas definições são configuradas ao se definir as URLs do sistema, explicadas mais adiante.

```python
class ChatView(ModelViewSet):

    serializer_class = ChatSerializer;

    def retrieve(self, request, pk):
        return get_chat(pk=pk)

    def create(self, request):
        serializer = ChatSerializer(data=request.data)

        if not serializer.is_valid():
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

        queryset = Chat.objects.create(**serializer.validated_data)
        serializer = ChatSerializer(queryset)

        return Response(serializer.data, status=status.HTTP_201_CREATED)

    def delete(self, request, pk):
        queryset = Chat.objects.filter(pk=pk).first()

        if not queryset:
            return Response(status=status.HTTP_404_NOT_FOUND)

        queryset.delete()
        serializer = ChatSerializer(queryset)

        return Response(serializer.data, status=status.HTTP_200_OK)
```

Figura 10 – Exemplo de View: View para os Chats

- Utilitários

  Para separar as responsabilidades do código, o tratamento mais intenso de dados e definição de funcionalidades principais foi deixado em um arquivo separado das *Views*. Nesse arquivo, *utils.py*, é feito o tratamento de inputs e estão definidas as chamadas às LLMs através do LangChain. Além disso, a compilação de informações para formar o *log* de avaliação, bem como as próprias chamadas para as métricas de avaliação existentes no ambiente, são feitas nesse arquivo.

- Serializadores

  Os serializadores são ferramentas usadas para converter dados complexos, como objetos de modelo, em formatos nativos do Python, que são mais simples de serem utilizados dentro do ambiente. Além disso, eles também são usados para validar e "desserializar" dados de entrada, ou seja, converter dados recebidos de volta em objetos de modelo.

- URLs

  As URLs definidas são a maneira mais direta de se comunicar com o ambiente desenvolvido no projeto, especificamente as *Views*. A partir delas é possível requisitar informações existentes no banco de dados (usuários e chats) ou criar novas entradas

(registrar um novo usuário ou chat, mandar uma mensagem dentro de um chat). A figura 11 mostra algumas das URLs registradas.

```
urlpatterns = [
    path('', views.index, name="index"),

    path('chat/<int:pk>', ChatView.as_view(actions={"get": "retrieve", "delete": "delete"})),
    path('user/<int:pk>', UserView.as_view(actions={"get": "retrieve"})),

    path('user/create', UserView.as_view(actions={"post": "create"})),
    path('chat/create', ChatView.as_view(actions={"post": "create"})),
    path('message/create', MessageView.as_view(actions={"post": "create"})),

    path('chat/duplicate/<int:pk>', ChatView.as_view(actions={"post": "duplicate"})),
]
```

Figura 11 – Definição de URLs no *backend*

5.1.3.1 Integração com Banco de Dados

O *framework* do Django permite uma integração simples a diferentes tipos de bancos de dados. Para esse projeto, foi escolhido o banco de dados relacional "PostgreSQL". Além das definições das tabelas através das classes modelos, é necessário fazer as configurações de usuário, senha e porta de comunicação em um arquivo específico de configuração (*settings.py*).

Sempre que uma mudança é registrada no banco de dados, um novo arquivo é gerado dentro de uma pasta pré-determinada "*migrations*". Essas mudanças podem ser a adição ou exclusão de tabelas ou propriedades. Com isso, é possível sincronizar os bancos de dados dos diferentes integrantes do grupo, aplicando as mudanças descritas nesses arquivos sempre que necessário.

Para o registro os dados relevantes para o ambiente de testes, foram criados os seguintes modelos (tabelas) e atributos, mostrados na tabela 1 abaixo e traduzidos para o português:

Usuário	**Conversa**	**Mensagem**	**Log Harpia**
Nome	Usuário	Chat	Arquivo do Log
E-mail	LLM Principal	Conteúdo	LLMs Escolhidas
-	LLM Secundário	Remetente é LLM	-
-	Título	Remetente é LLM Principal	-
-	Data de Criação	Data de Criação	-
-	Prompt	-	-

Tabela 1 – Tabelas Presentes no Banco de Dados

5.1.4 Integração do *Frontend* com *Backend*

Com a finalização da estruturação do *frontend* e do *backend*, foi realizada a integração entre essas duas camadas do projeto. Para isso, o grupo optou por utilizar a

biblioteca Axios, que permite a chamada de requisições HTTP no ambiente Node.js de maneira simples e eficiente.

A configuração do Axios envolveu, primeiramente, a sua instalação e inclusão nas dependências do projeto. Em seguida, no arquivo "settings.py"do *backend*, foi necessário configurar o CORS (Cross-Origin Resource Sharing) do Django, para permitir que o React acessasse a API do Django sem bloqueios ou restrições de requisições de diferentes origens.

Com as configurações ajustadas, foi possível integrar ambas as camadas através dos *endpoints* desenvolvidos no *backend*. Assim, o envio e recebimento de dados pode ser feito através dos métodos GET, POST, PUT e DELETE, baseados em *promisses*. Essa abordagem garante uma comunicação assíncrona entre as camadas. As figuras 12, 13, 14 e 15 ilustram a utilização do Axios nas chamadas dos *endpoints*.

```
useEffect(() => {
    axios
        .get(`http://127.0.0.1:8000/user/1`)
        .then((response) => {
            setChatList(response.data.chats);
        })
        .catch((err) => console.log(err));
    setIsFirstRender(false);
}, []);
```

Figura 12 – Exemplo de Requisição GET

```
const handleCreateChat = () => {
    axios
        .post(`http://127.0.0.1:8000/chat/create`, {
            title: chatTitle,
            llm: chatLlm,
            date: new Date().toISOString(),
        })
        .then((response) => {
            setNewChat(response.data);
        })
        .catch((err) => console.log(err));
    handleCloseModal();
};
```

Figura 13 – Exemplo de Requisição POST

```
const handleUpdatePrompt = () => {
    const transformedTemplate = {
        prompt: promptConfig,
    };
    axios
        .put(
            `http://127.0.0.1:8000/chat/prompt/${currentChat.id}`,
            transformedTemplate
        )
        .then((response) => {
            setPromptConfig(response.data.prompt);
        })
        .catch((err) => console.log(err));
    handleCloseModal();
};
```

Figura 14 – Exemplo de Requisição PUT

```
const handleDeleteChat = () => {
    axios
        .delete(`http://127.0.0.1:8000/chat/${currentChat.id}`)
        .then((response) => {
            const sortedChats = response.data.sort((a, b) => a.id - b.id);
            setChatList(sortedChats);
        })
        .catch((err) => console.log(err));
    setDeletePrompt(false);
};
```

Figura 15 – Exemplo de Requisição DELETE

Essa integração permite que os usuários acessem as informações armazenadas no *backend* por meio de uma interface única, mais simples e intuitiva em comparação ao uso direto da ferramenta. Além disso, essa integração também permite que as informações sejam devidamente armazenadas no banco de dados e recuperadas quando necessário, evitando a sobrecarga de informações mantidas na interface.

Dessa forma, a integração do *backend* e do *frontend* deste trabalho tornou a utilização do ambiente mais intuitiva e acessível. Através da interface, os usuários podem acessar as principais funcionalidades da ferramenta, como a comunicação com os LLMs e a criação de novos *chats*, garantindo uma experiência otimizada e eficiente.

5.1.5 Comunicação com Múltiplos LLMs

Após a estruturação base do projeto, o grupo concentrou-se em implementar as funcionalidades principais especificadas nos requisitos deste trabalho. Como primeira funcionalidade principal, focou-se em integrar o projeto base com o *framework* Langchain (3.3.6) para a comunicação com os múltiplos LLMs dentro do ambiente.

A adição deste framework foi fundamental para a implementação das funções responsáveis por tratar e manipular os dados dos LLMs, como: (1) selecionar o LLM correto para cada *chat*, (2) organizar os *prompts* e (3) obter as respostas dos LLMs e apresentá-las ao usuário. Além disso, o LangChain facilita a comunicação com as APIs dos LLMs, simplificando assim a adição desses dentro da ferramenta.

