Desenvolvimento De Uma Plataforma Web Assistida Por Inteligência Artificial Para O Ensino Da Língua Inglesa

Romulo L. Bezerra[1], **Iago Medeiros**[1]

[1]Universidade Federal do Pará(UFPA)

romulo.bezerra@tucurui.ufpa.br, iagomedeiros@ufpa.br

Abstract. *This work presents the development, architecture, and validation of the ChatEP platform, an open-source web ecosystem designed for autonomous English learning. Based on the Task-Based Language Teaching (TBLT) pedagogical approach, the system orchestrates heterogeneous specialized Artificial Intelligence models (Flan-T5 Large, Whisper, and TinyLlama-1.1B) fine-tuned via Low-Rank Adaptation (LoRA), aiming to improve listening, writing, speaking, and reading skills. Quantitative validation was based on data telemetry metrics, applying the BLEU index for textual correction, the WER rate for the acoustic pipeline, and the BERTScore to evaluate the virtual tutor's semantic similarity. Additionally, a pilot study with real users validated usability and mapped architectural trade-offs regarding synchronous cloud inference latency. The experimental results obtained demonstrate high levels of accuracy and technical compliance of the ecosystem against the specified requirements and business rules.*

Keywords: *Software Engineering. Microservices. Fine-Tuning. Low-Rank Adaptation. Natural Language Processing. Systems Validation. Architectural Trade-offs.*

Resumo. *Este trabalho apresenta o desenvolvimento, a arquitetura e a homologação da plataforma ChatEP, um ecossistema web open source projetado para o aprendizado autônomo de inglês. Baseado na abordagem pedagógica Task-Based Language Teaching (TBLT), o sistema orquestra modelos heterogêneos de Inteligência Artificial (Flan-T5 Large, Whisper e TinyLlama-1.1B) especializados e refinados via Low-Rank Adaptation (LoRA), visando o aprimoramento de listening, writing, speaking e reading. A validação quantitativa baseou-se em métricas de telemetria de dados, aplicando o índice BLEU para correção textual, a taxa WER para o pipeline acústico e o BERTScore para avaliar a similaridade semântica do tutor virtual. Complementarmente, um estudo piloto com usuários reais validou a usabilidade e mapeou trade-offs de latência de inferência síncrona em ambiente de nuvem. Os resultados experimentais obtidos demonstram elevados índices de precisão e conformidade técnica do ecossistema frente aos requisitos e regras de negócio especificados.*

Palavras-chave: *Engenharia de Software. Microsserviços. Fine-Tuning. Low-Rank Adaptation. Processamento de Linguagem Natural. Homologação de Sistemas. Validação de Arquitetura.*

1. Introdução

No ecossistema de desenvolvimento de *software* e comunicações científicas internacionais, a língua inglesa consolidou-se como o meio padrão e dominante para a transferência de conhecimento, gerando uma necessidade contínua de seu domínio e aprimoramento por meio de ferramentas tecnológicas [Kalimullah and Tabassum 2025]. Contudo, o mercado de trabalho e a academia no Brasil, enfrentam um gargalo técnico, uma vez que os índices formais classificam o país como uma nação de "baixa proficiência" no idioma [British Council 2014, Olajide-Owoyomi 2025]. Esse cenário, portanto, impõe a necessidade de sistemas computacionais escaláveis que consigam atuar na mitigação dessa lacuna de competência técnica.

Como resposta a essa demanda, a engenharia de sistemas de informação tem se beneficiado da evolução de arquiteturas de *deep learning* e do aumento do poder de processamento de hardware [Dhilleswararao et al. 2022]. No domínio de *Computer Assisted Language Learning* (*CALL*), essas inovações permitiram projetar e implantar pipelines complexos de *software*, compostos por módulos de *Speech-to-Text*, agentes de conversação baseados em grandes modelos de linguagem (*LLMs*) e serviços de análise semântica assíncrona [Kalimullah and Tabassum 2025, Bahari et al. 2025]. Esses componentes automatizam tarefas de avaliação que antes exigiam processamento e intervenção humana especializada.

Contudo, criar plataformas interativas em tempo real ainda é um grande desafio de engenharia. Os programas atuais são muito rígidos e seguem fluxos fixos, falhando quando precisam lidar com muitos acessos simultâneos ou dados dinâmicos [Bogner et al. 2021, Mohd Sapuan et al. 2024]. Do ponto de vista técnico, juntar inteligência artificial ao processamento de áudio e texto aumenta a latência (tempo de resposta), o que estraga a experiência do usuário. Além disso, as plataformas mais consolidadas tendem a adotar o modelo de negócio *freemium*, onde oferecem uma amostra gratuita e cobram por funcionalidades avançadas e, dessa forma, o mercado e academia carece de aplicações *open source*. [Osipov et al. 2015].

