UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUIS FELIPE MUNIZ DE ANGELO

TUTOR DE IA CONVERSACIONAL PARA O APRENDIZADO DE INGLÊS

BAURU
Novembro/2025

LUIS FELIPE MUNIZ DE ANGELO

TUTOR DE IA CONVERSACIONAL PARA O APRENDIZADO DE INGLÊS

Trabalho de Conclusão de Curso do Curso de Bacharelado em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Dr. João Paulo Papa

BAURU
Novembro/2025

Angelo, Luis Felipe Muniz de
D284t Tutor de IA Conversacional para o Aprendizado de Inglês / Luis Felipe Muniz de Angelo. – Bauru, 2025
62 p.

Trabalho de Conclusão de Curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (UNESP), Faculdade de Ciências, Bauru
Orientador: Dr. João Paulo Papa

1. Inteligência artificial. 2. Processamento de linguagem natural. 3. Arquiteturas de agentes. 4. Tutores de IA. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Dados fornecidos pelo autor(a).

Luis Felipe Muniz de Angelo

Tutor de IA Conversacional para o Aprendizado de Inglês

Trabalho de Conclusão de Curso do Curso de Bacharelado em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Dr. João Paulo Papa
Orientador
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Dra. Simone das G. Domingues Prado
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Dr. André Luis Debiasso Rossi
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Bauru, 11 de Novembro de 2025.

Dedico este trabalho a Deus por nortear minha vida, aos meus pais pelo apoio incondicional, e aos meus amigos que acompanharam minha jornada.

Agradecimentos

Agradeço primeiramente à minha família, pela oportunidade de poder ir atrás do meu sonho.

Agradeço ainda aos meus amigos, que estiveram comigo e compreenderam todas as minhas ausências.

À Ana Lara, por ser minha companheira em todos os momentos.

Aos professores do departamento de computação da UNESP Bauru, por todos os ensinamentos ao longo do curso.

Ao Prof. Dr. João Paulo Papa, pelo apoio e orientação no desenvolvimento deste trabalho.

E a todos que, direta ou indiretamente, contribuíram para a minha formação.

"Confia no Senhor de todo o teu coração e não te estribes no teu próprio entendimento. Reconhece-o em todos os teus caminhos, e ele endireitará as tuas veredas."
Provérbios 3:5-6

Resumo

Este trabalho apresenta o desenvolvimento de um tutor de inteligência artificial conversacional para o aprendizado de inglês, focado em falantes de português. O projeto aborda as limitações das abordagens puramente neurais no ensino, que carecem de precisão e personalização, bem como o alto custo e latência da arquitetura multiagente tradicional. A solução proposta é uma arquitetura composta por um pipeline de processamento em camadas implementado em *LangGraph* para análise gramatical, e um agente central unificado (AFM) que adota o paradigma *Chain-of-Agents* (CoA). Este agente simula internamente a colaboração de múltiplos agentes com diferentes papéis a fim de estabelecer raciocínios complexos com eficiência. O sistema integra tanto ferramentas simbólicas como o *LanguageTool* para validação gramatical determinística, quanto neurais para análise semântica, sendo implementado como um aplicativo móvel cliente usando *React Native*, e servidor em *FastAPI*. A eficiência da arquitetura foi validada frente aos pilares do framework *OAgents* e como resultado o trabalho entrega uma ferramenta pedagógica de baixa latência e com menor custo do que a abordagem multiagente tradicional, sendo capaz de oferecer correções instantâneas, proporcionando um ambiente seguro para a prática conversacional.

Palavras-chave: Inteligência artificial. Processamento de linguagem natural. Arquiteturas de agentes. Tutores de IA.

Abstract

This work presents the development of a conversational artificial intelligence tutor for English language learning, focused on Portuguese speakers. The project addresses the limitations of purely neural approaches in education, which lack precision and personalization, as well as the high cost and latency of traditional multi-agent architectures. The proposed solution is an architecture composed of a layered processing pipeline implemented using LangGraph for grammatical analysis, and a unified central agent (AFM) that adopts the Chain-of-Agents (CoA) paradigm. This agent internally simulates the collaboration of multiple agents with different roles to efficiently perform complex reasoning. The system integrates both symbolic tools, such as LanguageTool for deterministic grammatical validation, and neural tools for semantic analysis. It is implemented as a client mobile application using React Native and a server in FastAPI. The architecture's efficiency was validated against the pillars of the OAgents framework. As a result, the work delivers a pedagogical tool with low latency and lower cost than the traditional multi-agent approach, capable of offering instant corrections and providing a safe environment for conversational practice.

Keywords: Artificial Intelligence. Natural Language Processing. Agent Architectures. AI Tutors.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

AFM	Agent Foundation Model
API	Application Programming Interface
CALL	Computer-Assisted Language Learning
CoA	Chain-of-Agents
CoT	Chain-of-Thought
DTO	Data Transfer Object
ER	Entidade relacionamento
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
JWT	JSON Web Token
LLM	Large Language Model
LPU	Language Processing Unit
MALL	Mobile-Assisted Language Learning
MRKL	Modular Reasoning, Knowledge, and Language
MVVM	Model-View-ViewModel
ORM	Object-Relational Mapping
PLN	Processamento de linguagem natural
ReAct	Reasoning and Acting
REST	Representational State Transfer
SFT	Supervised Fine-Tuning
SQL	Structured Query Language
STT	Speech-to-Text
TTS	Text-to-Speech
URL	Uniform Resource Locator
URI	Uniform Resource Identifier

Sumário

1 Introdução

O domínio da língua inglesa representa um diferencial estratégico em um mundo globalizado, sendo frequentemente considerado requisito básico para o desenvolvimento profissional e acadêmico (Ilyosovna, 2020). Contudo, o processo para alcançar a proficiência em uma segunda língua enfrenta desafios significativos, incluindo a necessidade de prática contínua e de uma grande exposição ao idioma (Gilakjani; Sabouri, 2016).

Para isso, a evolução tecnológica tem transformado o aprendizado de idiomas nas últimas décadas. A abordagem tradicional, assistida por computador, evoluiu do *Computer-Assisted Language Learning (CALL)* (Levy, 1997) para o *Mobile-Assisted Language Learning (MALL)* (Shortt et al., 2023) após a popularização dos *Smartphones*. Ainda mais recentemente, com a ascensão dos *Large Language Models (LLMs)*, uma nova geração de sistemas conhecida como tutores de IA (*AI Tutors*) surgiu. Estes sistemas demonstram superioridade estatisticamente significativa sobre as abordagens anteriores, oferecendo conversação natural e feedback instantâneo, além de oferecer uma personalização muitas vezes ausente nos sistemas precursores, sendo essencial para o processo pedagógico (Lyu; Lai; Guo, 2025)(Abusahyon et al., 2023).

No entanto, o desenvolvimento de tutores de IA eficientes enfrenta um desafio em sua arquitetura. Abordagens puramente neurais, utilizadas nos LLMs, operam como sistemas conhecidos como "caixa-preta", em que não é possível verificar seus processos de decisão, apresentando limitações quando há a necessidade de precisão, podendo fornecer informações incorretas (Marcus, 2018)(Shahzad et al., 2025), falhando assim no principal propósito de um sistema de aprendizado. Por outro lado, sistemas puramente simbólicos, baseados em regras, são excessivamente rígidos e não conseguem capturar a fluidez da conversação natural (Mira, 2008).

Para superar essas limitações, novas arquiteturas de agentes que buscaram integrar os sistemas neurais com os simbólicos, como o MRKL (Karpas et al., 2022), que por sua vez evoluiu para paradigmas que integram raciocínio e ação, como o ReAct (Yao et al., 2023). Para lidar com problemas ainda mais complexos, ganharam destaque arquiteturas multiagentes que decompõem tarefas entre agentes especialistas. Embora as arquiteturas multiagente demonstrem um desempenho superior ao de agentes únicos (Hadfield et al., 2024), elas introduzem uma sobrecarga computacional significativa, com um aumento da latência e dos custos a cada interação entre os agentes especialistas.

Neste cenário, um estudo recente introduziu o framework *Chain-of-Agents* (CoA),

que propõe uma solução mais eficiente (Li et al., 2025). Ao invés de operar com múltiplos agentes, um único agente, chamado *Agent Foundation Model* (AFM) é treinado para simular internamente a colaboração entre os agentes especializados. O AFM atua como um único agente que adota diferentes papéis, como o de planejador, executor e crítico, replicando o comportamento de um sistema multiagente de forma otimizada.

O presente trabalho se insere nesta vertente supracitada de novos paradigmas de agentes. Propõe-se o desenvolvimento de um tutor de IA conversacional composto por um *pipeline* de processamento para análise gramatical em multiplas camadas (análise sintática e semântica) e um agente central (AFM), que utiliza o resultado do pipeline de processamento juntamente com o perfil e histórico do usuário. O objetivo é criar uma ferramenta de aprendizado de inglês para falantes de português que seja, simultaneamente, pedagógica, computacionalmente eficiente e capaz de raciocínio complexo.

1.1 Problema

A problemática central deste trabalho reside na discrepância entre os sistemas neurais atuais e as abordagens pedagógicas modernas no ensino de novas línguas. Enquanto os modelos neurais frequentemente ignoram o contexto individual do aprendiz (Marcus; Davis, 2019), o ensino eficaz exige a integração de instruções gramaticais tanto implícitas (fornecidas em contextos naturais de conversação) quanto explícitas (com explicação direta das regras), conforme evidenciado por Ellis e Shintani (2019).

Shahzad et al. (2025) por meio de um *framework*, analisaram os desafios do uso de LLMs no ensino superior, identificando limitações observadas em LLMs problemáticas no contexto educacional, destacando-se:

1. Controle de qualidade: Os LLMs frequentemente fornecem informações imprecisas ou incorretas, o que afeta a qualidade das informações fornecidas, resultando em falta de confiança nos dados produzidos por IA.

2. Limitações no aprofundamento da aprendizagem: A educação baseada exclusivamente em LLMs não consegue fornecer aos estudantes a profundidade de conhecimento, análise rigorosa e aprendizagem focada que professores qualificados oferecem. Os LLMs não capturam as pequenas características que distinguem um estudante de outro, ignorando preferências de aprendizagem, padrões e problemas específicos, sugerindo que falta aos LLMs a capacidade de educação personalizada.

Por outro lado, as abordagens puramente simbólicas, baseadas exclusivamente em regras gramaticais e bancos de conhecimento, enfrentam também desafios igualmente significativos que resultam em interações pobres em contexto que não refletem o uso verdadeiro da língua em conversas complexas (Mira, 2008).

No que tange os paradigmas modernos, o ReAct (Yao et al., 2023) ao incorporar o MRKL, se tornou o padrão ouro de utilização, sendo utilizado pelo framework LangGraph como parte do pacote de componentes prontos para um agente que utiliza de conhecimento externo. (Langgraph, 2025) Entretanto, agentes únicos como o ReAct, embora iterativos, carecem da especialização de arquiteturas multiagentes, que por sua vez, embora mais robustas que o sistema de agente único, introduzem latência e custos computacionais altos devido a uma grande necessidade de chamadas de API para orquestrar os agentes.

Em síntese, a problemática central deste estudo consiste em desenvolver um sistema que mitigue as limitações das abordagens puramente neurais, simbólicas e multiagente tradicionais, culminando nos objetivos a seguir.

1.2 Objetivos

1.2.1 Objetivo geral

Desenvolver e validar um tutor de IA conversacional, por meio de um aplicativo móvel, utilizando uma arquitetura composta com base no paradigma CoA com a finalidade de proporcionar uma experiência fluida e pedagogicamente eficaz de conversação para o falante nativo de português aprendiz de inglês.

1.2.2 Objetivos específicos

1. Implementar um *pipeline* de processamento utilizando LangGraph para análise linguística em camadas, integrando:
    - Conhecimento estruturado para verificação sintática via LanguageToolAPI
    - Um agente especialista para análise semântica e contextual
2. Desenvolver um agente central unificado (AFM) que implementa o paradigma de raciocínio CoA, capaz de:
    - Receber as análises do pipeline

- Gerar dinamicamente planos de ação
- Realizar autocrítica
- Invocar ferramentas externas quando necessário

1.3 Organização do trabalho

O presente trabalho está estruturado em quatro capítulos, descritos de forma breve a seguir:

- O Capítulo 1, se trata do capítulo atual, em que foi apresentado a introdução, o contexto, a evolução dos tutores de IA, as arquiteturas de agentes modernas, a problemática central e os objetivos que orientam o projeto.
- O Capítulo 2 justifica a relevância para a realização deste projeto.
- O Capítulo 3 fundamenta os detalhes teóricos necessários para o desenvolvimento do sistema proposto.
- O Capítulo 4 detalha a metodologia de desenvolvimento do sistema proposto.
- O Capítulo 5 determina os resultados do trabalho.
- O Capítulo 6 estabelece a conclusão do trabalho.

2 Justificativa

O desenvolvimento de tutores de IA conversacionais representa um campo emergente na intersecção entre inteligência artificial e educação. Porém, a implementação através de arquiteturas de agentes modernas, que incluem desde a multiagente até paradigmas recentes como o *CoA*, permanece pouco explorada em aplicações educacionais práticas, principalmente no contexto brasileiro, no que tange a prática de conversação para o aprendizado de inglês como segunda língua.

O presente trabalho busca contribuir para preencher esta lacuna por meio de uma aplicação prática que oferece:

- Disponibilidade contínua: A onipresença dos dispositivos móveis, com 98,8% dos brasileiros acessando a internet por celulares, fenômeno acompanhado pelo declínio do acesso à internet pelo uso de computadores de 63,2% em 2016 para 34,2% em 2023 (Ibge, 2024), possibilita o que Buchner e Andujar (2019) denominam como aprendizado informal (*informal learning*). Esse aprendizado caracteriza-se como a obtenção de conhecimento fora de ambientes estruturados, independente de tempo e espaço. O que se aplica a esse projeto por meio do desenvolvimento de um aplicativo móvel. Ainda, o paradigma adotado possibilita que o agente central possa acessar a memória de longo prazo do usuário, proporcionando uma continuidade no contexto mesmo que o usuário fragmente suas sessões de aprendizado, compondo assim, uma característica essencial para o estilo de vida atual móvel e conectado.
- Personalização do aprendizado: Como apontado por Escueta et al. (2017), as intervenções de aprendizado assistido por computador mais relevantes são aquelas que adaptam o conteúdo às necessidades individuais dos estudantes por meio do aprendizado personalizado, permitindo que avancem no próprio ritmo e recebam feedback imediato, o que se aplica neste projeto ao ser incorporado a um sistema de conversação aplicado em dispositivos móveis para o ensino de línguas com feedback imediato fornecido pelo agente central (Bernacki; Greene; Crompton, 2020).
- Interação por voz: A escolha da voz como meio primário não só visa a naturalidade da conversa como também se ampara em evidências de que chatbots com capacidade de interação vocal demonstram maior impacto positivo no aprendizado do que os chatbots somente com texto. (Lyu; Lai; Guo, 2025).