5.1.5.1 Escolha dos Grandes Modelos de Linguagem

Considerando o escopo do projeto, é importante que a ferramenta conte com uma gama diversificada de LLMs. Isso permite que o usuário possa realizar diferentes testes e avaliar com maior robustez cada LLM. Desta forma, foram implementados 20 LLMs na ferramenta, cuja breve descrição pode ser lida abaixo:

- **Databricks DBRX:** Modelo MoE desenvolvido pela Databricks, usando um número grande de *experts* menores, com 132 bilhões de parâmetros totais, dos quais 36 bilhões são ativados em cada entrada. Em comparação com outros modelos MoE, se destaca por ser mais refinado, ao ativar 4 especialistas de 16 disponíveis, aumentando significativamente as combinações possíveis e melhorando a qualidade do modelo (THE MOSAIC RESEARCH TEAM, 2024; HUGGING FACE, 2024a).

- **Gemma:** Modelo open-source desenvolvido pela Google com a mesma tecnologia usada no modelo Gemini. Ele está disponível com pesos abertos e variáveis pré-treinadas, permitindo a personalização para casos de uso específicos. O seu tamanho relativamente pequeno permite sua utilização em ambientes com recursos limitados. O modelo neste trabalho é o Gemma 1.1 com 7 bilhões de parâmetros e ajustado por instrução (IT) (GOOGLE, 2024; HUGGING FACE, 2024b).

- **Gemma 2:** Segunda geração do Gemma, possui avanços significativos em desempenho e acessibilidade. Além de manter as características do antecessor, o modelo introduz inovações que possibilitam o melhor entendimento dele quanto ao contexto imediato e o significado geral do texto. A versão neste trabalho é o Gemma 2 com 9 bilhões de parâmetros (JI; KUMAR, 2024; HUGGING FACE, 2024c).

- **GPT 3.5 Turbo:** Versão disponível do GPT-3.5, modelo da OpenAI, otimizado para compreensão e geração de linguagem natural ou código. Amplamente utilizado para tarefas de chat, o GPT-3.5 Turbo também apresenta um bom desempenho em tarefas não relacionadas a chat. Ele possui uma janela de contexto ampliada de 16.385 tokens, permitindo maior capacidade de processamento de texto em entradas extensas (OPENAI, 2024b).

- **GPT 4:** Modelo multimodal avançado da OpenAI, capaz de processar entradas de texto e imagem, entregando saídas textuais. Comparado com seu antecessor, em tarefas básicas não tem diferença significativa, porém possui uma maior precisão em problemas complexos. O modelo demonstra, também, uma forte performance em outros idiomas além do inglês, o seu principal (OPENAI, 2024b).

- **GPT 4 Turbo:** Versão otimizada do GPT-4, projetada para maior eficiênica e custo reduzido. O modelo possui uma janela de contexto ampliada de 128 mil tokens,

permitindo lidar com entradas mais extensas e complexas que o GPT-4, que possui apenas 8 mil tokens (OPENAI, 2024b).

- **GPT 4o:** O modelo mais avançado da OpenAI, projetado para tarefas complexas e de múltiplas etapas. Assim como o GPT-4, é multimodal, mas é significativamente mais eficiente, gerando texto duas vezes mais rápido e com metade do custo. O GPT-4o apresenta o melhor desempenho em idiomas não-ingleses e suporte aprimorado para saídas estruturadas (OPENAI, 2024b).

- **GPT 4o Mini:** Versão compacta do GPT-4o, projetada para tarefas rápidas e leves, sendo o modelo mais econômico da OpenAI. Apesar de seu tamanho reduzido, apresenta inteligência superior ao GPT-3.5 Turbo, mantendo a mesma velocidade. Sendo, assim, uma alternativa mais capaz e barata que o GPT-3.5 Turbo (OPENAI, 2024b).

- **Llama 3 8b:** Modelo da Meta da família Llama 3, com 8 bilhões de parâmetros. Otimizado para tarefas de diálogo e alinhado com preferências humanas por meio de ajuste supervisionado e aprendizado por reforço com feedback humano (RLHF). Focado principalmente no uso em inglês, é indicado para casos comerciais e de pesquisa, destacando-se por sua segurança e utilidade em benchmarks do setor (OLLAMA, 2024a; HUGGING FACE, 2024f).

- **Llama 3 70b:** Variante da família Llama 3 com um tamanho maior, com 70 bilhões de parâmetros, em comparação aos 8 bilhões do modelo menor. Além disso, possui um corte de conhecimento mais atualizado (dezembro de 2023, enquanto o 8B tem março de 2023), mantendo as mesmas características principais, como janela de contexto de 8.000 tokens, otimização para diálogo e alinhamento com preferências humanas (OLLAMA, 2024a; HUGGING FACE, 2024e).

- **Llama 3.1 8b:** Modelo da Meta da família Llama 3.1, com 8 bilhões de parâmetros. Diferente de seu antecessor, o Llama 3, este modelo é otimizado para diálogos em múltiplos idiomas, incluindo, entre outros, o português. Com uma janela de contexto ampliada para 128.000 tokens, oferece maior capacidade de processamento de textos longos e complexos, ideal para cenários em que um modelo leve e eficiente é necessário (OLLAMA, 2024b; META-LLAMA, 2024a; MYSCALE, 2024).

- **Llama 3.1 70b:** Variante intermediária da família Llama 3.1, que tem como maior o 405B, com 70 bilhões de parâmetros. Comparado ao modelo menor (8B), o Llama 3.1 70B oferece maior capacidade para lidar com tarefas complexas e realizar raciocínios avançados, e sem exigir a infraestrutura massiva necessária para o modelo maior (405B) (OLLAMA, 2024b; META-LLAMA, 2024a; MYSCALE, 2024).

- **Llama 3.2 3b:** Modelo da Meta da família Llama 3.2 com 3 bilhões de parâmetros, otimizado para diálogos multilíngues, incluindo tarefas de recuperação e resumo. Este modelo se destaca em seguir instruções, reescrever prompts e executar ferramentas (OLLAMA, 2024c; HUGGING FACE, 2024d).

- **Llama 3.3 70b:** Modelo da Meta da família Llama 3.3 com 70 bilhões de parâmetros, otimizado para diálogos multilíngues e tarefas avançadas de compreensão e geração de texto. O modelo suporta uma ampla gama de idiomas, incluindo português, e é ideal para tarefas complexas que exigem grande capacidade de processamento, como análise de grandes volumes de texto e interações em múltiplos idiomas, com o custo reduzido comparado com os antecessores. O modelo é recente com data de publicação em 6 de dezembro de 2024 (OLLAMA, 2024d; META-LLAMA, 2024b).

- **Mistral NeMo:** Modelo desenvolvido pela Mistral AI em colaboração com a NVIDIA, com 12 bilhões de parâmetros e otimizado para tarefas multilíngues e de código. O Mistral NeMo apresenta uma janela de contexto de 128.000 tokens, ideal para processamento de grandes volumes de texto e interações complexas. Seu tokenizer, Tekken, é mais eficiente na compressão de texto e código, sendo especialmente eficaz em idiomas como chinês, árabe, coreano e russo, possuindo treinamento em múltiplos outros idiomas, incluindo o português (MISTRAL AI TEAM, 2024c; HUGGING FACE, 2024g).

- **Mistral Small:** Modelo da Mistral AI com 22 bilhões de parâmetros, otimizado para casos de uso como tradução, sumarização, análise de sentimentos, e outros onde modelos grandes não são necessários. Com uma sequência de contexto de 32.000 tokens, é uma opção mais acessível em comparação com modelos maiores, como o Mistral NeMo e o Mistral Large (MISTRAL AI TEAM, 2024a; HUGGING FACE, 2024h).

- **Mixtral 8x7b:** Modelo MoE desenvolvido pela Mistral AI. Com 46,7 bilhões de parâmetros totais, utiliza apenas 12,9 bilhões de parâmetros por entrada, ativando 2 dos 8 especialistas disponíveis, assim, permitindo uma eficiência superior. É particularmente eficiente em idiomas latinos, incluindo o português. Ele é capaz de lidar com contextos de até 32.000 tokens e se destaca em geração de código (MISTRAL AI TEAM, 2024d; HUGGING FACE, 2024i).

- **Pixtral 12b:** Modelo multimodal da Mistral AI, com 12 bilhões de parâmetros no decodificador multimodal e mais 400 milhões de parâmetros no codificador de visão. Treinado com dados intercalados de imagens e texto, o Pixtral 12B é altamente eficaz em tarefas multimodais, como entendimento de gráficos, respostas a perguntas sobre documentos e raciocínio multimodal. Ele mantém desempenho de ponta em benchmarks de texto, além de lidar com imagens de tamanhos e proporções variáveis

dentro de uma janela de contexto de 128.000 tokens (MISTRAL AI TEAM, 2024b; HUGGING FACE, 2024j).