Dessa forma, este trabalho propõe o desenvolvimento de uma plataforma web *open source* baseada na arquitetura cliente-servidor com integração de microsserviço independente. O sistema distribuirá suas funções isolando a interface do usuário em *Next.js*, as regras de negócio em *NestJS* e a persistência de dados em *MongoDB*. O processamento de inteligência artificial será concentrado em um serviço *Python* com *FastAPI*, integrado ao ecossistema principal por meio do design *pattern Proxy*. O trabalho prevê o *fine-tuning* de modelos de código aberto utilizando *Low-Rank Adaptation* (*LoRA*) para a criação de modelos próprios. O sistema utilizará a métrica *Word Error Rate (WER)* para validar o modelo de *speech-to-text* no módulo de *speaking*, *BLEU* para o modelo de correção de texto no módulo de *writing*, e *BERTScore* para a avaliação do modelo de conversação do tutor virtual. Cumpre ressaltar que a concepção inicial do ecossistema ChatEp, bem como sua validação preliminar sob a ótica da Cultura Maker e da abordagem pedagógica Task-Based Language Teaching (TBLT), foi descrita em artigo aceito para publicação nos anais do II Workshop de Inovação, Desenvolvimento, Educação e Inclusão com Ações Maker (IDEIA), evento integrante do CSBC 2026 [Bezerra and Medeiros 2026]. Como continuação direta desse estudo anterior, o presente artigo aprofunda e expande essa proposta sob a ótica da Engenharia de Software.

2. Trabalhos relacionados

O desenvolvimento de plataformas de *Computer-Assisted Language Learning* (*CALL*) voltadas ao ensino de idiomas demanda uma modelagem que alinhe a engenharia de requisitos às regras de negócio instrucionais. Nesse cenário, o paradigma do *Task-Based Language Teaching* (*TBLT*) deixa de ser apenas uma abordagem metodológica e passa a atuar como uma especificação arquitetural central [Li 2021]. Sob a ótica da engenharia de *software*, o núcleo do sistema deve ser estruturado para instanciar tarefas de execução pragmática baseadas em fluxos de entrada (*input*) e saída (*output*) de dados comunicativos em tempo real. No entanto, mapeia-se um débito técnico crônico na maioria das aplicações vigentes, as quais priorizam árvores de decisão rígidas e funções acopladas a exercícios estruturais isolados [Verdecchia et al. 2021]. Essa dissociação cria um gargalo computacional entre a necessidade de interações dinâmicas e o comportamento estático do *software*.

Trazendo o conceito de *CALL* para ambientes móveis, tem-se o subcampo *Mobile-Assisted Language Learning (MALL)*. Esse ecossistema se destaca por adotar arquiteturas distribuídas e mais acessíveis, uma vez que os serviços podem ser utilizados em qualquer lugar [Wang et al. 2024]. Pesquisas recentes demonstram que tal abordagem proporciona um ambiente mais autônomo, gerando confiança e sensação de progresso aos usuários; plataformas comerciais consolidadas, como o *Duolingo* e o *ELSA Speak*, validam essa viabilidade [Ahmad et al. 2025]. Contudo, essas soluções esbarram em limitações severas de acoplamento técnico: a validação de dados orientada a regras rígidas impede o tratamento de fluxos de linguagem natural espontâneos. Como consequência, o sistema falha em realizar análises semânticas contextualizadas, limitando os caminhos de execução do código e restringindo a tolerância a falhas linguísticas complexas.

Análises empíricas coordenadas por [Coşkun and Solmaz 2024] e [Wang et al. 2024] confirmam que os principais gargalos de produtos *MALL* não residem na portabilidade ou nas capacidades dos dispositivos móveis, mas sim em falhas de design instrucional computacional. Embora o *MALL* tenha consolidado avanços significativos em acessibilidade e distribuição de interfaces, a lógica de processamento no *backend* frequentemente permanece atrelada a regras rígidas. O principal desafio técnico decorre do processamento sequencial em *pipelines* de áudio convencionais, que integram blocos independentes de Reconhecimento Automático de Fala *(ASR)* e Processamento de Linguagem Natural (PLN) [Jurafsky and Martin 2026], onde essa arquitetura em cadeia compromete a fluidez da interação em tempo real [Choi et al. 2026]. Para superar esse débito técnico, o sistema vai além da migração de interface, substituindo fluxos determinísticos por microsserviços baseados em modelos generativos customizados. Nessa abordagem, as diretrizes do *TBLT* funcionam como regras de negócio que orquestram o comportamento do *software*. Essa integração transforma a aplicação em um sistema adaptativo orientado a eventos, capaz de processar requisições contextuais dinamicamente a cada interação.

