UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Integração de LLMs com APIs: um estudo de caso

Luciano Lima Silva

Monografia - MBA em Inteligência Artificial e Big Data

ICMC USP
SÃO CARLOS

Luciano Lima Silva

Integração de LLMs com APIs: um estudo de caso

Monografia apresentada ao Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - ICMC/USP, como parte dos requisitos para obtenção do título de Especialista em Inteligência Artificial e Big Data.

Área de concentração: Inteligência Artificial

Orientador: Prof. Jean Roberto Ponciano

Versão original

São Carlos

2024

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

S586i

Silva, Luciano Lima
Integração de LLMs com APIs: um estudo de caso / Luciano Lima Silva; orientador Jean Roberto Ponciano; coorientador Solange Oliveira Rezende. -- São Carlos, 2024.
52 p.

Trabalho de conclusão de curso (MBA em Inteligência Artificial e Big Data) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2024.

1. Integração de LLMs com APIs. 2. Inteligência Artificial. 3. Integração de Sistemas. 4. AI. 5. API. I. Ponciano, Jean Roberto, orient. II. Rezende, Solange Oliveira, coorient. III. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Luciano Lima Silva

Integração de LLMs com APIs: um estudo de caso

Monograph presented to the Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - ICMC/USP, as part of the requirements for obtaining the title of Specialist in Artificial Intelligence and Big Data.

Concentration area: Artificial Intelligence

Advisor: Jean Roberto Ponciano

Original version

São Carlos

2024

Este trabalho é dedicado aos meus pais e à minha família, com gratidão à Força Divina que permeia o Universo. Que ele possa também servir como contribuição para todos aqueles que buscam conhecimento e crescimento.

AGRADECIMENTOS

Agradeço a Deus e à minha família pelo apoio e incentivo constante, que foram essenciais para que eu chegasse até aqui.

Ao meu orientador e aos professores, pela dedicação e pelos ensinamentos que me guiaram durante todo o processo.

E aos colegas, pela companhia e pelas trocas de conhecimento.

"O estudo, a busca da verdade e da beleza são domínios em que nos é consentido sermos crianças por toda a vida."

Albert Einstein

RESUMO

SILVA, L.S. **Integração de LLMs com APIs: um estudo de caso**. 2024. 52 p. Monografia (MBA em Inteligência Artificial e Big Data) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2024.

Este trabalho explora a integração de Grandes Modelos de Linguagem (LLMs) com APIs, visando modernizar e simplificar a interação entre o usuário e o sistema. A hipótese central é que a utilização de LLMs pode melhorar significativamente a identificação das intenções dos usuários e automatizar a execução de tarefas por meio de chamadas de API, utilizando linguagem natural. Para validar essa hipótese, foram selecionados três modelos de LLMs: tinyllama-bnb-4bit, gemma-7b-bnb-4bit e llama-3-8b-bnb-4bit, que passaram por um processo de ajuste fino (fine-tuning) utilizando um dataset criado a partir de registros anonimizados de clientes, associados a diferentes ações do sistema, como "ConsultarSaldo", "CadastrarCliente"e "CadastrarBoleto". Esse dataset foi ampliado com variações de frases em linguagem natural para aumentar a robustez do treinamento. O processo de fine-tuning demonstrou ser eficaz na redução da discrepância entre as saídas esperadas e as geradas pelos modelos, evidenciado por melhorias nas métricas de distância de string e de JSON. Os resultados indicam que a integração proposta pode oferecer uma interface mais intuitiva, aproximando a comunicação entre o usuário e o sistema.

Palavras-chave: LLM, API, JSON, Fine-tuning, Integração de Sistemas, Inteligência Artificial, AI.

ABSTRACT

SILVA, L.S. **Integração de LLMs com APIs: um estudo de caso**. 2024. 52 p. Monograph (MBA in Artificial Intelligence and Big Data) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2024.

This work explores the integration of Large Language Models (LLMs) with APIs, with the goal of modernizing and simplifying the interaction between the user and the system. The central hypothesis is that the use of LLMs can significantly improve the identification of user intentions and automate task execution through API calls using natural language. To validate this hypothesis, three LLM models were selected: tinyllama-bnb-4bit, gemma-7b-bnb-4bit, and llama-3-8b-bnb-4bit, which underwent a fine-tuning process using a dataset created from anonymized customer records associated with different system actions, such as "ConsultarSaldo," "CadastrarCliente," and "CadastrarBoleto." This dataset was expanded with variations of natural language phrases to increase the robustness of the training. The fine-tuning process proved effective in reducing the discrepancy between expected and generated outputs, as evidenced by improvements in string and JSON distance metrics. The results indicate that the proposed integration can offer a more intuitive interface, bringing the communication between the user and the system closer to human language.

Keywords: LLM, API, JSON, Fine-tuning, Systems Integration, Artificial Intelligence, AI.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
IBGE	Instituto Brasileiro de Geografia e Estatística
USP	Universidade de São Paulo
USPSC	Campus USP de São Carlos
LLM	Large Language Model (Grande Modelo de Linguagem)
API	Application Programming Interface (Interface de Programação de Aplicativos)
CPF	Cadastro de Pessoas Físicas
JSON	JavaScript Object Notation (Notação de Objetos JavaScript)
AI	Artificial Intelligence (Inteligência Artificial)
PLN	Processamento de Linguagem Natural

SUMÁRIO

1 INTRODUÇÃO

A operação de alguns sistemas empresariais tem se tornado gradualmente mais complexa, decorrente de diversos fatores, sendo os principais destacados pelas exigências governamentais, regras de negócio, legislação e obrigações legais. Podemos mencionar, também, as leis contábeis com regulamentações complexas que dependem de inúmeros fatores para serem aplicadas; essa complexidade é incorporada a esses sistemas. Nesse cenário, adicionam-se as regras e estratégias de negócio da empresa, nos mais diversos ramos, como comércio, cadeia de suprimentos, produtos e serviços, entre outros. Além disso, observa-se a presença de operações repetitivas, contribuindo para a sobrecarga do usuário e aumentando a probabilidade de ocorrência de erros.Todos esses fatores contribuem para um grande volume de dados, complexidade de operação e manutenção desses sistemas por parte dos usuários e operadores.

Atualmente, poucos sistemas empresariais fazem uso de Grandes Modelos de Linguagem (Large Language Model, LLM), evidenciando um vasto campo a ser explorado e muitas oportunidades a serem implementadas em sistemas existentes, uma vez que esses sistemas ainda executam eficientemente suas funções para as quais foram concebidos. Entretanto, observa-se que esses sistemas estão se tornando desatualizados e operacionalmente complexos, o que destaca a necessidade de modernização e simplificação em sua operação. A inteligência artificial e os LLM emergem como soluções promissoras para preencher essa lacuna, proporcionando uma abordagem inovadora e eficaz na interação entre usuários e sistemas, ao mesmo tempo em que melhoram a eficiência operacional e a experiência do usuário. Nesse contexto, os avanços recentes na área de Inteligência Artificial (AI), em particular no que diz respeito à criação e incorporação de LLM e agentes de (AI), têm o potencial de ajudar a contornar as dificuldades enfrentadas por usuários e mantenedores de sistemas.

