

UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO  
INSTITUTO MULTIDISCIPLINAR

FELLIPE BRAVO RIBEIRO PIMENTEL  
GABRIEL DE OLIVEIRA SEGOBIA

**AILFRED: Um assistente virtual  
utilizando Large Language Model**

Prof. Filipe Braidão do Carmo, D.Sc.  
Orientador

Nova Iguaçu, Julho de 2024

# AILFRED: Um assistente virtual utilizando Large Language Model

Fellipe Bravo Ribeiro Pimentel

Gabriel de Oliveira Segobia

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto Multidisciplinar da Universidade Federal Rural do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Apresentado por:

---

Fellipe Bravo Ribeiro Pimentel

---

Gabriel de Oliveira Segobia

Aprovado por:

---

Prof. Filipe Braida do Carmo, D.Sc.

---

Prof. Bruno José Dembogurski, D.Sc.

---

Prof. Fellipe Ribeiro Duarte, D.Sc.

NOVA IGUAÇU, RJ - BRASIL

Julho de 2024



**DOCUMENTOS COMPROBATÓRIOS Nº 11978/2024 - CoordCGCC (12.28.01.00.00.98)**

**(Nº do Protocolo: NÃO PROTOCOLADO)**

**(Assinado digitalmente em 29/07/2024 11:42 )**

**BRUNO JOSE DEMBOGURSKI**  
PROFESSOR DO MAGISTERIO SUPERIOR  
DeptCC/IM (12.28.01.00.00.83)  
Matrícula: ###249#4

**(Assinado digitalmente em 26/07/2024 17:49 )**

**FELLIPE RIBEIRO DUARTE**  
PROFESSOR DO MAGISTERIO SUPERIOR  
DeptCC/IM (12.28.01.00.00.83)  
Matrícula: ###890#4

**(Assinado digitalmente em 26/07/2024 16:12 )**

**FILIFE BRAIDA DO CARMO**  
PROFESSOR DO MAGISTERIO SUPERIOR  
DeptCC/IM (12.28.01.00.00.83)  
Matrícula: ###295#4

**(Assinado digitalmente em 26/07/2024 15:31 )**

**GABRIEL DE OLIVEIRA SEGOBIA**  
DISCENTE  
Matrícula: 2024#####5

**(Assinado digitalmente em 26/07/2024 15:00 )**

**FELLIPE BRAVO RIBEIRO PIMENTEL**  
DISCENTE  
Matrícula: 2024#####2

Visualize o documento original em <https://sipac.ufrrj.br/documentos/> informando seu número: **11978**, ano: **2024**,  
tipo: **DOCUMENTOS COMPROBATÓRIOS**, data de emissão: **26/07/2024** e o código de verificação: **d7aa1cbdac**

# Agradecimentos

Fellipe Bravo Ribeiro Pimentel

Primeiramente aos meus pais, Carlos Pimentel e Cláudia Bravo, que me moldaram para enfrentar esse mundo-cão em que vivemos. Me ensinaram a ser forte, otimista e ambicioso. Esse documento é o fruto do trabalho árduo de vocês em me dar educação de qualidade e em nenhum momento deixarem de acreditar em mim.

Às minhas irmãs, Thais Chiodo e Bruna Bravo. Thais por ser meu maior exemplo de luta e dedicação e Bruna por ter me ensinado a ser tão responsável desde pequeno.

À minha esposa, Carolina Baptista, por estar ao meu lado sempre como meu contraponto, me fazendo enxergar a vida como realmente é, me fazendo sonhar de maneira lúcida. Você é o carinho que eu preciso no meu dia e a voz que acalma as minhas decisões. Eu agradeço por todo apoio que você me dá e, principalmente, por você estar com a nossa filha agora para que eu possa escrever o TCC com foco. E pensar que quando comecei isso há anos atrás esse texto de agradecimento era "por você estar jogando nosso PS5 agora, me forçando a escrever esse TCC".

À minha filha, Luna Baptista Bravo. Você é o motivo principal de eu não ter desistido. Quero me formar para te mostrar o quão importante é estudar, seguir caminhos corretos na vida e tomar boas decisões mesmo que tardias.

Ao nosso orientador Filipe Braida, que aceitou nos orientar mesmo sabendo das nossas dificuldades de tempo devido ao trabalho e nos mantendo confiantes de que esse sonho seria possível.

Aos meus grandes amigos que a Rural apresentou, Gabriel Segobia, Paulo Xavier,

Reinaldo Moraes e Angelo Rodney, que estão sempre dispostos a ajudar e abertos à discussões sobre a vida ou sobre computação. Dois agradecimento especiais ao Reinaldo e ao Angelo: Reinaldo, obrigado por me deixar colocar algumas teorias à prova em nosso projeto paralelo. Angelo, obrigado por todos os dias que fomos ao shopping comer hambúrguer e chorar as pitangas, você foi o ombro amigo que eu precisei no final da faculdade prestes a desistir de tudo.

A todos os que passaram de forma momentânea em minha vida e me fizeram evoluir espiritualmente, enxergando o que fazer, o que não fazer e principalmente em quem confiar.

Agradecimentos também aos professores, Bruno Dembogurski e Fellipe Duarte, que estão em nossa banca e ao corpo docente de Ciência da Computação da Rural, que me ajudaram durante todo esse processo doloroso que foi chegar aqui. Obrigado por cada nota baixa que recebi, elas me fizeram estudar cada vez mais e reforçaram a ideia de que somos muito mais do que números inteiros ou em ponto flutuante.

Obrigado, nós conseguimos.

## Gabriel de Oliveira Segobia

Primeiramente, agradeço imensamente à minha mãe Adriana, por ter sempre sido sempre meu maior exemplo, fonte de inspiração e apoio, e por ter sempre guiado para fazer a coisa certa. Ao meu pai Waldemiro, que por mesmo sem entender muito o que eu fazia nunca deixou de me dar suporte.

À minha filha Júlia, por ter mudado minha vida de uma forma que eu jamais imaginaria e fazer com que tudo tenha sentido.

Ao meu amor Rayane, quase esposa, que me aturou por todos esses anos surtando com o TCC, desistindo e voltando, mas que em nenhum momento deixou de me apoiar, mesmo nas últimas semanas em que nem eu me aguentava mais. Muito obrigado por ter ficado do meu lado nas piores crises de ansiedade que esse TCC me trouxe, por ter sempre me dito que era possível e por sempre estar do meu lado, muito obrigado por tudo.

Aos meus avós, Darcilene e Hélio por sempre terem me incentivado e sido tão presentes na minha vida, sei que meu avô estaria muito orgulhoso de presenciar esse momento.

Aos meus cunhados, Yasmin e Antonio, por terem me ajudado a passar pelo caos que foi a pandemia e são pessoas que eu sei que sempre posso confiar, muito obrigado por cuidarem de mim.

Aos meus amigos, em especial ao Fellipe e ao Paulo. O Fellipe por ter entrado nessa loucura de terminar o TCC comigo e ter me incentivado a continuar por vários momentos. E ao Paulo, por todos os momentos vividos na faculdade e por ter me obrigado a tentar uma última vez quando eu já tinha desistido, sem você esse momento não aconteceria.

Aos amigos que São Paulo me trouxe, principalmente ao Michel e a Ana por terem feito absolutamente tudo para que a gente não abandonasse o TCC, fomos péssimos em seguir o planejamento, mas vocês nunca desistiram da gente, especialmente o Michel que depois de todos esse anos ainda se ofereceu e revisou o texto. Um muito obrigado também ao Cassiano, Débora e Wesley, sem vocês eu nunca teria sobrevivido

a loucura que foram aqueles anos em São Paulo.

Aos amigos que a Rural me trouxe, alguns ainda próximos, outros nem tanto, todos vocês foram muito importantes nessa jornada.

À todos os professores que eu tive o prazer de conhecer nos meus anos de Rural, muito do que vocês me ensinaram impactou diretamente na pessoa que eu sou hoje, mesmo que na época eu não entendesse o sentido de algumas coisas, depois de alguns anos certas coisas passam a fazer mais sentido.

Um agradecimento especial aos professores que fazem parte da banca, Bruno Dembogurski e Fellipe Duarte, não apenas por terem aceitado fazer parte dessa última etapa, mas por todo o conhecimento que vocês compartilharam em todos esses anos.

Por fim, gostaria de agradecer ao nosso orientador, Filipe Braida, por toda a paciência nesses 6 longos anos, por nunca ter desistido da gente e ter acreditado até mesmo quando a gente já não acreditava mais. Você foi um dos melhores professores que eu já tive e um dos que mais me inspira até hoje, muito obrigado por ter me ensinado a escrever minha primeira linha de código e agora quase 11 anos depois estar aqui me ajudando a fechar esse ciclo.

Por último, algumas menções honrosas, à minha psicóloga Dehbora Scoralich que disse que só me daria alta depois que eu acabasse o TCC, mas dado o estrago feito por ele nas últimas semanas vai precisar me aturar por mais um tempo e a Marina Rocha que me ajudou muito nesse final a me organizar e continuar escrevendo, eu não teria conseguido sem vocês.

Todos vocês foram essenciais para que eu me tornasse a pessoa que eu sou hoje, muito obrigado de verdade.

Dito isso, não esperem nenhum outro trabalho acadêmico meu pela próxima década. Tchau, obrigado!

## RESUMO

AILFRED: Um assistente virtual utilizando Large Language Model

Fellipe Bravo Ribeiro Pimentel e Gabriel de Oliveira Segobia

Julho/2024

Orientador: Filipe Braida do Carmo, D.Sc.

DCC - UFRRJ

Este trabalho apresenta o desenvolvimento de *Alfred*, um assistente virtual que utiliza Large Language Models (LLMs) para fornecer respostas contextuais e precisas aos usuários. O problema abordado é a necessidade de um assistente que possa processar e interpretar grandes volumes de dados textuais para fornecer respostas relevantes em diversas áreas de conhecimento. A proposta envolve a utilização de modelos de linguagem avançados, como o *gpt-3.5-turbo*, para categorizar e processar as solicitações dos usuários, enriquecendo o contexto das perguntas e utilizando o histórico de interações para refinar as respostas. A eficácia do *Alfred* foi avaliada comparando seu desempenho com outras soluções existentes na literatura, demonstrando superioridade em termos de precisão e relevância das respostas fornecidas.



## ABSTRACT

AILFRED: Um assistente virtual utilizando Large Language Model

Fellipe Bravo Ribeiro Pimentel and Gabriel de Oliveira Segobia

Julho/2024

Advisor: Filipe Braida do Carmo, D.Sc.

DCC - UFRRJ

*This paper presents the development of Ailfred, a virtual assistant that utilizes Large Language Models (LLMs) to provide contextual and accurate responses to users. The addressed problem is the need for an assistant capable of processing and interpreting large volumes of textual data to deliver relevant answers across various fields of knowledge. The proposed solution involves using advanced language models, such as gpt-3.5-turbo, to categorize and process user requests, enriching the context of questions and leveraging interaction history to refine responses. Ailfred's effectiveness was evaluated by comparing its performance with other existing solutions in the literature, demonstrating superiority in terms of accuracy and relevance of the provided answers.*

# Lista de Figuras

Figura 2.1: Exemplo de frase para Tokenização . . . . .	7
Figura 2.2: Resultado de Tokenização por palavras da Figura 2.1 . . . . .	8
Figura 2.3: Resultado de Tokenização por caracteres da Figura 2.1 . . . . .	8
Figura 2.4: Frase de exemplo para <i>Part-of-Speech (POS) Tagging</i> . . . . .	8
Figura 2.5: Resultado POS <i>Tagging</i> da Figura 2.4 . . . . .	9
Figura 2.6: Frase de exemplo para <i>Named Entity Recognition (NER)</i> . . . . .	9
Figura 2.7: Resultado NER da Figura 2.6 . . . . .	10
Figura 2.8: Exemplo de frase para Análise Sintática . . . . .	10
Figura 2.9: Exemplo de Árvore de Constituintes da Figura 2.8 . . . . .	10
Figura 2.10: Solução Cloud da <i>NVIDIA</i> para processamento de modelos Deep Learning . . . . .	16
Figura 3.1: Interface <i>Gradschool.chat</i> . . . . .	21
Figura 3.2: Fluxo de Execução <i>LLM-based Process Automation (LLMPA)</i> (GUAN et al., 2023) . . . . .	21
Figura 3.3: Fluxo de Execução LLMPA (VU et al., 2024) . . . . .	22
Figura 3.4: Arquitetura do Sistema . . . . .	24
Figura 3.5: Diagrama Entidade Relacionamento do Banco de Dados . . . . .	25

Figura 3.6: Diagrama dos Agentes . . . . .	27
Figura 3.7: Diagrama de Atividades do Fluxo de Processamento de Mensagens	28
Figura 4.1: Organização do projeto do <i>front-end</i> . . . . .	35
Figura 4.2: Diagrama arquitetural <i>Model-View-Controller</i> . . . . .	36
Figura 4.3: Documentação da Application Programming Interface (API) . . .	37
Figura 4.4: Tela Inicial . . . . .	38
Figura 4.5: Tela de Cadastro . . . . .	39
Figura 4.6: Tela de Login . . . . .	39
Figura 4.7: Tela de Edição de Usuário quando acessada por usuário básico . .	40
Figura 4.8: Tela de Listagem de Usuários disponível apenas para administradores	41
Figura 4.9: Tela de Mensagens . . . . .	41
Figura 4.10: Tela de Mensagens indicando a classificação . . . . .	46
Figura 4.11: Calendário do <i>Todoist</i> após criação de eventos . . . . .	46

# Lista de Tabelas

Tabela 4.1: Comparação de respostas fornecidas pelo <i>Alfred</i> , <i>GPT-4o</i> e <i>GPT-3.5 Turbo</i> . . . . .	48
--	----

# Lista de Abreviaturas e Siglas

<b>API</b>	Application Programming Interface
<b>BERT</b>	<i>Bidirectional Encoder Representations from Transformers</i>
<b>GPT</b>	<i>Generative Pre-Trained Transformer</i>
<b>GDPR</b>	<i>General Data Protection Regulation</i>
<b>GRU</b>	<i>Gated Recurrent Unit</i>
<b>HTML</b>	HyperText Markup Language
<b>IA</b>	Inteligência Artificial
<b>IoT</b>	<i>Internet of Things</i>
<b>I/O</b>	<i>Input/Output</i>
<b>JS</b>	<i>JavaScript</i>
<b>JSON</b>	<i>JavaScript Object Notation</i>
<b>LSA</b>	<i>Latent Semantic Analysis</i>
<b>LLM</b>	<i>Large Language Model</i>
<b>LLMPA</b>	<i>LLM-based Process Automation</i>
<b>LGPD</b>	Lei Geral de Proteção dos Dados
<b>LSTM</b>	<i>Long Short-Term Memory</i>
<b>MVC</b>	<i>Model-View-Controller</i>
<b>ML</b>	<i>Machine Learning</i>
<b>NLP</b>	<i>Natural Language Processing</i>

<b>NER</b>	<i>Named Entity Recognition</i>
<b>POS</b>	<i>Part-of-Speech</i>
<b>REST</b>	Representational State Transfer
<b>SGBD</b>	Sistema de Gerenciamento de Banco de Dados
<b>SVM</b>	<i>Support Vector Machines</i>
<b>TS</b>	<i>TypeScript</i>