Contudo, a implementação dessa orquestração esbarra nas restrições de infraestrutura dos modelos *Transformers* tradicionais. Como demonstrado por [Kalra and Sharma 2025], o processamento de bilhões de parâmetros exige alto consumo de hardware, o que inviabiliza o *fine-tuning* convencional em servidores menores ou dispositivos locais (*edge computing*), além de inflar os custos de nuvem. Como resposta

a essa limitação, os modelos de linguagem de menor escala (*Small Language Models* - SLMs) surgem como alternativas viáveis para a execução descentralizada do sistema. Ao serem otimizados, eles mitigam problemas de privacidade e reduzem drasticamente a latência de resposta, medida pelo *Time to First Token* (*TTFT*) [Islam et al. 2024].

Para viabilizar a customização desses *SLMs* sob essas restrições estritas de hardware, técnicas de *Parameter-Efficient Fine-Tuning* (*PEFT*) consolidam-se como o padrão necessário para otimizar a infraestrutura. A abordagem de *Low-Rank Adaptation (LoRA)* atua diretamente nessa camada ao decompor as matrizes de atualização de pesos do modelo em fatores de baixa dimensão matemática [Kalra and Sharma 2025]. Esse arranjo reduz drasticamente a pegada de memória e a alocação de *VRAM* e *RAM* durante o treinamento, preservando a acurácia semântica sem elevar a perplexidade computacional do sistema [Islam et al. 2024]. Além disso, o uso do *LoRA* impede o colapso de acurácia comum a processos agressivos de quantização uniforme, garantindo a robustez necessária para os microsserviços da aplicação [Choi et al. 2026].

Diante das lacunas identificadas, esta investigação caracteriza-se como uma pesquisa aplicada com desenvolvimento tecnológico sob abordagem empírico-experimental. A metodologia dividiu-se em três fases sequenciais: (I) a otimização de modelos de linguagem de menor escala e pipelines acústicos via *PEFT* com *LoRA* para mitigar restrições de hardware; (II) a implantação de uma arquitetura distribuída orientada a microsserviços para orquestrar as regras de negócio do método *Task-Based Language Teaching* (*TBLT*); e (III) a validação preliminar da estabilidade técnica e da telemetria de desempenho do ecossistema por meio de testes com usuários sob cenários de concorrência e carga.

3. Requisitos e Especificações do Sistema

A concepção da plataforma orientada ao ensino assistido por Inteligência Artificial (IA) exige o mapeamento sistemático de requisitos funcionais, não funcionais e regras de negócio. Esse processo traduz as diretrizes pedagógicas do método *Task-Based Language Teaching* (*TBLT*) em regras de domínio computacionais, definindo a lógica que governa a comunicação entre os módulos do sistema. A partir desse alinhamento, estabelecem-se as restrições arquiteturais necessárias para coordenar a orquestração de microsserviços e a execução dos modelos customizados.

No âmbito comportamental, os **Requisitos Funcionais (RF)** descrevem as ações diretas que o *software* deve executar para processar o fluxo interativo e a jornada do usuário. A Tabela 1 categoriza essas funcionalidades essenciais implementadas no ecossistema ChatEP.

Tabela 1. Requisitos Funcionais para o ecossistema ChatEP

Identificador	Requisito Funcional	Descrição Técnica
RF-01	Gestão de Sessão	Autenticar usuários e isolar históricos de *logs* individuais.
RF-02	Cenários *TBLT*	Expor tarefas pragmáticas parametrizadas por complexidade.
RF-03	*Streaming* de Áudio	Capturar *input* de voz do cliente e transmiti-lo via rede.
RF-04	Transcrição ASR	Processar o fluxo de áudio e convertê-lo em texto.
RF-05	Inferência SLM	Analisar semântica do texto através do modelo de linguagem.
RF-06	Módulo de Leitura	Exibir textos e instruções contextuais.
RF-07	Módulo de Escrita	Processar e avaliar a produção textual digitada pelo usuário.