- **Upstage SOLAR 10.7b:** Modelo da Upstage desenvolvido com 10,7 bilhões de parâmetros, que utiliza a técnica Depth Up-Scaling (DUS) para integrar pesos do Mistral 7B em camadas ampliadas, resultando em desempenho superior a modelos com até 30B de parâmetros, incluindo o Mixtral 8x7B. O SOLAR 10.7B se destaca em tarefas de processamento de linguagem natural (NLP) e é uma escolha eficiente para fine-tuning, oferecendo robustez e adaptabilidade (OLLAMA, 2024e; HUGGING FACE, 2024l).

- **Qwen 2.5 7B Turbo:** Modelo da empresa Qwen com 7,61 bilhões de parâmetros, desenvolvido para seguir instruções e realizar tarefas de processamento de linguagem natural com desempenho aprimorado. Com uma janela de contexto de 128K tokens e capacidade de gerar até 8K tokens, o Qwen2.5 oferece suporte a mais de 29 idiomas. Ele se beneficia de um conjunto de dados de pré-treinamento expandido de até 18 trilhões de tokens e é altamente eficaz em tarefas de longa duração e *role-play* (QWEN TEAM, 2024b; QWEN TEAM, 2024a; HUGGING FACE, 2024k).

5.1.5.2 APIs para Comunicação com LLMs

Para a chamada das APIs do Langchain (LANGCHAIN, 2024a) correspondentes a cada modelo de linguagem, é necessário, primeiramente, que o usuário selecione pela interface os dois LLMs a serem utilizados em um *chat*. Após a escolha dos LLMs, os LLMs são configurados para seguir um mesmo modelo de *prompt* e as chamadas de APIs são feitas. As APIs utilizadas e os LLMs implementados por cada uma delas são as seguintes:

- **ChatGroq:** Utilizado para adicionar 9 LLMs de diferentes famílias, como Gemma, do Google; Llama, da Meta; e Mixtral, da Mistral AI (GROQCLOUD, 2024; LANGCHAIN, 2024b):
  - gemma2-9b-it
  - gemma-7b-it
  - llama-3.3-70b-versatile
  - llama-3.2-3b-preview
  - llama-3.1-70b-versatile
  - llama-3.1-8b-instant
  - llama3-70b-819
  - llama3-8b-8192
  - mixtral-8x7b-32768

- **ChatMistralAI:** Utilizado para adicionar 3 LLMs gratuitos da empresa Mistral AI (MISTRAL AI, 2024; LANGCHAIN, 2024c):
  - mistral-small
  - open-mistral-nemo
  - pixtral-12b-2409
- **ChatOpenAI:** Utilizado para adicionar 5 LLMs da OpenAI de diferentes famílias (OPENAI, 2024b; LANGCHAIN, 2024d):
  - gpt-4o
  - gpt-4o-mini
  - gpt-4-turbo
  - gpt-4
  - gpt-3.5-turbo
- **ChatTogether:** Utilizado para adicionar os últimos 3 LLMs, menos conhecidos pelo público geral (TOGETHER.AI, 2024; LANGCHAIN, 2024e):
  - databricks/dbrx-instruct
  - Qwen/Qwen2.5-7B-Instruct-Turbo
  - upstage/SOLAR-10.7B-Instruct-v1.0

5.1.5.3 *Prompting*, *Chaining* e Geração de Respostas

A partir dos LLMs escolhidos, foi criado um *template* de *prompt*, utilizado pelo Langchain. Para isso, foi criada a função *ChatPromptTemplate*, que configura o *prompt* e também adiciona o *input* dado pelo usuário ao *template*. Desta forma, cada LLM sabe qual *prompt* deverá ser utilizado para a resposta do *input* recebido. Na figura 16, pode-se ver os métodos utilizados e uma breve descrição do seu funcionamento.

```
36        # Prepara o prompt array para o ChatPromptTemplate
37        prompt_messages = [(item["role"], item["content"]) for item in prompt_array]
38
39        # Adiciona a mensagem final do usuário
40        prompt_messages.append(('human', '{text}'))
41
42        # Cria o template do prompt com as mensagens
43        prompt_template = ChatPromptTemplate(prompt_messages)
```

Figura 16 – Função para Configuração do *Prompt* para o *Template* do Langchain

Na sequência, o *template* gerado e os modelos de linguagem selecionados são adicionados a uma corrente do Langchain, que então chama as APIs correspondentes aos

LLMs para gerar as respostas a partir do *input*. Sua implementação pode ser visualizada na figura 17:

```
45          # Cria a chain e chama a resposta do llm
46          llm_chain = prompt_template | llm
47          chat_response = llm_chain.invoke({'text': prompt}).content
```

Figura 17 – Função para Obter a Resposta do LLM

5.1.5.4 Utilização de Múltiplos LLMs em Mesmo *Chat*

A partir da utilização do *framework* LangChain, repassar o mesmo *input* para diferentes modelos e armazenar os seus *outputs* torna-se uma tarefa simples, limitada apenas pela disponibilidade de APIs e pela memória disponível para manter os vários contextos das conversas com os modelos. Dessa forma, a representação desse uso simultâneo de maneira intuitiva para o usuário na interface foi considerada como prioridade.

Com o objetivo de dar ao usuário acesso à visualização da atuação de qualquer um dos modelos enquanto responde aos *inputs*, o grupo decidiu por limitar a quantidade em apenas dois LLMs em um mesmo *chat*. Um desses modelos é sempre o de referência e o outro o comparado, com ambos podendo ser escolhidos pelo usuário na criação dos *chats*.

Para adequação na interface, foram alterados o modal de criação de novos *chats* e a cor de fundo das respostas devolvidas pelos modelos de linguagem, de forma a deixar claro ao usuário quais respostas pertencem ao LLM base e ao LLM comparado. As alterações podem ser visualizadas nas figuras 18 e 19.

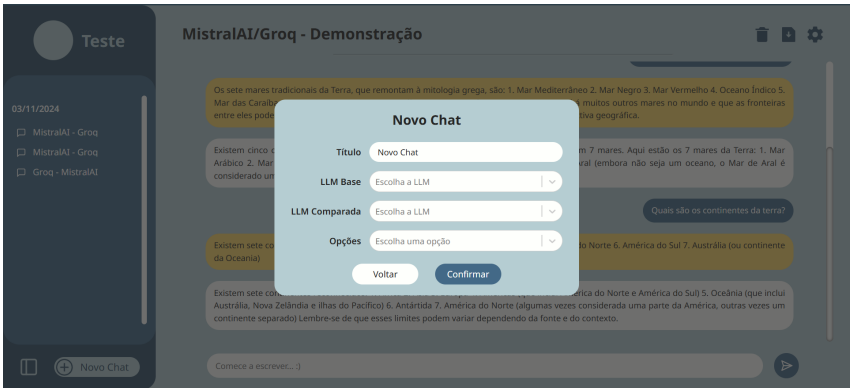


Figura 18 – Adição de Escolha do LLM Comparado no Modal de Criação de *Chats*

Figura 19 – Utilização de Dois LLMs em Mesmo *Chat*

5.1.6 Geração de *Logs* de Entrada para o HarpIA

Feita a implementação de uma comunicação estável entre dois modelos de linguagem diferentes simultaneamente, foram desenvolvidas *features* voltadas à avaliação do desempenho dos modelos, de maneira a cumprir os requisitos levantados anteriormente. Para tanto, foi feita uma integração entre o ambiente de múltiplos LLMs com o projeto de avaliação feitos pelo KEML, o HarpIA.