O presente trabalho de conclusão de curso propõe-se a auxiliar os usuários de sistemas de uma maneira mais humanizada na operação desses sistemas, utilizando o processamento de linguagem natural (PLN) em LLM com integração da Interface de Programação de Aplicativos (API, Application Programming Interface). Em particular, destacando as etapas necessárias para a adaptação e potencial incorporação futura. O Processamento de Linguagem Natural (PLN) é uma subárea da Inteligência Artificial (AI), que possibilita aos computadores entenderem e interpretarem o que os humanos falam ou escrevem, permitindo um diálogo mais próximo do utilizado em uma conversa entre humanos. Dessa maneira, torna-se possível conversar com o sistema e expressar o que se deseja obter. Em muitos cenários de negócio, o usuário sabe a operação que precisa realizar, mas pode não saber exatamente como operar o sistema para alcançar

o resultado desejado. Pode ser que não conheça a complexidade e as regras envolvidas nessa solicitação. Neste cenário, o usuário informa ao LLM a operação desejada, o qual pode solicitar as informações necessárias e executar a operação diretamente após validar todas as informações. Quando não for possível executar a operação diretamente devido à natureza da mesma, o usuário será guiado sobre como proceder para alcançar o resultado desejado. Fluxos e regras complexas podem ser explicados à medida que o usuário avança nessa conversa.

1.1 Hipótese e Objetivos

Como hipótese, espera-se que a integração de LLMs com APIs permita que o sistema identifique de maneira adequada a intenção do usuário e os dados inseridos, para executar a ação desejada de forma eficiente. Essa abordagem deverá proporcionar uma interface mais próxima da linguagem humana, facilitando a interação e melhorando significativamente a experiência do usuário. Além disso, a utilização de LLMs e APIs contribuirá para a modernização e otimização do sistema.

Com o ajuste do modelo, buscamos aumentar a probabilidade de que ele seja capaz de identificar corretamente as ações desejadas pelo usuário. Para isso, utilizaremos três tipos de exemplos de textos em linguagem natural e três tipos de chamadas de APIs correspondentes como base: ClienteCadastrar, BoletoCadastrar e SaldoConsultar. A API ClienteCadastrar é utilizada para registrar novos clientes no sistema, fornecendo informações como nome, endereço e CPF. A API BoletoCadastrar é responsável por gerar e registrar boletos de pagamento, incluindo detalhes como valor e data de vencimento. Já a API SaldoConsultar permite verificar o saldo disponível em uma conta, retornando as informações necessárias para o usuário.

Incorporar ao LLM funcionalidades que permitam executar operações a partir de solicitações do usuário. Por exemplo, se o usuário fornecer a instrução: 'Por favor, cadastre o cliente José Silva, Av. Paulista, 123, CPF 123456789-00', espera-se que o sistema seja capaz de gerar um JSON correspondente à chamada da API 'CadastrarCliente', com os parâmetros adequados, como no seguinte formato: { 'api': 'CadastrarCliente', 'nome': 'José Silva', 'endereço': 'Av. Paulista, 123', 'cpf': '123456789-00' }. Esse processo permitirá que o modelo transforme automaticamente comandos em linguagem natural em solicitações no formato JSON para a chamada de API, facilitando a interação entre o usuário e o sistema.".

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é integrar LLMs com APIs, com a finalidade de melhorar a capacidade do sistema em identificar intenções dos usuários, facilitar operações e automatizar processos em uma aplicação já existente, proporcionando uma interface mais próxima da linguagem humana.

1.1.2 Objetivos Específicos

1. Desenvolver a integração de LLMs com APIs que possibilite a interação do usuário por meio de linguagem natural.

2. Ajustar o modelo LLM de tal forma que, a partir de uma entrada em linguagem natural fornecida pelo usuário, o modelo seja capaz de compreender e identificar a ação desejada pelo usuário, bem como extrair os parâmetros necessários para executar essa ação. Com base nessas informações, o modelo deverá ser capaz de gerar uma saída estruturada em formato JSON, que incluirá os parâmetros essenciais para a chamada da API correspondente a ação solicitada.

3. Validar a saída do modelo com a saída esperada a partir de métricas quantitativas de distância ou dissimilaridade.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Inteligência Artificial

De acordo com (Negnevitsky, 2005), a Inteligência Artificial é um campo da ciência da computação que se dedica ao desenvolvimento de sistemas capazes de realizar tarefas que normalmente exigiriam inteligência humana. Estes sistemas são projetados para aprender com dados, fazer inferências, resolver problemas complexos, reconhecer padrões e tomar decisões autônomas. A Inteligência Artificial abrange uma ampla gama de técnicas e algoritmos, incluindo aprendizado de máquina, lógica, raciocínio probabilístico e processamento de linguagem natural, com o objetivo de simular ou reproduzir comportamentos inteligentes.

2.2 Processamento Linguagem Natural - PLN

Segundo (Nadkarni; Ohno-Machado; Chapman, 2011) a PLN, iniciou na década de 1950, surgiu da interseção entre inteligência artificial e linguística. Inicialmente separada da recuperação de informações de texto, que usa estatísticas para indexar e pesquisar textos. Hoje, a PLN é influenciada por diversos campos, demandando que os pesquisadores ampliem seu conhecimento. Abordagens iniciais, como tradução palavra por palavra, enfrentaram desafios com palavras com múltiplos significados e metáforas. A análise de Chomsky sobre gramáticas linguísticas, em 1956, influenciou a criação da notação BNF em 1963, usada para especificar a sintaxe de linguagens de programação. Chomsky também identificou gramáticas "regulares", a base das expressões regulares usadas para padrões de pesquisa de texto. A sintaxe de expressão regular, definida por Kleene em 1956, foi implementada pela primeira vez no UNIX por Ken Thompson com o utilitário grep.

Na PLN, o objetivo é entender o significado do texto usando gramáticas formais que descrevem como as palavras se relacionam, como substantivos, verbos e adjetivos. As gramáticas podem ser expandidas para incluir o significado em linguagem natural.

2.3 LLM

Segundo (Ozdemir, 2023), os Grandes Modelos de Linguagem (LLM), são projetados para compreender a linguagem humana. Para alcançar esse objetivo, esses modelos são treinados com grandes volumes de texto, permitindo que capturem nuances e complexidades da linguagem humana. Essa abordagem de treinamento massivo capacita os LLMs a entenderem a linguagem humana de maneira mais eficaz, possibilitando uma ampla gama de aplicações, desde a geração de texto até a compreensão de perguntas e respostas.

2.4 API

API ou Interface de Programação de Aplicativos é definida por (Jin; Sahni; Shevat, 2018) como uma interface que um programa de software oferece para outros programas, através desta interface diferentes programas podem se conectar, trocar informaçoes, realizar operações e permitem a interoperabilidade para as principais plataformas de negócios na web. Em (Bratić *et al.*, 2024) uma API (interface de programação de aplicativos) é um conjunto de regras e especificações que determinam como diferentes softwares podem se comunicar entre si. Ela facilita o acesso de aplicativos a funções e dados de outros aplicativos, serviços ou plataformas.