Para dar suporte a essas funcionalidades, os **Requisitos Não Funcionais (RNF)** especificam os critérios de qualidade, desempenho, segurança e infraestrutura que orientam a arquitetura física e a comunicação entre os microsserviços. Conforme apresentado na Tabela 2, essas diretrizes delimitam a capacidade operacional do ambiente.

Tabela 2. Requisitos Não Funcionais do ecossistema ChatEP

Identificador	Atributo de Qualidade	Restrição de Engenharia
RNF-01	Desempenho e Latência	Minimizar o *Time to First Token* (TTFT) nas respostas generativas.
RNF-02	Restrição de Infraestrutura	Otimizar o consumo de VRAM e RAM nos *hosts* de microsserviços.
RNF-03	Escalabilidade	Isolar o microsserviço de IA para evitar bloqueio de *threads* de negócio.
RNF-04	Interoperabilidade	Viabilizar comunicação assíncrona entre o *backend* NestJS e microsserviços *Python*.

Por fim, a coerência pedagógica e computacional do sistema é sustentada pelas **Regras de Negócio (RN)**, que estabelecem as diretrizes operacionais do ecossistema. A Tabela 3 detalha as validações lógicas e de domínio que governam o ciclo de vida das tarefas e as interações do usuário.

Tabela 3. Regras de Negócio e Diretrizes de Validação

Identificador	Premissa Operacional	Regra de Execução do Sistema
RN-01	Progresso Livre	Permitir o avanço de cenário a qualquer momento por decisão do usuário.
RN-02	Feedback Imediato	Disparar análise corretiva do SLM a cada interação gramatical ou sintática.

4. Arquitetura e Infraestrutura da Plataforma ChatEP

A plataforma ChatEP foi projetada como um sistema distribuído fundamentado em uma arquitetura cliente-servidor integrada a microsserviço. O *design* do ecossistema visa converter os requisitos instrucionais do método *Task-Based Language Teaching* (*TBLT*) em fluxos operacionais desacoplados. Essa abordagem mitiga o acoplamento de código e isola as rotinas intensivas de computação matemática nos servidores dedicados à Inteligência Artificial (IA), o que atende aos critérios de escalabilidade, otimização de recursos e latência mínima (RNF-01, RNF-02 e RNF-03). Para mediar as interações de forma assíncrona (RNF-04) e gerenciar o ciclo de vida dos dados, o ecossistema divide suas responsabilidades em quatro camadas lógicas: interface do cliente (*frontend*), núcleo de aplicação e regras de negócio (*backend* NestJS), microsserviço em *Python* e um banco para persistência de dados (*MongoDB*), esta topologia e fluxos de integração estão ilustrados na Figura 1.

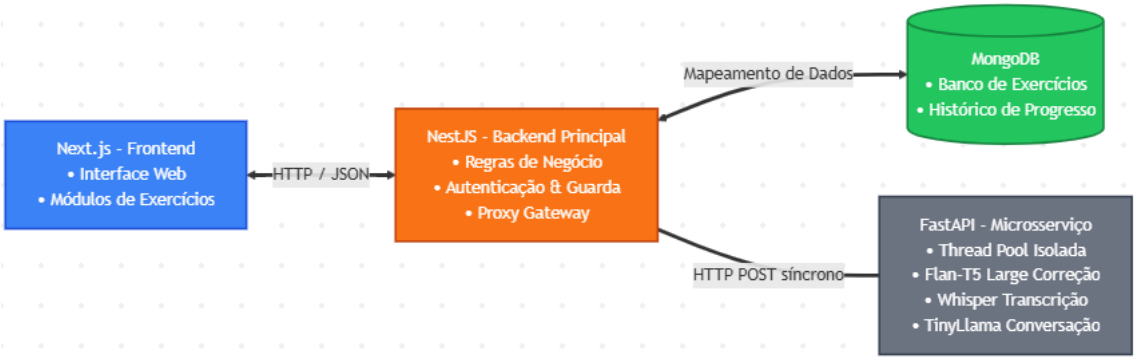

Figura 1. Topologia arquitetural distribuidora e pipeline de integração do ecossistema ChatEP.

A camada de interface e apresentação (frontend) foi desenvolvida utilizando o *framework Next.js* sob a biblioteca *React*, operando como uma aplicação híbrida com renderização otimizada e gerenciamento de rotas baseado em servidor/cliente. Essa camada desempenha a função de orquestrar os componentes de interface e gerenciar os estados de renderização reativa em tempo de execução. O cliente comunica-se exclusivamente via requisições assíncronas *HTTP/REST* com envelopes estruturados *JSON* para exibir textos contextuais de leitura (RF-06) e transmitir a produção textual digitada (RF-07), além de utilizar fluxos binários (*Multipart FormData*) para o tráfego de buffers de áudio brutos capturados (RF-03).