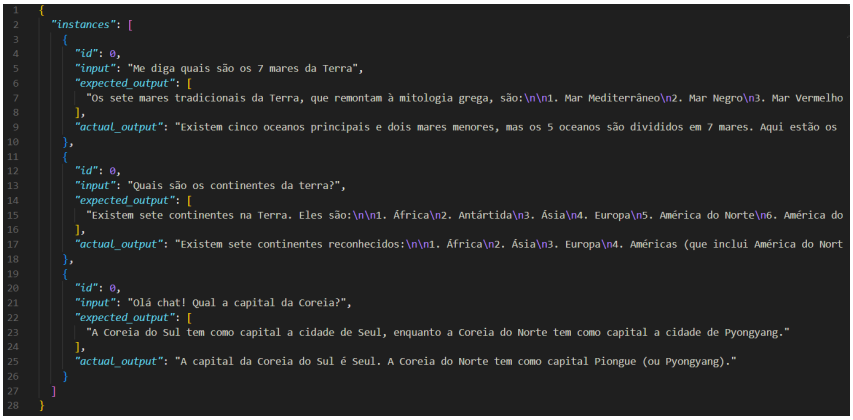
Essa integração permite que usuários do sistema possam facilmente utilizar o HarpIA para ter acesso a várias métricas de avaliação que permitam mensurar quantitativamente o desempenho dos modelos utilizados. Isso se dá pela geração automática de *logs* que podem ser diretamente utilizados pelo HarpIA para medir desempenho.

Os *logs* utilizados pelo HarpIA são arquivos .json que descrevem as respostas obtidas pelo modelo e uma lista de respostas esperadas para serem utilizados como referência. Além disso, também são enumeradas quais as métricas que devem ser utilizadas para avaliar o desempenho das respostas obtidas.

A geração desses *logs* seguiu duas abordagens, que visam atender necessidades diferentes dos usuários. Essas abordagens serão descritas a seguir.

5.1.6.1 *Logs* a partir de um Chat

A primeira abordagem feita trata-se da geração de um *log* a partir de um chat feito por um usuário. Esse *log* é construído utilizando os *outputs* gerados pelo LLM base como "*expected outputs*"e os *outputs* do LLM avaliado como "*actual outputs*". Essa abordagem prevê que o usuário defina manualmente quais métricas gostaria de utilizar na avaliação. De posse do *log* gerado e com as métricas definidas, o usuário final pode passá-lo diretamente ao HarpIA para que uma avaliação robusta seja feita.



```
{
  "instances": [
    {
      "id": 0,
      "input": "Me diga quais são os 7 mares da Terra",
      "expected_output": [
        "Os sete mares tradicionais da Terra, que remontam à mitologia grega, são:\n\n1. Mar Mediterrâneo\n2. Mar Negro\n3. Mar Vermelho
      ],
      "actual_output": "Existem cinco oceanos principais e dois mares menores, mas os 5 oceanos são divididos em 7 mares. Aqui estão os
    },
    {
      "id": 0,
      "input": "Quais são os continentes da terra?",
      "expected_output": [
        "Existem sete continentes na Terra. Eles são:\n\n1. África\n2. Antártida\n3. Ásia\n4. Europa\n5. América do Norte\n6. América do
      ],
      "actual_output": "Existem sete continentes reconhecidos:\n\n1. África\n2. Ásia\n3. Europa\n4. Américas (que inclui América do Nort
    },
    {
      "id": 0,
      "input": "Olá chat! Qual a capital da Coreia?",
      "expected_output": [
        "A Coreia do Sul tem como capital a cidade de Seul, enquanto a Coreia do Norte tem como capital a cidade de Pyongyang."
      ],
      "actual_output": "A capital da Coreia do Sul é Seul. A Coreia do Norte tem como capital Piongue (ou Pyongyang)."
    }
  ]
}
```


Figura 20 – *P*reenchimento do Campo *expected_output* no Log de um *Chat*

5.1.6.2 Preenchimento Automático do Campo *actual_output*

A utilização do sistema até então é baseada na necessidade de que haja um LLM que performe a tarefa a ser analisada suficientemente bem para que seja escolhido como o LLM base, referência com a qual outros modelos serão comparados. No entanto, esse nem sempre é o caso.

Para que haja uma maneira de medir o desempenho de múltiplos LLMs frente uma tarefa para a qual ainda não se tem um modelo base e que não dependam de comparações apenas um a um, a segunda abordagem foca em preencher um *log* com as respostas de múltiplos LLMs. O usuário pode mandar um *log* com as respostas esperadas já preenchidas mas com os campos de *actual outputs* vazio. A partir desse documento, o usuário lista todos os modelos que deseja testar. Assim, um documento com o *log* preenchido é gerado para cada LLM, de maneira que o HarpIA pode analisar individualmente o desempenho de cada modelo.

Assim, foi adicionado um modal para a inserção de um arquivo JSON de entrada do HarpIA previamente preenchido, com apenas o *actual_output* vazio. A partir de um modal, mostrado nas figuras 21 e 22, o usuário é capaz de realizar o *upload* de um arquivo no formato JSON e escolher os LLMs a serem utilizados. A partir destas informações, o sistema é preenche o campo com as respostas de cada LLM escolhido pelo usuário, retornando um arquivo "zip"contendo os *logs* totalmente preenchidos.

Figura 21 – *M*odal de Upload de Arquivo do HarpIA

Figura 22 – *M*odal de Upload de Arquivo do HarpIA com Arquivo Carregado

5.1.7 Adição de Métricas de Avaliação de LLMs

Por tratar-se de um projeto que visa atender às demandas de comparação entre diferentes modelos de linguagem, é pertinente que a própria ferramenta possibilite a utilização de processos avaliativos dos resultados em linguagem natural. Assim, com a integração com o *framework* Langchain e a comunicação com as APIs dos LLMs implementadas, foi possível adicionar a análise dos LLMs por meio de métricas de avaliação.

Cumprindo com os requisitos deste projeto, o usuário deve ser capaz de analisar e avaliar as respostas dos LLMs através das métricas de avaliação implementadas para uso direto no ambiente. Dessa forma, após uma sequência de mensagens trocadas entre o usuário e os LLMs, deve ser possível analisar o desempenho dos modelos, tendo como base de referência o LLM principal escolhido previamente pelo usuário.

5.1.7.1 Escolha das Métrica

Apesar do número de métricas integradas diretamente no ambiente desenvolvido ser limitado, a possibilidade de utilizar os dados gerados pelos modelos em um sistema de avaliação mais robusto através de *logs* gerados, como é o caso do HarpIA, permite que

análises mais profundas sejam feitas sobre os resultados obtidos. A seguir, cada uma das métricas implementadas é descrita sucintamente.

- **BERTScore**

  A métrica BERTScore é uma métrica de comparação de similaridade entre duas sentenças, utilizando modelos que levam em consideração as similaridades-cosseno entre cada *token* da frase. Essa métrica obteve resultados que se correlacionam mais fortemente com o julgamento de humanos (ZHANG et al., 2019).

  Essa métrica obtém valores para 3 diferentes atributos de similaridade: precisão (porcentagem das palavras da sentença avaliada que se encontram na sentença esperada), *recall* (porcentagem das palavras da sentença esperada que se encontram na sentença avaliada) e *f1 score* (média harmônica entre a precisão e o *recall*). No entanto, os modelos treinados não dependem de um *match* exato entre as palavras, mas sim de uma análise contextual, como ilustrado na figura 23.

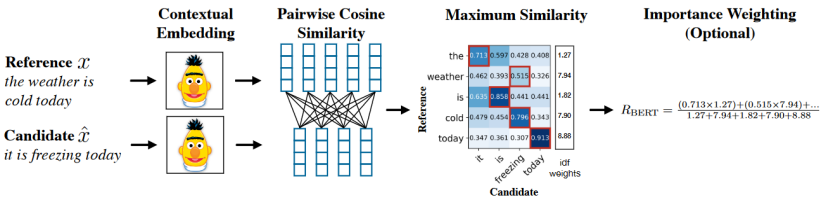


Figura 23 – Análise de uma sentença candidata, em que as palavras passam por um teste de similaridade de contexto para definir se ambas sentenças incluem a mesma ideia (ZHANG et al., 2019).

A implementação dessa métrica no ambiente foi feita seguindo o repositório de código criado por um dos autores do artigo que primeiro descreve a métrica BERTScore.

- **BLEU Score**

  A métrica BLEU foi proposta como um método de avaliação de traduções feitas por máquinas de maneira rápida e independente da linguagem, além de ser uma das primeiras métricas a se afirmar como tendo uma forte correlação com julgamento de humanos (PAPINENI et al., 2002). Ela se baseia no princípio básico de "quanto mais similar uma tradução de máquina é em relação a uma tradução profissional feita por um humano, melhor ela é."