# Sumário

Agradecimentos	i
Resumo	v
Abstract	vi
Lista de Figuras	vii
Lista de Tabelas	ix
Lista de Abreviaturas e Siglas	x
<b>1 Introdução</b>	<b>1</b>
<b>2 Large Language Model</b>	<b>4</b>
2.1 Inteligência Artificial . . . . .	5
2.1.1 Aprendizado de Máquina . . . . .	5
2.1.2 Deep Learning . . . . .	6
2.1.3 Processamento de Linguagem Natural . . . . .	7
2.1.3.1 Tokenização . . . . .	7
2.1.3.2 POS <i>Tagging</i> . . . . .	8

2.1.3.3	Reconhecimento de Entidades Nomeadas (NER) . . .	9
2.1.3.4	Análise sintática . . . . .	9
2.1.3.5	Análise semântica . . . . .	11
2.1.4	Transformers . . . . .	12
2.2	Large Language Models . . . . .	13
2.2.1	Generative Pre-Trained Transformer . . . . .	14
2.3	Desafios . . . . .	14
2.3.1	Escalabilidade, custos e meio-ambiente . . . . .	15
2.3.2	Segurança e privacidade . . . . .	16
2.3.2.1	Causas de alucinação . . . . .	17
<b>3</b>	<b>Allfred Assistente Virtual</b>	<b>18</b>
3.1	Motivação . . . . .	18
3.2	Trabalhos Relacionados . . . . .	20
3.3	Proposta . . . . .	22
3.3.1	Arquitetura do Sistema . . . . .	23
3.3.2	Armazenamento . . . . .	24
3.3.3	Integrações . . . . .	25
3.3.3.1	Serper . . . . .	26
3.3.3.2	Todoist . . . . .	26
3.3.3.3	ChatGPT . . . . .	26
3.3.4	Funcionalidades . . . . .	29
<b>4</b>	<b>Implementação da Arquitetura Proposta</b>	<b>31</b>



4.1	Tecnologias . . . . .	31
4.1.1	TypeScript . . . . .	32
4.1.2	Vue.js . . . . .	32
4.1.3	FastAPI . . . . .	32
4.1.4	Langchain . . . . .	33
4.1.5	PostgreSQL . . . . .	33
4.1.6	Redis . . . . .	33
4.2	Desenvolvimento . . . . .	34
4.2.1	Front-End . . . . .	34
4.2.2	Back-End . . . . .	35
4.3	Interfaces e Funcionalidades . . . . .	37
4.3.1	Página Inicial . . . . .	37
4.3.2	Cadastro de Usuário . . . . .	38
4.3.3	Login . . . . .	38
4.3.4	Configurações . . . . .	38
4.3.5	Mensagens . . . . .	40
4.4	Comparação e Análise de Resultados . . . . .	45
<b>5</b>	<b>Conclusões</b>	<b>49</b>
5.1	Considerações Finais . . . . .	49
5.2	Limitações e Trabalhos Futuros . . . . .	50
	<b>Referências</b>	<b>51</b>

# Capítulo 1

## Introdução

Nos últimos anos, a Inteligência Artificial (IA) consolidou-se como uma das áreas mais inovadoras e impactantes da tecnologia. Entre os avanços mais notáveis estão os Modelos de Linguagem de Grande Escala (*Large Language Model* (LLM)), que demonstraram uma capacidade impressionante de compreender e gerar texto em linguagem natural.

Esses modelos são fundamentados em arquiteturas avançadas do LLM como os *transformers* (VASWANI et al., 2017), demais detalhes estão descritos no capítulo 2. Estes são arquiteturas de redes neurais usadas para processamento de texto terão descrição destrinchada no capítulo 2, prometem revolucionar a interação humano-máquina, tornando-a mais eficiente e natural. A utilização desses modelos em assistentes virtuais, como o *ChatGPT*<sup>1</sup>, desenvolvido pela *OpenAI*, exemplifica como os LLM's podem aprimorar a qualidade das interações ao manter o contexto e fornecer respostas mais precisas e pertinentes.

Este estudo propõe o desenvolvimento de *Alfred*, um assistente virtual que emprega LLM para oferecer respostas contextuais e precisas em tempo real, utilizando dados extremamente atualizados. *Alfred* visa superar as limitações dos modelos convencionais ao integrar tecnologias avançadas de linguagem, como o *gpt-3.5-turbo*, para categorizar e processar solicitações, enriquecendo o contexto das interações e

---

<sup>1</sup><<https://chatgpt.com/>>

refinando as respostas com base no histórico.

*Alfred* é projetado para fornecer um desempenho comparável ao de modelos mais novos e robustos, mesmo utilizando recursos mais acessíveis, especialmente em termos de aplicação de contexto e integração com sistemas externos.

Este trabalho demonstra que é possível empregar tecnologias estabelecidas para executar funções avançadas por meio de integrações estratégicas, oferecendo uma interface de usuário intuitiva e amigável.

*Alfred* incorpora funcionalidades dos assistentes virtuais convencionais, como *Siri*<sup>2</sup>, *Alexa*<sup>3</sup> e *Google Assistant*<sup>4</sup>, mantendo o contexto, além de executar tarefas complexas envolvendo múltiplas etapas, além disso ao ser comparado as respostas com as de outras ferramentas, foi possível perceber uma melhoria na precisão e relevância das repostas obtidas.

Além disso, este estudo aborda os principais desafios e soluções no desenvolvimento de IA, incluindo aspectos de escalabilidade, segurança, privacidade e impacto ambiental. Ao comparar o desempenho de *Alfred* com outras soluções existentes, destacam-se as vantagens em termos de precisão e relevância das respostas fornecidas.

Em suma, a pesquisa não apenas introduz um novo assistente virtual, mas tem como objetivo contribuir para o avanço das aplicações de LLM's, ao enriquecer utilizar modelos já existentes para fornecer melhores respostas, utilizando múltiplos-agentes e buscas em tempo real em sistemas interativos, promovendo uma melhor compreensão e utilização dessas tecnologias emergentes em diferentes áreas do conhecimento.

O trabalho está dividido da seguinte forma:

- O capítulo 2, apresentou uma revisão bibliográfica a respeito dos LLM's, incluindo uma visão geral sobre o tema, seu contexto histórico e desafios;
- No capítulo 3, foi explicada a motivação deste trabalho, com problemas relacionados aos assistentes virtuais convencionais e como LLM's, podem

---

<sup>2</sup>[<https://www.apple.com/br/siri/>](https://www.apple.com/br/siri/)

<sup>3</sup>[<https://www.amazon.com.br/b?ie=UTF8&node=19949683011>](https://www.amazon.com.br/b?ie=UTF8&node=19949683011)

<sup>4</sup>[<https://assistant.google.com/>](https://assistant.google.com/)

ajudar na solução desses problemas, também foram citados alguns trabalhos relacionados ao tema e por fim a proposta do *Alfred* e como ele poderia trazer algumas soluções para esses problemas;

- O capítulo 4, detalha as etapas de desenvolvimento, tecnologias utilizadas e apresenta uma breve comparação dos resultados, mostrando que os objetivos propostos foram alcançados;
- Por fim, no capítulo 5, é apresentada a conclusão do trabalho, as considerações finais, limitações e ideias para trabalhos futuros.

# Capítulo 2

## Large Language Model

Os LLM's são modelos de IA treinados para entender e gerar textos em linguagem natural. Esta tecnologia tem ganhado notoriedade desde o lançamento público do *ChatGPT-3* em novembro de 2022 (DOUGLAS, 2023). De maneira geral, os LLM's utilizam arquiteturas baseadas em *transformers*, desenvolvidas por engenheiros do *Google* em 2017 (VASWANI et al., 2017); os detalhes dessas técnicas serão explorados ao longo deste capítulo.

Grande parte do interesse do público nos LLM's decorre da capacidade desses modelos em gerar respostas que se aproximam muito da forma como os humanos se comunicam, o que torna as interações com o usuário mais naturais. Apesar de haver espaço para melhorias contínuas, as aplicações que utilizam LLM's representam um avanço significativo na interação entre humanos e IA (KUMAR, 2023).

Nas próximas seções, será apresentado um contexto histórico dos modelos de IA, técnicas de *Natural Language Processing* (NLP) e como as arquiteturas *transformers* possibilitaram avanços nessa área. Em seguida, serão abordados os LLM's, com destaque para o modelo *Generative Pre-Trained Transformer* (GPT) desenvolvido pela *OpenAI*, e suas aplicações. Por fim, serão discutidos alguns desafios relacionados a esses novos modelos, tanto do ponto de vista tecnológico quanto em termos de privacidade dos usuários e impactos ambientais.

## 2.1 Inteligência Artificial

A capacidade de aprendizado, raciocínio e tomada de decisões automatizadas por máquinas tem revolucionado diversas áreas do mercado mundial, proporcionando soluções inovadoras para resolver problemas complexos. Utilizando algoritmos específicos, a IA é reconhecida por centralizar conhecimentos como o aprendizado de máquina analisando grandes volumes de dados para identificar padrões e realizar previsões.

Algumas áreas do mercado que estão sendo transformadas pelo uso da IA incluem Marketing, Finanças e Saúde. Por exemplo, na área da saúde, algoritmos de identificação de imagens têm contribuído para o rápido reconhecimento de padrões do *COVID-19* (FILHO et al., 2023).

Este trabalho explorará áreas específicas da IA, que serão detalhadas a seguir.

### 2.1.1 Aprendizado de Máquina

O aprendizado de máquina, também conhecido como *Machine Learning* (ML), é um subárea da IA que se concentra em modelos e algoritmos para sistemas que aprendem autonomamente ou com supervisão para melhorar experiências anteriores. Em ML, ao invés de programar modelos para executar uma tarefa específica, o foco principal está no treinamento desses modelos usando grandes conjuntos de dados para identificar padrões.

As duas abordagens principais em ML são o aprendizado supervisionado e o não supervisionado. No aprendizado supervisionado, utiliza-se um conjunto de dados rotulados, permitindo que o modelo aprenda a mapear entradas para saídas esperadas com base em exemplos fornecidos anteriormente. Por outro lado, no aprendizado não supervisionado, utiliza-se dados não rotulados para descobrir padrões e estruturas internas nos dados, explorando suposições sobre suas saídas.

Expandindo sobre as técnicas de aprendizado supervisionado, destacam-se a Regressão Linear, usada para prever valores contínuos como o preço de casas com

base em suas características. Além da Regressão Linear, algoritmos como *Support Vector Machines* (SVM), Árvores de Decisão e Redes Neurais são amplamente utilizados para categorizar os dados de treinamento.

No aprendizado não supervisionado, uma das técnicas principais é o *Clustering*, que agrupa dados em *clusters*, onde cada *cluster* pode representar uma segmentação com comportamentos, perfis e outras possibilidades de classificação.

### 2.1.2 Deep Learning

Aprofundando o conhecimento em ML, pode-se explorar especificamente a utilização de Redes Neurais, especialmente as Redes Neurais Artificiais com múltiplas camadas, conhecidas como *Deep Neural Networks* (*Deep Learning*), para modelar e compreender dados complexos.

Enquanto os modelos tradicionais de ML frequentemente requerem extração manual de características dos dados, o *Deep Learning* automatiza esse processo, permitindo análise direta dos dados brutos.

O *Deep Learning* pode ser considerado uma evolução das técnicas de ML existentes, onde os dados são representados em hierarquias, capturando essências em múltiplos níveis, tornando-se extremamente eficaz para lidar com dados complexos como vídeos, imagens, áudios e textos.

Dentro das Redes Neurais usadas em *Deep Learning*, destacam-se as *Convolutional Neural Networks* (CNNs) e *Recurrent Neural Networks* (RNNs). As CNNs são fundamentais no processamento de dados em Visão Computacional; por exemplo, em Krizhevsky, Sutskever e Hinton (2012), demonstra-se como CNNs são utilizadas para reconhecimento de objetos, identificação de texturas, bordas e formas.

Para dados sequenciais como séries temporais e NLP, as RNNs são amplamente empregadas devido à sua capacidade de capturar informações contextuais ao longo de sequências, como discutido em Bengio (2016). Modelos comuns incluem *Long Short-Term Memory* (LSTM) e *Gated Recurrent Unit* (GRU), que desempenham papéis essenciais no avanço do NLP, aplicados em áreas como tradução automática

e reconhecimento de fala.

### 2.1.3 Processamento de Linguagem Natural

Após exploração mais a fundo todas as principais áreas da IA, pôde-se chegar no Processamento de Linguagem Natural, ou em inglês NLP, que envolve o desenvolvimento de algoritmos e modelos para reconhecimento, interpretação e geração de linguagem natural por máquinas ou aplicações. Conforme citado anteriormente, graças o NLP novas nuances de aplicações puderam ser criadas como: chatbots, análise de sentimentos através da escrita, traduções instantâneas entre outros.

Através da componentização, o NLP subdivide-se em componentes centrais que são: tokenização, etiquetagem de partes de um discurso, reconhecimento de entidades nomeadas, análise sintática e análise semântica que serão destrinchados abaixo.

#### 2.1.3.1 Tokenização

A tokenização é um processo de segmentação de um determinado texto em unidades menores, os tokens. Os tokens podem ser obtidos de diversas maneiras como: divisão de um texto por palavras, divisão por subpalavras, caracteres, espaços ou outros tipos de unidades definidas pelo usuário.

A tokenização por palavras é um método comum por ser mais intuitivo, por que palavras são unidades semânticas de valor em diversas línguas.

*Eu gosto de Inteligência Artificial*

Figura 2.1: Exemplo de frase para Tokenização

O processo de tokenização com palavras retornará uma lista contendo as palavras identificadas é demonstrado na Figura 2.2.

Em contrapartida, a tokenização por caracteres subdividiria a frase da Figura 2.1 conforme resultado da Figura 2.3.

Todavia, a tokenização também possui os seus desafios como o tratamento de



Palavras =	Eu
	gosto
	de
	Inteligência
	Artificial

Figura 2.2: Resultado de Tokenização por palavras da Figura 2.1

Palavras =	E
	u
	g
	o
	...
	i
	a
l	

Figura 2.3: Resultado de Tokenização por caracteres da Figura 2.1

dados em idiomas de escrita complexa como o Japonês e o Chinês, palavras compostas como *e-mail* e contrações coloquiais em inglês como, por exemplo, *don't*.

### 2.1.3.2 POS Tagging

Como outro processo de marcação de textos, *POS Tagging* é um meio eficiente de marcar palavras com suas respectivas categorias gramaticais. Essas categorias incluem substantivos, verbos, adjetivos, advérbios, preposições, entre outros. Essa é uma técnica fundamental no NLP por que consegue fornecer informações sintáticas e gramaticais ao mesmo tempo, conforme estudos de (JURAFSKY; MARTIN, 2019).

*O gato preto dorme pacificamente.*

Figura 2.4: Frase de exemplo para POS *Tagging*

Onde assume-se que **DET** é o determinante, **NOUN** é o substantivo, **ADJ** é o adjetivo, **VERB** é o verbo e **ADV** sendo o advérbio.

Ainda segundo Jurafsky e Martin (2019), a etiquetagem POS pode utilizar

```
{  
  "DET": "O",  
  "NOUN": "gato",  
  "ADJ": "preto",  
  "VERB": "dorme",  
  "ADV": "pacificamente"  
}
```

Figura 2.5: Resultado POS *Tagging* da Figura 2.4

diversos métodos para análise, como por exemplo regras baseadas na gramática, modelos estatísticos e modelos baseados em *Deep Learning* utilizando CNNs e RNNs.