No lado do servidor, *backend* em *Nest.js*, adota uma arquitetura modular baseada em inversão de controle (*IoC*), atuando como um *API Gateway* para encapsular a autenticação (RF-01), sessões e regras de domínio dos cenários (RF-02), garantindo o progresso livre do estudante (RN-01). Para evitar o bloqueio de *threads* por inferências pesadas e manter o desacoplamento (RNF-03), o *backend* delega as demandas via comunicação assíncrona (RNF-04) para um microsserviço em *FastAPI* (*Python*). Esse componente de IA executa os modelos *Flan-T5 Large* para geração de análises gramaticais (RF-07 e RN-02), o *Whisper* para decodificação de áudio (RF-04) e o *TinyLlama-1.1B-Chat* para contextualização dos cenários conversacionais (RF-05 e RF-06). Este processo de desacoplamento atua diretamente na otimização de recursos de hardware, economizando memória RAM e garantindo a conformidade com os limites estritos do sistema (RNF-02).

Por fim, o armazenamento adota o paradigma não relacional (*NoSQL*) por meio do *MongoDB*, persistindo as informações na forma de documentos polimórficos estruturados sob a especificação *BSON*. A flexibilidade dos esquemas dinâmicos dispensa migrações complexas ao registrar os históricos de progresso e as interações do discente de forma independente do resultado obtido (RN-01), reduzindo o acoplamento entre as camadas de aplicação e dados.

Para viabilizar a implantação dessa topologia, o ecossistema foi hospedado em um ambiente de nuvem híbrido e descentralizado. A camada de interface e apresentação (*frontend*) está implantada na plataforma *Vercel*, aproveitando sua infraestrutura de *edge network* para otimizar a entrega das interfaces. O núcleo da aplicação e regras de negócio (*backend NestJS*) foi na *Render*, atuando de forma integrada à base de dados que utiliza o cluster gerenciado do *MongoDB Atlas*. Em paralelo, o microsserviço em *Python* responsável por isolar as *IAs* está hospedado em uma máquina virtual dedicada *VKM 2*, uma *VPS* fornecida pela *Hostinger*, garantindo o isolamento de recursos de *hardware* necessários para a execução dos modelos locais. A partir dessa topologia, o sistema distribui logicamente seus domínios de execução em quatro módulos voltados às habilidades fundamentais de aquisição de linguagem: compreensão auditiva (*listening*), escrita (*writing*), fala (*speaking*) e leitura (*reading*), detalhados a seguir.

4.1. Módulo de *Listening*

O módulo de compreensão auditiva (*listening*) funciona comparando o texto digitado pelo usuário com o gabarito oficial do exercício. O fluxo começa quando o *frontend* busca no banco de dados os metadados do exercício, que incluem o link do vídeo e o texto correto correspondente ao áudio. Assim que o usuário assiste ao vídeo e envia sua resposta

escrita, o *backend* recebe esse texto e faz uma limpeza inicial, padronizando tudo em letras minúsculas e removendo pontuações ou espaços extras para evitar que pequenos erros de digitação estraguem a avaliação.

Com o texto limpo, o sistema roda um algoritmo de comparação que analisa palavra por palavra em relação ao gabarito. Esse cálculo identifica exatamente quais palavras foram escritas corretamente, quais foram omitidas e quais foram digitadas com erro, gerando uma nota final automática de 0 a 100 (RF-07). Por fim, o servidor envia o resultado estruturado em um formato *JSON* para a interface. Isso permite que a tela destaque visualmente os acertos, erros e omissões do aluno de forma instantânea, sem precisar recarregar a página. A Figura 2 mostra a tela inicial do módulo, e a Figura 3 exibe o resultado após a correção do sistema.

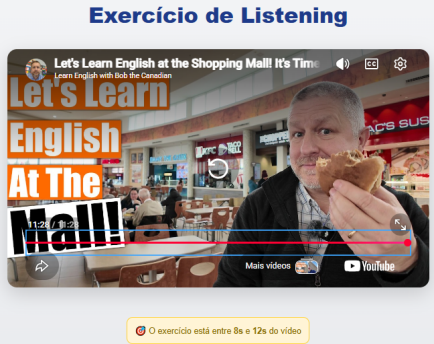

Figura 2. Módulo de *Listening* - Tela inicial.