Para quantificar o impacto dessas tecnologias, a metanálise recente de Lyu, Lai

e Guo (2025) combinou os resultados de 31 pesquisas empíricas quase-experimentais, comparando o aprendizado de uma segunda língua (L2) com e sem o uso de chatbots em contextos educacionais diversos. A métrica utilizada para medir a magnitude da diferença entre os grupos foi o **g** de Hedges, uma medida padronizada do tamanho do efeito baseada no **d** de Cohen que corrige vieses de amostras pequenas. O estudo identificou que esses fatores possuem efeito moderador na efetividade de chatbots para o aprendizado de L2, conforme resumido na Tabela 1.

Tabela 1 – Efeito moderador de características de chatbots no aprendizado de segunda língua

Fator moderador	Condição	g de Hedges
Mobilidade	Dispositivo móvel	0,790***
	Apenas computador	0,189*
Modalidade	Suporte a voz	0,809***
	Apenas texto	0,425***
Tecnologia	IA generativa	0,833***
	Baseado em regras	0,473***

Nota. Valores de **g** (tamanho do efeito calculado) próximos a 0.5 indicam um efeito médio.
*$p < 0{,}05$. ***$p < 0{,}001$.
Fonte: Adaptado de Lyu et al. (2025).

Horwitz, Horwitz e Cope (1986) identificam a ansiedade no aprendizado de línguas como obstáculo significativo, particularmente em contextos de prática oral, nos quais o medo de cometer erros inibe frequentemente a participação do aprendiz de L2. Sistemas conversacionais como o proposto neste trabalho possuem a capacidade de mitigar essa questão. Conforme discutido por (Zhang; Huang, 2024), agentes baseados em LLMs podem reduzir a ansiedade conversacional ao oferecer ambientes de práticas mais seguros. Nesses ambientes, o aprendiz pode cometer erros sem receio de julgamento social, ajudando a superar a ansiedade ou apreensão linguística, incentivando a prática da língua em estudo.

2.1 Trabalhos relacionados

2.1.1 Estudos com VCAs educacionais

Dentre os trabalhos relacionados na área, se enquadram os estudos no campo dos agentes VCAs (*voice-controlled conversational agents*)

Lee e Jeon (2024), em seu estudo exploratório sobre a percepção de agentes conversacionais (VCAs) por 67 crianças da Coreia no aprendizado de língua estrangeira em

aulas de inglês, descobriu que 71,6% delas perceberam que o agente possuía características humanas e era capaz de uma comunicação precisa em um segundo idioma. O estudo também observou que algumas dessas crianças, que tinham muito pouco contato com falantes nativos de inglês, perceberam que o agente poderia satisfazer sua necessidade de ter conhecidos fluentes em inglês. Este resultado fundamenta a viabilidade de agentes conversacionais como parceiros de aprendizado, principalmente no contexto de aprendizes de L2 que possuem limitado acesso a falantes nativos.

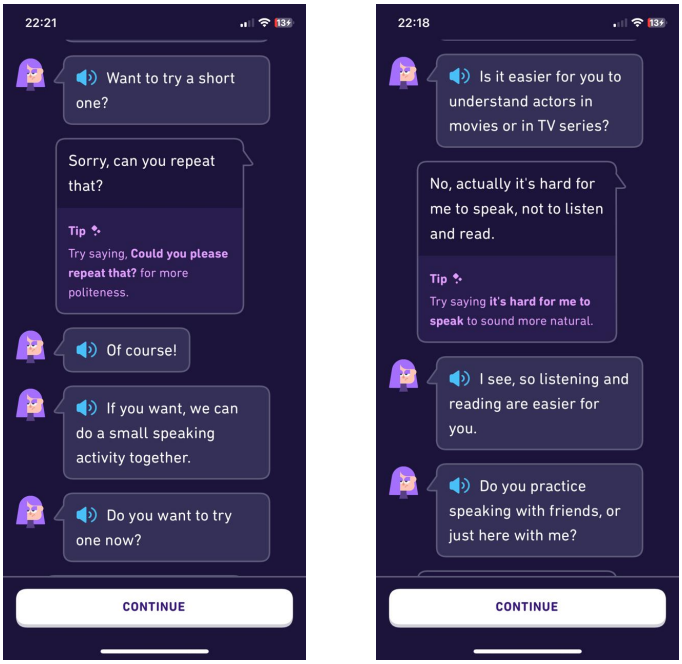
Xu et al. (2022) conduziram uma série de estudos, em que desenvolveram livros eletrônicos com apoio de alto-falantes inteligentes a fim de ajudar crianças de três a seis anos aprenderem a ler. Para isso, participaram 122 crianças para as quais o agente utilizava perguntas semelhantes às de um professor. As perguntas feitas às crianças seguiram um roteiro programado para apoiar o aprendizado, englobando tanto as histórias dos livros eletrônicos quanto o vocabulário. Esse estilo de leitura interativa é chamado de leitura dialógica e inclui fazer perguntas para estimular o pensamento das crianças, fornecendo feedback pela participação delas. Os estudos constataram que a leitura dialógica por meio de agentes melhorou tanto o aprendizado quanto o engajamento, sendo que simplesmente ouvir um agente foi a pior condição para o engajamento das crianças. No entanto, a adição do diálogo fez o engajamento subir ao mesmo nível de conversa com parceiros humanos.

2.1.2 Implementações comerciais de Tutores de IA

A análise de soluções comerciais emergentes evidencia a viabilidade de mercado para tutores baseados em IA.

Duolingo MAX: Este recurso se trata de um módulo do aplicativo Duolingo que utiliza o modelo GPT da OpenAI para oferecer funcionalidades como o *Roleplay* (conversação com personagens em cenários predefinidos) (Duolingo, 2023). No entanto, sua arquitetura limita-se a uma abordagem puramente neural, não fornecendo feedback em tempo real. A Figura 1 exibe a interface do aplicativo, destacando na subfigura (b) um exemplo de imprecisão dos sistemas neurais. Ainda, a aplicação não incorpora elementos centrais deste trabalho, como a integração de conhecimento simbólico externo, uma arquitetura multiagente ou o paradigma *CoA*.

Figura 1 – Comparativo da interface do Duolingo MAX, demonstrando as limitações de uma abordagem de agente único.



(a) Tela de transcrição padrão.

(b) Exemplo de imprecisão.

Fonte: Elaborado pelo autor.

Praktika: Esta plataforma implementa tutores com personalidades e roteiros predefinidos, permitindo ao usuário selecionar cenários de interesse para a prática de conversação. Seu principal apelo comercial é o uso de *Digital Humans* (Seres humanos digitais). No entanto, como essa tecnologia se aplica no modelo de geração de vídeo do avatar, ela foge ao escopo desta análise, cujo foco se concentra estritamente na arquitetura de raciocínio do agente.

3 Fundamentação teórica

Este capítulo apresenta os fundamentos teóricos que sustentam o desenvolvimento do tutor de IA proposto. Inicia-se com uma revisão da evolução histórica do processamento de linguagem natural (PLN), detalhando a transição das abordagens simbólicas para as conexionistas e estabelecendo o contexto para os desafios atuais. Em seguida, discute-se a abordagem neuro simbólica atual que permite inserir conhecimento explícito como um paradigma para a construção de agentes inteligentes mais robustos. Por fim, são analisadas as arquiteturas de raciocínio de agentes, progredindo do *Chain-of-Thought*, *ReAct* e sistemas multiagente até o paradigma *Chain-of-Agents*, que constitui o núcleo deste presente trabalho.

3.1 A Evolução do PLN

A capacidade dos sistemas de IA de interagir reflete historicamente a evolução dos paradigmas teóricos da área. A primeira grande abordagem no campo do PLN até a década de 1980, foi a era simbólica, caracterizada por sistemas especialistas que operavam com base em conhecimento estruturado em regras formais e processos lógicos(Russell; Norvig, 2016)(Caseli; Nunes, 2024). Porém, embora eficazes em determinados domínios bem definidos, os sistemas simbólicos mostraram-se frágeis diante da complexidade da linguagem humana.

A limitação evidente dos sistemas simbólicos, pela sua incapacidade de aprender com a experiência e de lidar com a incerteza, deu origem ao período de estagnação conhecido como o "inverno da IA". Tal limitação impulsionou a origem da era conexionista. Este novo paradigma, por sua vez, deixou de lado as regras explícitas dos modelos anteriores em favor de modelos estatísticos, e, mais posteriormente, redes neurais.(Russell; Norvig, 2016).

A ascensão definitiva da era conexionista foi marcada pelo surgimento do aprendizado profundo (*deep learning*) (Goodfellow; Bengio; Courville, 2016). A capacidade de treinar redes neurais com múltiplas camadas permitiu a criação de modelos que aprendem representações complexas de dados. A implementação desses modelos em arquiteturas como os Transformers, treinados em volumes massivos de texto, deu origem aos LLMs(Caseli; Nunes, 2024), que revolucionaram as capacidades de PLN, permitindo interações mais naturais e coerentes (Bommasani et al., 2021). Tais modelos possuem a capacidade de gerar

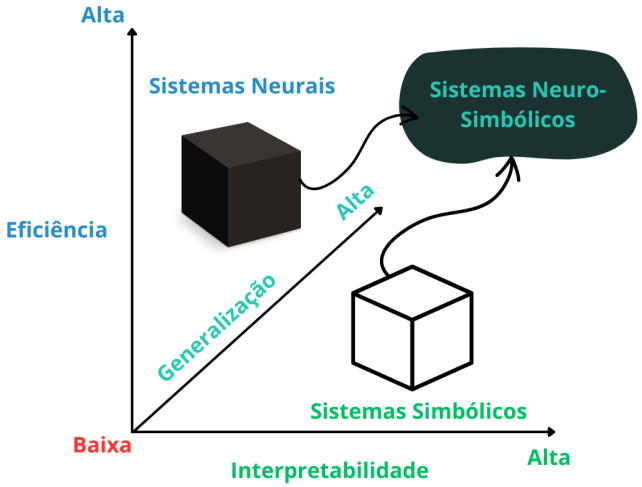
conteúdo semelhante ao humano, responder a perguntas complexas e executar diversas tarefas interagindo em linguagem natural (Chang et al., 2024).

3.2 A convergência neuro simbólica em arquiteturas de agentes

Apesar de seu sucesso, as abordagens puramente neurais, como os LLMs, apresentam limitações. Tais sistemas são frequentemente descritos como sistemas "caixa-preta", pois sua arquitetura de milhões ou até mesmo bilhões de parâmetros torna extremamente difícil rastrear o processo de raciocínio que leva a uma determinada saída, gerando um comportamento não determinístico. Essa falta de transparência é problemática em contextos que exigem precisão e alta confiabilidade, como na educação, pois os modelos podem alucinar, gerando informações incorretas (Marcus, 2018)(Goodfellow; Bengio; Courville, 2016).

Diante da insuficiência de abordagens isoladas, novos esforços de pesquisa têm se voltado para a combinação de redes neurais com sistemas simbólicos (Barnes; Hutson, 2024)(Caseli; Nunes, 2024). A abordagem neuro simbólica busca unir o melhor de ambas as abordagens: o aprendizado de padrões e a generalização fluente dos sistemas neurais com o raciocínio estruturado, a lógica e a saída determinística dos sistemas simbólicos. Conforme ilustrado na Figura 2 (Yu et al., 2023).

Figura 2 – As vantagens dos sistemas de aprendizado neuro simbólico com relação à eficiência, generalização e interpretabilidade do modelo.



Fonte: Adaptado de Yu et al. (2023).

3.3 Paradigmas de raciocínio em agentes autônomos

No campo dos agentes conversacionais, a abordagem neuro simbólica se materializa em arquiteturas que permitem a um LLM (o componente neural) interagir com ferramentas e processos externos (o componente simbólico) para resolver tarefas sofisticadas que vão além da geração de texto simples. Contudo, para que essa orquestração externa fosse possível, foi necessário primeiramente melhorar a precisão do modelo na sustentação de uma linha de raciocínio coerente.

3.3.1 *Chain-of-Thought*: Raciocínio em cadeia

Como precursor dessa evolução, a abordagem denominada *Chain-of-Thought* (CoT), ou raciocínio em cadeia, proposta por Wei et al. (2022), demonstrou que, por meio de manipulação do *prompt* é possível decompor problemas complexos em etapas menores, como no próprio processo de pensamento humano.

O CoT aprimora o raciocínio ao instruir o modelo a gerar uma sequência de passos intermediários $(\tau_1, \tau_2, ..., \tau_n)$ que levam à solução final. Diferente de um *prompt* padrão que busca uma resposta direta, o CoT instrui o modelo a pensar passo a passo, decompondo um problema complexo em uma sequência de etapas.(Wei et al., 2022).

Desse modo, por meio de benchmarks como o GSM8K (Problemas matemáticos de nível fundamental) com o modelo PaLM de 540B foi constatado que o paradigma melhorou o desempenho em tarefas de raciocínio aritmético, superando modelos que passaram por *fine-tuning* (Wei et al., 2022). Alem disso, a cadeia de pensamento gerada oferece uma janela melhor interpretável sobre o raciocínio do modelo, permitindo identificar em qual etapa o processo falhou, mesmo não demonstrando exatamente o raciocínio interno do modelo.