### 2.1.3.3 Reconhecimento de Entidades Nomeadas (NER)

O Reconhecimento de Entidades Nomeadas (NER) é uma tarefa fundamental em NLP, pois envolve a identificação e classificação de entidades mencionadas em um texto, de acordo com categorias pré-definidas, como nomes, locais e outros elementos.

A importância do NER reside na sua capacidade de extrair informações precisas de textos não estruturados, como documentos, e-mails e artigos científicos. Além da extração, a análise de grandes volumes de dados permite identificar tendências e padrões de frequência, o que aprimora a precisão dos algoritmos em tarefas de perguntas e respostas e melhora a distinção de ambiguidades entre entidades com nomes semelhantes.

*José nasceu no Brasil, em 2 de Setembro de 91*

Figura 2.6: Frase de exemplo para NER

O NER identificaria e classificaria as entidades como demonstrado na Figura 2.7.

### 2.1.3.4 Análise sintática

A análise sintática é um processo crucial no NLP devido à sua capacidade de determinar a estrutura gramatical de uma frase e como as palavras se combinam

```
{  
  "PESSOA": "José",  
  "LOCAL": "Brasil",  
  "DATA": "2 de Setembro de 91"  
}
```

Figura 2.7: Resultado NER da Figura 2.6

para formar unidades gramaticais maiores.

Por meio da análise sintática, é possível compreender a estrutura hierárquica das frases e resolver ambiguidades semânticas com base no posicionamento das palavras e suas relações estruturais dentro da frase. Essa análise é especialmente útil em aplicações que auxiliam o usuário na correção de textos em outros idiomas, como o *Grammarly*<sup>1</sup>.

*O gato preto dorme no sofá.*

Figura 2.8: Exemplo de frase para Análise Sintática

A análise sintática pode representar a estrutura de uma frase de duas maneiras principais: por meio de uma Árvore de Constituintes ou de uma Árvore de Dependências, conforme exemplificado abaixo.

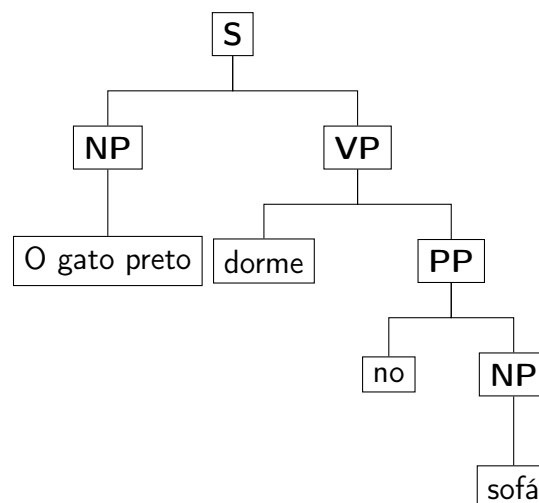


Figura 2.9: Exemplo de Árvore de Constituintes da Figura 2.8

<sup>1</sup><https://www.grammarly.com/>

A Árvore de Constituintes divide a frase principal em unidades menores, até alcançar as palavras individuais. No exemplo contido na Figura 2.9, **S** representa a frase principal, **NP** o sujeito, **VP** o predicado, **PP** a posição do verbo, e **NP** um objeto relacionado. Essa estrutura hierárquica ajuda a visualizar como as palavras se agrupam em constituintes menores e como esses constituem a frase completa.

Por outro lado, a Árvore de Dependências foca nas relações de interdependência entre as palavras da frase. Essa abordagem é recomendada para tarefas que requerem um entendimento detalhado das relações entre os elementos da frase, enfatizando como o verbo principal se conecta aos seus argumentos e outras palavras relacionadas.

O campo da análise sintática enfrenta desafios significativos, que vão desde a exploração completa da gramática de uma língua, incluindo o uso de gírias e expressões coloquiais, até a ambiguidade sintática, onde uma mesma frase pode ser analisada de diversas maneiras. Além disso, há a complexidade computacional envolvida na geração das árvores sintáticas.

#### 2.1.3.5 Análise semântica

A análise semântica tem como objetivo principal a extração de significados das palavras e frases dentro de um texto, mantendo a coerência semântica. Ela pode ser dividida em duas categorias principais: a análise semântica léxica e a análise semântica composicional.

Em termos de técnicas, são amplamente utilizados modelos baseados em regras linguísticas pré-definidas, modelos estatísticos como *Latent Semantic Analysis* (LSA) e *Word2Vec*, e modelos que empregam redes neurais. Atualmente, destacam-se modelos sofisticados como *Bidirectional Encoder Representations from Transformers* (BERT) e GPT, que se concentram em capturar nuances semânticas complexas por meio de contextos bidirecionais das palavras.

Ainda existem desafios significativos no campo da semântica, incluindo a ambiguidade de palavras, a compreensão de metáforas e a interpretação de expressões características de um idioma.

### 2.1.4 Transformers

O conceito de *transformers* revolucionou a IA e, conseqüentemente, o NLP, sendo inicialmente desenvolvido para tarefas como tradução automática. No entanto, os *transformers* se destacaram pela sua versatilidade e capacidade de lidar com uma ampla gama de tarefas devido às suas múltiplas facetas.

Nesta seção, revisaremos os principais aspectos dessa sub-área da IA, explorando suas características arquitetônicas e aplicações.

Conforme Vaswani et al. (2017), a arquitetura dos *transformers* se distingue por três principais técnicas: o modelo de atenção, a arquitetura *Encoder-Decoder* e as camadas de atenção múltipla, ou *Multi-Head Attention*.

Dessa forma, o modelo de atenção é baseado em mecanismos de atenção, especificamente o mecanismo de Atenção Escalonada (*Scaled Dot-Product Attention*). Este mecanismo permite ao modelo focar em diferentes partes do *input*, melhorando a capacidade de capturar dependências contextuais a longo prazo.

A arquitetura *Encoder-Decoder* é composta por duas partes principais: o *Encoder*, responsável pelo processamento dos dados de *input*, e o *Decoder*, que gera a saída. Ambos utilizam camadas de atenção e *feed-forward* para transformar e gerar sequências.

Por fim, Vaswani et al. (2017) define o mecanismo de Atenção Múltipla (*Multi-Head Attention*) como uma arquitetura que permite ao modelo aprender diferentes representações do contexto simultaneamente, aprimorando a capacidade do *transformer* em capturar informações complexas e variadas do *input*.

As aplicações dos *transformers* evoluíram significativamente, passando de simples traduções automáticas para modelos de linguagem complexos, como BERT e GPT. Esses modelos têm sido amplamente utilizados em tarefas como análise de sentimentos, resumo de textos, e sistemas de perguntas e respostas.

Destaca-se também o *GPT-3*, que possui habilidades impressionantes na produção de textos coerentes e contextualmente alinhados, e é capaz de auxiliar em uma

infinitude de tarefas, incluindo a assistência na programação como por exemplo o *Github Copilot*<sup>2</sup>.

Os desafios enfrentados pelos *transformers* são semelhantes aos encontrados em outras áreas da IA, incluindo a dependência de grandes quantidades de dados para treinamento, como discutido por Gang et al. (2020). Além disso, a interpretação dos modelos de *transformers* e a compreensão de como eles geram previsões continuam sendo áreas ativas de pesquisa. Esses desafios estão associados à explicabilidade e à transparência dos modelos, conforme apontado por Rogers, Kovaleva e Rumshisky (2020).

## 2.2 Large Language Models

Os LLM's têm se tornado centrais nas discussões sobre IA e NLP, incluindo arquiteturas como GPT e BERT. Esses modelos são treinados em vastas quantidades de dados textuais e possuem bilhões de parâmetros, permitindo-lhes compreender e produzir texto de maneira sofisticada.

Uma das principais características dos LLM é a escala de treinamento, que se refere à quantidade de dados textuais ingeridos pelo modelo, permitindo a captura de uma ampla gama de conhecimentos e padrões linguísticos. Outro aspecto crucial é a quantidade de parâmetros; por exemplo, o *GPT-3* conta com 175 bilhões de parâmetros (BROWN et al., 2020).

Esses modelos são capazes de gerar texto coerente e relevante em resposta a prompts fornecidos, sendo utilizados em tarefas como tradução automática, resumo de textos, respostas a perguntas e criação de conteúdo. Geralmente, os LLM passam por um processo de pré-treinamento em um corpus grande e genérico, seguido por um ajuste (*fine-tuning*) em conjuntos de dados específicos para melhorar seu desempenho em tarefas particulares.

---

<sup>2</sup><<https://github.com/features/copilot>>

### 2.2.1 Generative Pre-Trained Transformer

O modelo GPT, como introduzido por Vaswani et al. (2017), é baseado na arquitetura de *transformers*. O *GPT-1*, desenvolvido pela *OpenAI* e lançado em 2018, foi treinado em um grande corpus de texto e posteriormente ajustado para tarefas específicas por meio de *fine-tuning* (RADFORD; NARASIMHAN, 2018).

O sucesso do *GPT-1* levou ao desenvolvimento do *GPT-2*, que expandiu significativamente o número de parâmetros para 1,5 bilhão (RADFORD et al., 2019). O *GPT-3*, lançado posteriormente, aumentou ainda mais essa capacidade, alcançando 175 bilhões de parâmetros (BROWN et al., 2020).

Atualmente, a *OpenAI* disponibiliza o *GPT-4* através de suas APIs e da aplicação *ChatGPT*<sup>3</sup>. Embora o número exato de parâmetros do *GPT-4* não tenha sido divulgado, a *OpenAI* afirma que ele possui significativamente mais parâmetros que o *GPT-3* (OPENAI et al., 2024).

No benchmark *ARC*, o *GPT-4* atingiu uma precisão de 86,4%, comparado a 81,2% do *GPT-3*, demonstrando um aumento significativo. Além disso, o *GPT-4* apresentou uma redução substancial nas respostas consideradas tóxicas ou prejudiciais, com uma taxa de cerca de 82% em comparação com seu antecessor.

## 2.3 Desafios

A rápida e contínua evolução dos modelos de IA está marcando o início de uma nova era de inovação tecnológica, impactando profundamente diversos setores. O avanço desses modelos, impulsionado por melhorias em arquiteturas como o *Deep Learning*, tem proporcionado capacidades sem precedentes em análise de dados, predição e automação de processos. No entanto, à medida que esses modelos se tornam mais sofisticados e amplamente adotados, surgem desafios complexos que precisam ser abordados para assegurar um desenvolvimento sustentável e ético da tecnologia.

---

<sup>3</sup><<https://chat.openai.com/>>

Nesta seção, serão discutidos tópicos relevantes para o estudo e compreensão dos impactos e das questões emergentes à medida que a tecnologia avança.

### 2.3.1 Escalabilidade, custos e meio-ambiente

A utilização de máquinas robustas com *Central Processing Units* (CPUs) e *Graphics Processing Units* (GPUs), que suportam os novos *Software Development Kit* (SDKs) do mercado, é essencial para o treinamento de modelos complexos, como, por exemplo, em *Deep Learning*.

No treinamento de modelos deve-se levar em consideração a quantidade de GPUs e de *Tensor Processing Units* (TPUs), além do tempo total de treinamento. Como é de conhecimento, o *GPT-3* desenvolvido pela *OpenAI* demora dias ou semanas para ser treinado, dependendo do volume de dados e tamanho do modelo. Em estudo por Strubell, Ganesh e McCallum (2019), estimou-se que um treinamento de modelo de IA de grande escala pode consumir 626.000 kWh de eletricidade, o equivalente ao consumo de energia anual de cinco residências americanas médias.

Para aplicações em escala, os modelos devem ser replicados em data centers ao redor do mundo, gerenciados 24/7 e, portanto, aumenta-se o consumo energético devido ao custo de resfriamento desses data centers. Grandes companhias como a *Microsoft* detalham seus compromissos com o meio ambiente através de relatórios anuais referentes a compensação de pegadas de carbono, conforme emitido em Microsoft Corporation (2024).

Todavia, máquinas que atingem essa capacidade de processamento ainda são inacessíveis para parte da população, mantendo uma distância entre o uso cotidiano, entusiastas da tecnologia e os interesses das grandes corporações.

O uso cotidiano de IA limita-se a tarefas básicas de baixa latência e tempo de resposta aceitáveis, a partir de modelos treinados anteriormente. Por outro lado, para grandes corporações, o estudo de comportamento humano, previsões e análises são fundamentais e nesse caso são utilizadas máquinas robustas.



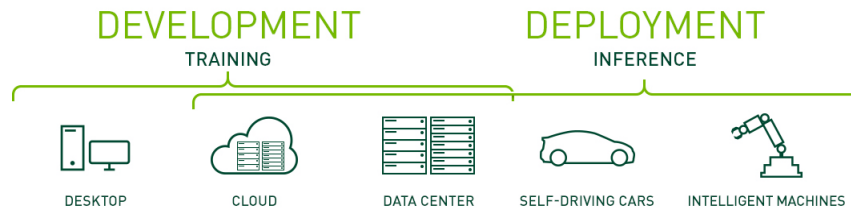


Figura 2.10: Solução Cloud da *NVIDIA* para processamento de modelos Deep Learning

### 2.3.2 Segurança e privacidade

Durante a utilização de modelos, podem ocorrer ataques adversariais. Estes ataques consistem na manipulação dos dados de entrada de um modelo de IA, de forma a enganar o sistema e gerar informações incorretas ou indesejáveis. Ataques adversariais são comuns entre usuários que desejam explorar e testar novas plataformas, muitas vezes levando à geração de resultados distorcidos.

Em 2016, um experimento da *Microsoft* saiu do controle devido a inputs mal-intencionados de usuários do *Twitter*, resultando em aprendizado incorreto por parte da IA, conforme relatado por TecMundo (2016).

Segundo Ji et al. (2022), alucinações em sistemas de IA são outputs que aparentam ser coerentes, seguindo uma lógica interna, mas que distorcem fatos ou geram novos conhecimentos falsos. Exemplos disso incluem a criação de citações e eventos históricos fictícios.

À medida que grandes corporações intensificam a análise dos perfis de consumo, é crucial que as mesmas enviem todos os dados anonimizados aos seus modelos de IA e que estejam em conformidade com padrões como a Lei Geral de Proteção dos Dados (LGPD) no Brasil, o *General Data Protection Regulation* (GDPR) na Europa e as normativas específicas de cada país.

Além disso, recomenda-se que os usuários finais de plataformas baseadas em IA evitem compartilhar dados pessoais e sensíveis. Isso se deve ao fato de que *inputs* e *outputs* podem ser reutilizados para o treinamento de novas versões dos modelos, o que pode comprometer a privacidade dos dados.

### 2.3.2.1 Causas de alucinação

Conforme IBM (2024), a ambiguidade dos dados durante o treinamento é um dos principais fatores que levam uma IA a gerar respostas incorretas. Caso os dados não sejam devidamente tratados antes do treinamento, o modelo tende a replicar esses erros.

Ainda de acordo com IBM (2024), a má formulação dos inputs ou *prompts* também pode induzir o sistema a alucinar, tentando adivinhar o que o usuário deseja inserir, o que pode resultar na geração de informações irreais.

Portanto, as alucinações podem comprometer a confiabilidade das aplicações e seus *outputs*, especialmente em setores como saúde, jornalismo e jurídico, onde dados incorretos podem influenciar decisões e prejudicar outros seres humanos.

# Capítulo 3

## Alfred Assistente Virtual

Este capítulo tem como objetivo apresentar o sistema proposto, sua motivação e a modelagem da solução. A seção 3.1, tem como objeto discutir os problemas relacionados aos assistentes virtuais pessoais. Na seção 3.2, foram citados sistemas de assistentes virtuais que buscam resolver o problema através de uma diferente abordagem. Por fim, a seção 3.3, apresenta a modelagem proposta por este trabalho, com base nos problemas apresentados.