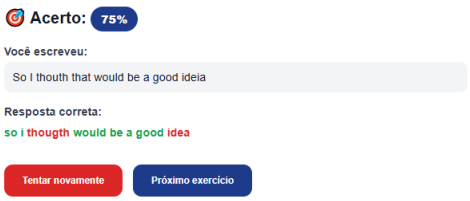

Figura 3. Módulo de *Listening* - Tela pós-correção.

4.2. Módulo de *Writing*

O módulo de escrita (*writing*) foi estruturado em duas ferramentas principais que gerenciam o fluxo de dados de formas diferentes: a validação de exercícios e o corretor gramatical.

A primeira ferramenta cuida dos exercícios baseados em contextos. Quando o usuário envia sua resposta em texto para o *backend*, o servidor *NestJS* intercepta a requisição e faz uma checagem direta no banco de dados. A validação aqui acontece de forma exata e rápida, comparando a resposta enviada com uma lista de traduções e strings corretas já armazenadas na base de dados (RF-02). O sistema processa essa comparação e retorna imediatamente uma resposta dizendo se o texto está correto ou se houve algum erro. A Figura 4 ilustra a tela de exercícios após essa validação do sistema.

A segunda ferramenta é a aba de correção textual, que lida com textos livres e exige processamento inteligente. O fluxo de dados funciona da seguinte forma: o usuário insere um texto qualquer na interface (*frontend*) e envia uma requisição *HTTP POST*. O servidor *NestJS* recebe esse *payload*, valida os dados e, usando o padrão de projeto *Proxy*, repassa o texto para o microsserviço em *Python* (*FastAPI*) (RNF-04). No microsserviço, o modelo *Flan-T5 Large* é acionado em uma *thread* isolada para processar o texto sem travar as outras funções do sistema (RNF-03). O modelo analisa a estrutura, identifica os erros gramaticais e gera a versão corrigida. O microsserviço devolve o resultado estruturado em um formato *JSON* contendo duas chaves principais: o texto original e o texto

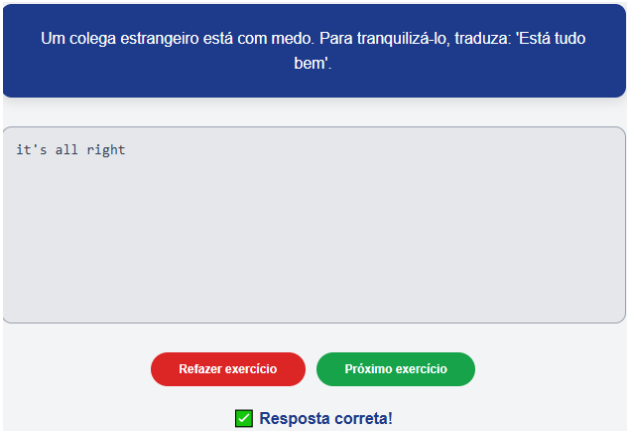

Figura 4. Aba de exercícios de *writing* após a validação.

corrigido. O *frontend* recebe esse *JSON* e renderiza as duas versões lado a lado para o usuário. A Figura 5 demonstra o resultado visual desse processo na interface.