O estudo demonstra que os benefícios só se tornam evidentes em modelos com aproximadamente 100 bilhões de parâmetros ou mais, que parecem adquirir a capacidade de gerar cadeias de pensamento lógicas e coerentes, enquanto para modelos menores a técnica pode até prejudicar o desempenho (Wei et al., 2022).

O paradigma CoT, portanto, estabeleceu a base para o desenvolvimento de agentes mais sofisticados. Porém, o CoT opera gerando a sequência de pensamentos de uma só vez. Desse modo, a evolução natural desse conceito foi buscar formas de permitir que o agente viesse a acessar conhecimentos externos.

3.3.2 MRKL: Uma arquitetura neuro simbólica modular

Enquanto o CoT aprimorou o raciocínio interno dos LLMs, ele não resolveu as limitações fundamentais dos modelos de linguagem quando confrontados com tarefas que exigem conhecimento atualizado, informações proprietárias ou raciocínio matemático preciso (Karpas et al., 2022). Um LLM, por sua natureza, é um sistema estático o que o torna incapaz de acessar informações dinâmicas, além de possuir elevada propensão a erros (Karpas et al., 2022).

Neste contexto, foi proposta a arquitetura MRKL (*Modular Reasoning, Knowledge, and Language*), um sistema neuro simbólico projetado para diminuir essas fraquezas (Karpas et al., 2022). A premissa central do MRKL é que um único LLM não é a solução ideal para todas as tarefas. Em vez disso, a arquitetura propõe um sistema modular composto por um roteador inteligente e módulos especialistas.

A estrutura de um sistema MRKL consiste em:

- Um roteador (*Router*): Tipicamente um LLM, responsável por receber a entrada do usuário em linguagem natural e direcioná-la para o módulo mais adequado para resolver a tarefa.
- Um conjunto de especialistas (*Experts*): Módulos que variam desde sistemas neurais, como outros LLMs, até ferramentas simbólicas como calculadoras, conversores de moeda, consulta em APIs (*Application Programming Interface*), ou qualquer outra ferramenta que execute uma tarefa discreta (Karpas et al., 2022).

Essa abordagem modular oferece diversos benefícios como a extensibilidade, em que novos módulos especialistas podem ser adicionados ao sistema sem a necessidade de retreinar o LLM central de modo que apenas o roteador precisa ser atualizado para aprender a invocar o novo módulo (Karpas et al., 2022). Além disso, a arquitetura permite que o sistema acesse informações atualizadas e proprietárias através de APIs, resolvendo a limitação de conhecimento estático dos LLMs.

A arquitetura MRKL, portanto, representa uma aplicação das abordagens neuro simbólicas, estabelecendo um projeto para sistemas que combina a fluidez dos LLMs com a precisão dos sistemas simbólicos. Porém, embora defina quais componentes usar, ela não estabelece com clareza como o modelo deve alternar entre eles dinamicamente e como garantir que o estado não se perca durante essa interação.

3.3.3 ReAct: Raciocínio e ação

Tornando possível o projeto neuro simbólico de sistemas modulares, o ReAct (*Reasoning and Acting*) (Yao et al., 2023) superou as limitações do CoT ao gerar raciocínios e ações de maneira intercalada. Essa abordagem diminui problemas de alucinação, pois substitui a especulação interna por um ciclo de *feedback* contínuo com o mundo real. Desse modo, o agente não apenas planeja suas ações, mas utiliza as respostas obtidas de ferramentas externas para atualizar seu conhecimento e refinar decisões subsequentes (Yao et al., 2023).

Como retratado por Yao et al. (2023), o ReAct implementa um ciclo iterativo de pensamento, ação e observação da seguinte forma:

1. Pensamento: O agente analisa a tarefa e seu estado atual, deliberando sobre a próxima melhor ação. Esta etapa corresponde à função do roteador do MRKL, mas de forma mais dinâmica, pois ocorre a cada passo do processo.
2. Ação: O agente executa uma ação, que consiste tipicamente na chamada de uma ferramenta externa, como uma API de busca na web.
3. Observação: O agente recebe o resultado da ação e o incorpora ao seu contexto, atualizando o seu estado e utilizando essa nova informação no próximo ciclo de pensamento.

O ciclo iterativo do ReAct pode ser representado formalmente, conforme proposto por Li et al. (2025), pela Equação 3.1:

$$(\tau_1, a_1, o_1, \tau_2, a_2, o_2, \ldots, \tau_T) \tag{3.1}$$

em que $\tau_t \in \mathcal{T}$ representa o pensamento, os passos de raciocínio no instante t, sendo $\mathcal{T}$ o espaço de todos os possíveis passos de raciocínio, $a_t \in \mathcal{A}$ define a ação executada, sendo $\mathcal{A}$ o espaço de ações disponíveis, $o_t \in \mathcal{O}$ captura a observação resultante da ação, sendo $\mathcal{O}$ o espaço de observações, e T é o número total de passos até a resolução da tarefa.

Cada pensamento τ_t é condicionado ao histórico completo $h_t = [\tau_{1:t-1}, a_{1:t-1}, o_{1:t-1}]$, permitindo que o agente incorpore o feedback do ambiente (observações) no processo de raciocínio subsequente (Yao et al., 2023).

Para validar a arquitetura, o estudo original avaliou o ReAct em dois domínios. O primeiro foca no raciocínio intensivo, utilizando *benchmarks* como HotpotQA (Yang et al., 2018), um conjunto de dados de perguntas e respostas que exige que o modelo

busque informações dispersas em múltiplos documentos (*multi-hop*). O segundo domínio se concentra na tomada de decisão, representada por *benchmarks* como ALFWorld (Shridhar et al., 2020), que proporciona um ambiente no qual o agente deve determinar localizações prováveis de itens domésticos via comando de texto.

Os resultados demonstraram que, em tarefas como o HotpotQA, o ReAct supera os problemas de alucinação do CoT ao interagir com uma API da Wikipedia (Yao et al., 2023). Entretanto, conforme elucidado pela Anthropic em estudos de casos reais sobre sistemas de pesquisa autônomos, agentes individuais tendem a falhar em tarefas que exigem alta complexidade por enfrentarem limitações físicas na janela de contexto quando precisam processar grandes volumes de informação (Hadfield et al., 2024).

3.3.4 Sistemas multiagente

Em resposta à limitação do ReAct, uma evolução natural para lidar com problemas de maior complexidade é a decomposição da tarefa entre múltiplos agentes autônomos e especializados que colaboram para um objetivo comum. Esta abordagem baseia-se no princípio da divisão para conquistar, em que um problema é fragmentado em tarefas menores, cada uma atribuída a um único agente que detém as ferramentas ideais para sua responsabilidade (Hadfield et al., 2024; Li et al., 2025).

Em sistemas multiagente, cada agente a_i mantém um estado interno s_i^t que evolui através de comunicação com outros agentes. A transição de estado de um agente j no tempo t é dada, segundo Li et al. (2025), pela Equação 3.2:

$$s_j^t = f_j(s_j^{t-1}, \{m_{i \to j}^{t-1}\}_{a_i \in \mathcal{A}}) \tag{3.2}$$

em que $\mathcal{A}$ representa o conjunto de todos os agentes no sistema, $m_{i \to j}^{t-1}$ são as mensagens enviadas do agente i para o agente j no passo de tempo $t-1$, e f_j é a função de transição de estado específica do agente j (Li et al., 2025).

Arquiteturas multiagente têm se mostrado mais eficientes para problemas reais do que sistemas compostos por um único agente. Conforme documentado por Hadfield et al. (2024), sistemas multiagente demonstram melhoria de 90,2% em desempenho comparado a agentes únicos em tarefas mais complexas que exigem decomposição. Entretanto, esse desempenho elevado consome cerca de 15 vezes mais tokens do que interações de chat padrão. Essa eficiência no resultado é obtida pela especialização, já que, em vez de um único agente generalista ter que ser responsável por todas as nuances de um problema, agentes especialistas podem ser otimizados para suas funções, levando a um resultado geral melhor.

Para gerenciar as interações entre esses especialistas, o framework LangGraph (Pelluru, 2025) emergiu como o padrão utilizado na indústria. Ele permite modelar fluxos de trabalho multiagente como grafos de estado, nos quais cada nó representa um agente e as arestas representam as transições de controle entre eles.

Esta abordagem é aplicada no presente trabalho por meio do módulo de processamento gramatical. Este módulo funciona como um pipeline multiagente determinístico que realiza o processamento da entrada do usuário de forma estruturada. No entanto, a aplicação dessa arquitetura introduz um maior custo e alta latência, limitações que o paradigma *Chain-of-Agents* visa solucionar.

3.4 O paradigma *Chain-of-Agents*

Essa ineficiência ocorre porque, conforme apontado por Li et al. (2025), sistemas multiagente sofrem com alta sobrecarga computacional devido à comunicação contínua entre seus componentes, enfrentando desafios ao lidar com novas tarefas sem uma grande reengenharia de *prompts* (Li et al., 2025). Cada transição no grafo geralmente requer uma nova chamada para a API do LLM, o que aumenta tanto a latência, impactando a experiência do usuário, quanto os custos.

Para mitigar essa situação, o recente trabalho de Li et al. (2025) introduziu o paradigma *Chain-of-Agents* (CoA), introduzindo a ideia de que ao invés de coordenar agentes externos, um único modelo pode ser treinado para se tornar um *Agent Foundation Model* (AFM), capaz de simular internamente a colaboração de múltiplos agentes especialistas (Li et al., 2025). O CoA, portanto, internaliza a estrutura multiagente dentro de um único processo de inferência, o que permite que o AFM desempenhe de forma análoga a um sistema multiagente completo (Li et al., 2025).

Os agentes no CoA não são instâncias de LLMs separadas, mas sim modos de raciocínio que um modelo aprende a adotar dinamicamente. Estes dividem-se em *Role-playing Agents*, responsáveis pela estratégia (como *Plan Agent*, *Reflection Agent*, *Verification Agent*), e *Tool Agents*, responsáveis pela execução (como *Search Agent*, *Crawl Agent*, *Code Generate Agent*) (Li et al., 2025). A simulação deste conceito é o cerne do agente central neste projeto. Através de um prompt mestre, um LLM de propósito geral é instruído a assumir esses diferentes papéis, fazendo com que ele se comporte como um AFM sem a necessidade de *fine-tuning*.

A criação de um AFM é realizada através de um processo chamado destilação de conhecimento multiagente (*multi-agent distillation*) (Li et al., 2025). Nesse processo,

as trajetórias de sucesso de um sistema multiagente explícito são coletadas, filtradas, e usadas como dados que irão servir para o *fine-tuning* do AFM. Desse modo, o modelo treinado aprende a replicar os padrões de raciocínio colaborativo. Embora este trabalho não realize o *fine-tuning* de um novo modelo, ele induz o comportamento desejado através de *in-context learning*.

3.4.1 Formalização do CoA

O paradigma CoA introduz a orquestração dinâmica de múltiplos papéis dentro de um único modelo. Essa transição de estado é formalizada, segundo Li et al. (2025), pela Equação 3.3:

$$S_t = f_\theta(S_{t-1}, \phi_{t-1}, o_{t-1}), \quad \phi_t \sim P(\phi \mid S_t) \tag{3.3}$$

em que S_t representa o estado persistente de raciocínio no passo t, $\phi_t \in \{\phi_{\text{think}}, \phi_{\text{plan}}, \phi_{\text{reflect}}, \ldots\}$ é o papel (agente) ativado dinamicamente, o_{t-1} é a observação do passo anterior, f_θ é a função de transição parametrizada pelo modelo, e $P(\phi \mid S_t)$ é a distribuição de probabilidade sobre papéis dado o estado atual.

Diferentemente do ReAct (Equação 3.1), que alterna rigidamente entre pensamento e ação, o CoA permite que o modelo escolha dinamicamente qual agente interno ativar em cada passo, adaptando-se à complexidade da tarefa (Li et al., 2025).

Assim, uma trajetória completa no paradigma CoA é definida, conforme Li et al. (2025), pela Equação 3.4:

$$\tau = \{(S_t, \phi_t, o_t)\}_{t=1}^{T} \tag{3.4}$$

em que $S_t \in \mathcal{S}$ é o estado de raciocínio que mantém o contexto acumulado e evolui a cada passo através da função de transição f_θ, $\phi_t \sim P(\phi \mid S_t)$ é a escolha do papel, representando qual agente especializado será ativado de forma dinâmica com base no estado atual, e $o_t \in \mathcal{O}$ é a observação, resultado da execução do papel escolhido, que alimenta o próximo ciclo de raciocínio.

O processo iterativo de construção dessa trajetória segue um ciclo de refinamento expresso pela Equação 3.5:

$$S_t = \Gamma(S_{t-1}, o_{t-1}), \quad a_t \sim \pi(\cdot \mid S_t), \quad o_t = \Phi(a_t) \tag{3.5}$$

em que Γ é a função de atualização de estado, π é a política de seleção de ação, e Φ é o ambiente de execução (ferramentas, APIs).

3.4.2 Destilação multiagente

Na implementação original do CoA proposta por Li et al. (2025), a criação de um AFM envolve um processo de destilação de conhecimento multiagente seguido de refinamento por meio de aprendizado por reforço. Nesse processo, as trajetórias de sucesso de sistemas multiagente são coletadas e utilizadas para treinar um modelo único através de *Supervised Fine-Tuning* (SFT).

Após o SFT, o modelo é refinado via aprendizado por reforço em tarefas verificáveis, utilizando funções de recompensa específicas por domínio. Para agentes web, por exemplo, adota-se $R_{\text{web}}(\tau) = \text{score}_{\text{answer}}$, em que $\text{score}_{\text{answer}} \in \{0, 1\}$ é determinado através da abordagem *LLM-as-a-Judge* (Zheng et al., 2023).

Para validar a eficácia do CoA, os autores utilizaram *benchmarks* como o GAIA (Mialon et al., 2023), que avalia a capacidade de raciocínio do agente em múltiplas etapas e o correto uso de ferramentas para resolver questões que exigem raciocínio complexo. Já para o raciocínio lógico, foi utilizado um conjunto de problemas do AIME 2025, uma competição de matemática com nível superior aos exames escolares tradicionais.