2.5 JSON

Como descrito em (Marrs, 2017), o JSON (JavaScript Object Notation) é um formato de troca de dados leve e de fácil leitura, amplamente utilizado para a transmissão de dados entre sistemas. Ele é baseado em uma estrutura simples de chave-valor, onde cada chave é associada a um valor específico, permitindo a organização de dados de forma clara e hierárquica. Devido à sua simplicidade e flexibilidade, JSON se tornou um padrão na comunicação entre serviços web, APIs, e aplicações que precisam trocar informações de maneira eficiente. A estrutura do JSON é composta por objetos e arrays. Um objeto é definido por um conjunto de pares chave-valor, delimitados por chaves , enquanto um array é uma lista ordenada de valores, delimitados por colchetes []. Cada valor dentro de um objeto ou array pode ser um número, uma string, um booleano, outro objeto, um array, ou o valor nulo. Essa flexibilidade permite que JSON represente dados complexos de forma simples, compacta e fácil de interpretar tanto por humanos quanto por máquinas. Por exemplo, um JSON que represente as informações de um cliente pode ser estruturado da seguinte forma:

```
{
  "nome": "Joao Silva",
  "idade": 30,
  "email": "teste@email.com",
  "endereco" : "Av. Paulista, 123"
}
```

2.6 ChatBots e LLM

Segundo (Adamopoulou; Moussiades, 2020) os chatbots são programas de computador que podem simular conversas humanas e entreter os usuários, mas eles não são criados apenas com esse propósito. Eles têm uma variedade de aplicações úteis em áreas como educação, recuperação de informações, negócios e comércio eletrônico. Eles se tornaram

muito populares devido às muitas vantagens que oferecem tanto para os usuários quanto para os desenvolvedores. Além disso, o contato com um chatbot pode ser feito de forma integrada com o ambiente social do usuário, sem a necessidade de sair do aplicativo de mensagens onde o chatbot está inserido, o que garante a identidade do usuário.

Ainda segundo o autor, em 1950, Alan Turing levantou a questão de saber se um programa de computador seria capaz de interagir com um grupo de pessoas sem que elas percebessem que estavam conversando com uma entidade artificial. Esse conceito, conhecido como o teste de Turing, é amplamente considerado como o ponto inicial para o desenvolvimento dos chatbots.O primeiro chatbot conhecido foi o Eliza, desenvolvido em 1966, seguido pelo PARRY em 1972 e pelo ALICE em 1995. Desde então, assistentes pessoais virtuais como Siri, Cortana, Alexa, Assistant e Watson foram desenvolvidos. O interesse por chatbots cresceu significativamente após 2016, conforme evidenciado por pesquisas.

Na pesquisa de (Brachten; Kissmer; Stieglitz, 2021) os autores citam que embora haja um aumento significativo nas pesquisas sobre o uso de chatbots no setor privado, a aplicação desses assistentes no contexto empresarial ainda não foi adequadamente examinada. Essa observação está alinhada com a ideia de que, no ambiente empresarial, a adoção de novas tecnologias geralmente é mais lenta devido a interesses institucionais, em comparação com ambientes pessoais. Além disso, a decisão de implementar chatbots em um ambiente corporativo é considerada complexa, pois é vista como uma decisão de longo prazo, com o objetivo de aumentar as vendas e melhorar a eficiência. Estudos indicam que a interação diária com sistemas de conversação, como os chatbots, está se tornando cada vez mais comum no ambiente de trabalho.

Segundo (Kar; Haldar, 2016), os chatbots têm a capacidade de executar ações relevantes para o usuário, levando em conta suas preferências e o ambiente em que estão inseridos. Além disso, eles são eficazes na automação de tarefas repetitivas, utilizando chamadas de API, Websockets ou outros métodos.

Neste artigo, o autor descreve um cenário em que múltiplos agentes, incluindo agentes de (AI), colaboram com humanos para executar tarefas. Um agente de IA utiliza recursos avançados, como (LLMs), ou se comunica com outros sistemas externos para gerar uma solução inicial para a tarefa. Essa solução é então passada para o agente proxy, que pode solicitar entradas adicionais dos humanos ou executar o código fornecido pelo assistente. Os resultados são enviados de volta ao assistente como feedback, permitindo ajustes ou refinamentos adicionais, se necessário. Essa abordagem facilita uma colaboração eficaz entre agentes assistentes e humanos, melhorando a qualidade e a eficiência da execução da tarefa.

2.7 LLMs Existentes

2.7.1 Llamma3

Segundo a (Meta, 2024) o Llama3 é um LLM desenvolvido com foco em ser uma ferramenta de código aberto. Ele foi projetado para oferecer capacidades em termos de raciocínio, geração de texto e compreensão de linguagem natural. Este modelo vem em diferentes tamanhos, variando de 7 a 70 bilhões de parâmetros, permitindo uma ampla gama de aplicações, desde a pesquisa acadêmica até implementações empresariais complexas. Além disso, o Llama3 foi projetado para ser eficiente, com otimizações que permitem que ele seja implementado em diferentes infraestruturas, desde servidores de grande porte até dispositivos menores. Isso torna o modelo acessível e prático para desenvolvedores e empresas que buscam soluções de (AI). O Llama 3 também se destaca por seu desempenho, oferecendo uma capacidade de processamento aprimorada, que melhora significativamente a precisão e a rapidez na geração de respostas. Além disso, o modelo suporta um ajuste fino de instrução que permite personalizar seu comportamento para atender a necessidades específicas, aumentando sua versatilidade e aplicabilidade em diferentes contextos.

2.7.2 Gemma 7b

Como descrito pela (Google, 2024) o Gemma 7B é um LLM desenvolvido pela Google, sendo parte de uma nova família de modelos de código aberto. Projetado para ser leve e eficiente, o Gemma 7B se baseia na mesma pesquisa e tecnologia que deram origem aos modelos Gemini. Com 7 bilhões de parâmetros, ele é otimizado para fornecer desempenho, permitindo uma ampla gama de aplicações, desde o desenvolvimento de software até a integração em sistemas empresariais. Além disso, o modelo foi otimizado para funcionar de maneira eficiente em diversos frameworks, ferramentas e hardwares, garantindo flexibilidade e facilidade de uso em diferentes ambientes. O modelo também é personalizável, permitindo ajustes finos para tarefas específicas, o que amplia sua versatilidade e aplicabilidade em diferentes contextos.

2.7.3 Tinnyllama

O TinyLlama é um LLM leve e de código aberto desenvolvido por Jingyu Zhang (Zhang *et al.*, 2024), com foco na eficiência em dispositivos de recursos limitados. Este projeto, se destaca por sua capacidade de operar em ambientes com poder computacional restrito, sem comprometer a qualidade na geração de texto.Com um vocabulário de 32 mil tokens, o TinyLlama oferece uma excelente relação entre desempenho e economia de recursos, sendo ideal para desenvolvedores que buscam personalizações específicas em seus projetos de IA. Além disso, é um projeto open-source.