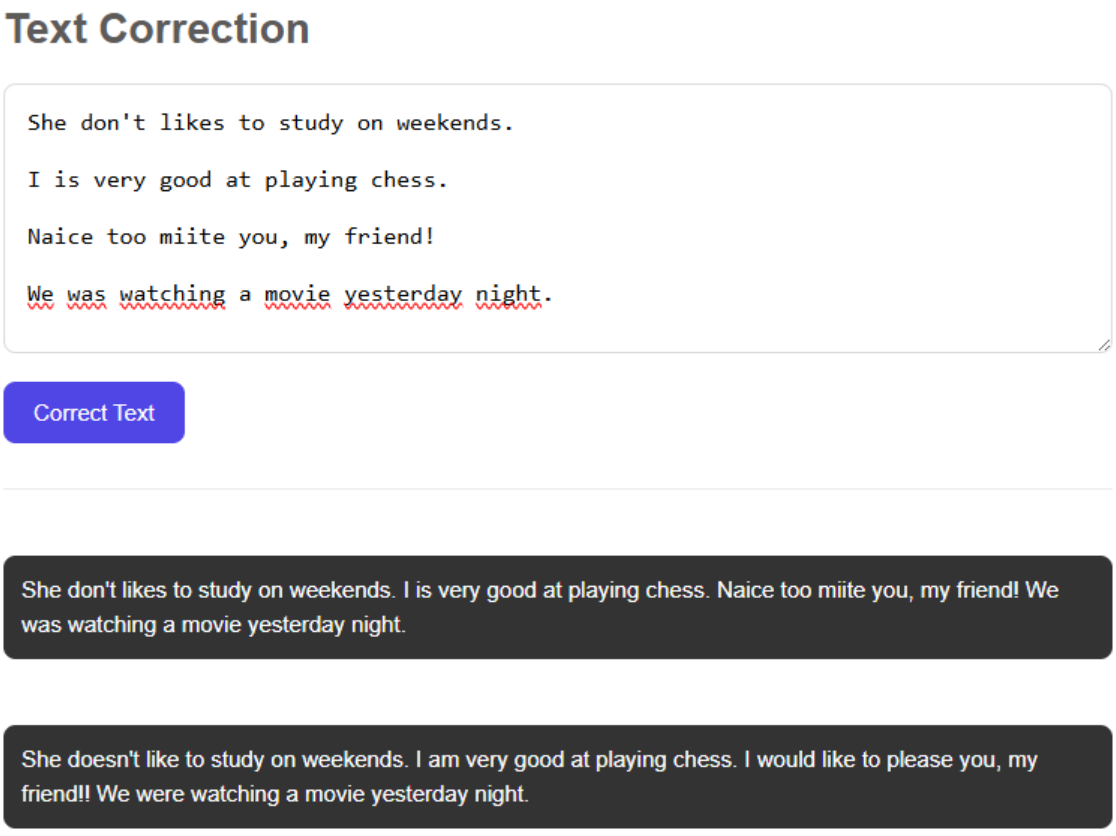

Figura 5. Interface do corretor textual exibindo o retorno do microsserviço.

4.3. Módulo de *Speaking*

O módulo de fala (*speaking*) é o componente de maior complexidade de integração do sistema, dividindo-se entre o *pipeline* de avaliação de pronúncia e o motor de diálogo do tutor virtual.

Na aba de exercícios, o fluxo técnico começa na interface do usuário, onde a *API* nativa do navegador (*MediaRecorder*) captura o áudio do microfone. Esse áudio é encapsulado e enviado assincronamente para o *backend* através de uma requisição HTTP do tipo *Multipart FormData* (RF-03). O servidor *NestJS* atua como um intermediário e repassa o arquivo binário para o microsserviço *FastAPI*. Dentro do microsserviço *Python*, o *buffer* de áudio é convertido em memória para o formato linear *PCM, pulse-code modulation*, a 16kHz, que é o padrão exigido pelo modelo *Whisper* (RF-04). O modelo processa o sinal acústico e gera uma transcrição em texto (RF-07), onde, por fim, o resultado do processamento retorna em formato *JSON* para atualizar a interface do usuário. A Figura 6 demonstra o estado da tela após o processamento completo desse pipeline de áudio.

Na aba de tutoria, o foco muda para o gerenciamento de estado e contexto da conversa. Como modelos de linguagem não possuem memória própria entre as requisições, o *backend* foi projetado para manter um *buffer* dinâmico com o histórico do diálogo. Toda vez que o usuário envia uma nova mensagem de texto, o sistema resgata as interações anteriores armazenadas no banco de dados, anexa-as ao novo texto e monta o *prompt* estruturado que é enviado ao modelo *TinyLlama-1.1B-Chat* (RF-05). Para mitigar as limitações de capacidade de *tokens* impostas pelo hardware atual e evitar o estouro de memória, o sistema adota uma janela de contexto restrita, limitando o *buffer* histórico ao armazenamento das últimas 4 interações.

O modelo gera a resposta textual, manda para o *backend* que, por sua vez, a devolve para o *frontend*. Para complementar o fluxo de dados, a interface intercepta esse texto de retorno e aciona uma biblioteca de síntese de fala (*Text-to-Speech* - TTS) local. Isso faz com que o navegador converta o texto do tutor em áudio em tempo real, permitindo que o usuário ouça a resposta gerada. A Figura 7 ilustra a interface de chat exibindo a persistência do histórico de diálogo.

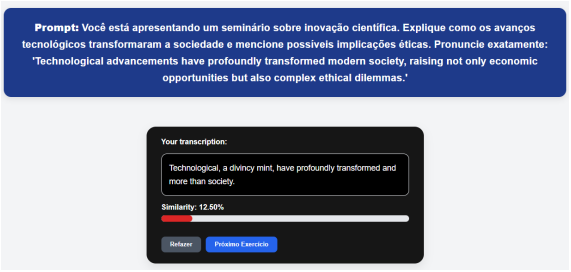

Figura 6. Aba de exercícios de *speaking* após o processamento do áudio e cálculo de métricas.