### 3.1 Motivação

Os assistentes virtuais são ferramentas focadas em executar solicitações dos usuários, com o objetivo de simplificar tarefas comuns do dia-a-dia como criar lembretes, alarmes, etc. Nos últimos anos, diversas empresas de tecnologia desenvolveram e aprimoraram seus assistentes virtuais, visando oferecer mais funcionalidades, fazendo com que esses sistemas se tornem cada vez mais comuns. Com a popularização de assistentes virtuais como a *Alexa* da *Amazon*, o *Google Assistant* da *Google* e a *Siri* da *Apple*, que em sua maioria já estão embarcados em *smartphones*, auto-falantes e televisores inteligentes, o interesse do público geral nessas tecnologias vem aumentando ano após ano (HOY, 2018).

Porém, os sistemas atuais ainda possuem uma série de limitações que vêm se

mostrando uma barreira para que os assistentes atendam a todas as necessidades da rotina dos usuários, ainda em Hoy (2018) é mostrado que apesar de cerca de 98% dos usuários de *iPhone* já terem utilizado a *Siri* em algum momento, apenas 30% utilizam de forma cotidiana. Mesmo sendo capazes de realizar inúmeras integrações com dispositivos de *Internet of Things* (IoT) e sistemas de casas inteligentes, ainda é notável a dificuldade que os assistentes virtuais possuem para executar tarefas mais complexas que envolvam múltiplas etapas. Uma das principais reclamações dos usuários se dá pelo fato dessas aplicações terem problemas para executar tarefas sequenciais e, principalmente, o fato de elas não serem capazes de manter o contexto de tarefas executadas anteriormente. Ou seja, elas não são capazes de guardar informações fornecidas previamente pelo usuário. Estas limitações tornam as experiências dos usuários por vezes frustrantes, criando assim uma barreira para a utilização dos assistentes de forma mais geral (GÄRTLER; SCHMIDT, 2021).

Visando melhorar a comunicação entre humanos e máquinas, nos últimos anos, diversos estudos focaram em aprimorar técnicas de *Machine Learning* para a criação de inteligências artificiais conversacionais (SUNDAR; HECK, 2022). Porém, o avanço mais significativo aconteceu em Novembro de 2022, quando a *Open AI* lançou para o público o *ChatGPT*. O *ChatGPT* se trata de um LLM que, de forma muito similar a um chat, é capaz de processar perguntas dos usuários e respondê-las utilizando linguagem natural, sendo capaz de manter o contexto de interações anteriores. Desde então, inúmeras empresas de tecnologia vêm aprimorando e disponibilizando para o público os seus próprios LLM's para competirem com a *Open AI* e o *ChatGPT*. Com esses avanços, começaram a surgir inúmeras possibilidades de aplicações, que podem se aproveitar dessas tecnologias para aprimorar sistemas que já existem nos dias de hoje (RAY, 2023).

Apesar de serem muito eficientes em responder perguntas de forma natural e similar a como um humano responderia, os LLM's ainda são incapazes de executar tarefas objetivas, que exijam interagir com plataformas externas de modo geral, por outro lado os assistentes virtuais são muito eficientes ao executar esse tipo de tarefa. Logo, é inevitável pensar que essas tecnologias podem ser utilizadas em conjunto, em um assistente virtual que tenha sua conversação melhorada através do uso de

LLM's.

Dessa forma, assistentes virtuais poderiam se aproveitar principalmente da capacidade que esses modelos possuem de armazenar contexto, para assim criarem uma interface mais fluída de comunicação entre os usuários e o assistente virtual, ao mesmo tempo que executam tarefas objetivas que podem ser relevantes no dia-a-dia dos usuários. Devido à capacidade desses modelos de serem enriquecidos com informações relevantes e principalmente com o histórico de interações anteriores, assistentes virtuais que se aproveitarem dessas tecnologias poderiam oferecer uma experiência muito mais assertiva e personalizada.

## 3.2 Trabalhos Relacionados

Nesta seção, serão apresentados alguns artigos e trabalhos relacionados, que visam melhorar o desempenho de assistentes virtuais através da utilização de LLM's.

Em Park e Kulkarni (2024), foi proposto um modelo de assistente virtual chamado de *thinking assistants*, nele foi desenvolvido o assistente virtual *Gradschool.chat*. Seu principal objetivo vai além de simplesmente responder as perguntas feitas pelo usuário. Por se tratar de um assistente focado no ensino, ele tem como objetivo principal engajar debates acerca de um tema, oferecer ideias e recomendar artigos relevantes.

Um fato interessante sobre o *Gradschool.chat* é que ele foi treinado com dados de pesquisa de professores reais que aceitaram participar do projeto, então ao acessar a plataforma o aluno pode escolher um agente específico que representa um professor real. O sistema está disponível online em *gradschool.chat*<sup>1</sup> como mostra a Figura 3.1.

Já em Guan et al. (2023), foi proposta a automatização de processos através do uso de LLM, chamado LLMPA, onde os *prompts* são responsáveis por decompor a solicitação que o usuário deseja através de um *chatbot* e emular o comportamento de uma humano interagindo diretamente com um dispositivo móvel para realizar a tarefa solicitada, como representado na Figura 3.2. O artigo conclui que um assistente baseado em LLM é capaz de executar tarefas objetivas, até mesmo no caso

---

<sup>1</sup><https://gradschool.chat/>

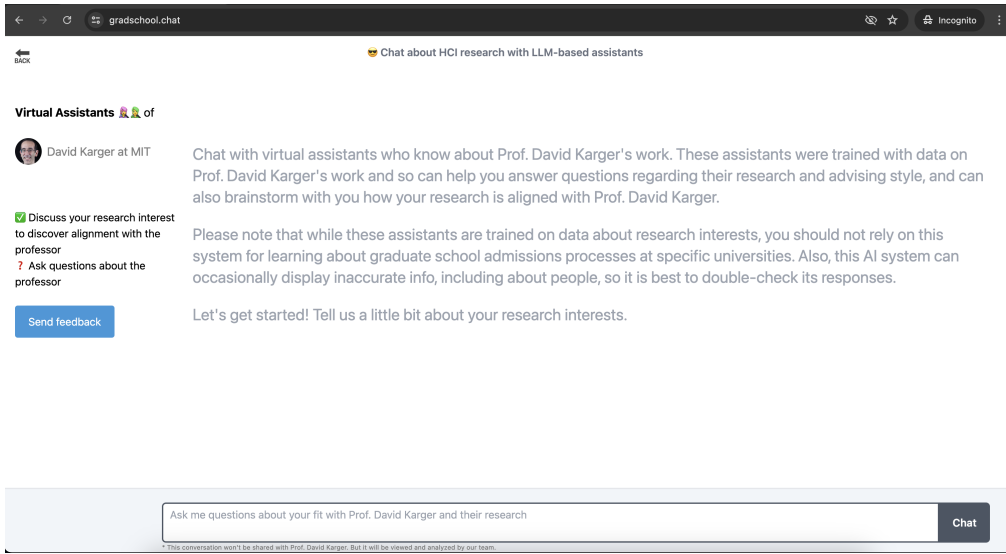


Figura 3.1: Interface *Gradschool.chat*

de cenários mais complexos, porém avanços ainda são necessários para melhorar a performance e solucionar possíveis ambiguidades relacionadas à análise de contexto.

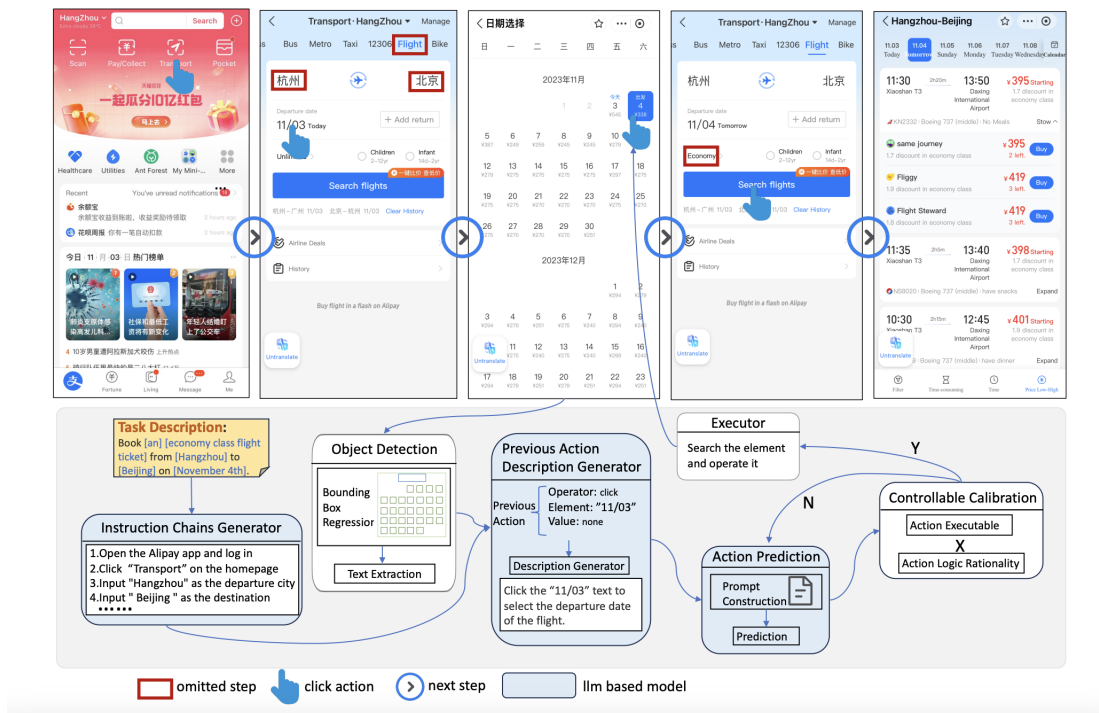


Figura 3.2: Fluxo de Execução LLMPA (GUAN et al., 2023)

Uma proposta similar ao LLMPA, foi feita em Vu et al. (2024), porém com a interação entre o usuário e o assistente sendo feita por voz, conforme mostra a Figura 3.3. O assistente, chamado *GptVoiceTasker*, também tem como objetivo

interagir com a tela de um dispositivo móvel ao analisá-las para interpretar como a solicitação do usuário pode ser executada.

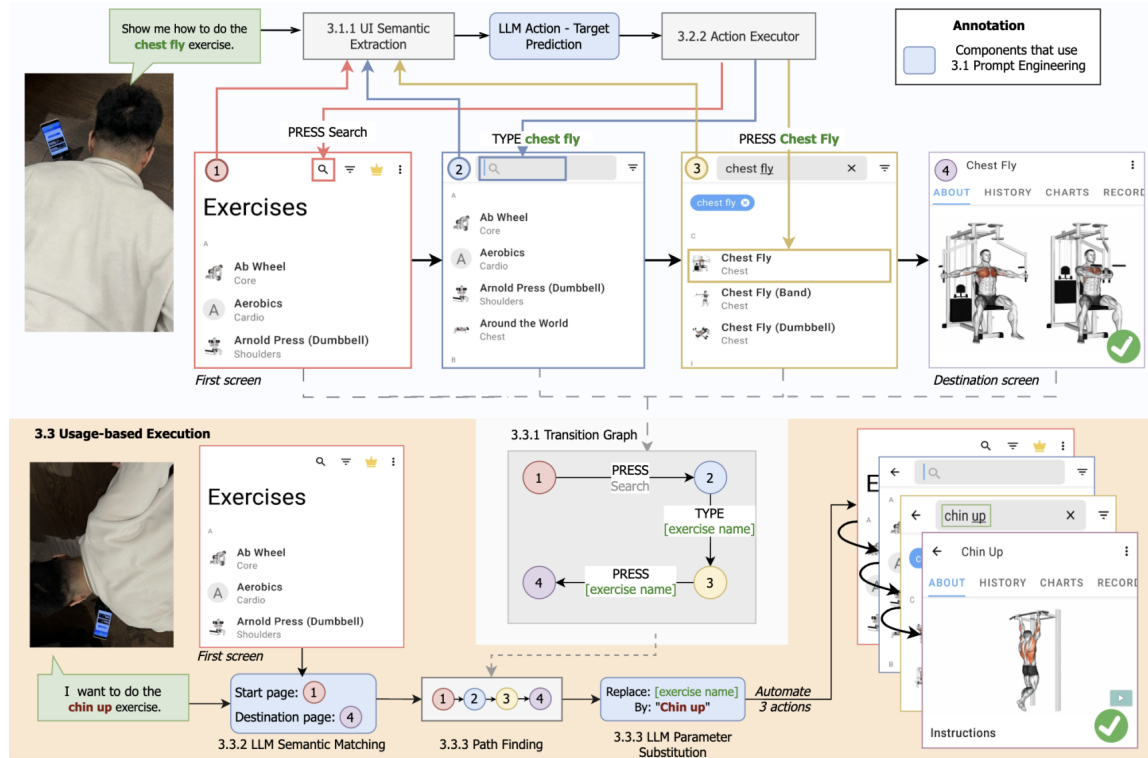


Figura 3.3: Fluxo de Execução LLMPA (VU et al., 2024)

Como é possível notar diversos esforços estão sendo realizados para criar assistentes virtuais baseados em LLM e realizar tarefas objetivas, com sua forma e objetivo principal podendo variar de acordo com necessidades específicas.

### 3.3 Proposta

Considerando o contexto citado anteriormente, a proposta deste trabalho foi desenvolver um assistente virtual *web*, chamado *Alfred*, que através da interação entre múltiplos agentes executa as tarefas solicitadas pelos usuários via texto. O principal objetivo do sistema é fornecer uma experiência personalizada para cada usuário, já que o histórico de cada interação entre o usuário e o sistema será mantido, visando melhorar respostas em interações futuras.

No intuito de tornar experiência do usuário mais completa e personalizada, a aplicação utilizou a API do *ChatGPT*, a API de busca *Serper*, que foi utilizada para enriquecer as conversas e alimentar os *prompts* enviados ao *ChatGPT*, com objetivo de buscar informações atualizadas e dados em tempo real.

Como assistentes virtuais convencionais executam tarefas objetivas, o *Allfred* conta também com uma integração com a API do *Todoist*, para permitir a criação de eventos na conta do usuário a partir das mensagens enviadas para o assistente.

A subseção 3.3.1, tratou da arquitetura do sistema, explicitando como ele está estruturado. Na subseção 3.3.2, foram abordadas todas as estratégias de armazenamento, incluindo banco de dados, *cache* e as estruturas que serão responsáveis por armazenar os contextos das conversas. Já na subseção 3.3.3, foram descritas todas as integrações com sistemas externos e a utilidade de cada um deles, além de mostrar como ocorre a interação entre os agentes. E, por último, na subseção 3.3.4, estão descritas as funcionalidades incluídas no sistema, além de um diagrama explicando como funciona o fluxo de mensagens do sistema.

### 3.3.1 Arquitetura do Sistema

A aplicação é composta por quatro camadas, como apresentado na Figura 3.4, um sistema *web* (*front-end*), um servidor *web* (*back-end*), uma camada de persistência e uma camada responsável pelas as integrações com sistemas externos.