2.7.4 Gorilla LLM

Segundo o autor (Patil *et al.*, 2023) Gorilla é um modelo aprimorado baseado em LLaMA, projetado para superar as limitações dos Grandes Modelos de Linguagem (LLMs) ao realizar chamadas de API de maneira eficaz. Ele se destaca por sua capacidade de adaptabilidade às mudanças na documentação da API em tempo real e pela redução significativa do problema de alucinação comumente encontrado em modelos de LLMs. A avaliação do Gorilla é realizada utilizando o conjunto de dados APIBench, demonstrando sua eficácia na integração de sistemas de recuperação de documentos e seu potencial para aumentar a precisão e a confiabilidade no uso de ferramentas externas.

3 TRABALHOS RELACIONADOS

O estudo de (Bratić *et al.*, 2024) examina os desafios relacionados ao armazenamento de materiais educacionais em um banco de dados fragmentado e diversificado.Nesse artigo o autor propõe um modelo híbrido que combina a estrutura de (LLM) / chatbot já existente e chamadas de API. Essa combinação assegura respostas precisas, provenientes de um amplo banco de dados educacionais.

Ainda segundo o autor, os chatbots podem ser divididos em duas categorias principais: os baseados em regras e os baseados em inteligência artificial (IA). Os chatbots baseados em regras oferecem aos usuários opções específicas para escolher, geralmente sendo usados em tarefas simples, como responder a perguntas frequentes (FAQs). Por outro lado, os chatbots baseados em IA utilizam tecnologias como inteligência artificial, processamento de linguagem natural (PNL) e aprendizado de máquina (ML) para compreender as palavras-chave que os usuários usam durante a conversa. Esses chatbots são treinados ao longo do tempo para aprender quais respostas oferecer com base nas consultas dos usuários. Por fim, um chatbot híbrido combina características de ambos os tipos, mesclando abordagens baseadas em regras e em IA. O modelo híbrido se torna relevante, sendo desenvolvido para superar essa limitação ao gerenciar as solicitações do usuário de forma local. Ao combinar uma estrutura de Chatbot com LLM e chamadas de API, o modelo híbrido se sobressai no processamento das solicitações do usuário e na recuperação de informações pertinentes de um banco de dados de materiais educacionais fornecido pelo usuário.

No trabalho de (Kim *et al.*, 2023) os autores descrever a capacidade dos LLMs de integrar várias ferramentas e "chamada de função"(function calling) esta técnica, permite que LLMs invoquem APIs para automatizar a execução de tarefas específicas o que pode revolucionar a maneira como o software baseado em LLM é desenvolvido. No entanto, isso apresenta um desafio significativo: qual é a maneira mais eficaz de incorporar múltiplas chamadas de função? Uma abordagem notável (ReAct) foi introduzida o por (Yao *et al.*, 2022), onde o LLM faz uma chamada de função, analisa os resultados e então decide sobre a próxima ação, que geralmente envolve uma chamada de função subsequente. No entanto, os métodos atuais para chamadas de múltiplas funções geralmente requerem raciocínio e execução sequencial para cada função, o que pode resultar em alta latência, custos elevados e, por vezes, comportamento impreciso. O autor também apresenta o LLMCompiler, que executa funções em paralelo para orquestrar eficientemente chamadas de múltiplas funções. Inspirado nos princípios dos compiladores clássicos, o LLMCompiler simplifica a chamada de funções em paralelo com três componentes principais: (i) um Planejador LLM, que formula planos de execução; (ii) uma Unidade de Busca de Tarefas, que despacha tarefas de chamada de função; e (iii) um Executor, que executa essas tarefas em paralelo. O autor

em seu trabalho conclue que o ReAct é um método simples, porém eficaz, para integrar raciocínio e ação em grandes modelos de linguagem. Através de uma série diversificada de experimentos, incluindo respostas a perguntas multi-hop, verificação de fatos e tarefas interativas de tomada de decisão, os autores demonstraram que o ReAct alcança um desempenho superior com traços de decisão interpretáveis.

Gorilla é um LLM baseado em LLaMA que pode fornecer chamadas de API apropriadas. Os autores (Patil *et al.*, 2023) da Univercidade de Berkeley, mencionam em seus estudos que o Gorilla é capaz de superar o desempenho do GPT-4 (OpenAI, 2023), um modelo de inteligência artificial desenvolvido pela OpenAI. O autor apresenta estudos comparativos entre os modelos Gorilla, GPT-4 e Claude (Anthropic, 2023), evidenciando a capacidade do modelo Gorilla de identificar corretamente a tarefa e sugerir uma chamada de API mais apropriada. Foi utilizada a técnica de correspondência de subárvore AST (*Abstract Syntax Trees*) para avaliar a correção funcional das APIs geradas, onde o código é analisado em uma árvore AST e uma subárvore contendo a chamada de API é identificada para avaliação. A Gorilla demonstra capacidade de sugerir a API apropriada com base na consulta do usuário durante a inferência. Para compilar um conjunto de dados completo de APIs de ML, foram coletadas diversas chamadas de API do TensorFlow Hub, Torch Hub e HuggingFace. Dados sintéticos foram gerados usando o GPT-4, com exemplos de contextualizados e referências à documentação da API. Este processo resultou em pares instrução-API que fornecem uma base sólida para análises subsequentes. Da mesma forma, é importante que o Gorilla respeite restrições como precisão, número de parâmetros do modelo e consumo de recursos. O autores apresentam um estudo de para avaliar como diferentes modelos se comportam em respeito a uma restrição específica.

Os autores concluem que os LLMs estão rapidamente se tornando populares em diversos domínios. Em seus estudos, destacam técnicas projetadas para melhorar a precisão na identificação da API adequada por parte dos LLMs. Como as APIs funcionam como uma linguagem universal para a comunicação eficaz entre sistemas, seu uso correto pode aumentar a capacidade dos LLMs de interagir com diversas ferramentas. Propondo o Gorilla, como um novo pipeline para ajustar LLMs para invocar APIs, o Gorila gera chamadas de API confiáveis para modelos de ML, demonstrando capacidade de adaptação às mudanças de uso da API e considerando restrições ao selecionar APIs. O autor destaca as limitações das APIs focadas em aprendizado de máquina (ML), ressaltando uma desvantagem significativa, a propensão dessas APIs a gerar previsões enviesadas quando treinadas com conjuntos de dados distorcidos, o que pode acarretar impactos negativos.

4 METODOLOGIA

Neste capítulo, é apresentado a metodologia para integrar LLM com APIs. Para a experimentação foram selecionados três modelos de LLMs: tinyllama-bnb-4bit, gemma-7b-bnb-4bit, e llama-3-8b-bnb-4bit. Esses modelos foram escolhidos com base em suas especificações técnicas, como o tamanho do vocabulário, a quantidade de parâmetros, e a VRAM requerida, que variam de 0,631 bilhões a 8 bilhões de parâmetros e de 0,8 GB a 5,7 GB de tamanho de arquivo.

A manutenção e distribuição desses modelos é realizada pela startup Unsloth, utilizando como base os modelos treinados originalmente pela Google, Meta e pela Comunidade OpenSource. Na tabela 1 podemos ver as especificações técnicas dos modelos selecionados para o experimento.