  A partir disso, a métrica BLEU mede a similaridade entre uma tradução candidata e uma série de traduções de referência utilizando a "*word error rate*"(taxa de erros em palavras), uma medida utilizada frequentemente em pesquisas de reconhecimento de fala, porém adaptada para traduções. No caso, compara-se a sentença candidata a

uma lista de sentenças para procurar sub-sentenças que apareçam em ambas, através de uma *"n-gram match"*.

- **Character Error Rate**

  Character Error Rate, ou CER, é uma métrica comum para se medir a performance de sistemas de reconhecimento de fala. Essa métrica se baseia na metodologia descrita na abordagem do *Word Error Rate* (WER) que mede a performance de uma sentença analisada levando em consideração o custo de transformá-la na sentença de referência (MORRIS; MAIER; GREEN, 2004).

  Desta forma, a CER mede a distância entre a sentença avaliada e a sentença de referência. Dessa forma, quanto menor for o resultado dessa métrica, mais próxima a sentença avaliada está de sua referência. A diferença entre a CER e a WER é que a primeira faz essa medição em respeito a cada caractere da sentença, enquanto a WER leva em consideração apenas cada palavra da sentença.

- **CharacTer**

  Essa métrica, tal como o CER, também se baseia na distância entre a sentença avaliada e a sentença de referência a nível de caracteres utilizados (WANG et al., 2016). No entanto, o custo é normalizado pelo tamanho da sentença avaliada. Novamente, quanto menor o resultado da métrica CharacTer, mais próxima a sentença avaliada está de sua referência.

- **chrf**

  A métrica chrf, que também foca em avaliação de traduções realizadas por máquinas, utiliza uma combinação de valores F1 sobre uma avaliação em *n-gram match*. Os valores F1, ou pontuação F1, são uma forma de medir a precisão de um determinado resultado face a um resultado esperado. Uma pontuação F1 pode ser calculada com a fórmula abaixo, na qual *precision* é a proporção dos verdadeiros positivos dentre todas as hipóteses de positivos e *recall* é a proporção de verdadeiros positivos dentre todos os positivos.

  $$\frac{2 \cdot (\text{precision} \cdot \text{recall})}{\text{precision} + \text{recall}} \quad ;$$

  Ou seja, a pontuação F1 é uma média harmônica entre a *precision* e o *recall*. Nesse sentido, a métrica chrf utiliza uma comparação de *n-gram matches* para verificar as sub-sentenças presentes na sentença avaliada e na sentença de referência e calcula uma pontuação F1. (POPOVIĆ, 2015).

- **COMET**

Crosslingual Optimized Metric for Evaluation of Translation ou "COMET"é um *framework open-source* para treinamento de modelos de avaliação de tradução feitas por máquinas (REI et al., 2020). Um dos modelos treinados pelos autores do *framework* é utilizado para avaliar os resultados dos LLMs. Nesse contexto, a menor nota possível é zero e a maior é 1.

- **Google BLEU**

  Uma adaptação da métrica BLEU feita pela Google visando aumentar a usabilidade desse tipo de métrica para avaliar um conjunto de apenas uma sentença candidata à tradução e um conjunto de referências (WU, 2016). Isso pois o BLEU original não funciona tão bem quando não está avaliando um corpo de sentenças de tradução de máquina. Para tanto, o Google BLEU limita as sub-sentenças a serem utilizadas nas técnicas de *n-gram matching* a terem, no máximo, 4 tokens (ou palavras).

- **METEOR**

  METEOR é uma métrica de avaliação de traduções de máquina baseada na pontuação F1 das sentenças analisadas utilizando um *n-gram matching* de apenas um elemento. O valor dessa métrica representa o quão bem ordenadas estão as palavras da sentença avaliada que também correspondem (exatamente ou semanticamente) às palavras na sentença de referência (BANERJEE; LAVIE, 2005).

- **Rouge**

  Rouge, ou "*Recall-Oriented Understudy for Gisting Evaluation*"é um conjunto de métricas para avaliação de traduções e resumos feitos de maneira automatizada. O conjunto de métricas utiliza *n-gram matches* de 1 a 2 elementos, produzindo valores diferentes para cada uma das abordagens. Além disso, também calcula uma pontuação com base na maior sub-sentença presente em ambas as sentenças avaliadas e de referência (LIN, 2004). Quanto mais a pontuação se aproximar de 1, mais próximas as sentenças avaliadas são de suas referências.

- **SacreBLEU**

  A métrica SacreBLEU busca resolver os problemas existentes na métrica BLEU relacionados aos seus parâmetros pouco conhecidos, que dificultam a comparação de *scores* entre casos de uso (POST, 2018). Desta forma, o SacreBLEU resolve os problemas de pré-processamento do BLEU, explicitando as parametrizações utilizadas e mantendo um conjunto de testes padrão. Isso possibilita a comparação direta entre scores do BLEU.

- **Translation Edir Rate**

  A métrica "*Translation Edit Rate*", ou "TER", mede a proximidade da sentença avaliada da sentença de referência através do número de edições necessárias para

tornar a sentença avaliada igual à sentença de referência (SNOVER et al., 2006). A equação do cálculo dessa métrica está dada abaixo.

$$\text{ter score} = 100 \cdot \frac{\text{number of edits}}{\text{averaga reference length}} \tag{5.1}$$

- **Word Error Rate**

  Word Error Rate, ou WER, é também uma métrica que mensura a proximidade da sentença avaliada à sua sentença de referência, porém, diferentemente do CER, se baseia em mudanças de palavras ao invés de caracteres (MORRIS; MAIER; GREEN, 2004).

5.1.7.2 Avaliação dos *Chats* pelas Métricas

Dado um chat feito com múltiplos LLMs e tendo um como base, é possível aplicar as métricas avaliativas para analisar o desempenho de um dos LLMs utilizando o outro como fonte dos resultados esperados dentro do próprio ambiente, sem a necessidade de gerar *logs* para o HarpIA. Por exemplo, para uma tarefa de tradução para a qual o modelo do ChatGPT seja considerado capaz de gerar respostas corretas, poderemos analisar as traduções geradas por outro LLM em comparação às respostas dadas pelo ChatGPT utilizando a métrica "TER".

Essa funcionalidade foi integrada no ambiente de maneira que seja possível facilmente requisitar uma avaliação sobre o chat gerado em qualquer momento através do botão de avaliação. No entanto, para uma avaliação mais robusta e com várias métricas simultaneamente, o HarpIA ainda seria uma ferramenta mais adequada. Desse modo, foram implementados (1) um modal contendo a descrição de cada uma das métricas utilizadas dentro do projeto e (2) um modal de avaliação das mensagens de um determinado *chat*, permitindo que o usuário escolha a métrica a ser utilizada, retornando o *score* final.

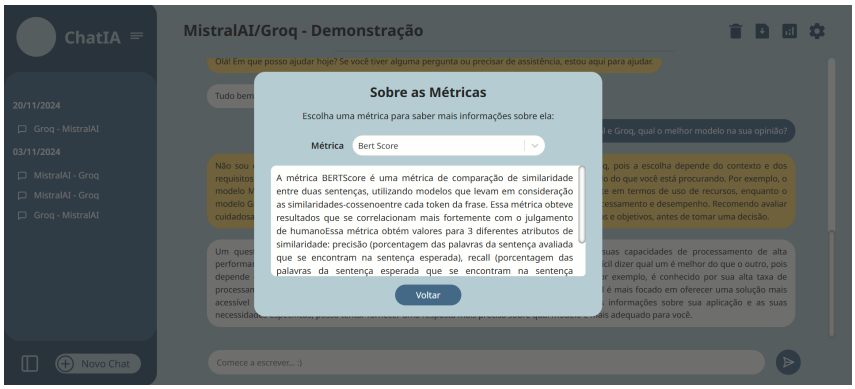


Figura 24 – *M*odal de Descrição das Métricas

Figura 25 – *M*odal de Avaliação - Escolha da Métrica

Figura 26 – *M*odal de Avaliação - *Score* da Métrica

5.1.8 Edição do *Prompt* de Cada *Chat*

Apesar do usuário ser capaz de alterar o *prompt* utilizado nas conversas de maneira direta pelo código-fonte, essa opção não torna a experiência otimizada nem eficiente. Desta forma, foi pensado na possibilidade do próprio usuário poder alterar o *prompt* de um *chat* diretamente pela interface.