Este processo de treinamento com AFMs alcançou 55,3% no *benchmark* GAIA e 59,8% no AIME 2025 (Li et al., 2025). Além da performance, o paradigma destacou-se pela eficiência, atingindo uma redução de 84,6% no consumo de *tokens* comparado a sistemas multiagente tradicionais (Li et al., 2025). Vale ressaltar que os autores relataram uma melhora notável na latência, embora não tenham fornecido métricas detalhadas na análise principal. Apesar dos benefícios, a implementação completa deste processo (destilação e RL) requer recursos computacionais substanciais e dados de treinamento especializados que fogem do escopo deste trabalho.

3.5 Comparação dos paradigmas apresentados

A discussão sobre a evolução das arquiteturas de agentes, desde o *Chain-of-Thought* até o *Chain-of-Agents*, demonstra o direcionamento para sistemas mais flexíveis e de baixo custo. Nesse comparativo, a flexibilidade refere-se à capacidade de adaptação a novas tarefas, enquanto o custo corresponde ao consumo computacional, medido em *tokens* e latência. A Tabela 2 demonstra essas características entre os paradigmas abordados.

Tabela 2 – Comparação entre paradigmas de raciocínio em agentes

Paradigma	Formalização	Flexibilidade	Custo
CoT	$h_t = [\tau_{1:t-1}]$	Baixa	Baixo
ReAct	$(\tau_t, a_t, o_t)_{t=1}^{T}$	Média	Médio
Multiagente	$s_j^t = f_j(s_j^{t-1}, \{m_{i \to j}^{t-1}\})$	Alta	Alto
CoA	$S_t = f_\theta(S_{t-1}, \phi_{t-1}, o_{t-1})$	Alta	Baixo

Fonte: Elaborado pelo autor.

Assim, a arquitetura adotada neste trabalho combina um pipeline de correção gramatical com a eficiência de um agente unificado. O próximo capítulo detalha a implementação dessa arquitetura.

4 Materiais e métodos

Este capítulo detalha os recursos tecnológicos utilizados e a metodologia de desenvolvimento empregada na construção do tutor de IA conversacional. A abordagem metodológica seguiu um processo de desenvolvimento experimental, focado na implementação dos paradigmas discutidos no capítulo 3, referente à fundamentação teórica.

4.1 Materiais e recursos tecnológicos

4.1.1 Servidor

O servidor foi desenvolvido com tecnologias que proporcionam agilidade no desenvolvimento de sistemas de IA.

- Python 3.11: Foi a linguagem escolhida devido ao seu ecossistema voltado para IA. Sua ampla comunidade ativa proporciona o desenvolvimento contínuo de bibliotecas, conforme fundamentado por Talati (2021).
- FastAPI: Foi o framework web selecionado para a construção da API RESTful. Sua escolha se justifica por sua performance superior em relação a outros frameworks para Python, pela sua natureza assíncrona nativa e por sua capacidade de gerar documentação automática (via Swagger UI) a partir dos modelos de dados, acelerando o desenvolvimento (Lubanovic, 2023).
- PostgreSQL: Atua como o banco de dados relacional principal, servindo como armazenamento de memória permanente de todas as entidades do sistema, como usuários, conversas e correções (Juba; Vannahme; Volkov, 2015).
- SQLAlchemy: É o *Object-Relational Mapper* (ORM) utilizado para mediar a comunicação entre a aplicação Python e o PostgreSQL. Ele funciona como um facilitador, abstraindo as consultas SQL em objetos Python, o que aumenta por sua vez a produtividade, evitando a necessidade de escrever código SQL diretamente (Myers; Copeland; Copeland, 2015).
- Redis: Desempenha um papel duplo no sistema. Primeiramente, atua como um banco de dados em memória de alta velocidade para cache, armazenando perfis de usuário para acesso instantâneo. Em segundo lugar, serve como forma de persistência

de estado para o LangGraph (implementando *store* e *checkpointer*), gerenciando o histórico de conversação e proporcionando o contexto necessário para a operação do agente (Carlson, 2013).

- Docker & Docker Compose: São as ferramentas de containerização que encapsulam a aplicação e seus serviços (API, PostgreSQL, Redis). Seu uso simplifica a implantação através dos *containers* (Miell; Sayers, 2019).

4.1.2 Modelos de linguagem e APIs de IA

A parte central do sistema utiliza um conjunto de modelos e serviços externos especializados, cada um selecionado para uma tarefa específica.

- Groq API: Foi escolhida para os modelos de linguagem pois sua arquitetura é baseada em LPUs (*Language Processing Units*), o que oferece uma latência mais baixa, sendo fundamental para uma aplicação conversacional em tempo real. A API foi utilizada para servir dois modelos principais: o Llama 3.1 70B, para todas as tarefas de raciocínio complexo, e o Whisper-large-v3, para transcrição de áudio (*Speech-to-Text*) com alta precisão (Groq, 2025).
- ElevenLabs API: É o serviço utilizado para a síntese de voz (*Text-to-Speech*). Foi selecionado pela naturalidade das vozes. A fim de caracterizar o agente, foi utilizado uma voz específica chamada Rachel com o intuito de tornar a interação mais imersiva (Elevenlabs, 2025).
- LanguageTool API: Serve como o componente simbólico e determinístico da arquitetura. Se trata de uma ferramenta especialista, baseada em regras, para verificação gramatical, garantindo uma primeira camada de análise confiável para erros sintáticos (Languagetool, 2025).
- Tavily Search API: É a ferramenta de busca na web integrada ao agente. Otimizada para agentes de IA, ela retorna resultados já processados de uma forma mais simples para a IA (Tavily, 2025).

4.1.3 Frameworks e bibliotecas de agentes

A lógica do agente foi implementada com bibliotecas especializadas.

- LangChain: É a biblioteca que fornece as abstrações para interagir com LLMs, gerenciar prompts, definir ferramentas e encadear as chamadas. (Oshin; Campos, 2025)
- LangGraph: É uma extensão do LangChain usada para a implementação da arquitetura. Permite a criação de agentes como grafos de estado, sendo um facilitador para o desenvolvimento do pipeline de processamento gramatical. (Oshin; Campos, 2025)
- Pydantic: Atua como biblioteca de validação de dados, integrando-se ao FastAPI para serialização e validação automática, além de definir os esquemas das ferramentas do agente, assegurando que as chamadas do LLM sigam estruturas bem definidas (Narayanan, 2024).

4.1.4 Cliente

A seleção de tecnologias para o cliente foi orientada para o desenvolvimento de uma aplicação multiplataforma de forma simples e moderna.

- React Native com Expo: Foi a plataforma escolhida para o desenvolvimento da aplicação. O React Native permite a construção de aplicações nativas para iOS e Android a partir de uma única base de código que pode ser tanto em JavaScript quanto em TypeScript. O framework Expo foi utilizado sobre o React Native para simplificar e acelerar o ciclo de desenvolvimento com foco em abstrair complexidades de configuração do ambiente nativo (Expo, 2025).
- TypeScript: A linguagem de programação adotada para todo o frontend. O uso de TypeScript garante a segurança de tipos, o que reduz a ocorrência de erros em tempo de execução (Goldberg, 2022).
- Expo Router: Para o gerenciamento da navegação e das rotas da aplicação com uma abordagem declarativa a fim de simplificar a transição de telas (Expo, 2025).
- NativeWind (Tailwind CSS): A estilização da interface do usuário foi implementada utilizando NativeWind, que se trata de uma versão do framework simplificador de CSS chamado TailwindCSS, porém para React Native. Seu uso permite escrever as classes de estilo diretamente nos componentes, acelerando o desenvolvimento (Nativewind, 2025).
- Axios: Biblioteca utilizada para realizar requisições HTTP ao backend da aplicação. O Axios foi escolhido por sua simplicidade de uso, ampla adoção na comunidade TypeScript e recursos nativos como interceptadores de requisições e tratamento automático de erros (Axios, 2025).

- TanStack Query: Para facilitar o gerenciamento de estado do servidor, a comunicação com a API e o cache, foi utilizada a biblioteca TanStack Query (Tanstack, 2025).

4.2 Métodos de desenvolvimento e arquitetura do sistema

A metodologia de desenvolvimento focou na implementação de uma arquitetura modular, bem como na arquitetura do agente.

4.2.1 Arquitetura do agente

Conforme justificado no capítulo 3, o sistema adota uma arquitetura que separa a correção gramatical do raciocínio mais complexo, visando assim, maior precisão na correção gramatical.

4.2.1.1 Componente 1: Pipeline de análise gramatical

O primeiro componente se trata do pipeline de processamento inicial, utilizando LangGraph para o gerenciamento de dois agentes especialistas em correção gramatical:

1. Agente sintático: O primeiro nó do grafo recebe a entrada do usuário e a envia para a LanguageTool API. O resultado é uma análise sintática determinística com a correta identificação dos possíveis erros.
2. Agente semântico: O segundo nó recebe a entrada original e a análise sintática. Utilizando o Llama 3.1 70B com um prompt especializado na correção, ele realiza uma análise semântica para detectar tanto erros contextuais quanto problemas de naturalidade em frases que, embora não contenham erros gramaticais, soam estranhas para um nativo.

A saída deste pipeline é usada como contexto para o componente principal de raciocínio.

4.2.1.2 Componente 2: Componente principal de raciocínio.

O segundo componente, se trata do componente principal de raciocínio que aplica o paradigma CoA através de engenharia de prompt. Seu fluxo se dá por meio de um LLM

de propósito geral que é instruído a se comportar como um AFM, seguindo um ciclo de raciocínio que intercala diferentes papéis, apresentados a seguir:

- Planejamento e reflexão: O agente é instruído a adotar um plano de ação explícito através da tag(`<plan>`) e a refletir sobre os resultados das suas ações através da tag (`<reflection>`), permitindo a autocrítica e o ajuste dinâmico de sua estratégia conforme as interações fluem.
- Ação e uso de ferramentas: O agente pode invocar as ferramentas disponíveis como `WebSearch`, `UpdateUserProfile`, etc. através da tag `<tool_call>`. Isso permite que ele interaja com o mundo externo, seja a web ou o banco de dados para buscar conhecimento e o armazenar em memória, seguindo os princípios das arquiteturas MRKL e ReAct.

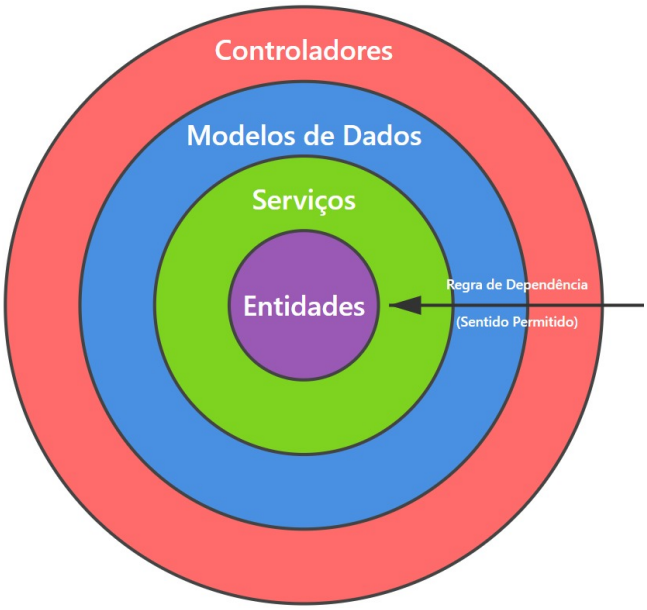
4.2.2 Implementação do *backend* e processamento de áudio

A interação do usuário com o sistema de agente é mediada por uma API RESTful desenvolvida com FastAPI. A estrutura do código foi projetada seguindo os princípios da *Clean Architecture* (Martin, 2017), uma abordagem em camadas que promove a separação de responsabilidades para aumentar a manutenibilidade, conforme detalhado por Valente (2020) e ilustrado na Figura 3. As quatro camadas principais implementadas no sistema são:

- Entidades (`/entities`): Representam a camada mais interna da arquitetura. Contêm os modelos de dados principais da aplicação, definidos com SQLAlchemy ORM. Esta camada é o núcleo do domínio de negócio e não depende de nenhuma outra.
- Serviços (`/services`): Contêm a lógica de negócio e as regras da aplicação. Esta camada orquestra as operações, interagindo com as entidades para acessar o banco de dados e invocando os componentes do agente principal. Ela atua como intermediária entre os controladores e o núcleo do sistema.
- Modelos de dados (`/models`): Esta camada, implementada com Pydantic, define os contratos de dados da API, também conhecidos como DTOs (*Data Transfer Objects*). Estes objetos atuam como estrutura dos dados esperados nas requisições e fornecidos nas respostas. O uso de Pydantic é fundamental, pois o FastAPI o utiliza para serialização, validação e documentação.
- Controladores (`/controllers`): Constituem a camada mais externa da arquitetura. São responsáveis por gerenciar as rotas da API (*endpoints*), receber as requisições

HTTP e retornar as respostas. Os controladores também utilizam os modelos Pydantic para validar os dados de entrada e formatar os dados de saída.

Figura 3 – A arquitetura em camadas proposta



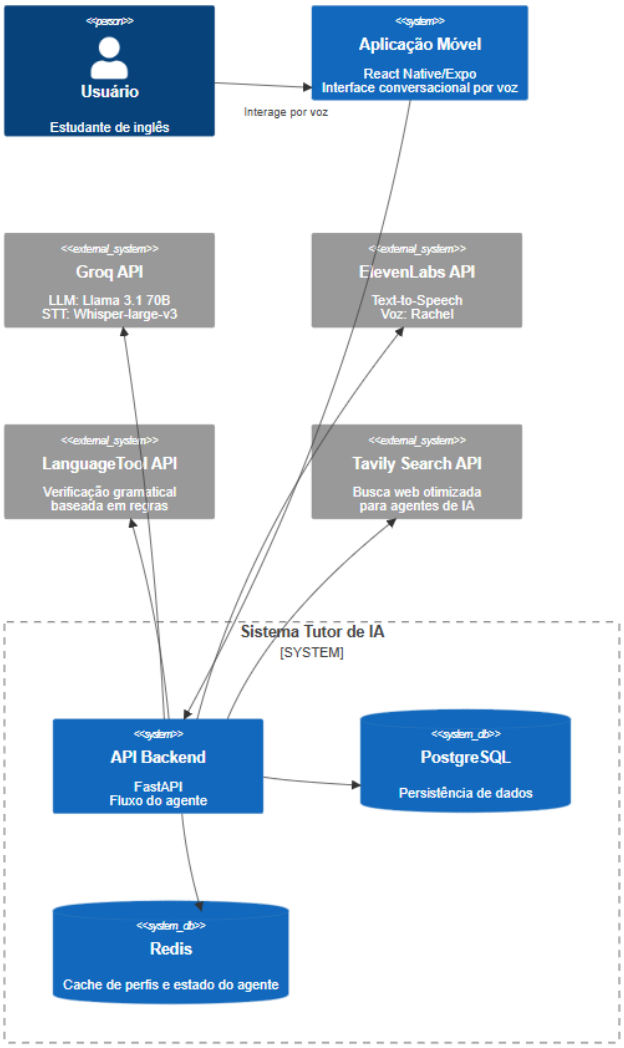
Fonte: Adaptado de Valente (2020).