Tabela 1 – Especificações Técnicas dos Modelos Selecionados

	tinyllama-bnb-4bit	**gemma-7b-bnb-4bit**	**llama-3-8b-bnb-4bit**
Modelo Base	Tinyllama OpenSource	Gemma-7b Google	Llama-3 Meta
Modelo Unsloth	tinyllama-bnb-4bit	gemma-7b-bnb-4bit	llama-3-8b-bnb-4bit
Licença	apache-2.0	apache-2.0	llama3 / Meta
Vocabulario	32.000	256.000	128.256
Vram Requerida	0.8Gb	5.6Gb	5.6Gb
Tamanho Arquivo	0.8Gb	5Gb	5.7Gb
Nº Parametros	0.631b	7b	8b

Os modelos considerados são quantizados para 4 bits, gerados a partir do modelo base. Esses modelos são distribuídos e mantidos pela Unsloth (UnslothAI, 2024), com o propósito de reduzir a quantidade de memória utilizada durante a inferência e treinamento. A quantização de um modelo de linguagem, como um LLM (Large Language Model), é uma técnica que consiste em reduzir a precisão dos números que representam os pesos do modelo, passando de 32 bits (padrão) para 4 bits. Essa redução diminui o consumo de memória e o uso da largura de banda, permitindo que o modelo seja executado em hardware menos potente sem perder significativamente a precisão ou a qualidade das respostas. Entre os benefícios da quantização estão a redução do tempo de inferência, a diminuição do tamanho do modelo e a viabilidade de execução em dispositivos com menos recursos, como GPUs com menor capacidade de memória. Além disso, a quantização permite economizar custos operacionais, pois modelos mais leves consomem menos energia e recursos computacionais. Essas vantagens influenciaram a escolha dessa técnica no desenvolvimento deste projeto.

4.1 Coleta dos dados

Para a criação do dataset, utilizou-se uma base de dados composta por 2 mil registros cadastrais de clientes que já utilizam o sistema BoletoFast[1]. Devido a questões de privacidade e confidencialidade, esses dados não serão disponibilizados.

4.2 Criação de Dataset

A partir da coleta de dados, foi criado um dataset que possui dois campos principais: "prompt de entrada do usuário"e "saída esperada em JSON". O objetivo é fornecer ao modelo um prompt de entrada em linguagem natural, a partir do qual o modelo deve gerar um JSON correspondente para a chamada da API. A partir da base de dados composta por 2 mil registros cadastrais de clientes, o objetivo seguinte foi associar esses registros a três ações específicas: "ConsultarSaldo", "CadastrarCliente"e "CadastrarBoleto". Para cada uma dessas ações, foram geradas diversas frases em linguagem natural que representam as intenções dos usuários. Para melhorar a eficácia do modelo em interpretar comandos em linguagem natural, foram criados conjuntos específicos de frases para cada ação selecionada. No caso da ação "CadastrarCliente", o modelo foi exposto a frases como "Inscreva", "Cadastre", "Registre o cliente", "Adicione o cliente", entre outras. Para "CadastrarBoleto", foram utilizadas expressões como "Registre Boleto", "Cadastre Boleto", "Inclua Boleto". Por fim, para "ConsultarSaldo", o modelo recebeu frases com termos como "Saldo do cliente", "Balance", "Extrato do cliente", "Qual é o saldo". Com essa variedade de expressões incluídas no dataset espera-se que o modelo desenvolva a capacidade de identificar corretamente as intenções e os parâmetros necessários para cada ação, gerando um JSON bem estruturado para a chamada das APIs correspondentes.Esse processo de criação do dataset visa aumentar o número de exemplos disponíveis, com o objetivo de melhorar a precisão do modelo em relação ao comportamento esperado em cenários de uso real. Ao incluir mais exemplos variados, busca-se aprimorar a capacidade do modelo de lidar com diferentes solicitações dos usuários de forma mais eficaz, aproximando-o dos resultados desejados no contexto prático.

Ao final desse processo foram totalizando 6.071 registros. Deste total de registros, 80% foi utilizado para treinamento e 20% para testes, visando que o modelo identifique corretamente a intenção e os parâmetros necessários para formar o JSON adequado.

[1] BoletoFast https://boletofast.com.br é um software desenvolvido e mantido pelo autor desta monografia.

4.3 Fine-Tuning

Fine-tuning (ajuste fino) é uma técnica de aprendizado de máquina utilizada para adaptar um modelo previamente treinado a um novo conjunto específico de dados (datasets) (Friederich, 2017). Em vez de treinar um modelo do zero, aproveita-se um modelo que já foi treinado em um grande volume de dados gerais. Esse modelo é então ajustado com dados mais específicos e relevantes ao problema que estamos tentando resolver. Esse processo não só economiza tempo e recursos computacionais, mas também melhora o desempenho do modelo em tarefas específicas, como a integração com APIs de sistemas existentes. Para realizar o Fine-Tuning, será utilizado o algoritmo desenvolvido pela Unsloth (UnslothAI, 2024), que facilita o ajuste fino desses modelos por meio de um código padronizado. O código-fonte é disponibilizado no GitHub (UnslothAI, 2024), permitindo que seja personalizado conforme as necessidades específicas do projeto. Na Figura 1 podemos ver o fluxo desse processo.

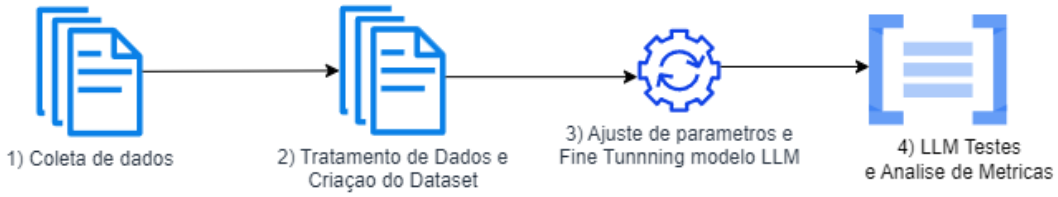

Figura 1 – Diagrama dos passos executados

4.4 Avaliação dos Resultados

Para avaliar os resultados do ajuste do LLM, utilizaremos duas métricas[2] principais. A primeira será a distância entre a string do prompt de entrada e a string de saída gerada pelo modelo, o que permitirá medir o quão próximo o modelo chega da transformação esperada das solicitações. A segunda métrica será a distância entre o JSON esperado e o JSON gerado pelo modelo, ajudando a avaliar a proximidade dos resultados com as expectativas. Essas duas métricas são utilizadas conjuntamente porque abordam diferentes aspectos, enquanto a distância entre as strings foca na precisão do conteúdo textual, avaliando diretamente quão bem o modelo gera texto alinhado com a saída esperada, a distância entre os JSONs se concentra na precisão do conteúdo textual e também na precisão de estar em conformidade com as especificações do formato JSON. Adicionalmente, a loss do modelo durante o treinamento e o tempo de resposta são monitorados para fornecer

[2] Para as metricas de distância entre a string e distância entre o JSON, foi utilizada a biblioteca em Python disponível em https://python.langchain.com/v0.1/docs/guides/productionization/evaluation/string/json/ para implementar essas métricas.

uma avaliação mais completa da performance operacional e da eficácia do aprendizado do modelo.