Assim, para permitir que o usuário configure os modelos de linguagem de maneira simplificada, foi implementada a edição de *prompts*. Essa funcionalidade permite que os usuários possam definir as variáveis *system*, *human* e *ai* do *prompt* utilizado pelos LLMs em um *chat*. Com essa configuração, é possível definir o comportamento dos modelos de linguagem.

Para a edição de *prompt*, fez-se necessário a adição de um modal onde o usuário pode ajustar livremente os *templates* do *prompt* de cada conversa criada. A implementação pode ser visualizada na figura 27.

Figura 27 – Modal de Edição de *Prompt*

5.1.9 Exclusão de *Chats*

Por fim, para que os usuários possam montar diferentes tipos de testes dos LLMs, foi implementada a deleção de *chats*, permitindo que o usuário possa gerar diferentes *logs* e respostas de acordo com a necessidade. Desta forma, uma vez deletado, o usuário não será capaz de recuperar o *chat*, e esse não aparecerá na lista de conversas.

Figura 28 – Modal de Deleção de um *Chat*

5.1.10 Disponibilização do Código Fonte e Licença

Com a finalização da implementação do projeto, a disponibilização do código-fonte do ambiente desenvolvido torna-se um requisito essencial para que esse trabalho cumpra com os seus objetivos propostos. Essa disponibilização deve ser feita tanto para a comunidade científica quanto para grupos que desejem comparar e escolher LLMs para seus projetos

Portanto, o projeto está sob a licença *open-source* Apache License 2.0 (ASL 2.0). Essa licença permite o uso do trabalho desenvolvido em projetos em outras licenças, permitindo o uso em projetos comerciais, além de prever a possibilidade de modificação do código-fonte original.

5.2 Testes e Validação

Ao longo do desenvolvimento deste projeto, foram feitas validações dos conceitos utilizados e algumas reformulações dos objetivos finais que o sistema deveria alcançar, em concordância com o orientador do projeto. As reformulações foram fundamentais para o desenvolvimento de um trabalho conciso e que agregasse tanto às pesquisas desenvolvidas pelo grupo KEML-USP quanto para projetos externos na área de Grandes Modelos de Linguagem.

Na finalização do projeto, o grupo convidou o mestre Vinícius Bitencourt Matos, responsável principal no desenvolvimento do projeto HarpIA, para análise e testagem do sistema. Dessa forma, foi possível recolher feedbacks importantes para a melhoria e conclusão do projeto, principalmente para a sua utilização em conjunto com o HarpIA e visões de trabalhos futuros.

Por fim, como etapa final do desenvolvimento do projeto, foi feita uma última validação juntamente com o orientador do projeto, Fábio Cozman. Esta validação foi de extrema importância para a aferição dos resultados esperados e cumpridos do projeto. Os resultados e conclusões feitos a partir dos testes estarão descritos em mais detalhes no capítulo 6.

6 Considerações Finais

Neste capítulo, por fim, é feito o balanço do trabalho realizado neste projeto de conclusão de curso, apresentando os resultados atingidos e não atingidos, justificando cada resultado. Além disso, também são descritas as contribuições feitas para este trabalho e as perspectivas de continuidade a partir do ambiente desenvolvido.

6.1 Conclusões do Projeto de Formatura

Com base nos requisitos e objetivos propostos descritos neste trabalho, é possível dizer que o ambiente para testes e comparações de múltiplos LLMs cumpriu com as expectativas esperadas. A ferramenta desenvolvida mostrou-se útil na comparação de modelos de linguagem, oferecendo ao usuário a utilização de diferentes métricas de avaliação, acesso a diversos LLMs e suporte para o seu uso juntamente com o projeto HarpIA.

Durante o desenvolvimento do projeto, o grupo focou principalmente em funcionalidades que auxiliassem no uso do HarpIA. Desta forma, as funcionalidades de preenchimento automático de *logs*, seja apenas pela resposta ou pelo download das mensagens, demonstraram-se úteis para o projeto. Ademais, essas questões já haviam sido levantadas anteriormente em discussões sobre o HarpIA.

Os testes realizados com o orientador do projeto e com o mestre Vinícius Bitencourt Matos, que coordena o desenvolvimento do HarpIA, atestaram a utilidade do ambiente desenvolvido para diferentes projetos do KEML. A integração do projeto com o HarpIA mostra-se importante para incentivar a sua ampla utilização pela comunidade acadêmica, podendo consolidar a ferramenta como uma solução para avaliar e comparar modelos de linguagem.

Além disso, também entende-se que o projeto possa ser utilizado pelos projetos do KEML que se beneficiem da utilização de diferentes LLMs em um mesmo ambiente integrado. Com a finalização deste trabalho, o domínio do código-fonte será repassado para os coordenadores do KEML, de forma que este possa ser utilizado livremente pelo grupo.

6.2 Contribuições

A construção deste projeto foi feita em alinhamento com o grupo KEML-USP, buscando auxiliar o grupo na pesquisa e no desenvolvimento de projetos na área de Grandes Modelos de Linguagem, colaborando sobretudo para o projeto HarpIA. A principal parceria deu-se pela implementação de métricas de avaliação no contexto de perguntas e

respostas, utilizando classificações *word-based*, *char-based* e de *accuracy* tanto no ambiente desenvolvido quanto no próprio HarpIA. Assim, o grupo pôde contribuir diretamente com um projeto do KEML para além do sistema descrito neste trabalho.

Desta forma, é importante destacar que todas as métricas de avaliação implementadas neste projeto foram baseadas na literatura já existente, disponível em artigos e pesquisas acadêmicas. Além disso, também foram utilizadas bibliotecas prontas disponíveis em Python, cuja implementação foi feita por terceiros, para a construção de algumas métricas presentes neste trabalho.

O ambiente desenvolvido disponibiliza uma maneira rápida para que pesquisadores do KEML gerem dados avaliativos sobre múltiplos LLMs simultaneamente. Esses dados, disponibilizados num formato que pode ser utilizado no HarpIA, podem facilmente ser submetidos a uma série de avaliações. É também esperado que a disponibilização do projeto auxilie as escolhas direcionadas dos modelos de linguagem a serem utilizados em futuros projetos e pesquisas, além de incentivar o uso do HarpIA.

6.3 Perspectivas de Continuidade

A partir da ferramenta e do ambiente desenvolvidos neste projeto, disponíveis na plataforma do GitHub, espera-se que seu uso facilite o desenvolvimento de novos métodos de avaliação e comparação do desempenho de modelos de linguagem. Ademais, entende-se que o ambiente também poderá ser utilizado como base para a implementação de projetos que visem à realização de testes de LLMs em um único ambiente integrado.

Por fim, o ambiente pode ser utilizado como ponte de integração para futuros e presentes projetos do KEML que se beneficiem do acesso a múltiplos LLMs simultaneamente. Além disso, o sistema apresenta a possibilidade de ser publicado para sua utilização pelo público geral, com a devida implementação de requisitos de segurança e financiamento do projeto.

Referências

6SENSE. *Top 5 Source Code Management technologies in 2024*. 2024. Disponível em: <https://6sense.com/tech/source-code-management>. Acesso em: 27 set. 2024. Citado na página 30.

AGGARWA, S. Modern Web-Development using ReactJS. *International Journal of Recent Research Aspects*, 2018. Citado na página 31.

AGUILERA, F. M. *Mistral LLM: A New Era in Language Models*. 2024. Disponível em: <https://www.linkedin.com/pulse/mistral-llm-new-era-language-models-frank-62cde/?trk=article-ssr-frontend-pulse_more-articles_related-content-card>. Acesso em: 26 fev. 2024. Citado na página 28.

ALURA. *React: o que é, como funciona e um guia dessa popular ferramenta JS*. 2024. Disponível em: <https://www.alura.com.br/artigos/react-js#:~:text=%C3%89%20popular%20e%20amplamente%20utilizado,todos%20os%20tipos%20de%20projetos.> Acesso em: 27 set. 2024. Citado na página 31.

ANDERSON, P. et al. Spice: Semantic propositional image caption evaluation. In: SPRINGER. *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part V 14*. [S.l.], 2016. p. 382–398. Citado na página 25.