A funcionalidade de voz é implementada através de um fluxo de processamento de áudio que atravessa essas camadas:

1. O áudio do usuário é recebido pelo controlador de áudio e salvo em um arquivo temporário.
2. O serviço de áudio envia o arquivo para a Groq API para transcrição via modelo Whisper.
3. O texto transcrito é passado para o serviço de conversação, que gerencia a arquitetura de agente.
4. A resposta final em texto, gerada pelo agente, é enviada pelo serviço de TTS para a ElevenLabs API para ser convertida em áudio.
5. Por fim, o arquivo de áudio gerado é retornado ao usuário como resposta da API pelo controlador.

Este fluxo permite uma interação conversacional completa por voz, integrando os componentes internos e serviços externos conforme detalhado no Diagrama C4 apresentado na Figura 4.

Figura 4 – Diagrama C4



Fonte: Elaborado pelo autor.

4.2.3 Modelagem de dados e persistência

A modelagem de dados do sistema foi dividida em uma camada de dados permanentes que utiliza o banco de dados relacional e outra para o cache de acesso rápido, visando alimentar o agente com contexto e informações sobre o perfil de usuário visando performance e acesso rápido.

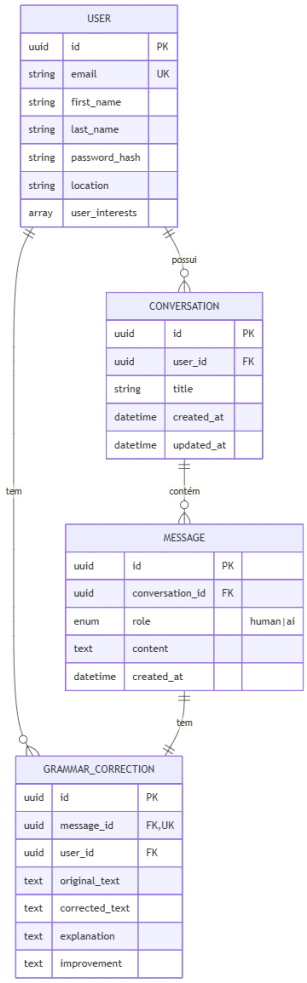
4.2.3.1 Modelagem relacional e entidades

A persistência de dados principal do sistema foi projetada com uma abordagem relacional, utilizando PostgreSQL como banco de dados e o ORM SQLAlchemy para a definição das entidades. A modelagem foi estruturada em torno de quatro entidades principais que representam o núcleo do domínio da aplicação:

- `User`: Representa o usuário do sistema. A entidade armazena informações de autenticação e dados de perfil que são utilizados para a personalização da experiência, como localização do usuário e uma lista de interesses que serão extraídos durante a conversa com o agente.
- `Conversation`: Representa uma sessão de conversa entre um usuário e o agente de IA, associada a um `User` através de uma chave estrangeira.
- `Message`: Representa uma única mensagem dentro de uma conversa, possuindo um campo `role` (do tipo Enum: `HUMAN` ou `AI`) para identificar o autor da mensagem.
- `GrammarCorrection`: Entidade que armazena os resultados da análise gramatical para uma mensagem específica do usuário, possuindo uma relação um-para-um com a entidade `Message`.

As relações entre essas entidades foram definidas para garantir a integridade referencial dos dados. Conforme ilustrado na Figura 5, um `User` pode ter múltiplas `Conversations`, uma `Conversation` contém múltiplas `Messages`, e uma `Message` pode possuir, no máximo, uma `GrammarCorrection`.

Figura 5 – Diagrama ER



Fonte: Elaborado pelo autor.

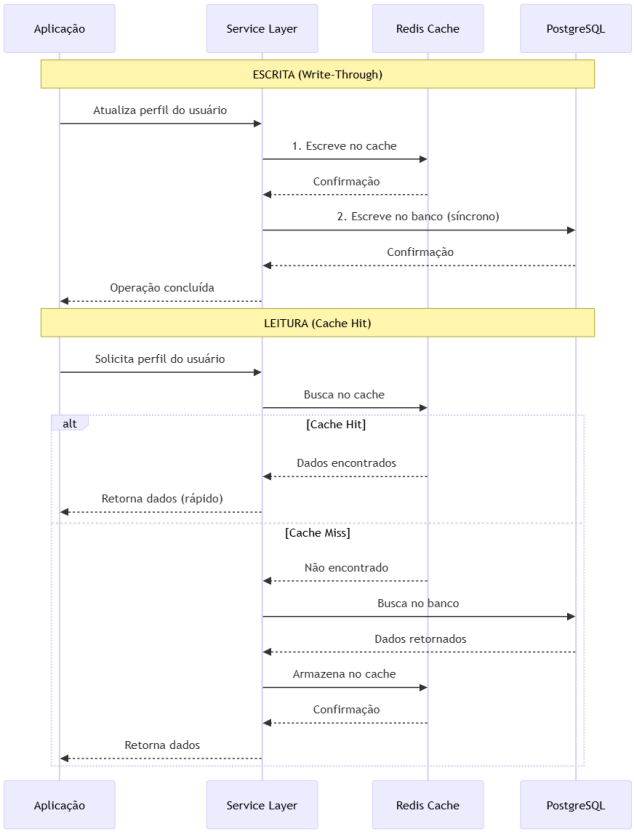
4.2.3.2 Estratégia de cache

Para otimizar o tempo de resposta e reduzir a carga sobre o banco de dados principal, foi implementada uma camada de cache utilizando Redis. A interação entre o PostgreSQL, para dados permanentes, e o Redis, para os dados de perfil do usuário segue o padrão *Write-Through Cache*, chamado de cache com escrita direta. (Jouppi, 1993)

Neste padrão, toda operação de escrita de dados é realizada de forma síncrona tanto no cache quanto no banco de dados. A principal vantagem desta abordagem é a garantia de consistência dos dados pois o cache nunca conterá informações desatualizadas.

Essa performance de leitura é crucial para garantir que o agente possa personalizar a conversação em tempo real, acessando dados do perfil do usuário instantaneamente para adaptar suas respostas ao estado atual da conversa. O funcionamento dessa estratégia é ilustrado na Figura 6.

Figura 6 – Técnica *Write-Through Cache*



Fonte: Elaborado pelo autor.

4.2.4 *Endpoints* e fluxo de conversação

A interação com o sistema é realizada através dos endpoints definidos nos controladores do FastAPI. O fluxo de conversação por voz foi projetado para ser intuitivo com base na observação do autor aos grandes sistemas de IA, como Claude, ChatGPT e Gemini.

4.2.4.1 Criação de uma nova conversa

O fluxo para iniciar uma nova conversa é gerenciado pelo *endpoint* `POST /audio/new`. Todo o processo, ilustrado no diagrama de sequência da Figura 7, ocorre na seguinte ordem:

1. O cliente, através do aplicativo móvel, envia um arquivo de áudio para o *endpoint*.
2. O controlador da API recebe o áudio, valida seu formato e o salva de forma temporária.

3. A função de transcrição de áudio é chamada, enviando o arquivo para a API da Groq e recebendo o texto transcrito pelo modelo Whisper.

4. O controlador invoca a função de criar nova conversa, pertencente ao serviço de conversação.

5. Dentro deste serviço, uma nova instância da entidade `Conversation` é criada no banco de dados. Em seguida, a primeira mensagem do usuário, já transcrita, é salva como uma instância da entidade `Message`, associada a essa nova conversa.

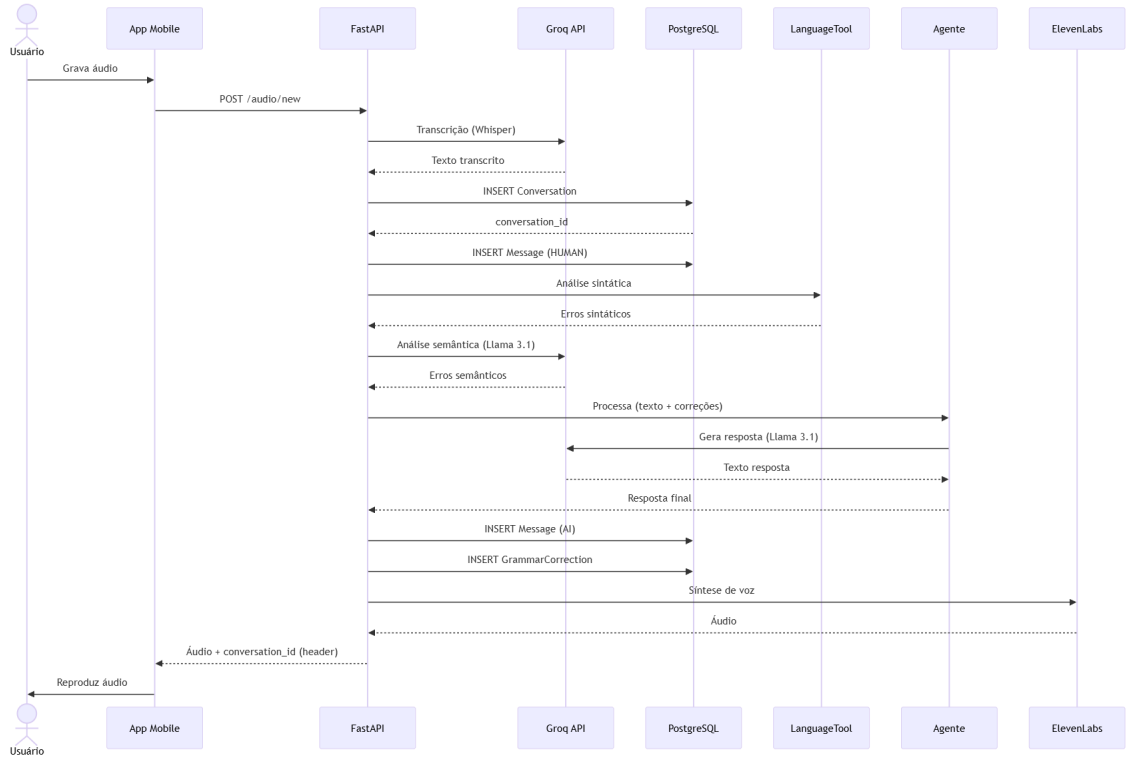
6. O serviço de conversação, por sua vez, aciona a arquitetura de agente completa: o texto do usuário passa pelo *pipeline* de análise gramatical e o resultado, junto com a mensagem original, é enviado para o agente principal de raciocínio.

7. A resposta textual gerada pelo agente é salva no banco de dados como uma nova `Message` com o papel `AI`.

8. O texto da resposta é enviado para a API da ElevenLabs para síntese de voz.

9. Por fim, a API retorna o áudio gerado ao cliente, juntamente com o ID da nova conversa nos cabeçalhos da resposta HTTP.

Figura 7 – Diagrama de sequência sobre o fluxo de uma nova conversa



Fonte: Elaborado pelo autor.

4.2.4.2 Continuação de uma conversa existente

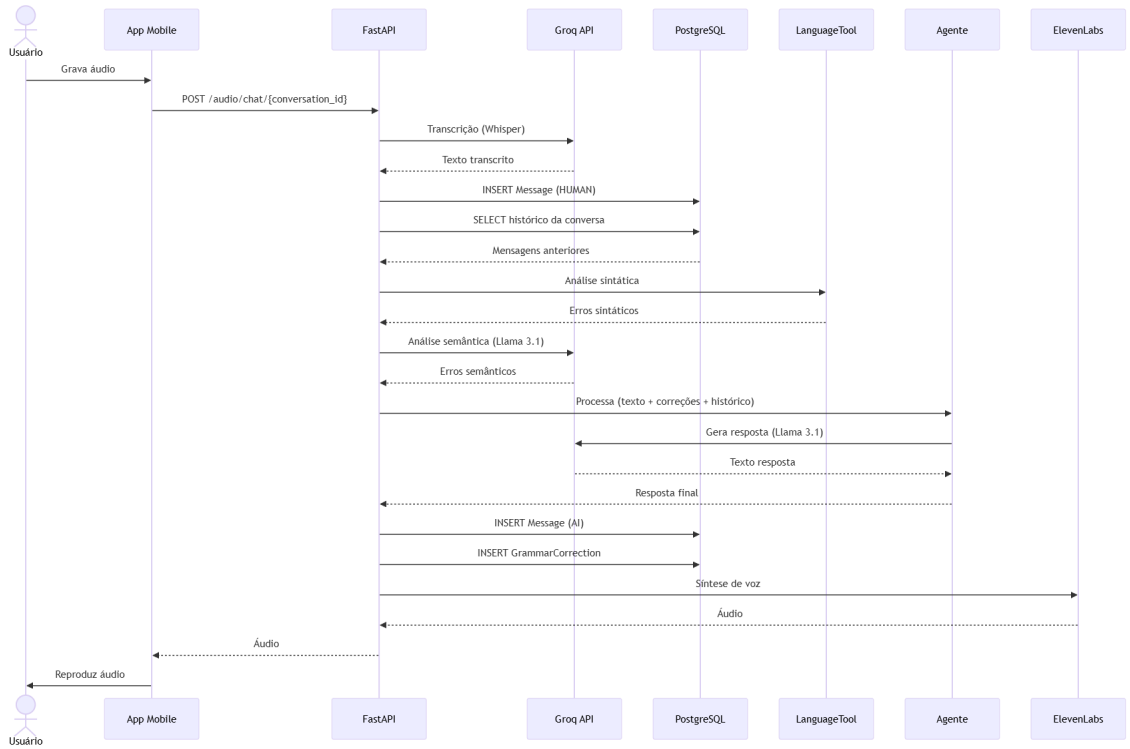
Uma vez iniciada a conversa, as interações subsequentes são tratadas pelo *endpoint* `POST /audio/chat/{conversation_id}`. Este fluxo, detalhado na Figura 8, é mais direto que o anterior:

1. O cliente envia um novo arquivo de áudio, desta vez incluindo o `ID` na URL.
2. O áudio é transcrito para texto, da mesma forma que no fluxo de criação.
3. O controlador invoca a função de processar nova mensagem do serviço de conversação, passando o `ID` e o novo texto do usuário.
4. O serviço salva a nova mensagem do usuário no banco de dados, recupera o histórico da conversa existente e executa o fluxo do agente para gerar a resposta.
5. A resposta do agente é salva como uma nova mensagem na conversa e seu texto é convertido em áudio.