4.5 Resultados Esperados

Ao final do processo, espera-se que o usuário possa inserir uma solicitação em linguagem natural, como por exemplo "Por favor, cadastre no sistema o cliente José Silva, residente na Av. Paulista, 123 com o CPF 123456789-00."O LLM deve ser capaz de identificar automaticamente a ação desejada (neste caso, "CadastrarCliente") e extrair os parâmetros necessários, como nome, endereço e CPF, da solicitação. Com base nessa identificação, o modelo deve então gerar uma resposta em formato JSON, estruturada corretamente para a chamada da API correspondente. Por exemplo, o JSON retornado deve incluir todos os campos necessários, como {"api": "CadastrarCliente", "nome": "José Silva", "endereco": "Av. Paulista, 123", "cpf": "123456789-00"}. A precisão deste processo é avaliada pelas métricas discutidas anteriormente, assegurando que o modelo, mesmo em sua fase experimental, esteja próximo de produzir respostas adequadas e utilizáveis em um contexto real. Como mostrado na Figura 2, esse fluxo é demonstrado, ilustrando como o LLM processa a entrada, identifica os parâmetros e forma o JSON adequado para a API.

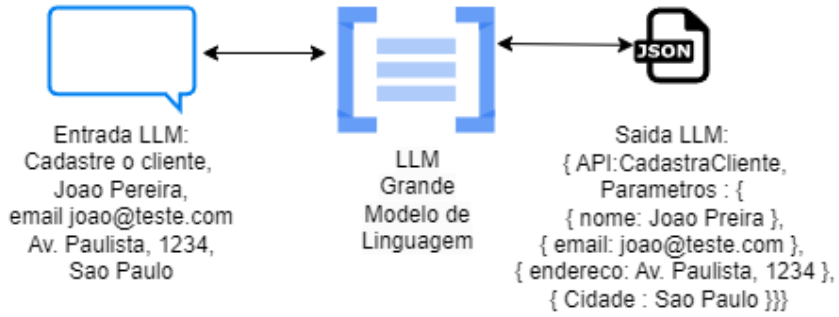

Figura 2 – LLM processa a entrada, identifica a ação, os parâmetros e forma o JSON adequado para chamada da API

5 AVALIAÇÃO EXPERIMENTAL

5.1 Fine Tunning

Como mencionado na Seção 4.3 para o processo de Fine-Tuning, foi utilizado o algoritmo proposto pela Unsloth, que, por meio de um código padronizado, permite o Fine-Tuning desses modelos. O código-fonte está disponível no GitHub (UnslothAI, 2024), permitindo personalização conforme as necessidades do projeto.

Além do algoritmo proposto para o Fine-Tuning, a escolha dos parâmetros é essencial para descobrir as melhores combinações que otimizem o desempenho do modelo. Parâmetros como o número de épocas e a taxa de aprendizado (*Learning Rate*) têm um impacto significativo no processo de treinamento. O número de épocas determina quantas vezes o modelo passará pelos dados de treinamento (Goodfellow, 2016), o que pode influenciar a qualidade da resposta do modelo. Um número muito baixo de épocas pode resultar em um modelo subajustado , enquanto um número muito alto pode causar sobreajuste. A taxa de aprendizado, por sua vez, controla o tamanho dos passos que o algoritmo de otimização dá na busca por melhores pesos para o modelo. Taxas de aprendizado muito altas podem fazer com que o modelo perca padrões importantes nos dados, enquanto taxas muito baixas podem levar a um treinamento excessivamente longo ou a um modelo preso em mínimos locais.

Para este experimento inicial, foram escolhidas três variações de épocas (30, 60 e 90) e três variações de Learning Rate (1e-5, 2e-4, 5e-4), totalizando seis combinações de parametrizações para cada modelo e um total de 18 variações nesta fase do experimento. Na Tabela 2, podemos ver os parâmetros utilizados.

Tabela 2 – Combinações de Parâmetros Utilizados nos Experimentos

Modelo	**Épocas**	**Learning Rate**
llama-3-8b-bnb-4bit	30	1e-5
		2e-4
		5e-4
	60	1e-5
		2e-4
		5e-4
gemma-7b-bnb-4bit	30	1e-5
		2e-4
		5e-4
	60	1e-5
		2e-4
		5e-4
tinyllama-bnb-4bit	30	1e-5
		2e-4
		5e-4
	60	1e-5
		2e-4
		5e-4

5.2 Execução do Fine Tunning

Todo o processo de Fine-Tuning foi executado no Google Colab PRO, utilizando instâncias idênticas para garantir uma análise mais precisa dos dados. As principais características das instâncias utilizadas foram: 13 GB de RAM do sistema, 15 GB de RAM em uma GPU Tesla T4, e 80 GB de espaço em disco. Na tabela 3, são apresentados os dados referentes ao Fine-Tuning inicial das 18 variações apresentadas na Tabela 2, onde estão listados o Loss Inicial, o Loss Final, e o tempo de treinamento para cada variação. Esses dados são essenciais para avaliar a eficiência e a performance de cada configuração durante o processo de ajuste fino. A Loss Inicial representa o erro do modelo na primeira época do processo de ajuste fino, enquanto a Loss Final reflete o erro na última época do mesmo, ambos medidos utilizando a Cross Entropy Loss como métrica de erro. Comparando os resultados da Tabela 3, observa-se que a configuração com tinyllama-bnb-4bit, 60 épocas e uma Learning Rate de 5e-4 apresentou a menor Loss Final (0.5555), indicando uma melhor adaptação ao ajuste fino. Por outro lado, a configuração com llama-3-8b-bnb-4bit, 60 épocas e Learning Rate de 1e-5 apresentou uma Loss Final maior (2.3169), sugerindo uma menor eficácia nesse cenário específico. Porém, uma análise da redução de loss revela que a configuração que demonstrou a maior taxa de aprendizado foi a do modelo llama-3-8b-bnb-4bit, com 60 épocas e uma learning rate de 5e-4, que alcançou uma redução de loss de 2.3625. Estes resultados ajudam a identificar quais combinações de parâmetros otimizam o desempenho do modelo durante o processo de Fine-Tuning. Na Tabela 3, podemos ver esses resultados.

Tabela 3 – Resultados da Execução do Fine-Tuning Inicial

Modelo	**Épocas**	**Learning Rate**	**Loss Inicial**	**Loss Final**	**Redução de Loss**
llama-3-8b-bnb-4bit	30	1e-5	3.0198	2.5900	0.4298
		2e-4	3.0198	0.9020	2.1178
		5e-4	3.0198	0.7876	2.2322
	60	1e-5	3.0198	2.3169	0.7029
		2e-4	3.0198	0.6959	2.3239
		5e-4	3.0198	0.6573	2.3625
gemma-7b-bnb-4bit	30	1e-5	2.7780	1.9186	0.8594
		2e-4	2.7780	0.7421	2.0359
		5e-4	2.7780	0.7304	2.0476
	60	1e-5	2.7780	1.2260	1.5520
		2e-4	2.7780	0.6016	2.1764
		5e-4	2.7780	0.5977	2.1803
tinyllama-bnb-4bit	30	1e-5	2.4316	2.2381	0.1935
		2e-4	2.4316	0.8786	1.5530
		5e-4	2.4316	0.7040	1.7276
	60	1e-5	2.4316	2.2154	0.2162
		2e-4	2.4316	0.6421	1.7895
		5e-4	2.4316	0.5555	1.8761

Uma análise realizada em sequência foi a da distância de similaridade entre strings, uma métrica utilizada para avaliar a precisão das saídas geradas pelos modelos. A distância de similaridade entre strings é uma métrica utilizada para quantificar o grau de semelhança entre duas sequências de caracteres, ou "strings". A distância de similaridade pode ser usada para avaliar quão semelhante uma string gerada por um modelo está em relação a uma string de referência ou esperada. O valor da distância varia entre 0 e 1, onde 0 indica que as strings comparadas são idênticas, e 1 indica que elas são completamente diferentes. Neste trabalho, utizamos inicialmente essa métrica para avaliar a performance dos modelos durante o processo de Fine-Tuning.