Todas as regras de negócio, integrações externas e acesso à camada de persistência são responsabilidade do *back-end*, nele também serão disponibilizadas todas funcionalidades que poderão ser acessadas por meio de *endpoints* em uma API *Rest*. O *front-end* por sua vez será a página acessada pelo usuário, que irá se conectar com os *endpoints* disponibilizados pelo *back-end* e necessita de uma chave de acesso para autenticar as requisições.

O acesso aos dados do sistema foi dividido de duas formas: a primeira se trata de um banco de dados relacional para armazenar todas as informações dos usuários e novas entradas no sistema. Já a segunda se trata de uma camada de armazenamento



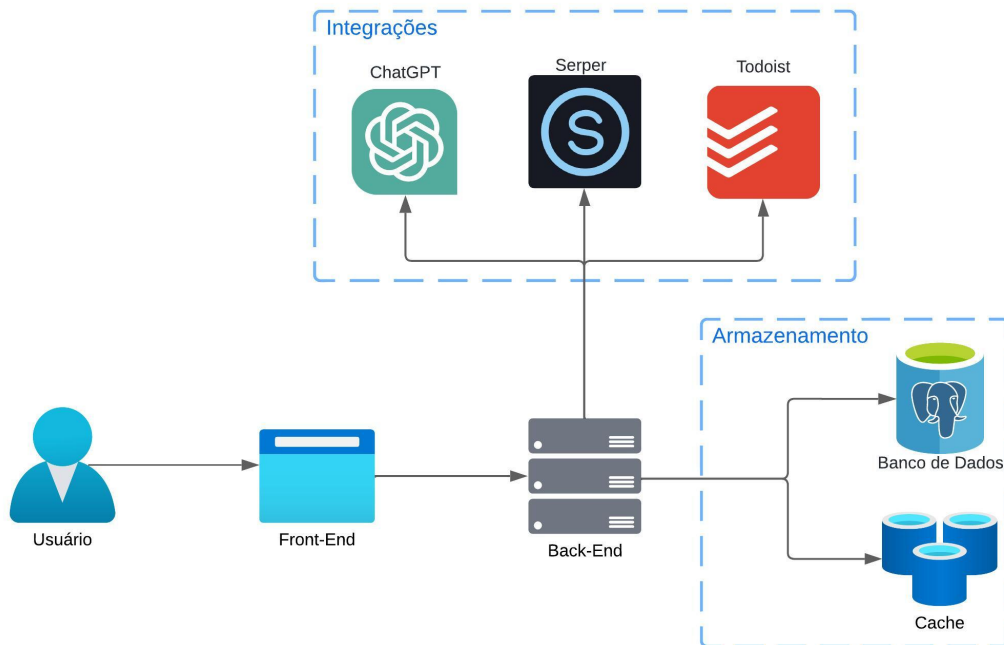


Figura 3.4: Arquitetura do Sistema

chave-valor em memória, que devido a sua velocidade de acesso, se mostra muito útil para validar informações que são constantemente acessadas pelo sistema, como a chave de autorização do usuário, e também permitem uma forma mais rápida de acesso ao histórico de mensagens enviadas pelo usuário.

O sistema também conta com a camada de integrações, responsável por se conectar com outras três plataformas externas: A API do *ChatGPT*, para acessar o LLM que irá processar as solicitações do usuário, a API do *Serper*, para buscar informações atualizadas em tempo real à respeito das solicitações feitas pelo usuário e a API do *Todoist* que a permite a criação de eventos na conta do usuário.

### 3.3.2 Armazenamento

Como mencionado anteriormente, a camada de armazenamento foi dividida em duas estruturas, o banco de dados relacional representado pela Figura 3.5 é responsável por armazenar de forma permanente as informações do usuário tendo sido dividido em cinco entidades, sendo elas:

- **User:** Responsável pelas informações básicas do usuário como nome, *email* e

senha;

- **Message:** Têm como objetivo armazenar as mensagens enviadas pelo usuário, sua categoria e a resposta fornecida pelo sistema;
- **Model:** Essa entidade existe para armazenar os modelos de LLM que podem ser atribuídos a um usuário, neste momento apenas *gpt-3.5-turbo* está disponível;
- **Role:** Armazena as possibilidades de permissionamento de um usuário, que pode ser *básico* ou *administrador*;
- **Category:** Contém as possíveis categorias de uma mensagem, que no caso estão divididas em *pergunta*, *informação* e *evento*.

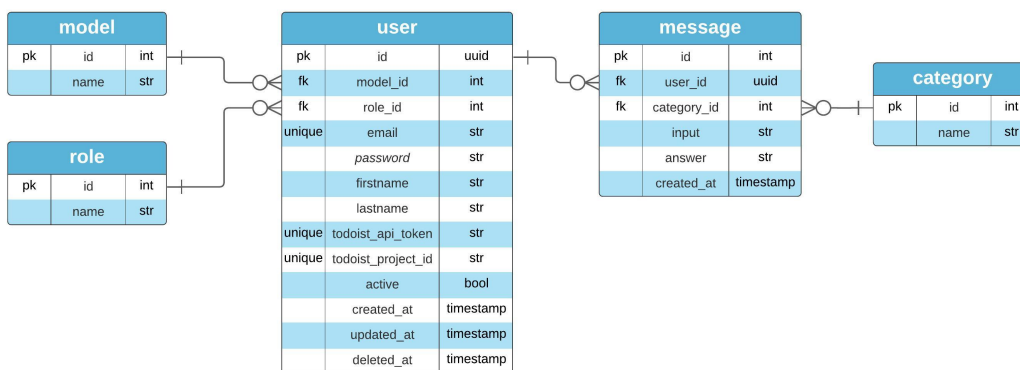


Figura 3.5: Diagrama Entidade Relacionamento do Banco de Dados

Já a estrutura responsável pelo *cache* é um banco de dados chave-valor não relacional em memória, ele está dividido em duas instâncias, a primeira responsável por armazenar os *tokens* de permissão para autenticar os usuários e a segunda mantém o histórico de mensagens de cada usuário, essa abordagem tem como objetivo melhorar a performance do sistema, porém caso o histórico não esteja disponível na camada de *cache* a informação deve ser recuperada do banco de dados relacional.

### 3.3.3 Integrações

Para executar as ações solicitadas pelo usuário, o sistema precisará se conectar com alguns serviços externos como mostra a arquitetura do sistema na subseção 3.3.1. São essas integrações:

- *Serper*;
- *Todoist*;
- *ChatGPT*.

#### 3.3.3.1 *Serper*

A API do *Serper* é responsável por executar consultas no *Google*, de modo que se possa buscar informações atualizadas relacionadas a solicitação do usuário. Seu principal objetivo é enriquecer o contexto da solicitação e melhorar a resposta fornecida pelo sistema.

Modelos como o *gpt-3.5-turbo* possuem limitações ao buscar informações posteriores aos dados utilizados em seu treinamento. Por isso, a utilização do *Serper* se faz necessária no contexto da aplicação, já que dessa forma, foi possível buscar informações mais atualizadas para o usuário mesmo que o sistema esteja utilizando modelos mais limitados e sem acesso a dados atualizados.

#### 3.3.3.2 *Todoist*

A integração com o *Todoist* tem como objetivo permitir que o sistema crie eventos na conta do usuário de forma automatizada. Sendo assim, uma vez que o sistema interpreta que a solicitação do usuário se trata de uma criação de evento, o mesmo irá construir um objeto que atende as necessidades da API do *Todoist* e um novo evento será criado.

#### 3.3.3.3 *ChatGPT*

O *ChatGPT* é a principal integração do sistema, uma vez que ela é responsável por executar as ações solicitadas pelo usuário. De modo que, o sistema se conecta através da API disponibilizada pela *OpenAI* onde é possível se conectar com o modelo de LLM responsável por executar as ações de cada agente. Por padrão, o sistema utiliza o modelo *gpt-3.5-turbo*, principalmente devido ao alto custo de utilizar modelos mais recentes. Contudo, o modelo poderá ser alterado para cada usuário

por um administrador da plataforma.

Visando uma melhor precisão na execução das tarefas solicitadas, o sistema foi desenvolvido em um modelo multi-agentes, onde cada agente será responsável por processar uma tarefa específica, tendo como principal objetivo, melhorar a qualidade da resposta fornecida ao usuário.

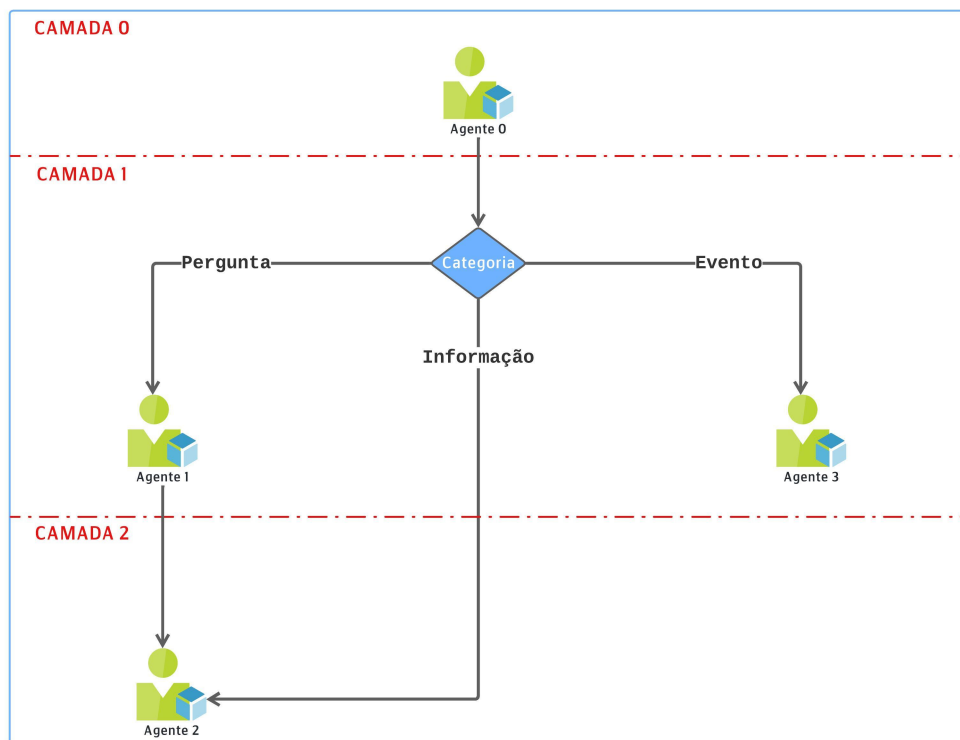


Figura 3.6: Diagrama dos Agentes

Como exibido na Figura 3.6, cada agente é executado seguindo uma ordem estabelecida de acordo com a requisição do usuário, onde cada um deles será responsável por executar uma tarefa específica. Vale ressaltar que cada agente possui um *prompt* que será executado na API do *ChatGPT*, sendo assim, ao final de uma requisição, o sistema pode utilizar dois ou três agentes à depender da categoria da solicitação feita, os *prompts* de cada agente podem ser encontrados na subseção 4.3.5. São esses os papéis de cada agente:

**Agente 0:** responsável por definir a categoria da solicitação feita pelo usuário, essas categorias podem ser classificadas em **pergunta**, **informação** ou **evento**. As *perguntas* podem ser qualquer requisição feita pelo o usuário sobre algum tema

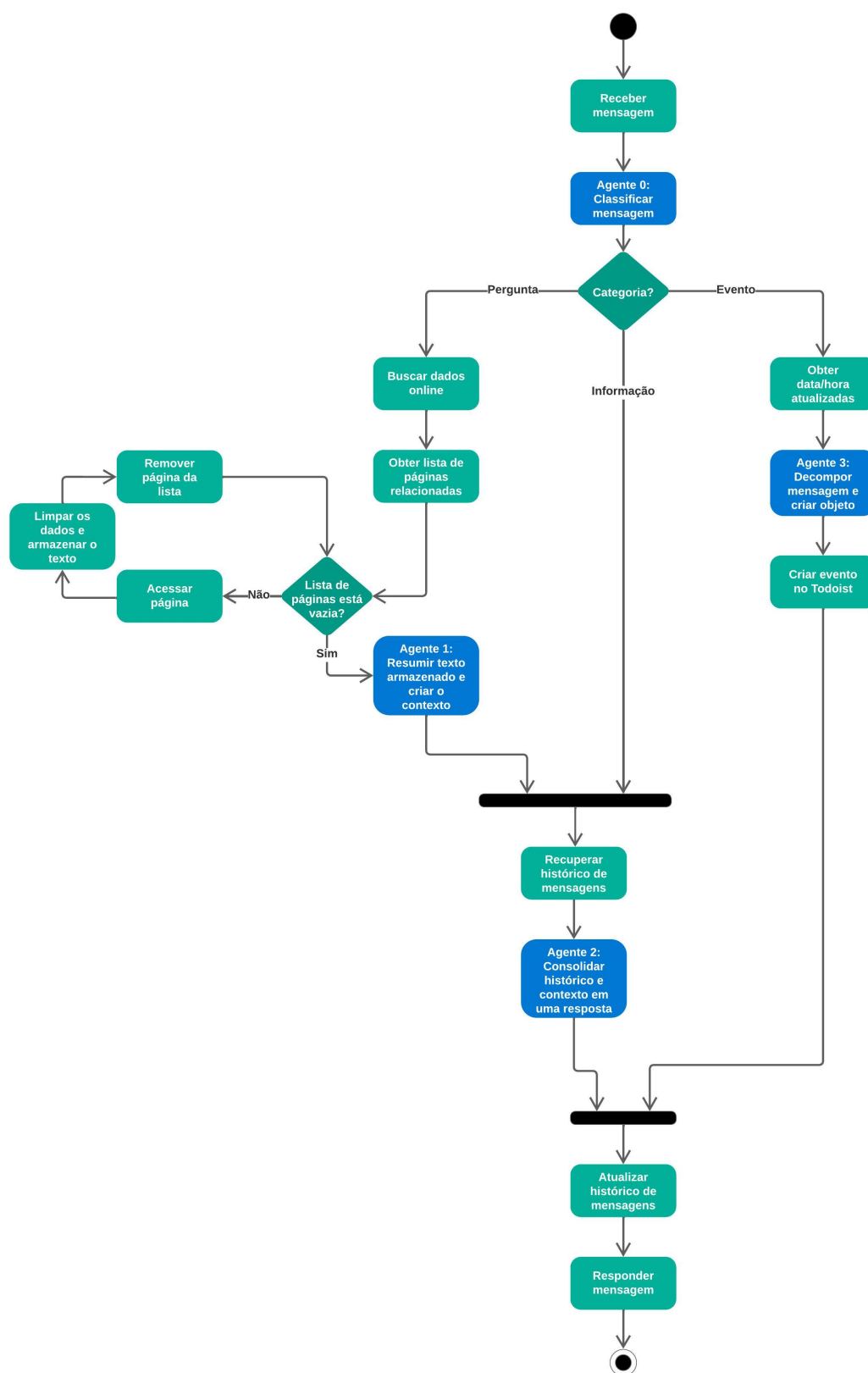


Figura 3.7: Diagrama de Atividades do Fluxo de Processamento de Mensagens

específico. As *informações* são dados que o usuário deseja fornecer para o assistente, podendo enriquecer o contexto e melhorar a qualidade das respostas fornecidas. Já os *eventos* se tratam de tarefas possuem data e hora específicas e precisam ser lembrados ao usuário;

**Agente 1:** é executado caso o *Agente 0* identifique que a solicitação feita pelo usuário se trata de uma *pergunta*. Este agente é responsável por aprimorar o contexto da pergunta feita pelo usuário, fornecendo um resumo criado a partir dos dados obtidos na busca feita através da API do *Serper*;

**Agente 2:** é responsável por fornecer a resposta final ao usuário. Identificado como *Alfred*, ele pode ser executado imediatamente após o *Agent 0*, caso a categoria da solicitação seja *informação* ou após o *Agente 1*, caso a solicitação se trate de uma pergunta. Esse agente pode se utilizar tanto dos dados fornecidos pelo contexto gerado pelo *Agent 1*, quanto do histórico de mensagens. Sua resposta é armazenada no histórico de mensagens para que possa servir de contexto em futuras interações;

**Agente 3:** é executado imediatamente após o *Agent 0* quando a solicitação do usuário corresponde à um *evento*. Seu objetivo é decompor a solicitação do usuário de modo que seja possível extrair informações necessárias para a criação de uma tarefa na API do *Todoist*. Para melhorar a qualidade de sua resposta este agente recebe informações atualizadas de data e hora.