6. A API retorna apenas o arquivo de áudio da resposta, pois o cliente já possui o ID da conversa.

Este *design* segregado em dois *endpoints* otimiza a comunicação, permitindo que as mensagens subsequentes à criação da conversa troquem apenas o conteúdo essencial.

Figura 8 – Diagrama de sequência sobre o fluxo de continuação da conversa



Fonte: Elaborado pelo autor.

4.2.4.3 Endpoints de gerenciamento e suporte

Além do fluxo principal de conversação por voz, a API foi projetada com um conjunto de endpoints de suporte para gerenciar o ciclo de vida do usuário e suas interações com o sistema.

4.2.4.3.1 Gerenciamento de autenticação e usuários

Os endpoints localizados nos controladores `/auth` e `/users` são responsáveis respectivamente pela segurança e pela gestão dos dados do usuário:

- `POST /auth/`: Endpoint para o registro de novos usuários no sistema.
- `POST /auth/token`: Endpoint de login que autentica o usuário e retorna um token de acesso JWT (*JSON Web Token*), que é necessário para autorizar todas as requisições subsequentes.
- `GET /users/me/profile`: Permite que o cliente recupere as informações do perfil do usuário logado, possibilitando a personalização da interface e da experiência.

4.2.4.3.2 Gerenciamento de conversas

Para que o usuário possa revisitar e gerenciar seu histórico de aprendizado, foram criados os seguintes endpoints no controlador `/conversations`:

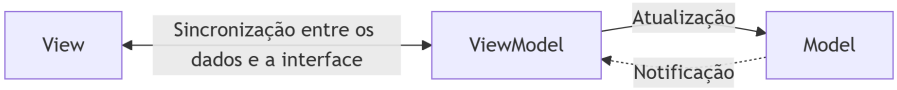
- `GET /conversations/`: Retorna uma lista com todas as conversas anteriores do usuário, permitindo que ele navegue por seu histórico.
- `GET /conversations/{conversation_id}`: Recupera o histórico completo de mensagens de uma conversa específica, incluindo as transcrições de texto e as correções gramaticais que foram salvas. Este endpoint é fundamental para o aspecto pedagógico, permitindo ao usuário revisar seus erros e o feedback recebido.
- `DELETE /conversations/{conversation_id}`: Permite ao usuário apagar uma conversa de seu histórico, garantindo o controle sobre seus próprios dados.

Este conjunto de endpoints complementa o fluxo de conversação principal.

4.2.5 Implementação do *frontend* com a arquitetura MVVM

A interface do cliente constitui uma aplicação móvel multiplataforma, desenvolvida com o objetivo de proporcionar uma experiência de usuário intuitiva. Para alcançar este propósito, a arquitetura do *frontend* foi fundamentada no padrão de projeto *Model-View-ViewModel* (MVVM) (Fuksa; Speth; Becker, 2024), que promove a separação de responsabilidades, conforme ilustrado na Figura 9.

Figura 9 – A arquitetura MVVM



Fonte: Adaptado de Fuksa, Speth e Becker (2024).

4.2.5.1 Implementação da arquitetura MVVM

A arquitetura MVVM foi seguida na estrutura de diretórios do projeto, principalmente dentro de `src/viewModels`. Cada componente foi dividido em suas três responsabilidades principais:

- View: Representa a camada de interface do usuário (UI). No projeto, as Views são implementadas como componentes React, localizados nos arquivos `view.tsx`. A única responsabilidade da View é renderizar a interface com base nos dados fornecidos pelo ViewModel e notificar o ViewModel sobre as interações do usuário.
- ViewModel: Atua como a ponte entre a View e o Model. Os ViewModels, implementados nos arquivos `model.tsx`, são desenvolvidos como *hooks* do React. Eles expõem o estado para a View e as funções que a View pode chamar. O ViewModel contém toda a lógica de apresentação e formatação de dados, mantendo a View limpa.
- Model: Se trata da camada de dados e lógica de negócio da aplicação. No frontend, esta camada é composta pelos serviços em `src/shared/services`, que possuem a lógica de comunicação com a API backend e o gerenciamento de dados locais. O Model é independente da interface, sendo responsável por tratar os dados.

4.2.5.2 Gerenciamento de estado e comunicação com a API

O gerenciamento de estado, especialmente o estado do servidor, é um aspecto fundamental da aplicação.

- TanStack Query: Em vez de gerenciar manualmente estados de carregamento, erro e sucesso, a biblioteca TanStack Query abstrai essas complexidades, simplificando a lógica nos serviços e ViewModels.
- Axios e Interceptadores: Uma instância centralizada do Axios, foi implementada com interceptadores de requisição e resposta. O interceptador de requisição é responsável

por anexar automaticamente o token de autenticação JWT a todas as chamadas protegidas. O interceptador de resposta detecta automaticamente respostas expiradas com status `401 Unauthorized`, executando um fluxo de logout forçado e redirecionando o usuário para a tela de login.

4.2.5.3 Estrutura de navegação e rotas protegidas

A navegação da aplicação foi feita de forma a separar os fluxos de autenticação e o conteúdo principal.

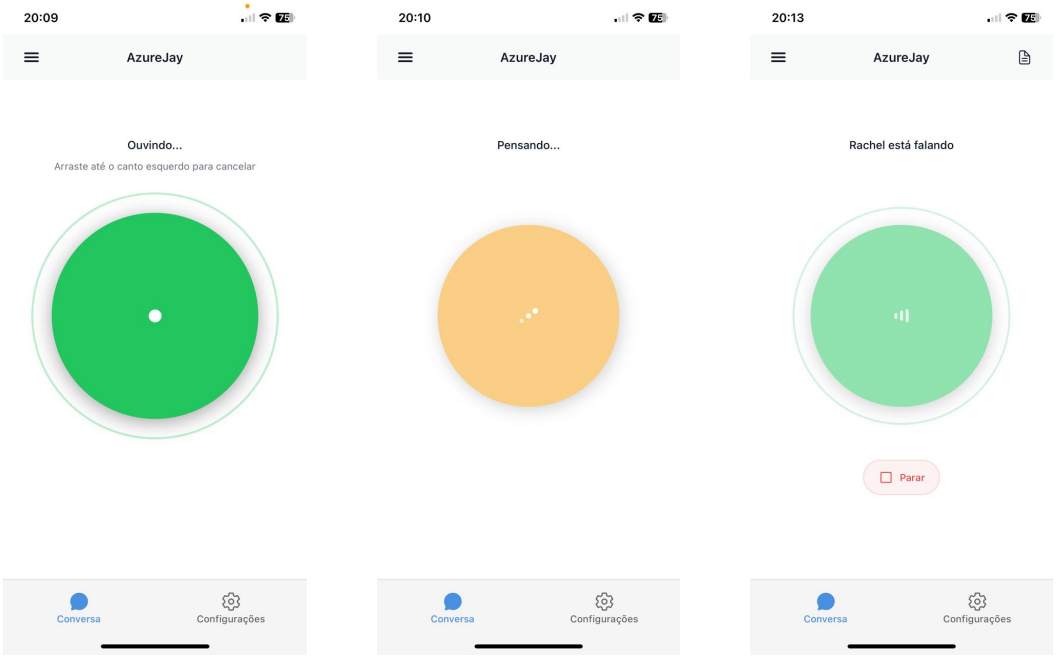
- Roteamento: Utilizando o Expo Router, as rotas foram organizadas em grupos, sendo o `(auth)` para as telas de login e registro, e `(tabs)` para as telas principais da aplicação após a autenticação, como conversa e configurações.
- Rotas protegidas (*Protected Routes*): Para garantir que apenas usuários autenticados possam acessar o grupo `(tabs)`, foi implementado um componente que envolve os layouts dos grupos de rotas e utiliza o hook de autenticação para verificar o estado de autenticação do usuário. Caso um usuário não autenticado tente acessar uma rota protegida, ele é automaticamente redirecionado para a tela de login.

4.2.5.4 Componentização e experiência do usuário

Além da estruturada arquitetura MVVM, foram implementadas diversas estratégias de *design* e componentização para aprimorar a experiência do usuário, dentre as quais se destaca:

- Feedback visual e tátil em tempo real: O componente central da interação do usuário foi projetado para fornecer *feedback* imediato. Utilizando as APIs de animação do React Native, o componente exibe estados visuais distintos para cada fase da interação: ocioso (*idle*), gravando (*recording*), processando (*processing*) e reproduzindo (*playing*), conforme demonstrado na Figura 10. A interação de tocar e segurar para falar é gerenciada pela biblioteca `react-native-gesture-handler`, que também detecta o gesto de arrastar para o lado para cancelar a gravação.

Figura 10 – Telas de *feedback* que demonstram os estados da reprodução do áudio



(a) Tela durante o envio do áudio.

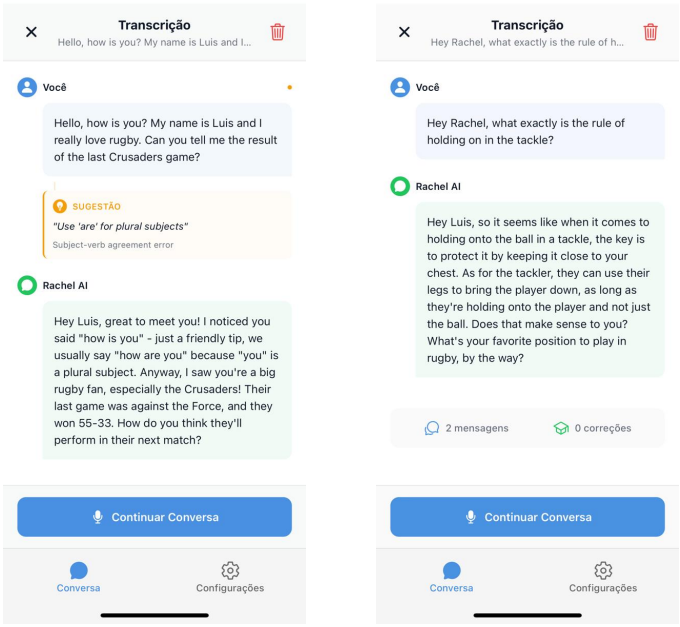
(b) Tela de processamento do áudio.

(c) Tela de reprodução do áudio.

Fonte: Elaborado pelo autor.

- Apresentação pedagógica de correções: Na tela de transcrição da conversa, a apresentação do *feedback* é o elemento pedagógico mais forte do sistema. O componente `MessageItemView` é responsável por verificar os dados de cada mensagem recebida da API. No servidor, a entidade `GrammarCorrection` armazena a análise da correção, caso exista. Quando o cliente solicita o histórico de uma conversa através do *endpoint* `GET /conversations/{id}`, a camada de serviço a transforma em um DTO (*Data Transfer Object*) por meio de um modelo Pydantic. O componente `MessageItemView` então verifica a presença do objeto `correction` dentro de cada mensagem. Se presente, ele exibe a sugestão de correção em destaque na tela, representando a correção gramatical diretamente abaixo da mensagem original do usuário, conforme exemplificado na Figura 11.

Figura 11 – Telas de transcrição com e sem erros



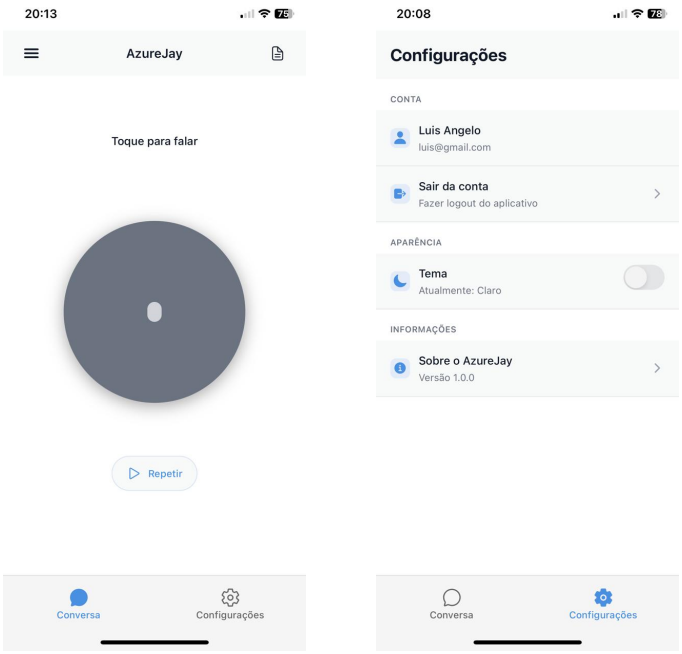
(a) Tela de transcrição com correção.

(b) Tela de transcrição sem erro.

Fonte: Elaborado pelo autor.