ATTARD, S. *The Evolution of LLMs Over the Last 12 Months*. 2023. Disponível em: <https://medium.com/@simon_attard/the-evolution-of-llms-over-the-last-12-months-188a04edb3ac>. Acesso em: 17 mar. 2024. Citado na página 19.

AWAN, A. A. *Introduction to Meta AI's LLaMA*. 2023. Disponível em: <https://www.datacamp.com/blog/introduction-to-meta-ai-llama>. Acesso em: 26 fev. 2024. Citado na página 27.

AWS. *O que é o LangChain?* 2024. Disponível em: <https://aws.amazon.com/pt/what-is/langchain/>. Acesso em: 05 mar. 2024. Citado na página 32.

AWS AMAZON. *O que é GPT?* 2024. Disponível em: <https://aws.amazon.com/pt/what-is/gpt/>. Acesso em: 26 fev. 2024. Citado na página 27.

AXIOS. *Getting Started*. 2024. Disponível em: <https://axios-http.com/docs/intro>. Acesso em: 16 out. 2024. Citado na página 31.

BANERJEE, S.; LAVIE, A. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In: *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. [S.l.: s.n.], 2005. p. 65–72. Citado na página 56.

C4AI. *Publicações no C4AI*. 2021. Disponível em: <https://c4ai.inova.usp.br/pt/publicacoes/>. Acesso em: 17 mar. 2024. Citado na página 19.

C4AI. *KEML: Aprendizado de Máquinas com Conhecimento Aprimorado*. 2024. Disponível em: <https://c4ai.inova.usp.br/pt/pesquisas_2/#KEML_B_eng>. Acesso em: 12 fev. 2024. Citado na página 19.

DAIR.AI. *Prompt Engineering Guide*. 2024. Disponível em: <https://www.promptingguide.ai/>. Acesso em: 17 mar. 2024. Citado na página 24.

ESTÊVÃO, J. M.; ESTÊVÃO, M. D. Inteligência Artificial na avaliação tradicional: aquisição de conhecimento vs prompt engineering. *de Práticas Pedagógicas no Ensino Superior*, p. 73, 2023. Citado na página 19.

GAGLIARDI, V. Modern Django and the Django REST framework. In: *Decoupled Django: Understand and Build Decoupled Django Architectures for JavaScript Front-ends*. [S.l.]: Springer, 2021. p. 31–40. Citado na página 29.

GITHUB. *About branches*. 2024. Disponível em: <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches>. Acesso em: 27 set. 2024. Citado na página 30.

GITHUB. *About pull requests*. 2024. Disponível em: <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests>. Acesso em: 27 set. 2024. Citado na página 30.

GITHUB. *Sobre o GitHub e o Git*. 2024. Disponível em: <https://docs.github.com/pt/get-started/start-your-journey/about-github-and-git>. Acesso em: 27 set. 2024. Citado na página 30.

GITHUB. *The tools you need to build what you want.* 2024. Disponível em: <https://github.com/features>. Acesso em: 27 set. 2024. Citado na página 30.

GOOGLE. *Gemma models overview*. 2024. Disponível em: <https://ai.google.dev/gemma/docs>. Acesso em: 06 dez. 2024. Citado na página 45.

GROOTENDORST, M. *A Visual Guide to Mixture of Experts (MoE)*. 2024. Disponível em: <https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mixture-of-experts>. Acesso em: 06 dez. 2024. Citado na página 23.

GROQCLOUD. *Supported Models*. 2024. Disponível em: <https://console.groq.com/docs/models>. Acesso em: 06 dez. 2024. Citado na página 48.

HUGGING FACE. *Model Card: databricks/dbrx-instruct*. 2024. Disponível em: <https://huggingface.co/databricks/dbrx-instruct>. Acesso em: 06 dez. 2024. Citado na página 45.

HUGGING FACE. *Model Card: google/gemma-1.1-7b-it*. 2024. Disponível em: <https://huggingface.co/google/gemma-1.1-7b-it>. Acesso em: 06 dez. 2024. Citado na página 45.

HUGGING FACE. *Model Card: google/gemma-2-9b-it*. 2024. Disponível em: <https://huggingface.co/google/gemma-2-9b-it>. Acesso em: 06 dez. 2024. Citado na página 45.

HUGGING FACE. *Model Card: meta-llama/Llama-3.2-3B*. 2024. Disponível em: <https://huggingface.co/meta-llama/Llama-3.2-3B>. Acesso em: 06 dez. 2024. Citado na página 47.

HUGGING FACE. *Model Card: meta-llama/Meta-Llama-3-70B-Instruct*. 2024. Disponível em: <https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct>. Acesso em: 06 dez. 2024. Citado na página 46.

HUGGING FACE. *Model Card: meta-llama/Meta-Llama-3-8B-Instruct*. 2024. Disponível em: <https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct>. Acesso em: 06 dez. 2024. Citado na página 46.

HUGGING FACE. *Model Card: mistralai/Mistral-Nemo-Instruct-2407*. 2024. Disponível em: <https://huggingface.co/mistralai/Mistral-Nemo-Instruct-2407>. Acesso em: 06 dez. 2024. Citado na página 47.

HUGGING FACE. *Model Card: mistralai/Mistral-Small-Instruct-2409*. 2024. Disponível em: <https://huggingface.co/mistralai/Mistral-Small-Instruct-2409>. Acesso em: 06 dez. 2024. Citado na página 47.

HUGGING FACE. *Model Card: mistralai/Mixtral-8x7B-Instruct-v0.1*. 2024. Disponível em: <https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1>. Acesso em: 06 dez. 2024. Citado na página 47.

HUGGING FACE. *Model Card: mistralai/Pixtral-12B-2409*. 2024. Disponível em: <https://huggingface.co/mistralai/Pixtral-12B-2409>. Acesso em: 06 dez. 2024. Citado na página 48.

HUGGING FACE. *Model Card: Qwen/Qwen2.5-7B-Instruct*. 2024. Disponível em: <https://huggingface.co/Qwen/Qwen2.5-7B-Instruct>. Acesso em: 06 dez. 2024. Citado na página 48.

HUGGING FACE. *Model Card: upstage/SOLAR-10.7B-Instruct-v1.0*. 2024. Disponível em: <https://huggingface.co/upstage/SOLAR-10.7B-Instruct-v1.0>. Acesso em: 06 dez. 2024. Citado na página 48.

IBM. *What is LangChain?* 2024. Disponível em: <https://www.ibm.com/topics/langchain>. Acesso em: 05 mar. 2024. Citado na página 32.

IBM. *What is mixture of experts?* 2024. Disponível em: <https://www.ibm.com/topics/mixture-of-experts>. Acesso em: 6 dez. 2024. Citado na página 23.

JI, J.-y.; KUMAR, R. *Gemma explained: What's new in Gemma 2*. 2024. Disponível em: <https://developers.googleblog.com/en/gemma-explained-new-in-gemma-2/>. Acesso em: 06 dez. 2024. Citado na página 45.

LANGCHAIN. *Chat models*. 2024. Disponível em: <https://python.langchain.com/docs/integrations/chat/>. Acesso em: 06 dez. 2024. Citado na página 48.

LANGCHAIN. *ChatGroq*. 2024. Disponível em: <https://python.langchain.com/docs/integrations/chat/groq/>. Acesso em: 06 dez. 2024. Citado na página 48.

LANGCHAIN. *ChatMistralAI*. 2024. Disponível em: <https://python.langchain.com/docs/integrations/chat/mistralai/>. Acesso em: 06 dez. 2024. Citado na página 49.

LANGCHAIN. *ChatOpenAI*. 2024. Disponível em: <https://python.langchain.com/docs/integrations/chat/openai/>. Acesso em: 06 dez. 2024. Citado na página 49.

LANGCHAIN. *ChatTogether*. 2024. Disponível em: <https://python.langchain.com/docs/integrations/chat/together/>. Acesso em: 06 dez. 2024. Citado na página 49.

LANGCHAIN. *LangChain*. 2024. Disponível em: <https://www.langchain.com/langchain>. Acesso em: 17 mar. 2024. Citado 2 vezes nas páginas 29 e 32.

LIGABUE, P. d. M. et al. Applying a Context-based Method to Build a Knowledge Graph for the Blue Amazon. *Data Intelligence*, MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . , p. 1–63, 2024. Citado na página 25.