### 3.3.4 Funcionalidades

O sistema conta com dois níveis de permissões de usuário o *básico* e o *administrador*. O usuário básico terá acesso apenas às funcionalidades básicas do sistema, como iniciar um chat e vincular sua conta do *Todoist*. Também possui o poder de gerir informações do seu próprio usuário, como alteração de senha. Já um usuário administrador, além de incluir todas funcionalidades de um usuário básico, também possui um grupo de ferramentas para gerir todos os usuários do sistema, podendo alterar informações, aprovar ou bloquear o acesso de outros usuários.

O sistema permite que o usuário crie uma conta fornecendo o primeiro nome, o

último nome, *email* e senha. Após solicitar o cadastro o usuário, deverá ser aprovado por um administrador para conseguir iniciar um *chat* com o assistente virtual. Uma vez aprovado, o usuário deverá se autenticar ao sistema para poder conversar com o assistente. Já para ter acesso às funcionalidades de criação de evento, o usuário deverá permitir o acesso à sua conta do *Todoist*, ao fornecer a chave de API disponível no próprio aplicativo do *Todoist*.

O principal fluxo do sistema é o de processamento de mensagens do usuário, representado pela Figura 3.7. Este fluxo representa a forma como cada um dos agentes é acionado e as etapas que englobam cada uma das categorias de mensagens mencionadas anteriormente, nele estão descritas cada uma das etapas necessária para o processamento das solicitações do usuário e mostra diferença entre cada um dos fluxos.

Como por exemplo, os fluxos de *informação* e *pergunta*, se unificam na etapa de recuperar os histórico de mensagem para executar o *Agente 2*, porém no fluxo de perguntas existe uma etapa de obtenção de contexto que inclui uma busca em tempo real, acesso à páginas *web* que contenham informações relevantes para a solicitação do usuário, até finalmente o *Agente 1* ser executado para resumir as informações obtidas e de fato criar um contexto.

Por outro, lado para a de criação de *eventos* o fluxo executado é totalmente separado dos demais, tendo uma etapa de enriquecimento de contexto fornecendo a data e hora da solicitação, para que então o *Agente 3* possa decompor a mensagem do usuário, para enfim se comunicar com a API do *Todoist* e efetivar a criação do evento. Todos os fluxos se unificam na etapa final para armazenar o histórico e enviar a resposta ao usuário.

# Capítulo 4

## Implementação da Arquitetura Proposta

Neste capítulo, será apresentada a implementação da solução proposta. Abordando as tecnologias utilizadas, as etapas e camadas necessárias para o desenvolvimento, além das funcionalidades do sistema detalhando os *prompts* necessários para executar as tarefas solicitadas.

### 4.1 Tecnologias

Para desenvolver a aplicação foi necessária a utilização de diversas tecnologias. No *front-end* foi utilizado o *framework* *Vue.js*<sup>1</sup> com *TypeScript* (TS)<sup>2</sup>, que por se tratar de uma linguagem fortemente tipada pode ajudar na detecção e prevenção de alguns erros. Já no *back-end* foi escolhido o *framework* para *Python*<sup>3</sup>, *FastAPI*<sup>4</sup>, que fornece uma interface simples e objetiva para a criação de API's Representational State Transfer (REST). Na camada de persistência foram utilizados o *PostgreSQL*<sup>5</sup> e o *Redis*<sup>6</sup>, que são responsáveis pelo armazenamento dos dados do sistema e *cache*,

---

<sup>1</sup><<https://vuejs.org/>>

<sup>2</sup><<https://www.typescriptlang.org/>>

<sup>3</sup><<https://www.python.org/>>

<sup>4</sup><<https://fastapi.tiangolo.com/>>

<sup>5</sup><<https://www.postgresql.org/>>

<sup>6</sup><<https://redis.io/>>



respectivamente. Vale destacar também a utilização do framework *Langchain*<sup>7</sup> que é responsável tanto pela integração com os *LLMs* da *OpenAI*, quanto pela integração com a *API* do *Serper*. O detalhes de cada tecnologia mencionada serão abordados nas subseções a seguir.

#### 4.1.1 TypeScript

O TS se trata de um *superset* fortemente tipado para *JavaScript* (JS) que tem como principal objetivo garantir a segurança de tipos das variáveis da aplicação. Com isso ele é capaz de melhorar a qualidade e legibilidade em comparação à aplicações JavaScript, além de apresentar também pequenas melhoras em relacionadas a erros (BOGNER; MERKEL, 2022).

Além dos benefícios citados anteriormente, por se tratar de um *superset*, o TS é capaz de manter interoperabilidade com código escrito em JS, então caso seja necessário utilizar alguma biblioteca desenvolvida em JS, está poderá ser integrada ao sistema de forma transparente.

#### 4.1.2 Vue.js

O *Vue.js* é um *framework* para desenvolvimento de interfaces de usuário, que devido a sua simplicidade, desempenho e flexibilidade se mostra uma boa opção de ferramenta para aplicações *front-end*. Uma das principais características do *Vue.js* é a sua arquitetura baseada em componentes, o que permite que cada parte da interface seja desenvolvida em módulos independentes, facilitando sua manutenção e reutilização.

#### 4.1.3 FastAPI

O *FastAPI* é um *framework Python* que tem como objetivo facilitar a criação de APIs REST. Por ter sido desenvolvido utilizando bibliotecas e *frameworks* consolidados de *Python*, como o *Starlette* e o *Pydantic*, disponibiliza uma API

---

<sup>7</sup><https://www.langchain.com/>

simples para criação de rotas através de *decorators* e gera a documentação da API de forma automática. O *FastAPI* têm se mostrado, então, uma boa alternativa para o desenvolvimento de aplicações *Python*.

#### 4.1.4 Langchain

O *Langchain* foi escolhido devido a sua capacidade de abstrair a integração com diversos LLM's, além de possuir um grande número de integrações com aplicações externas, como é o caso do *Serper* que foi utilizado para o enriquecimento de contexto dos *prompts* executados pela aplicação.

#### 4.1.5 PostgreSQL

O *PostgreSQL* é um Sistema de Gerenciamento de Banco de Dados (SGBD) de código aberto que vem sendo amplamente utilizado nas últimas décadas. Possui confiabilidade, robustez, grande variedade de tipos de dados e estruturas de dados avançadas. Isto aliado ao fato de ser gratuito para utilização mesmo em ambientes de produção, o *PostgreSQL* se destaca na escolha de um SGBD. Além disso, sua conexão em aplicações *Python* é bem simples e consolidada o que o torna uma excelente opção para este projeto.

#### 4.1.6 Redis

O *Redis* se trata de um banco de dados chave-valor, não relacional e em memória, que é amplamente utilizado para armazenar os *caches* das aplicações. A principal vantagem da utilização do *Redis* para este projeto será manter os *tokens* de autenticação e o contexto das conversas do usuário, tendo como principal objetivo reduzir a quantidade de acessos ao banco de dados relacional e melhorar o tempo de acesso à informações que precisam ser acessadas de forma constante.

## 4.2 Desenvolvimento

O projeto é constituído por duas aplicações, a primeira delas, o *front-end*, é responsável pela a parte visual do projeto e que será acessada pelo usuário durante suas interações e se conectar a segunda camada, o *back-end*. Já o *back-end* é responsável pelas regras de negócio do projeto, conexão com as camadas de persistência e integração com sistemas externos.

### 4.2.1 Front-End

O *front-end* foi desenvolvido utilizando *Vue.js* com TS, como mencionado anteriormente. De modo geral o *front-end* precisa possuir as páginas HyperText Markup Language (HTML) correspondentes a cada uma das interfaces que pode ser acessada pelo usuário, ser capaz fazer requisições a API para permitir que o usuário possa utilizar as funcionalidades do sistema e além disso deve ser capaz de armazenar algumas informações de forma local, como o *token* de usuário, para permitir interação do usuário com o sistema.

O sistema foi dividido em seis páginas que o usuário poderá ter acesso, onde cada uma delas é responsável por uma funcionalidade, são elas:

- Página Inicial;
- Página de Login;
- Página de Cadastro;
- Página de Mensagens;
- Página de Edição de Usuário;
- Página de Listagem de Usuários.

Já a aplicação foi organizada conforme mostra a Figura 4.1, onde a pasta *components* possui todos os componentes reutilizáveis da aplicação, como cabeçalhos e campos de texto, já na pasta *views* é onde ficam todas as páginas listadas anteriormente e é também onde são utilizados os componentes, a pasta *routers* é responsável por definir a rota de acesso de cada uma das páginas através do navegador e a pasta

*store* mantém todas as regras de armazenamento local de acordo com cada uma das funcionalidades ou componentes. Por fim, o arquivo *api.services.ts* é responsável por centralizar todas as chamadas de API realizadas pelo *front-end*. As interfaces de usuário e seus respectivos *endpoints* serão detalhados nas próximas seções.

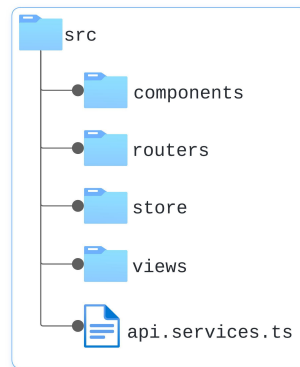


Figura 4.1: Organização do projeto do *front-end*

#### 4.2.2 Back-End

Como mencionado anteriormente, o *back-end* foi desenvolvido na linguagem *Python* utilizando o *framework FastAPI*, que por sua vez não possui um padrão arquitetural rígido, podendo ser adaptado de acordo com a necessidade de cada sistema. No caso do *Allfred*, pode-se dizer que, no *back-end*, a arquitetura utilizada é uma adaptação do *Model-View-Controller* (MVC), onde os *routers* e *schemas*, desempenham o papel comumente atribuído ao *view*, além disso foi incluída uma camada adicional, chamada *services*, responsável pela integração com aplicações externas, como mostra a Figura 4.2. Os papéis de cada uma dessas camadas são:

- **Schema:** camada onde são definidos os objetos aceitos pela API e também os objetos de retorno, nele é possível incluir validações específicas para cada um dos atributos, além de ser possível fazer alguma manipulação de valores no momento de criação do objeto, geralmente seus atributos possuem correlação com as colunas do banco de dados;
- **Router:** nele são definidos os *endpoints* disponíveis e os respectivos *schemas* de cada um deles, além disso também é responsável por validações mais iniciais como autenticação e permissionamento;

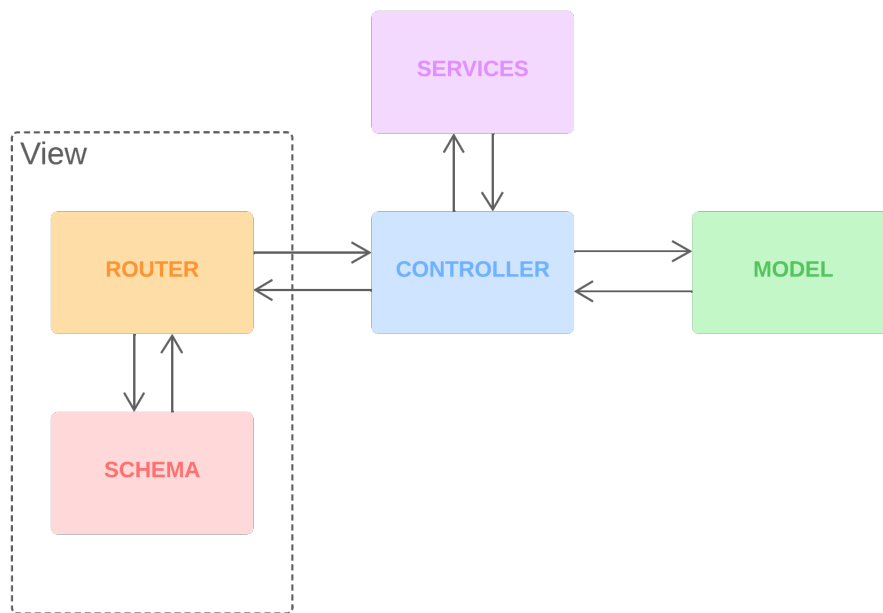


Figura 4.2: Diagrama arquitetural *Model-View-Controller*

- **Controller:** se trata de uma camada crucial do sistema já que se comunica com *routers*, *services* e *models*, sendo responsável pelas regras de negócios de cada um dos *endpoints* disponíveis;
- **Services:** é a camada de integração com serviços externos, onde no caso do *Allfred* é responsável por gerenciar as integrações com *ChatGPT*, *Serper* e *Todoist*;
- **Model:** responsável por mapear e abstrair todas as interações com o banco de dados, tem como principal objetivo garantir a recuperação e persistência dos dados do sistema.

Uma das vantagens de utilizar o *FastAPI* é o fato dele criar automaticamente uma documentação *Swagger* contendo as informações de todos os *endpoints* disponibilizados pelo sistema, vide a Figura 4.3, nela é possível ver todos os *endpoints* disponíveis na API do *Allfred*, separados por contexto. Para acessar essa documentação basta acessar o endereço onde a API está sendo executada seguido do caminho */docs*.

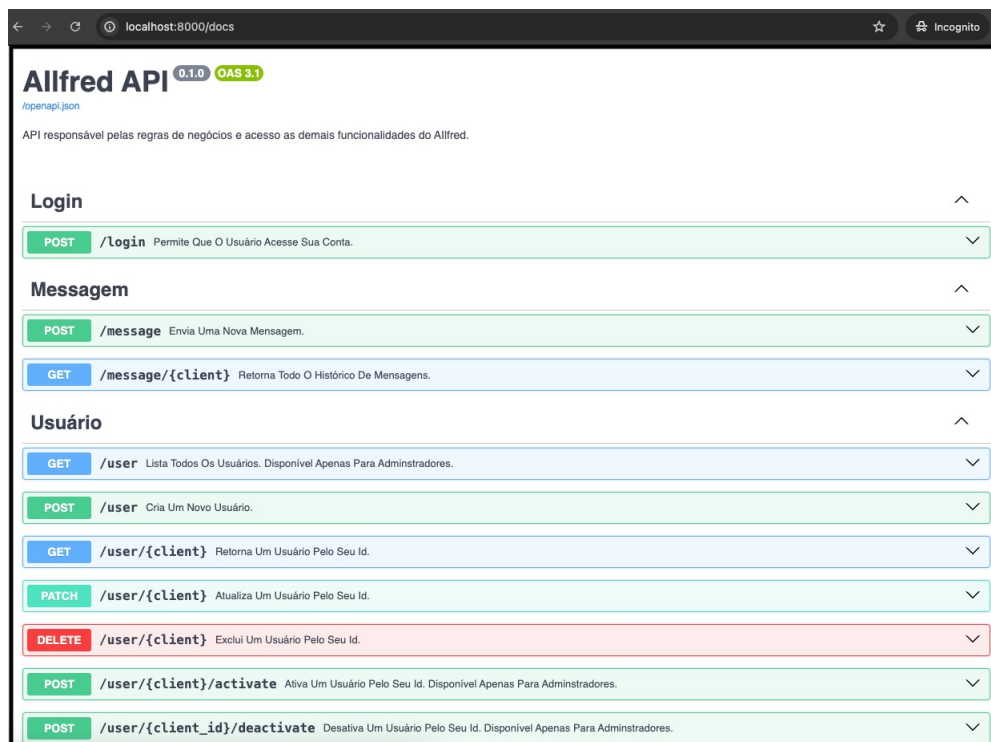


Figura 4.3: Documentação da API

## 4.3 Interfaces e Funcionalidades

O sistema possui seis páginas que são responsáveis por permitir que o usuário acesse cada uma das funcionalidades. Cada uma dessas páginas pode acessar um ou mais *endpoints* da API. A lista *endpoints* está disponível na documentação da API como mostra a Figura 4.3. As próximas subseções contêm detalhes da implementação e a correlação de cada tela com as suas respectivas funcionalidades.