Nos dados apresentados na Tabela 4, podemos observar uma evolução significativa na performance dos modelos após o processo de Fine-Tuning. Notamos que a "distância média string", ou seja, a média das distâncias obtidas para as tuplas (json esperado, json obtido), foi consideravelmente reduzida em quase todas as configurações, evidenciando a eficácia do ajuste fino em melhorar a precisão dos modelos na geração de saídas esperadas. Por exemplo, no modelo llama-3-8b-bnb-4bit com 60 épocas e uma learning rate de 5e-4, a distância média foi reduzida de 0,8746 para 0,00253, mostrando uma significativa melhoria. De maneira semelhante, o modelo gemma-7b-bnb-4bit com 60 épocas e uma learning rate de 5e-4 também apresentou uma redução, passando de 0,8250 para 0,00654. Além disso, o modelo tinyllama-bnb-4bit com 60 épocas e uma learning rate de 5e-4 reduziu a distância de 0,8275 para 0,01241. Esses resultados demonstram que o Fine-Tuning foi particularmente eficaz nessas configurações, resultando em saídas mais próximas do

esperado. Segundo esse critério, a melhor configuração foi o modelo llama-3-8b-bnb-4bit com 60 épocas e Learning Rate de 5e-4.

Tabela 4 – Média da Distância String Antes e Após o Fine-Tuning

Modelo	**Épocas**	**Learning Rate**	**Média Antes**	**Média Após**
llama-3-8b-bnb-4bit	30	1e-5	0,8746	0,85627
		2e-4	0,8746	0,01255
		5e-4	0,8746	0,00891
	60	1e-5	0,8746	0,84524
		2e-4	0,8746	0,09707
		5e-4	0,8746	**0,00253**
gemma-7b-bnb-4bit	30	1e-5	0,8250	0,84755
		2e-4	0,8250	0,00857
		5e-4	0,8250	0,00597
	60	1e-5	0,8250	0,42922
		2e-4	0,8250	0,00667
		5e-4	0,8250	**0,00654**
tinyllama-bnb-4bit	30	1e-5	0,8275	0,85161
		2e-4	0,8275	0,08218
		5e-4	0,8275	0,02543
	60	1e-5	0,8275	0,89350
		2e-4	0,8275	0,02180
		5e-4	0,8275	**0,01241**

Outra análise realizada foi a distância de similaridade entre objetos JSON contidos nas strings geradas pelos modelos. Essa métrica é particularmente útil para avaliar não apenas a precisão textual, mas também a exatidão estrutural e semântica das saídas geradas. Ao comparar os objetos JSON, podemos quantificar o quão próxima a estrutura gerada pelo modelo está em relação à estrutura esperada, levando em conta tanto a presença dos campos corretos quanto os valores atribuídos a eles. O valor dessa distância, assim como na comparação entre strings, varia entre 0 e 1, onde 0 indica que os objetos JSON comparados são idênticos em estrutura e conteúdo, enquanto 1 indica que eles são completamente diferentes. Essa métrica permite uma avaliação mais robusta do desempenho do modelo, já que considera a correta formação e estruturação dos dados em formato JSON, o que é muito importante para aplicações que dependem da precisão na comunicação de dados entre sistemas.

Na Tabela 5, podemos observar os resultados das distâncias JSON após o Fine-Tuning para diferentes modelos e configurações. Primeiramente, é importante destacar as três configurações que apresentaram a pior performance em termos de distância JSON. O modelo tinyllama-bnb-4bit com 30 épocas e uma learning rate de 1e-5 apresentou uma das maiores distâncias, com um valor de 0,99852, indicando uma grande discrepância entre o JSON gerado e o esperado. Outra configuração que apresentou um desempenho fraco foi o modelo llama-3-8b-bnb-4bit com 30 épocas e uma learning rate de 1e-5, que também obteve uma alta distância JSON de 0,99835. Por fim, o modelo gemma-7b-bnb-4bit com 30 épocas

e uma learning rate de 1e-5 resultou em uma distância de 0,97571, mostrando dificuldades semelhantes na geração precisa do JSON. Em contrapartida, as três configurações que demonstraram a melhor performance foram significativamente mais eficazes. O modelo llama-3-8b-bnb-4bit com 60 épocas e uma learning rate de 5e-4 alcançou uma das menores distâncias JSON, com um valor de 0,00312, indicando uma excelente proximidade entre o JSON gerado e o esperado. De forma semelhante, o modelo gemma-7b-bnb-4bit com 60 épocas e uma learning rate de 5e-4 apresentou uma distância muito baixa, de 0,00549, evidenciando uma melhoria considerável em comparação com suas outras configurações. Finalmente, o modelo tinyllama-bnb-4bit com 60 épocas e uma learning rate de 5e-4 também obteve um bom resultado, com uma distância JSON de 0,00997.

Tabela 5 – Distância JSON Após o Fine-Tuning

Modelo	**Épocas**	**Learning Rate**	**Distância JSON**
llama-3-8b-bnb-4bit	30	1e-5	0,99835
		2e-4	0,00944
		5e-4	0,00526
	60	1e-5	0,98553
		2e-4	0,08707
		5e-4	**0,00312**
gemma-7b-bnb-4bit	30	1e-5	0,97571
		2e-4	0,00636
		5e-4	0,00487
	60	1e-5	0,51458
		2e-4	0,00167
		5e-4	**0,00549**
tinyllama-bnb-4bit	30	1e-5	0,99852
		2e-4	0,19677
		5e-4	0,02373
	60	1e-5	0,95654
		2e-4	0,14803
		5e-4	**0,00997**

Ao comparar as configurações com melhor e pior desempenho, ficou evidente que o aumento no número de épocas e o ajuste da learning rate para valores maiores, como 5e-4, foram fatores decisivos para a melhora na precisão dos modelos. As configurações com uma learning rate muito baixa (1e-5) e menos épocas não conseguiram convergir de forma eficaz, resultando em distâncias JSON mais elevadas, enquanto as configurações com mais épocas e uma learning rate mais elevada (5e-4), foram capazes de minimizar as discrepâncias na geração dos objetos JSON. Segundo esse critério, a melhor configuração foi o modelo llama-3-8b-bnb-4bit com 60 épocas e Learning Rate de 5e-4.