- Persistência de áudio: Para permitir que o usuário possa ouvir novamente a resposta do tutor, o sistema implementa um serviço chamado `conversationAudioService`, que gerencia a gravação local do último áudio de cada conversa. Este serviço utiliza o `AsyncStorage` para manter um mapa entre o ID da conversa e o URI do arquivo de áudio salvo no dispositivo. Além disso, ao receber um novo áudio para uma conversa, o serviço apaga o arquivo anterior para economizar espaço e, ao carregar a lista de conversas, executa uma rotina de limpeza para remover arquivos de áudio que pertenciam a conversas anteriores que foram deletadas, conforme ilustrado na Figura 12.

Figura 12 – Telas de persistência do último áudio de cada conversa e de configuração



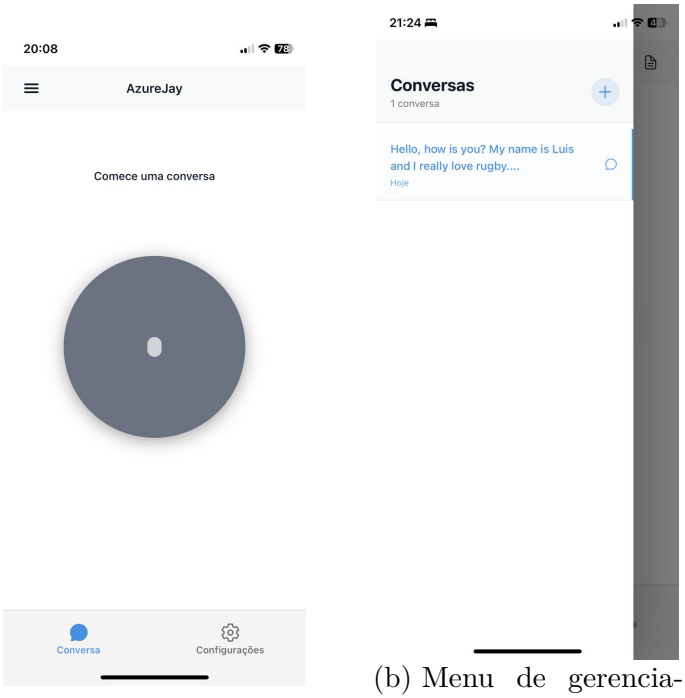
(a) Tela de conversa (com ID).

(b) Tela de configurações.

Fonte: Elaborado pelo autor.

- Fluxo de interação e gerenciamento de conversas: O fluxo de interação do usuário é gerenciado pelo estado da sessão. Para iniciar uma nova conversa, o usuário utiliza uma ação no menu lateral que redefine este estado para `null`; a primeira gravação de áudio é então enviada ao *endpoint* de criação (`POST /audio/new`). Uma vez que a conversa é criada e seu ID é mantido no estado do cliente, todas as gravações seguintes são direcionadas ao *endpoint* de continuação (`POST /audio/chat/{id}`). O acesso ao histórico de sessões é realizado através de um menu lateral que permite ao usuário selecionar uma conversa anterior. As interfaces correspondentes a esse fluxo estão dispostas na Figura 13. Ao selecionar uma conversa, ou ao acionar o ícone de transcrição na tela principal, o usuário é direcionado para a `TranscriptionView`. Esta tela serve para revisão da conversa e verificação das correções e sugestões de melhoria.

Figura 13 – Telas do fluxo de interação que exibem nova conversa e o histórico de conversas



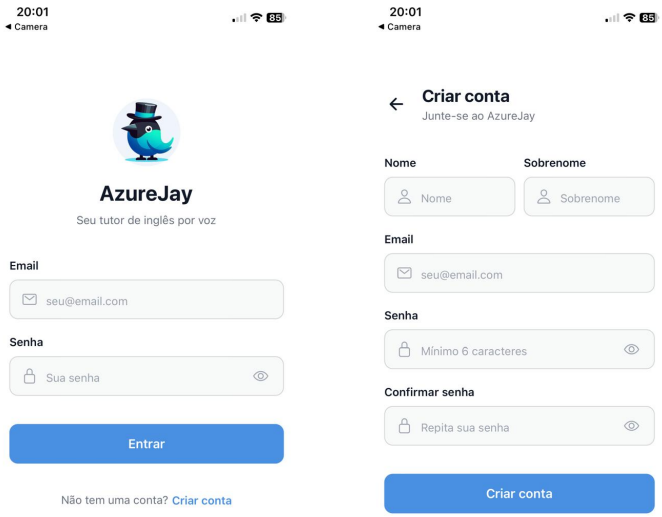
(a) Tela inicial (sem ID).

(b) Menu de gerenciamento.

Fonte: Elaborado pelo autor.

- Fluxo de autenticação e validação de dados: A tela inicial de entrada da aplicação é composta pelas telas de *login* e registro. As credenciais validadas são enviadas aos *endpoints* do servidor (`POST /auth/token` e `POST /auth/`), que cuidam da verificação, criptografia de senhas e geração de *tokens* JWT. Caso o usuário se autentique com sucesso, ocorre o redirecionamento para a tela principal da aplicação, conforme apresentado na Figura 14.

Figura 14 – Telas de autenticação iniciais



(a) Tela de *login*. (b) Tela de registro.

Fonte: Elaborado pelo autor.

5 Resultados e discussão

Este capítulo apresenta uma análise de resultados de natureza qualitativa cujo objetivo é discutir a eficácia da arquitetura proposta frente aos componentes fundamentais que definem um agente de IA eficaz.

Para isso, Zhu et al. (2025) conduziram uma investigação sistemática para identificar quais componentes são cruciais para o desempenho de um agente inteligente, culminando no framework *OAgents*. Neste presente trabalho, a fim de analisar tanto as capacidades quanto as limitações do sistema desenvolvido, foram adotados os principais pilares do framework: planejamento, ferramentas e memória, que serão explorados a seguir.

5.1 Análise qualitativa frente aos pilares do OAgents

5.1.1 Capacidade de planejamento (*Planning*)

O estudo do OAgents destaca que um planejamento eficaz é o componente mais crítico para o sucesso em tarefas complexas, propondo mecanismos como a revisão dinâmica do plano (*Dynamic Plan Revise*) e a decomposição em subtarefas (*Subtask Decomposition*)(Zhu et al., 2025).

O sistema proposto implementa essas capacidades forçando o agente a gerar um plano explícito dentro da tag `<plan>` antes de qualquer ação. Além disso, a tag `<reflection>` serve como um mecanismo de revisão e autocrítica do agente. Após executar uma ferramenta e receber uma observação, o agente pode então refletir sobre o resultado, identificar se o plano original ainda é válido e, caso seja necessário, ajustar os passos seguintes. Isso se equipara ao ciclo de revisão dinâmica apontado pelo estudo do *OAgents*. A decomposição da tarefa, por sua vez, ocorre naturalmente na geração do plano inicial, no qual o agente detalha os passos que tomará para chegar na resposta final.

5.1.2 Utilização de ferramentas (*Ensembled tools*)

O framework OAgents demonstra que a eficácia de um agente está diretamente ligada à qualidade e diversidade das ferramentas disponibilizadas para ele, incluindo ferramentas para busca (*Search Agent*) e capacidades multimodais (Zhu et al., 2025).

A arquitetura deste trabalho, embora focada no domínio da linguagem, adota o mesmo princípio de delegação de atividades. O agente principal não tenta realizar todas as tarefas sozinho, ele cria um plano e invoca as ferramentas tanto simbólicas quanto neurais conforme a necessidade para executar o plano com sucesso:

- Para informações externas atualizadas, ele delega a tarefa à ferramenta `WebSearch`.
- Para persistir conhecimento, ele utiliza `UpdateUserProfile` e `SaveGrammarCorrection`, que interagem com o banco de dados.
- Para a análise gramatical inicial, todo o pipeline de processamento gramatical pode ser visto como uma ferramenta especializada que o sistema invoca no início de cada ciclo.

5.1.3 Gerenciamento de memória

O *OAgents* propõe um sistema de memória hierárquico, com memória de curto e longo prazo, mecanismos de sumarização e também de recuperação vetorial para garantir a continuidade do sistema ao longo do tempo (Zhu et al., 2025).

O sistema desenvolvido implementa esses conceitos, alinhando-se com a estrutura hierárquica proposta no estudo:

- Memória de curto prazo: É implementada passando o histórico da conversa atual para o prompt do agente a cada turno. Isso serve como a memória de trabalho volátil, o que disponibiliza ao agente o contexto imediato da interação.
- Memória de longo prazo: É realizada pela ferramenta `UpdateUserProfile` e pela persistência das correções gramaticais. Conforme detalhado na metodologia, o sistema utiliza o padrão *Write-Through Cache* para manter a consistência entre o banco de dados PostgreSQL e o cache Redis. Isso visa garantir que o perfil persistente do usuário esteja sempre atualizado e seja acessível rapidamente a fim de personalizar a experiência de aprendizado em futuras conversas.

5.2 Análise quantitativa de eficiência

Para complementar a análise qualitativa, foi conduzido também um teste de eficiência. O ambiente de execução para ambos os testes utilizou Docker e foram executados em uma máquina local cujas especificações estão detalhadas na tabela 3 a seguir.

Tabela 3 – Especificações do ambiente de teste

Componente	Especificação
Máquina *(Host)*	
Sistema	Notebook Dell G15 5511
Processador (CPU)	Intel Core i5 (9ª Geração) @ 2.69 GHz
Memória (RAM)	16 GB
Sistema Operacional	Windows 11 Home Single Language (x64)
Ambiente de execução	
Imagens Docker	Python 3.11-slim, Postgres 17, Redis Alpine

Fonte: Elaborado pelo autor.

Neste ambiente, foi conduzido um teste para quantificar o custo computacional e a latência da arquitetura proposta em comparação com uma implementação supervisor multiagente tradicional usando o LangGraph.

Ambas as arquiteturas receberam o mesmo prompt de entrada, projetado para o pior cenário em termos de latência e custos de API. O prompt busca tanto a extração de memória de perfil do usuário, como nome e interesses, quanto correções gramaticais e a busca na web por informações atualizadas:

> *Hello, my name is Luis and I really loves Rugby. Can you tell me the result of the last Crusaders game?*

A arquitetura supervisor multiagente opera invocando sequencialmente diferentes agentes (planejador, agente de busca, agente de resposta), o que gera uma sobrecarga de comunicação e múltiplas chamadas de LLM para orquestração. A arquitetura proposta neste trabalho, por sua vez, internaliza esses papéis em um único ciclo de inferência, conforme detalhado no Capítulo 4.

Para a validação, ambas as arquiteturas utilizaram a API do Groq com o modelo Llama 3.1 70B, sendo que foram contabilizadas apenas as chamadas realizadas ao LLM. Os resultados obtidos demonstram uma melhora em eficiência e foram resumidos na Tabela 4,

Tabela 4 – Comparativo de eficiência entre arquiteturas

Métrica	Supervisor multiagente	Arquitetura proposta (AFM)
Chamadas na API (LLM)	28	8
Tempo de Resposta (s)	32,21	8,79

Fonte: Elaborado pelo autor.

A redução de 71,43% no número de chamadas à API (de 28 para 8) é um resultado da internalização do raciocínio. Enquanto a abordagem multiagente gasta chamadas para a comunicação entre diversos agentes, o paradigma CoA permite ao agente planejar, refletir e agir dentro de menos ciclos de inferência.

Consequentemente, a latência total foi reduzida em 72,71% (de 32,21s para 8,79s). Isso valida a hipótese levantada na fundamentação teórica de que o CoA oferece uma alternativa de baixo custo e baixa latência, crucial para a viabilidade de aplicações em produção.

5.3 Discussão e limitações

As análises apresentadas neste capítulo demonstram a eficiência da arquitetura por meio tanto da análise qualitativa quanto da análise quantitativa. A análise qualitativa implementa os princípios mais fundamentais apontados pelo estudo do *Oagents* que definem um agente eficaz, especificamente o planejamento dinâmico via reflexão, separação de responsabilidades e o uso de ferramentas e memória.(Zhu et al., 2025)

Somado a isso, a análise quantitativa demonstra que a abordagem do *Chain-of-Agents* (CoA), ainda que implementada de forma parcial, atinge os objetivos de eficiência, reduzindo o número de chamadas de API e a latência em comparação com uma arquitetura multiagente tradicional, se mostrando assim, uma opção viável para uma aplicação real.

Entretanto, é importante reconhecer as limitações do presente trabalho. O sistema não implementa todos os componentes do OAgents, notavelmente a estratégia do *Best-of-N* (Zhu et al., 2025) que consiste em gerar trajetórias de resposta candidatas e selecionar a de maior qualidade através de um modelo de recompensa, a fim de aumentar a precisão das respostas do agente, embora acarrete um custo maior de latência e chamadas de API. Além disso, não foi implementado o processo de destilação de conhecimento, o que poderia otimizar ainda mais a eficiência do agente.

Ainda, uma avaliação mais ampla se faz necessária. Um trabalho futuro seria a condução de uma avaliação quantitativa em larga escala, criando um benchmark especializado a fim de validar a eficácia da arquitetura frente a outras implementações.

6 Considerações finais

Neste trabalho, foi desenvolvido um tutor de IA conversacional para o aprendizado de inglês, um desafio caracterizado pelas limitações das abordagens puramente neurais (LLMs), que podem fornecer informações incorretas e carecem de personalização pedagógica, bem como das arquiteturas multiagente tradicionais, que sofrem com alta latência e custos computacionais elevados. Observando a evolução das arquiteturas de agentes, ficou evidente que novos paradigmas como o *Chain-of-Agents* (CoA) surgem como uma solução promissora.

Assim, este trabalho adotou um *pipeline* de processamento em camadas para análise gramatical sintática e semântica, e um agente central unificado (AFM) que simula internamente a colaboração de especialistas. Isso possibilitou não apenas a análise linguística precisa, mas também a manutenção de uma conversação fluida e com menor custo computacional, superando a latência dos sistemas multiagente.

A aplicação móvel desenvolvida teve como objetivo fornecer uma ferramenta prática de conversação voltada ao aprendizado de uma segunda língua. Ela oferece suporte ao aprendizado por meio de *feedback* instantâneo e personalização baseada no histórico do usuário, o que proporciona um ambiente seguro para a prática oral para quem sofre de ansiedade conversacional. No entanto, é essencial enfatizar que esta aplicação possui caráter majoritariamente acadêmico, sendo um protótipo de pesquisa.