### 4.3.1 Página Inicial

A página inicial é uma página simples com a explicação do projeto, porém não existem chamadas ao *back-end*. Essa página é responsável por apresentar o projeto de forma resumida, explicando seus objetivos principais e forma de funcionamento. Como mostra a Figura 4.4.



Figura 4.4: Tela Inicial

### 4.3.2 Cadastro de Usuário

A tela de cadastro é responsável pela criação de novos usuários nela o usuário precisa informar seu primeiro nome, último nome, *email*, que precisa ser único, e senha, como observado na Figura 4.5. Para realizar o cadastro o *front-end* irá executar uma requisição no *endpoint POST /user* e caso as informações sejam válidas o novo usuário será criado.

### 4.3.3 Login

Para realizar o *login* o usuário precisará informar seu *email* e senha, conforme a Figura 4.6, o *front-end* irá fazer uma requisição no *endpoint POST /login*, caso a conta esteja ativada e as informações estejam corretas, um *token* de sessão será criado e o usuário redirecionado para a tela de mensagens.

### 4.3.4 Configurações

A tela de configurações pode ser apresentada de duas formas, na visão de um usuário *básico*, conforme a Figura 4.7 onde ele poderá alterar sua senha e incluir

< home

nome Alfred sobrenome Pennyworth

email alfred@mail.com

senha \*\*\*\*\*

cadastre-se

já possui cadastro? [acesse sua conta](#)

Figura 4.5: Tela de Cadastro

< home

email alfred@mail.com

senha \*\*\*\*\*

Lembre-se de mim

entrar

novo por aqui? [cadastre-se](#)

Figura 4.6: Tela de Login

sua chave de API do *Todoist* ou na visão de um usuário *administrador*, onde ele terá acesso à lista de usuários, representada na Figura 4.8, onde um administrador poderá alterar informações de outros usuários, podendo também bloqueá-los ou desbloqueá-los.



O *endpoint* utilizado para atualizar as informações dos usuários é o *PATCH /user* e para listar os usuários, funcionalidade disponível apenas para *administradores*, é *GET /user*. Vale também ressaltar, que caso a chave de API do *Todoist* seja enviada, um projeto no *Todoist* será criado com o nome *Allfred* onde serão criados os eventos do usuário.



**editar usuário**

nome  sobrenome

email

senha

 chave de api todoist <sup>ⓘ</sup>

Figura 4.7: Tela de Edição de Usuário quando acessada por usuário básico

#### 4.3.5 Mensagens

Após ser aprovado por um usuário administrador, o novo usuário poderá começar a interagir com o assistente, a tela de mensagens como mostra a Figura 4.9, é composta por uma caixa de texto onde o usuário pode digitar e enviar suas mensagens, o histórico de perguntas e respostas fica acima da caixa de texto, onde em vermelho do lado direito estão as solicitações do usuário, enquanto as respostas fornecidas pelo *Allfred* ficam em azul do lado esquerdo.

Quando o usuário enviar uma mensagem o *front-end* irá enviar uma requisição ao *back-end* através do *endpoint POST /message*. O sistema irá verificar se o *token* de usuário é válido e caso seja o primeiro agente (*Agent 0*) será executado. De modo geral o *prompt* de um agente é composto três informações principais, a especialidade



nome	email	status	
Alfred Pennyworth	ailfred@mail.com	ativo	
Fellipe Bravo	fellipe@mail.com	ativo	
Gabriel Segobia	gabriel@mail.com	inativo	 
Paulo Roberto	paulo@mail.com	ativo	
Reinaldo Moraes	reinaldo@mail.com	inativo	 

Figura 4.8: Tela de Listagem de Usuários disponível apenas para administradores

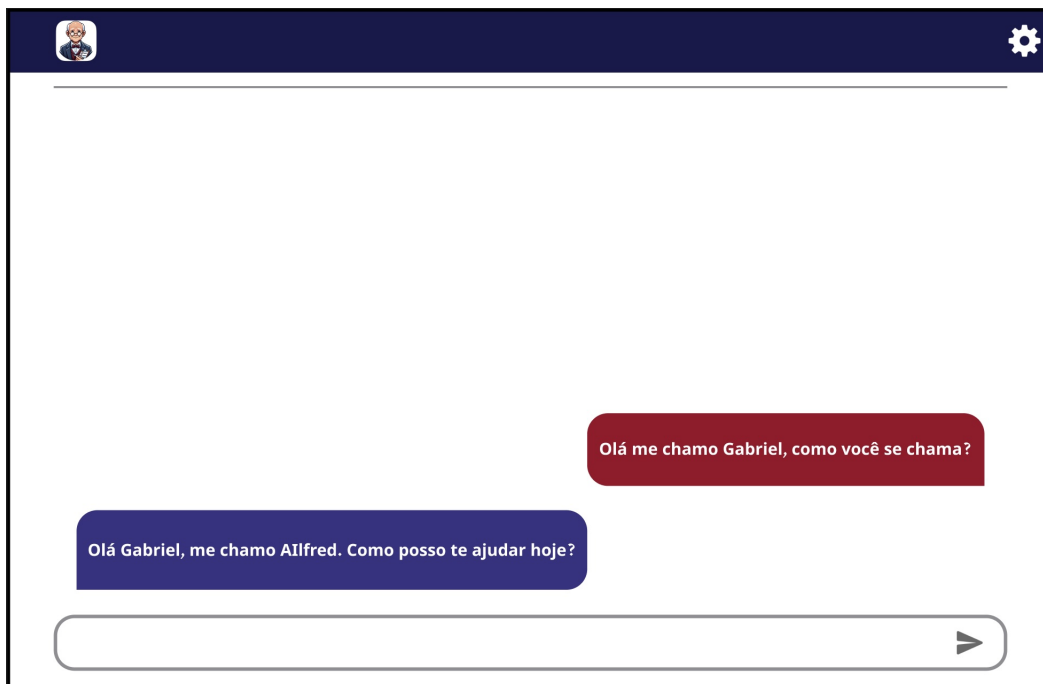


Figura 4.9: Tela de Mensagens

do agente que diz suas características e como ele deve executar sua tarefa, chamado de *backstory*, um *template* que informa o que é agente precisará executar e um *input* que de modo geral se trata da mensagem enviada pelo usuário, em alguns casos o agente precisará de informações extras, um contexto, que será adicionado ao

*backstory* conforme o código avança.

No caso o *Agent 0* é responsável por classificar a mensagem do usuário em três categorias *PERGUNTA*, *INFORMAÇÃO* ou *EVENTO*, como representado no Código *Prompt Agent 0*. Essa classificação é essencial para determinar o fluxo que será executado e quais agentes precisarão ser acionados.

---

**Prompt Agent 0:** Categoriza as requisições do usuário em *PERGUNTA*, *AFIRMAÇÃO* ou *EVENTO*

---

```
Data: input          /* Qual o nome do maior animal do mundo? */
Result: category    /* PERGUNTA */
llm ← chat-gpt-3.5-turbo
```

```
backstory ← "Você é um Agente especializado em categorizar instruções de
forma objetiva. As possíveis categorias são PERGUNTA, INFORMAÇÃO
ou EVENTO. A instrução a ser classificada estará representada entre os
delimitadores ###."
```

```
template ← "{backstory} Dada a seguinte instrução classifique-a
respondendo apenas a categoria: ###{input}###"
```

```
agent ← create-agent(llm, backstory, input, template)
category ← agent.execute()
```

```
return category
```

---

Caso a mensagem do usuário seja classificada como uma *PERGUNTA*, o sistema irá fazer uma busca no *Google* através da API do *Serper*, essa busca irá retornar uma lista de *links* que podem ajudar a responder a solicitação do usuário e em alguns casos pode retornar também uma resposta objetiva para pergunta. Em seguida, o sistema fará uma requisição em cada um dos links para retornar suas respectivas páginas HTML, dessas páginas serão extraídas as informações textuais contidas em cada uma delas, removendo *tags* e *scripts* que não são relevantes no processo, após essa etapa de busca e limpeza dos dados o *Agent 1* entra em ação.

Como mostra o Código *Prompt Agent 1*, o *Agent 1* é responsável por, dado o contexto adquirido na etapa anterior, resumir todas as informações obtidas focando em responder a pergunta feita pelo usuário, como não se trata de um agente responsável por responder de fato responder ao usuário a pergunta feita, esse resumo pode ser

mais longo e conter informações adicionais à respeito do tema. Ao final do processo o *Agente 1* irá fornecer uma resumo que traga informações relevantes referentes à pergunta do usuário, onde o mesmo será utilizado para aprimorar a resposta do agente responsável pela resposta final.

---

**Prompt Agent 1:** Responsável por enriquecer o contexto das perguntas feitas pelo usuário e responder com um resumo sobre o tema

---

**Data:** input, content, answer

**Result:** summary

*llm* ← chat-gpt-3.5-turbo

*backstory* ← "Você é um Agente especializado em extrair informações relevantes de um texto coletado na internet. Seu objetivo é resumir essas informações em uma resposta que contenha entre 3 e 5 parágrafos. O resumo deverá conter apenas informações que sejam capazes de responder a pergunta do usuário e todo o restante deverá ser descartado. Caso aja uma resposta correta objetiva ela estará delimitada por ==== e você deve considerá-la como a única resposta correta ao elaborar o resumo. Conteúdo a ser resumido estará delimitado entre <<<. A pergunta do usuário estará delimitada entre ###. Análise o seguinte o texto <<<{content}<<<. Resposta: ===={answer}====."

*template* ← "{backstory} Crie um resumo capaz de responder a seguinte pergunta: ###{input}###"

*agent* ← create-agent(*llm*, *backstory*, *input*, *template*)

*summary* ← agent.execute()

**return** summary

---

Por último, fechando o fluxo da categoria *PERGUNTA*, o *Agent 2* será executado. Por se tratar de um agente responsável por dar a resposta final ao usuário, ele irá utilizar tanto o contexto fornecido pelo *Agent 1*, quanto o histórico de interações do usuário para fornecer a resposta, como mostra o Código *Prompt Agent 2*.

Por outro lado, caso a mensagem do usuário seja classificada como uma *INFORMAÇÃO*, o fluxo executado pelo sistema será similar ao anterior, porém não haverá a necessidade de busca, limpeza e resumo dos dados, tendo em vista que logo após o *Agent 0* o *Agent 2* é executado, porém dessa vez sem a inclusão contexto como ocorre no fluxo de *PERGUNTA*.

---

**Prompt Agent 2:** Allfred: Responsável por consolidar o histórico de mensagens e contexto, quando houver, para responder ao usuário

---

**Data:** input, history, context

**Result:** answer

*llm* ← chat-gpt-3.5-turbo

*backstory* ← "Você é um assistente virtual chamado Allfred. Sua função é responder perguntas feitas pelo usuário. Sempre que houver um contexto ou histórico de mensagens eles devem ser lavados em consideração na hora de responder a pergunta. Se for possível responder apenas com as informações contidas no histórico, ignore o contexto. O contexto, quando houver estará delimitado entre <<<. O histórico, quando houver estará delimitado entre ===". A solicitação do usuário estará delimitada entre ###"

*template* ← "{backstory} Contexto: <<<{context}<<< Histórico: ==={history}=== Responda a seguinte solicitação do usuário: ###{input}###"

*agent* ← create-agent-with-memory(*llm*, *backstory*, *input*, *template*, *context*, *memory*)  
*answer* ← *agent*.execute-with-memory()

**return** answer

---

Por fim, caso a mensagem do usuário seja classificada como um *EVENTO*, para que o sistema possa prosseguir o usuário precisa ter cadastrado sua chave do *Todoist*, como explicado em 4.3.4, caso contrário o sistema irá informar que não pode prosseguir com essa solicitação.

Se o usuário possuir uma chave cadastrada, o *Agent 3* será responsável por decompor a solicitação do usuário, para extrair informações como assunto, data e hora. Para isso o agente irá receber como contexto data e a hora exatas da solicitação, dando a ele uma referência precisa na hora de extrair as informações do evento. Após seu processamento o *Agent 3* deverá responder apenas um *JavaScript Object Notation* (JSON), onde cada atributo será uma das informações solicitadas como mostra o Código *Prompt Agent 3*.

A partir dessas informações extraídas, o sistema irá criar um objeto capaz de solicitar a criação de uma tarefa na API do *Todoist* e com isso o irá retornar para o usuário que um evento foi criado em sua conta.

---

**Prompt Agent 3:** Responsável por decompor a solicitação do usuário e extrair os elementos necessários para a criação de um evento.

---

**Data:** input

**Result:** event

*llm* ← chat-gpt-3.5turbo

*now* ← datetime.now() /\* 14 de Julho de 2024 às 22:41:00 no  
 timezone America/Sao\_Paulo \*/

*backstory* ← "Você é um Agente especializado em decompor a solicitação do usuário afim de retornar um objeto JSON. Esse objeto deve possuir um atributo subject, um atributo date no formato yyyy-mm-dd e um atributo time no formato H:M:S e caso a solicitação do usuário não possua informações de tempo o horário padrão deve ser às 9h da manhã. Considere que hoje {now}"

*template* ← "{backstory} De acordo com o objeto definido anteriormente, responda apenas o objeto JSON que corresponda ao seguinte comando: {input}"

*agent* ← create-agent(*llm*, *backstory*, *input*, *template*)

*category* ← agent.execute()

**return** event

---

A Figura 4.10 mostra o exemplo de um chat que contempla todas as categorias de mensagens e a resposta fornecida pelo chat, já a Figura 4.11 mostra os eventos criados no calendário do *Todoist* após essas interações.

## 4.4 Comparação e Análise de Resultados

Nesta seção foram comparadas as respostas dadas pelo *AIfred* em relação aos modelos *GPT-3.5 Turbo* e modelo *GPT-4o*. Para simplificar a comparação foram feitas apenas solicitações classificadas como *PERGUNTAS* e no caso do *AIfred*, apenas o *Agente 1* e o *Agente 2* foram utilizados. Para se conectar aos demais modelos foram criados dois agentes genéricos que tem como única função enviar a pergunta feita para cada um desses modelos.