5.3 Discussão

Os resultados obtidos a partir da integração dos LLMs com APIs revelam o potencial significativo desta abordagem para simplificar a interação do usuário com sistemas empresariais. A utilização de LLMs para interpretar comandos em linguagem natural e convertê-los em chamadas estruturadas de API mostrou-se eficaz, com os modelos ajustados atingindo níveis satisfatórios de precisão tanto na identificação de intenções quanto na geração de JSONs. Em particular, para as análises efetuadas e dataset utilizado, o modelo llama-3-8b-bnb-4bit com 60 épocas e Learning Rate de 5e-4, mostrou-se superior para essa tarefa. No entanto, a discussão dos resultados também levanta algumas questões importantes. Primeiro, a dependência do Fine-Tuning para alcançar a precisão desejada sugere que, apesar dos avanços, os LLMs ainda exigem ajustes específicos ao contexto de aplicação para funcionarem de maneira correta. Isso pode implicar em desafios adicionais quando se trata de adaptar esses modelos para diferentes sistemas ou contextos empresariais sem uma fase extensiva de ajuste fino.

Além disso, a análise das métricas de distância entre strings e JSONs evidenciou que, embora os modelos tenham atingido altos níveis de precisão, ainda existem cenários onde a formação de JSONs não é perfeita. Isso sugere a necessidade de estratégias adicionais, como a implementação de técnicas de pós-processamento para corrigir ou ajustar saídas que não estejam em conformidade com o esperado. Por fim, a aplicação prática desta solução em um ambiente empresarial real ainda requer mais estudos. A transição do ambiente experimental para o ambiente de produção pode introduzir novos desafios, como a necessidade de escalabilidade, a gestão de diferentes contextos de uso e a adaptação a mudanças nas APIs ou nos requisitos do sistema.

6 CONCLUSÕES

Neste trabalho, foi proposta a integração de Large Language Models (LLMs) com APIs, visando a modernização e simplificação das interações entre o usuário e o sistema. Através do ajuste fino (Fine-Tuning) de modelos como tinyllama-bnb-4bit, gemma-7b-bnb-4bit, e llama-3-8b-bnb-4bit, foi possível observar uma melhoria significativa na capacidade dos modelos em transformar comandos em linguagem natural em requisições estruturadas em JSON, prontas para a chamada de APIs.

Os resultados experimentais demonstraram que o processo de Fine-Tuning, especialmente com a variação de parâmetros como o número de épocas e a taxa de aprendizado, foi eficaz em reduzir a discrepância entre as saídas esperadas e as geradas pelo modelo. A melhoria nas métricas de distância entre strings e na precisão da formação de JSONs indicam que o modelo ajustado pode ser uma ferramenta valiosa na automação de processos empresariais, proporcionando uma interface mais intuitiva e próxima da linguagem natural humana.

6.1 Trabalhos Futuros

Para trabalhos futuros, é sugerido algumas melhorias que podem ser implementadas com base nas observações dos resultados atuais. Primeiramente, um aumento no número de épocas pode ser explorado, uma vez que o incremento de épocas demonstrou ser eficaz na melhoria da precisão na geração dos objetos JSON. Esse aumento no número de épocas pode permitir que os modelos tenham mais tempo para ajustar seus parâmetros e melhorar a convergência, resultando em distâncias JSON ainda menores. Além disso, seria interessante testar uma variação ainda maior na learning rate, mantendo os valores como 5e-4, mas também experimentando valores intermediários, como 3e-4 e 4e-4, para encontrar um equilíbrio ideal entre velocidade de aprendizagem e precisão. Outra área de melhoria seria a implementação de técnicas de regularização, como dropout, bayesian optimization ou weight decay, que poderiam ajudar a evitar overfitting, especialmente em configurações com mais épocas.

REFERÊNCIAS

ADAMOPOULOU, E.; MOUSSIADES, L. An overview of chatbot technology. *In*: SPRINGER. **IFIP international conference on artificial intelligence applications and innovations**. [*S.l.: s.n.*], 2020. p. 373–383.

Anthropic. **Claude**. 2023. https://www.anthropic.com/claude. Accessed: 2023-09-01.

BRACHTEN, F.; KISSMER, T.; STIEGLITZ, S. The acceptance of chatbots in an enterprise context–a survey study. **International Journal of Information Management**, Elsevier, v. 60, p. 102375, 2021.

BRATIĆ, D. *et al.* Centralized database access: Transformer framework and llm/chatbot integration-based hybrid model. **Applied System Innovation**, Multidisciplinary Digital Publishing Institute, v. 7, n. 1, p. 17, 2024.

FRIEDERICH, S. Fine-tuning. **The Stanford encyclopedia of philosophy**, 2017.

GOODFELLOW, I. **Deep learning**. [*S.l.: s.n.*]: MIT press, 2016. v. 196.

GOOGLE. **Gemma: Google introduces new state-of-the-art open models**. 2024. Acessado em: Agosto, 2024. Disponível em: https://blog.google/technology/developers/gemma-open-models/.

JIN, B.; SAHNI, S.; SHEVAT, A. **Designing Web APIs: Building APIs That Developers Love**. [*S.l.: s.n.*]: "O'Reilly Media, Inc.", 2018.

KAR, R.; HALDAR, R. Applying chatbots to the internet of things: Opportunities and architectural elements. **arXiv preprint arXiv:1611.03799**, 2016.

KIM, S. *et al.* An llm compiler for parallel function calling. **arXiv preprint arXiv:2312.04511**, 2023.

MARRS, T. **JSON at work: practical data integration for the web**. [*S.l.: s.n.*]: "O'Reilly Media, Inc.", 2017.

META. **Apresentando Meta Llama 3: o grande modelo de linguagem de código aberto mais capaz até hoje**. 2024. Acessado em: Agosto, 2024. Disponível em: https://about.fb.com/br/news/2024/04/apresentando-meta-llama-3-o-grande-modelo-de-linguagem-de-codigo-aberto-mais-capaz-ate-hoje/.

NADKARNI, P. M.; OHNO-MACHADO, L.; CHAPMAN, W. W. Natural language processing: an introduction. **Journal of the American Medical Informatics Association**, BMJ Group BMA House, Tavistock Square, London, WC1H 9JR, v. 18, n. 5, p. 544–551, 2011.

NEGNEVITSKY, M. **Artificial intelligence: a guide to intelligent systems**. [*S.l.: s.n.*]: Pearson education, 2005.

OpenAI. **GPT-4**. 2023. https://openai.com/index/gpt-4/. Accessed: 2023-09-01.

OZDEMIR, S. **Quick Start Guide to Large Language Models: Strategies and Best Practices for Using ChatGPT and Other LLMs**. [*S.l.: s.n.*]: Addison-Wesley Professional, 2023.

PATIL, S. G. *et al.* Gorilla: Large language model connected with massive apis. **arXiv preprint arXiv:2305.15334**, 2023.

UNSLOTHAI. **Unsloth: Finetune Llama 3.1, Mistral, Phi & Gemma LLMs 2-5x faster with 80% less memory**. 2024. Acessado em: Agosto 22, 2024. Disponível em: https://github.com/unslothai/unsloth.

YAO, S. *et al.* React: Synergizing reasoning and acting in language models. **arXiv preprint arXiv:2210.03629**, 2022.

ZHANG, P. *et al.* **TinyLlama: An Open-Source Small Language Model**. 2024.