LIGABUE, P. de M. et al. Blabkg: a Knowledge Graph for the Blue Amazon. In: IEEE. *2022 IEEE International Conference on Knowledge Graph (ICKG)*. [S.l.], 2022. p. 164–171. Citado na página 25.

LIN, C.-Y. Rouge: A package for automatic evaluation of summaries. In: *Text summarization branches out*. [S.l.: s.n.], 2004. p. 74–81. Citado na página 56.

LIN, C.-Y.; HOVY, E. Automatic evaluation of summaries using n-gram co-occurrence statistics. In: *Proceedings of the 2003 human language technology conference of the North American chapter of the association for computational linguistics*. [S.l.: s.n.], 2003. p. 150–157. Citado na página 25.

MAHOWALD, K. et al. Dissociating language and thought in large language models. *Trends in Cognitive Sciences*, Elsevier, 2024. Citado na página 23.

META. *Discover the power of LLama*. 2024. Disponível em: <https://llama.meta.com/>. Acesso em: 26 fev. 2024. Citado na página 27.

META-LLAMA. *llama-models/models/llama3*$_1$*/MODEL*$_C$*ARD.md*.2024.*Disponívelem* : <>. Acesso em: 06 dez. 2024. Citado na página 46.

META-LLAMA. *llama-models/models/llama3*$_3$*/MODEL*$_C$*ARD.md*.2024.*Disponívelem* : <>. Acesso em: 06 dez. 2024. Citado na página 47.

META OPEN SOURCE. *Introducing JSX*. 2024. Disponível em: <https://legacy.reactjs.org/docs/introducing-jsx.html>. Acesso em: 27 set. 2024. Citado na página 31.

META OPEN SOURCE. *React*. 2024. Disponível em: <https://react.dev/>. Acesso em: 27 set. 2024. Citado na página 31.

MISTRAL AI. *Models Overview*. 2024. Disponível em: <https://docs.mistral.ai/getting-started/models/models_overview/>. Acesso em: 06 dez. 2024. Citado na página 49.

MISTRAL AI TEAM. *AI in abundance*. 2024. Disponível em: <https://mistral.ai/news/september-24-release/>. Acesso em: 06 dez. 2024. Citado na página 47.

MISTRAL AI TEAM. *Announcing Pixtral 12B*. 2024. Disponível em: <https://mistral.ai/news/pixtral-12b/>. Acesso em: 06 dez. 2024. Citado na página 48.

MISTRAL AI TEAM. *Mistral Nemo*. 2024. Disponível em: <https://mistral.ai/news/mistral-nemo/>. Acesso em: 06 dez. 2024. Citado na página 47.

MISTRAL AI TEAM. *Mixtral of experts*. 2024. Disponível em: <https://mistral.ai/news/mixtral-of-experts/>. Acesso em: 06 dez. 2024. Citado na página 47.

MISTRALAI. *Mistral AI API (0.0.1)*. 2024. Disponível em: <https://docs.mistral.ai/api/>. Acesso em: 26 fev. 2024. Citado na página 28.

MOLLMAN, S. Chatgpt gained 1 million users in under a week. here's why the AI chatbot is primed to disrupt search as we know it. *Fortune*, 2022. Disponível em: <https://fortune.com/2022/12/09/ai-chatbot-chatgpt-could-disrupt-google-search-engines-business>. Citado na página 19.

MORRIS, A. C.; MAIER, V.; GREEN, P. D. From WER and RIL to MER and WIL: improved evaluation measures for connected speech recognition. In: *Interspeech*. [S.l.: s.n.], 2004. p. 2765–2768. Citado 2 vezes nas páginas 55 e 57.

MYSCALE. *Llama 3.1 405B, 70B, 8B: A Quick Comparison*. 2024. Disponível em: <https://myscale.com/blog/llama-3-1-405b-70b-8b-quick-comparison/>. Acesso em: 06 dez. 2024. Citado na página 46.

OLLAMA. *Llama 3*. 2024. Disponível em: <https://ollama.com/library/llama3>. Acesso em: 06 dez. 2024. Citado na página 46.

OLLAMA. *Llama 3.1*. 2024. Disponível em: <https://ollama.com/library/llama3.1>. Acesso em: 06 dez. 2024. Citado na página 46.

OLLAMA. *Llama 3.2*. 2024. Disponível em: <https://ollama.com/library/llama3.2>. Acesso em: 06 dez. 2024. Citado na página 47.

OLLAMA. *Llama 3.3*. 2024. Disponível em: <https://ollama.com/library/llama3.3>. Acesso em: 06 dez. 2024. Citado na página 47.

OLLAMA. *Solar*. 2024. Disponível em: <https://ollama.com/library/solar>. Acesso em: 06 dez. 2024. Citado na página 48.

OPENAI. *GPT-4*. 2024. Disponível em: <https://openai.com/gpt-4>. Acesso em: 26 fev. 2024. Citado na página 27.

OPENAI. *Models*. 2024. Disponível em: <https://platform.openai.com/docs/models>. Acesso em: 06 dez. 2024. Citado 3 vezes nas páginas 45, 46 e 49.

OPENAI. *Prompt engineering*. 2024. Disponível em: <https://platform.openai.com/docs/guides/prompt-engineering>. Acesso em: 10 dez. 2024. Citado na página 24.

PAPINENI, K. et al. Bleu: a method for automatic evaluation of machine translation. In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. [S.l.: s.n.], 2002. p. 311–318. Citado 2 vezes nas páginas 25 e 54.

POPOVIĆ, M. chrF: character n-gram f-score for automatic mt evaluation. In: *Proceedings of the tenth workshop on statistical machine translation*. [S.l.: s.n.], 2015. p. 392–395. Citado na página 55.

POST, M. A Call for Clarity in Reporting BLEU scores. In: BOJAR, O. et al. (Ed.). *Proceedings of the Third Conference on Machine Translation: Research Papers*. Brussels, Belgium: Association for Computational Linguistics, 2018. p. 186–191. Disponível em: <https://aclanthology.org/W18-6319>. Citado na página 56.

QWEN TEAM. *Qwen*. 2024. Disponível em: <https://qwen.readthedocs.io/en/latest/>. Acesso em: 06 dez. 2024. Citado na página 48.

QWEN TEAM. *Qwen2.5-LLM: Extending the boundary of LLMs*. 2024. Disponível em: <https://qwenlm.github.io/blog/qwen2.5-llm/>. Acesso em: 06 dez. 2024. Citado na página 48.

REI, R. et al. COMET: A neural framework for MT evaluation. *arXiv preprint arXiv:2009.09025*, 2020. Citado na página 56.

SNOVER, M. et al. A study of translation edit rate with targeted human annotation. In: *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*. [S.l.: s.n.], 2006. p. 223–231. Citado na página 57.

SU, K.-Y.; WU, M.-W.; CHANG, J.-S. A new quantitative quality measure for machine translation systems. In: *COLING 1992 Volume 2: The 14th international conference on computational linguistics*. [S.l.: s.n.], 1992. Citado na página 21.

THE MOSAIC RESEARCH TEAM. *Introducing DBRX: A New State-of-the-Art Open LLM*. 2024. Disponível em: <https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm>. Acesso em: 06 dez. 2024. Citado na página 45.

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *About PostgreSQL*. 2024. Disponível em: <https://www.postgresql.org/about/>. Acesso em: 27 set. 2024. Citado na página 32.

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *What Is PostgreSQL?* 2024. Disponível em: <https://www.postgresql.org/docs/current/intro-whatis.html>. Acesso em: 27 set. 2024. Citado na página 32.

THIRUNAVUKARASU, A. J. et al. Large language models in medicine. *Nature medicine*, Nature Publishing Group US New York, v. 29, n. 8, p. 1930–1940, 2023. Citado na página 23.

TOGETHER.AI. *Serverless models*. 2024. Disponível em: <https://docs.together.ai/docs/serverless-models>. Acesso em: 06 dez. 2024. Citado na página 49.

WANG, W. et al. Character: Translation edit rate on character level. In: *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*. [S.l.: s.n.], 2016. p. 505–510. Citado na página 55.

WU, Y. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016. Citado na página 56.

ZHANG, T. et al. Bertscore: Evaluating text generation with BERT. *arXiv preprint arXiv:1904.09675*, 2019. Citado 2 vezes nas páginas 11 e 54.

ZHAO, W. X. et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023. Citado na página 19.