A Tabela 4.1 mostra a lista de perguntas realizadas e outras quatro colunas adicionais, com a resposta esperada e as respostas dadas pelo *AIfred*, *GPT-4o* e o

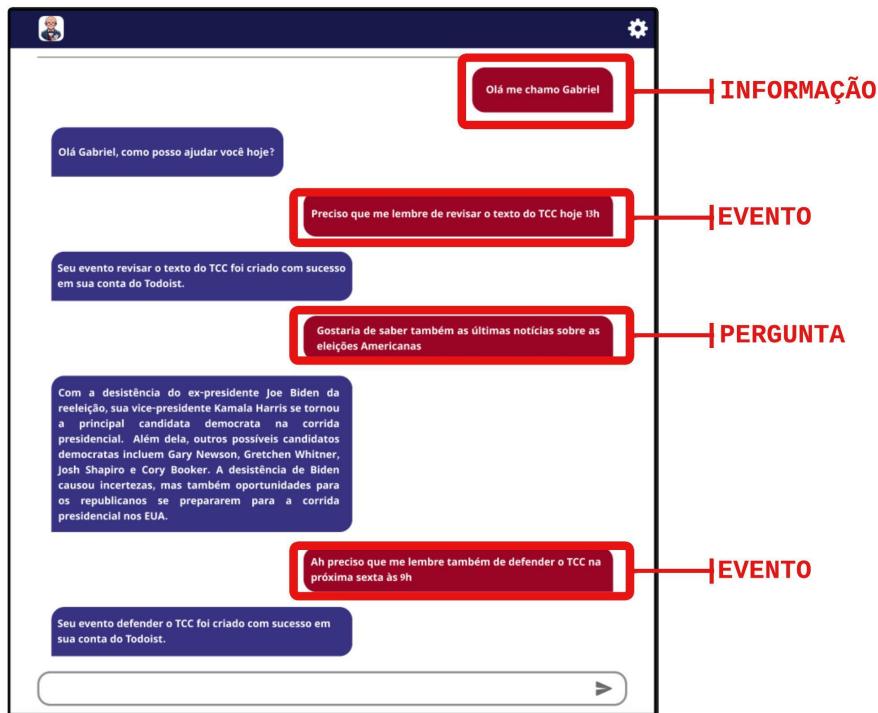


Figura 4.10: Tela de Mensagens indicando a classificação

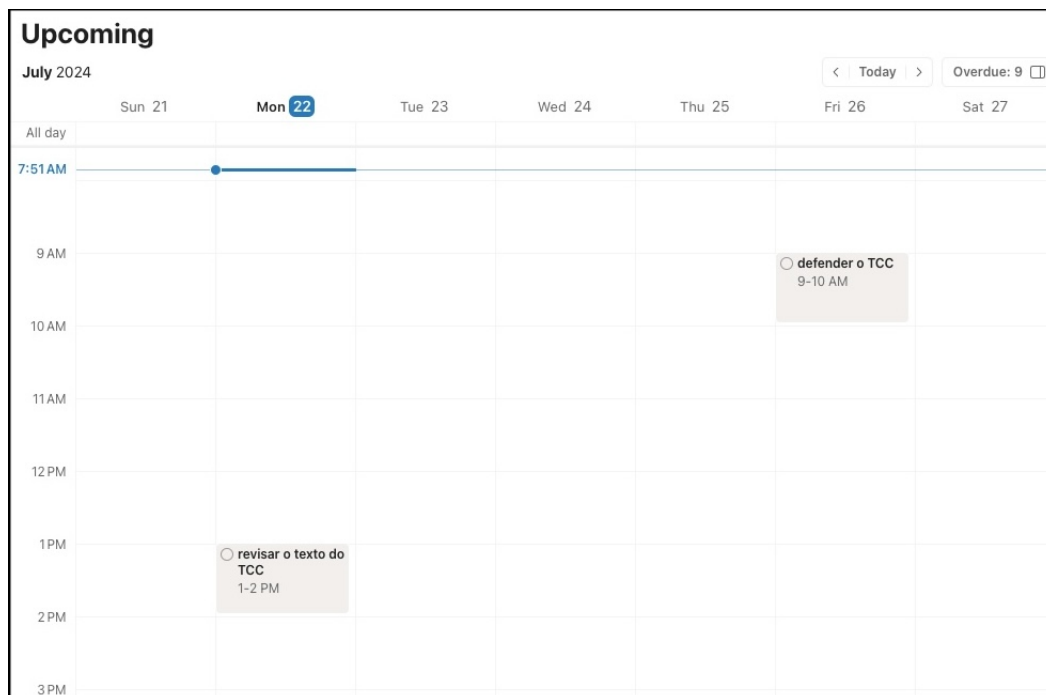


Figura 4.11: Calendário do *Todoist* após criação de eventos

*GPT-3.5 Turbo*. Vale ressaltar que os resultados foram obtidos em 16 de Julho de 2024, logo as respostas de informações sensíveis ao tempo podem sofrer alterações, outro detalhe importante é que algumas respostas foram reduzidas na tabela para

exibir apenas as informações objetivas referentes a pergunta feita, porém foi feito de modo que não prejudique o entendimento e análise das respostas.

Como é possível notar, em comparação ao *GPT-4o* e ao *GPT-3.5 Turbo*, o *Allfred* foi capaz de responder a maioria das perguntas de forma correta, tendo errado apenas a pergunta de número 5, onde o *GPT-3.5 Turbo* falhou em responder todas as perguntas sensíveis a tempo e até mesmo o *GPT-4o*, que é um modelo mais recente e robusto, por não ser capaz de obter informações em tempo real falhou em responder a maior parte das perguntas realizadas.

De qualquer forma, apesar do *Allfred* ter respondido corretamente a maior parte das perguntas, esses acertos não podem ser considerados constantes, já que eles dependem da qualidade das informações obtidas em tempo real utilizadas para enriquecer o contexto, podendo sofrer variações. Como não foi o foco deste trabalho fazer uma análise comparativa, nenhum método objetivo foi utilizado com os dados apresentados na Tabela 4.1, seus resultados ilustram apenas como a experiência de usuário pode ser aprimorada.



Tabela 4.1: Comparação de respostas fornecidas pelo *AIIfred*, *GPT-4o* e *GPT-3.5 Turbo*

Pergunta	Resposta Esperada	AIIfred	GPT-4o	GPT-3.5 Turbo
1 Quem é o presidente do Brasil hoje?	Lula	O presidente do Brasil atualmente é Lula.	Até minha última atualização em outubro de 2023, o presidente do Brasil é Luiz Inácio Lula da Silva[...]	O presidente do Brasil hoje é Jair Bolsonaro.
2 Quando o foi atentado contra Donald Trump e quem é o principal suspeito?	13 de Julho de 2024. Thomas Crooks.	O atentado contra Donald Trump ocorreu em 13 de julho de 2024[...] O principal suspeito do ataque é Thomas Mathew Crooks[...]	Até a minha última atualização em outubro de 2021, não houve nenhum relato de atentado significativo contra Donald Trump.[...]	Até a data presente não houve nenhum atentado contra o ex-presidente dos Estados Unidos Donald Trump.[...]
3 Qual o maior animal do mundo?	Baleia-azul	O maior animal do mundo é a baleia-azul (Balaenoptera musculus).	O maior animal do mundo é a baleia-azul (*Balaenoptera musculus*).[...]	A baleia azul é o maior animal do mundo.[...]
4 Quem jogou e qual foi o placar da final da Copa América?	Argentina 1 x 0 Colômbia	Na final da Copa América de 2024, a Argentina enfrentou a Colômbia, com um placar de 1 a 0 na prorrogação a favor da Argentina.	A última edição da Copa América foi realizada em 2021, e a final foi disputada entre o Brasil e a Argentina.[...] A Argentina venceu o Brasil por 1 a 0[...]	A final da Copa América de 2021 foi disputada entre Argentina e Brasil. O Brasil venceu por 1 a 0, com gol de Richarlison.
5 Quanto foi o último jogo do Vasco?	Vasco 2 x 0 Corinthians	Desculpe, mas o placar específico do último jogo do Vasco contra o Cuiabá não foi mencionado no histórico.[...]	Lamento, mas eu não tenho acesso a informações em tempo real. Sugiro que você verifique os resultados mais recentes do Vasco da Gama[...]	O último jogo do Vasco foi contra o Atlético-GO, no dia 01 de maio de 2021, e o Vasco venceu por 2 a 1.[...]

# Capítulo 5

## Conclusões

Esse capítulo irá apresentar as considerações finais referentes à proposta e desenvolvimento da ferramenta. Além disso, serão levantadas as limitações da ferramenta, possíveis melhorias e projetos futuros que podem complementar o projeto desenvolvido neste trabalho.

### 5.1 Considerações Finais

A expectativa do surgimento de ferramentas que utilizem LLM's para executar tarefas objetivas para o usuário se mostra cada vez maior. Recentemente, a *Apple* anunciou que em sua próxima atualização de sistemas operacionais irá fornecer diversas funcionalidades que utilizam LLM para executar tarefas mais complexas e até mesmo a *Siri* irá receber atualizações nesse sentido. Outras empresas como *Google* e *Microsoft* também estão trabalhando para aprimorar seu ecossistema com essas tecnologias.

Este projeto teve como principal objetivo demonstrar que é possível criar um assistente virtual utilizando LLM. Pode-se concluir que o trabalho não apenas foi capaz de responder perguntas e de executar tarefas objetivas, mas também de melhorar as respostas fornecidas pelo modelo utilizado.

## 5.2 Limitações e Trabalhos Futuros

Apesar dos bons resultados obtidos no desenvolvimento do *Allfred*, foram identificados alguns pontos que poderiam melhorar ainda mais a qualidade do sistema e principalmente a experiência do usuário, tornando as interações mais fluidas e podendo aumentar o engajamento do usuário com a plataforma.

O primeiro ponto seria, realizar algumas operações como chamadas de API e busca de dados externos de forma assíncrona. Por serem necessárias diversas operações dessa natureza, algumas interações do usuário podem demorar para obter um retorno. Isso acontece principalmente pelo fato de todas as operações de *Input/Output* (I/O) serem executadas de forma sequencial, torná-las assíncronas traria uma grande melhoria para a experiência do usuário.

Também poderiam ser criados novos agentes responsáveis por refinar ainda mais as etapas do processo e principalmente melhorar as respostas fornecidas, já que apesar de ter apresentado uma boa margem de acertos, o *Allfred* respondeu algumas perguntas de forma muito objetiva e até mesmo alguns ajustes nos *prompts* existentes poderiam ajudar a melhorar as respostas para que elas pareçam mais naturais.

Além disso, diversas funcionalidades poderiam ser incorporadas ao projeto, como a possibilidade de integrar a API do *Allfred* como algum sistema móvel de mensagens e permitir com o que o usuário interaja com ele diretamente através desses aplicativos e até mesmo incluir a possibilidade do usuário interagir por mensagens de voz.

Outra funcionalidade interessante seria a possibilidade de ter mais integrações focadas em tarefas objetivas e talvez uma versão do *Allfred* voltado para casas inteligentes. Por fim, também seria possível testar como o sistema funcionaria utilizando um LLM local, sem a necessidade de se conectar com um modelo via API, isso provavelmente reduziria o tempo de resposta para o usuário e o custo para execução do sistema.

# Referências

- BENGIO, Y. *Deep Learning*. London, England: MIT Press, 2016. (Adaptive Computation and Machine Learning series).
- BOGNER, J.; MERKEL, M. To type or not to type? a systematic comparison of the software quality of javascript and typescript applications on github. p. 658–669, 2022.
- BROWN, T. B. et al. *Language Models are Few-Shot Learners*. 2020. Disponível em: <<https://arxiv.org/abs/2005.14165>>.
- DOUGLAS, M. R. Large language models. 2023.
- FILHO, E. S. et al. An online platform for covid-19 diagnostic screening using a machine learning algorithm. *Revista da Associação Médica Brasileira*, v. 69, n. 4, p. e20221394, Apr 2023.
- GANG, P. et al. *A Visual Kinematics Calibration Method for Manipulator Based on Nonlinear Optimization*. 2020. Disponível em: <<https://arxiv.org/abs/2005.08420>>.
- GÄRTLER, M.; SCHMIDT, B. Practical challenges of virtual assistants and voice interfaces in industrial applications. In: *Hawaii International Conference on System Sciences*. [s.n.], 2021. Disponível em: <<https://api.semanticscholar.org/CorpusID:232414445>>.
- GUAN, Y. et al. *Intelligent Virtual Assistants with LLM-based Process Automation*. 2023. Disponível em: <<https://arxiv.org/abs/2312.06677>>.
- HOY, M. B. Alexa, siri, cortana, and more: An introduction to voice assistants. *Medical Reference Services Quarterly*, Med Ref Serv Q, v. 37, n. 1, p. 81–88, 2018.
- IBM. Ai hallucinations. *IBM Topics*, 2024. Accessed: 2024-07-14. Disponível em: <<https://www.ibm.com/topics/ai-hallucinations>>.
- JI, Z. et al. Survey of hallucination in natural language generation. *arXiv preprint arXiv:2202.03629*, 2022.
- JURAFSKY, D.; MARTIN, J. H. *Speech and Language Processing*. 3rd. ed. [s.n.], 2019. Draft available at *Speech and Language Processing*. Disponível em: <<https://web.stanford.edu/~jurafsky/slp3/>>.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems*. [s.n.], 2012. p. 1097–1105. Disponível em: <<https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>>.

KUMAR, P. Large language models humanize technology. arXiv (Cornell University), 2023.

Microsoft Corporation. *Microsoft's Commitment to Sustainability*. 2024. Disponível em: <<https://www.microsoft.com/en-us/sustainability>>.

OPENAI et al. *GPT-4 Technical Report*. 2024. Disponível em: <<https://arxiv.org/abs/2303.08774>>.

PARK, S.; KULKARNI, C. *Thinking Assistants: LLM-Based Conversational Assistants that Help Users Think By Asking rather than Answering*. 2024. Disponível em: <<https://arxiv.org/abs/2312.06024>>.

RADFORD, A.; NARASIMHAN, K. Improving language understanding by generative pre-training. In: . [s.n.], 2018. Disponível em: <<https://api.semanticscholar.org/CorpusID:49313245>>.

RADFORD, A. et al. Language models are unsupervised multitask learners. In: . [s.n.], 2019. Disponível em: <<https://api.semanticscholar.org/CorpusID:160025533>>.

RAY, P. P. Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems*, v. 3, p. 121–154, 2023. ISSN 2667-3452. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S266734522300024X>>.

ROGERS, A.; KOVALEVA, O.; RUMSHISKY, A. *A Primer in BERTology: What we know about how BERT works*. 2020. Disponível em: <<https://arxiv.org/abs/2002.12327>>.

STRUBELL, E.; GANESH, A.; MCCALLUM, A. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019. Disponível em: <<https://arxiv.org/abs/1906.02243>>.

SUNDAR, A.; HECK, L. *Multimodal Conversational AI: A Survey of Datasets and Approaches*. 2022. Disponível em: <<https://arxiv.org/abs/2205.06907>>.

TECMUNDO, R. *Tay: como o Twitter conseguiu corromper a IA da Microsoft em menos de 24 horas*. 2016. Disponível em: <<https://www.tecmundo.com.br/inteligencia-artificial/102782-tay-twitter-conseguiu-corromper-ia-microsoft-24-horas.htm>>.

VASWANI, A. et al. Attention is all you need. 2017.

VU, M. D. et al. *GPTVoiceTasker: LLM-Powered Virtual Assistant for Smartphone*. 2024. Disponível em: <<https://arxiv.org/abs/2401.14268>>.