É importante salientar que a aplicação não foi desenvolvida para ser vista como uma substituição para a instrução formal de um professor qualificado. Em vez disso, ela fornece uma ferramenta de prática adicional, destinada a auxiliar o estudante em seu próprio ritmo de estudo.

Ademais, destaca-se a natureza da avaliação realizada, que foi conduzida, primeiramente, através de uma análise qualitativa, utilizando os pilares fundamentais do *framework OAgents* para validar a eficácia teórica da arquitetura. Em complemento, uma análise quantitativa de eficiência demonstrou que a abordagem proposta atinge uma redução de latência (72,71%) e de chamadas de API (71,43%) em comparação com uma implementação multiagente tradicional usando LangGraph. Embora esta validação confirme a viabilidade da arquitetura, é essencial pontuar que o trabalho não se estendeu a uma avaliação de *performance* em *benchmarks* padronizados utilizando uma quantidade maior de casos de teste.

Adicionalmente, é crucial reconhecer as limitações do protótipo desenvolvido. O

sistema não implementa todos os componentes do *OAgents*. Ainda, o fluxo do AFM foi realizado via engenharia de *prompt* (*in-context learning*), não implementando o processo de destilação de conhecimento. Esta destilação, embora fora do escopo deste trabalho, trata-se do método que otimizaria ainda mais a eficiência do agente, conforme apontado por Li et al. (2025).

Como trabalho futuro, destaca-se a oportunidade de expandir a validação quantitativa, analisando-a em *benchmarks* padronizados para comparar sua *performance* de forma mais ampla. Como principal evolução técnica, sugere-se ainda a implementação do processo de destilação de conhecimento para a criação de um AFM especializado, superando a engenharia de *prompt*. Por fim, é importante conduzir estudos de caso com usuários reais, bem como obter *feedback* de professores qualificados no ensino de inglês a fim de aprimorar os aspectos pedagógicos da aplicação. A busca pela melhoria contínua dos modelos de agente e da experiência do usuário é fundamental para que aplicações como essa possam contribuir de maneira relevante para o aprendizado de idiomas no futuro.

Referências

ABUSAHYON, A. S. E.; ALZYOUD, A.; ALSHORMAN, O.; AL-ABSI, B. Ai-driven technology and chatbots as tools for enhancing english language learning in the context of second language acquisition: A review study. *International Journal of Membrane Science and Technology*, v. 10, n. 1, p. 1209–1223, 2023.

AXIOS. *Axios*. 2025. Disponível em: https://axios-http.com/ptbr/docs/intro. Acesso em: 12 nov. 2025.

BARNES, E.; HUTSON, J. Natural language processing and neurosymbolic ai: The role of neural networks with knowledge-guided symbolic approaches. *Journal of Artificial Intelligence and Robotics*, v. 2, n. 1, 2024.

BERNACKI, M. L.; GREENE, J. A.; CROMPTON, H. Mobile technology, learning, and achievement: Advances in understanding and measuring the role of mobile technology in education. *Contemporary Educational Psychology*, Elsevier, v. 60, p. 101827, 2020.

BOMMASANI, R.; HUDSON, D. A.; ADELI, E.; ALTMAN, R.; ARORA, S.; ARX, S. von; BERNSTEIN, M. S.; BOHG, J.; BOSSELUT, A.; BRUNSKILL, E. et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

BUCHNER, J.; ANDUJAR, A. The expansion of the classroom through mobile immersive learning. *International Association for Development of the Information Society*, ERIC, 2019.

CARLSON, J. *Redis in action*. [S.l.]: Simon and Schuster, 2013.

CASELI, H. M.; NUNES, M. G. V. (Ed.). *Processamento de Linguagem Natural: Conceitos, Técnicas e Aplicações em Português*. 3. ed. [S.l.]: BPLN, 2024. Disponível em: https://brasileiraspln.com/livro-pln/3a-edicao/. ISBN 978-65-01-20581-6.

CHANG, Y.; WANG, X.; WANG, J.; WU, Y.; YANG, L.; ZHU, K.; CHEN, H.; YI, X.; WANG, C.; WANG, Y. et al. A survey on evaluation of large language models. *ACM transactions on intelligent systems and technology*, ACM New York, NY, v. 15, n. 3, p. 1NOT–45, 2024.

Duolingo. *Introducing Duolingo Max, a learning experience powered by GPT-4*. 2023. Disponível em: https://blog.duolingo.com/duolingo-max/. Acesso em: 23 mar. 2025.

ELEVENLABS. *ElevenLabs API Documentation*. 2025. Disponível em: https://elevenlabs.io/docs/overview. Acesso em: 13 nov. 2025.

ELLIS, R.; SHINTANI, N. Implicit and explicit instruction in l2 learning: Norris & ortega (2000) revisited and updated. *Routledge handbook of instructed second language acquisition*, Routledge New York, NY, p. 24–46, 2019.

ESCUETA, M.; QUAN, V.; NICKOW, A. J.; OREOPOULOS, P. Education technology: An evidence-based review. National Bureau of Economic Research, 2017.

EXPO. *Expo Documentation*. 2025. Disponível em: https://docs.expo.dev/. Acesso em: 12 nov. 2025.

FUKSA, M.; SPETH, S.; BECKER, S. Mvvm revisited: Exploring design variants of the model-view-viewmodel pattern. In: SPRINGER. *International Conference on Enterprise Design, Operations, and Computing*. [S.l.], 2024. p. 163–181.

GILAKJANI, A. P.; SABOURI, N. B. Learners' listening comprehension difficulties in english language learning: A literature review. *English language teaching*, ERIC, v. 9, n. 6, p. 123–133, 2016.

GOLDBERG, J. *Learning TypeScript*. [S.l.]: "O'Reilly Media, Inc.", 2022.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press, 2016.

GROQ. *Groq API Quickstart*. 2025. Disponível em: https://console.groq.com/docs/quickstart. Acesso em: 13 nov. 2025.

HADFIELD, J.; ZHANG, B.; LIEN, K.; SCHOLZ, F.; FOX, J.; FORD, D. *How we built our multi-agent research system*. 2024. Disponível em: https://www.anthropic.com/engineering/multi-agent-research-system/. Acesso em: 02 ago. 2025.

HORWITZ, E. K.; HORWITZ, M. B.; COPE, J. Foreign language classroom anxiety. *The Modern language journal*, JSTOR, v. 70, n. 2, p. 125–132, 1986.

IBGE. *Pesquisa IBGE 2023*. 2024. Disponível em: https://agenciadenoticias.ibge.gov.br/agencia-noticias/2012-agencia-de-noticias/noticias/41026-em-2023-87-2-das-pessoas-com-10-anos-ou-mais-utilizaram-internet/. Acesso em: 21 mar. 2025.

ILYOSOVNA, N. A. The importance of english language. *International Journal on Orange Technologies*, Research Parks Publishing, v. 2, n. 1, p. 22–24, 2020.

JOUPPI, N. P. Cache write policies and performance. *ACM SIGARCH Computer Architecture News*, ACM New York, NY, USA, v. 21, n. 2, p. 191–201, 1993.

JUBA, S.; VANNAHME, A.; VOLKOV, A. *Learning PostgreSQL*. [S.l.]: Packt Publishing Ltd, 2015.

KARPAS, E.; ABEND, O.; BELINKOV, Y.; LENZ, B.; LIEBER, O.; RATNER, N.; SHOHAM, Y.; BATA, H.; LEVINE, Y.; LEYTON-BROWN, K. et al. Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*, 2022.

LANGGRAPH. *Agent development using prebuilt components*. 2025. Disponível em: https://langchain-ai.github.io/langgraph/agents/overview/. Acesso em: 05 set. 2025.

LANGUAGETOOL. *LanguageTool HTTP API*. 2025. Disponível em: https://languagetool.org/http-api/. Acesso em: 13 nov. 2025.

LEE, S.; JEON, J. Visualizing a disembodied agent: Young efl learners' perceptions of voice-controlled conversational agents as language partners. *Computer Assisted Language Learning*, Taylor & Francis, v. 37, n. 5-6, p. 1048–1073, 2024.

LEVY, M. *Computer-assisted language learning: Context and conceptualization*. [S.l.]: Oxford university press, 1997.

LI, W.; LIN, J.; JIANG, Z.; CAO, J.; LIU, X.; ZHANG, J.; HUANG, Z.; CHEN, Q.; SUN, W.; WANG, Q. et al. Chain-of-agents: End-to-end agent foundation models via multi-agent distillation and agentic rl. *arXiv preprint arXiv:2508.13167*, 2025.

LUBANOVIC, B. *FastAPI*. [S.l.]: "O'Reilly Media, Inc.", 2023.

LYU, B.; LAI, C.; GUO, J. Effectiveness of chatbots in improving language learning: A meta-analysis of comparative studies. *International Journal of Applied Linguistics*, Wiley Online Library, v. 35, n. 2, p. 834–851, 2025.

MARCUS, G. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*, 2018.

MARCUS, G.; DAVIS, E. *Rebooting AI: Building artificial intelligence we can trust*. [S.l.]: Vintage, 2019.

MARTIN, R. C. *Clean architecture: a craftsman's guide to software structure and design*. [S.l.]: Prentice Hall Press, 2017.

MIALON, G.; FOURRIER, C.; WOLF, T.; LECUN, Y.; SCIALOM, T. Gaia: a benchmark for general ai assistants. In: *The Twelfth International Conference on Learning Representations*. [S.l.: s.n.], 2023.

MIELL, I.; SAYERS, A. *Docker in practice*. [S.l.]: Simon and Schuster, 2019.

MIRA, J. M. Symbols versus connections: 50 years of artificial intelligence. *Neurocomputing*, Elsevier, v. 71, n. 4-6, p. 671–680, 2008.

MYERS, J.; COPELAND, R.; COPELAND, R. D. *Essential SQLAlchemy*. [S.l.]: "O'Reilly Media, Inc.", 2015.

NARAYANAN, P. K. Getting started with data validation using pydantic and pandera. In: *Data Engineering for Machine Learning Pipelines: From Python Libraries to ML Pipelines and Cloud Platforms*. [S.l.]: Springer, 2024. p. 163–196.

NATIVEWIND. *NativeWind*. 2025. Disponível em: https://www.nativewind.dev/. Acesso em: 12 nov. 2025.

OSHIN, M.; CAMPOS, N. *Learning LangChain*. [S.l.]: "O'Reilly Media, Inc.", 2025.

PELLURU, K. Langchain & langgraph in production: Architectures for multi-agent llm systems. *Journal of Data and Digital Innovation (JDDI)*, v. 2, n. 3, p. 1–9, 2025.

RUSSELL, S. J.; NORVIG, P. *Artificial intelligence: a modern approach*. [S.l.]: pearson, 2016.

SHAHZAD, T.; MAZHAR, T.; TARIQ, M. U.; AHMAD, W.; OUAHADA, K.; HAMAM, H. A comprehensive review of large language models: issues and solutions in learning environments. *Discover Sustainability*, Springer, v. 6, n. 1, p. 27, 2025.

SHORTT, M.; TILAK, S.; KUZNETCOVA, I.; MARTENS, B.; AKINKUOLIE, B. Gamification in mobile-assisted language learning: A systematic review of duolingo literature from public release of 2012 to early 2020. *Computer Assisted Language Learning*, Taylor & Francis, v. 36, n. 3, p. 517–554, 2023.

SHRIDHAR, M.; YUAN, X.; CÔTÉ, M.-A.; BISK, Y.; TRISCHLER, A.; HAUSKNECHT, M. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

TALATI, D. Python: The alchemist behind ai's intelligent evolution. *Available at SSRN 5201760*, 2021.

TANSTACK. *TanStack Query*. 2025. Disponível em: https://tanstack.com/query/latest/docs/framework/react/overview. Acesso em: 12 nov. 2025.

TAVILY. *Tavily Search API Documentation*. 2025. Disponível em: https://docs.tavily.com/welcome. Acesso em: 13 nov. 2025.

VALENTE, M. T. Engenharia de software moderna. *Princípios e práticas para desenvolvimento de software com produtividade*, v. 1, n. 24, 2020.

WEI, J.; WANG, X.; SCHUURMANS, D.; BOSMA, M.; XIA, F.; CHI, E.; LE, Q. V.; ZHOU, D. et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, v. 35, p. 24824–24837, 2022.

XU, Y.; AUBELE, J.; VIGIL, V.; BUSTAMANTE, A. S.; KIM, Y.-S.; WARSCHAUER, M. Dialogue with a conversational agent promotes children's story comprehension via enhancing engagement. *Child Development*, Wiley Online Library, v. 93, n. 2, p. e149–e167, 2022.

YANG, Z.; QI, P.; ZHANG, S.; BENGIO, Y.; COHEN, W.; SALAKHUTDINOV, R.; MANNING, C. D. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In: *Proceedings of the 2018 conference on empirical methods in natural language processing*. [S.l.: s.n.], 2018. p. 2369–2380.

YAO, S.; ZHAO, J.; YU, D.; DU, N.; SHAFRAN, I.; NARASIMHAN, K.; CAO, Y. React: Synergizing reasoning and acting in language models. In: *International Conference on Learning Representations (ICLR)*. [S.l.: s.n.], 2023.

YU, D.; YANG, B.; LIU, D.; WANG, H.; PAN, S. A survey on neural-symbolic learning systems. *Neural Networks*, Elsevier, v. 166, p. 105–126, 2023.

ZHANG, Z.; HUANG, X. The impact of chatbots based on large language models on second language vocabulary acquisition. *Heliyon*, Elsevier, v. 10, n. 3, 2024.

ZHENG, L.; CHIANG, W.-L.; SHENG, Y.; ZHUANG, S.; WU, Z.; ZHUANG, Y.; LIN, Z.; LI, Z.; LI, D.; XING, E. P.; ZHANG, H.; GONZALEZ, J. E.; STOICA, I. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

ZHU, H.; QIN, T.; ZHU, K.; HUANG, H.; GUAN, Y.; XIA, J.; YAO, Y.; LI, H.; WANG, N.; LIU, P. et al. Oagents: An empirical study of building effective agents. *arXiv preprint arXiv:2506.15741*, 2025.