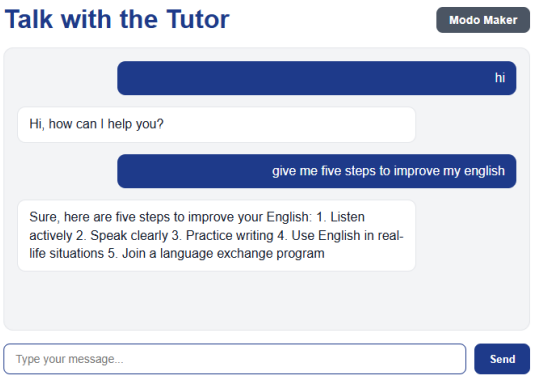

Figura 7. Interface da aba de tutoria pós conversa.

4.4. Módulo de *Reading*

O módulo de leitura (*reading*) foi projetado para gerenciar árvores de exercícios com diferentes níveis de complexidade gramatical e validar as respostas de forma atômica no servidor.

O fluxo de dados começa quando o *frontend* solicita ao banco de dados as

informações do cenário escolhido. O *backend NestJS* retorna um documento *JSON* contendo o texto base, o nível de dificuldade e um *array* com as questões de múltipla escolha, incluindo as alternativas e o índice da resposta correta (RF-02). O sistema foi modelado para suportar desde textos com vocabulário básico até estruturas mais avançadas, como *phrasal verbs* e verbos modais, sem a necessidade de alterar a estrutura da tabela ou do código, graças à flexibilidade do banco de dados não relacional.

Assim que o usuário escolhe uma alternativa e confirma o envio, a interface dispara uma requisição *HTTP* contendo o identificador do exercício e o índice da resposta marcada. O *backend* recebe esse *payload*, valida se a opção escolhida bate com o gabarito oficial e computa a atualização do progresso do aluno no banco de dados (RF-01). Em seguida, o servidor devolve um *JSON* de resposta que indica o status da validação. Com esses dados, a interface atualiza o estado da tela na hora, destacando os acertos e erros do usuário de forma visual e instantânea. A Figura 8 exemplifica o comportamento da interface do módulo de *reading* após a validação da resposta pelo sistema.

Figura 8. Tela de exercícios de *reading* pós correção

5. Avaliação

Para a avaliação experimental do ecossistema, estabeleceu-se uma taxonomia de classificação adaptada e calibrada especificamente para este projeto, tomando como base os levantamentos e estudos analíticos estruturados por [Sai et al. 2020]. Sob esse referencial teórico de suporte, fixou-se que a métrica *Bilingual Evaluation Understudy (BLEU)* delimita os intervalos de qualidade muito baixa (0–9%), baixa (10–29%), atingindo um patamar aceitável entre 30% e 49%, e evoluindo para boa qualidade (50–69%) e excelente correspondência (70–100%). Para a taxa *Word Error Rate (WER)*, os critérios adotados estipulam faixas de excelente reconhecimento (inferior a 9%), bom (10–19%), moderado (20–39%), baixa precisão (40–59%) e inadequação técnica acima de 60%. Por fim, a similaridade semântica medida pelo *BERTScore* foi mapeada de forma incremental em

baixa (0,00 a 0,59), moderada (0,60 a 0,74), boa (0,75 a 0,84), alta similaridade semântica (0,85 a 0,94) e correspondência quase perfeita (0,95 a 1,00). Tomando como base esses parâmetros unificados e parametrizados, os modelos foram submetidos a uma esteira de avaliação para aferir sua precisão operacional.

Para o módulo de correção de texto livre, gerenciado pela versão customizada do modelo *Flan-T5 Large* após o processo de ajuste fino com *LoRA* utilizando as bases *grammar_samples1.json* e *Mistake-To-Meaning*, utilizou-se a métrica *BLEU*. O processo de avaliação experimental concentrou-se em quantificar a precisão da geração de texto do componente homologado, confrontando suas predições com as referências e gabaritos para validar o nível de proximidade com a resposta padrão esperada.

Após o processo de ajuste fino (*fine-tuning*), o corretor de texto alcançou a marca de 40,62% no índice BLEU. Conforme ilustrado no gráfico da Figura 9, esse resultado enquadra o componente na faixa de qualidade ”aceitável”para tarefas de geração e reestruturação livre de sentenças, indicando um salto significativo em relação à sua versão pura.

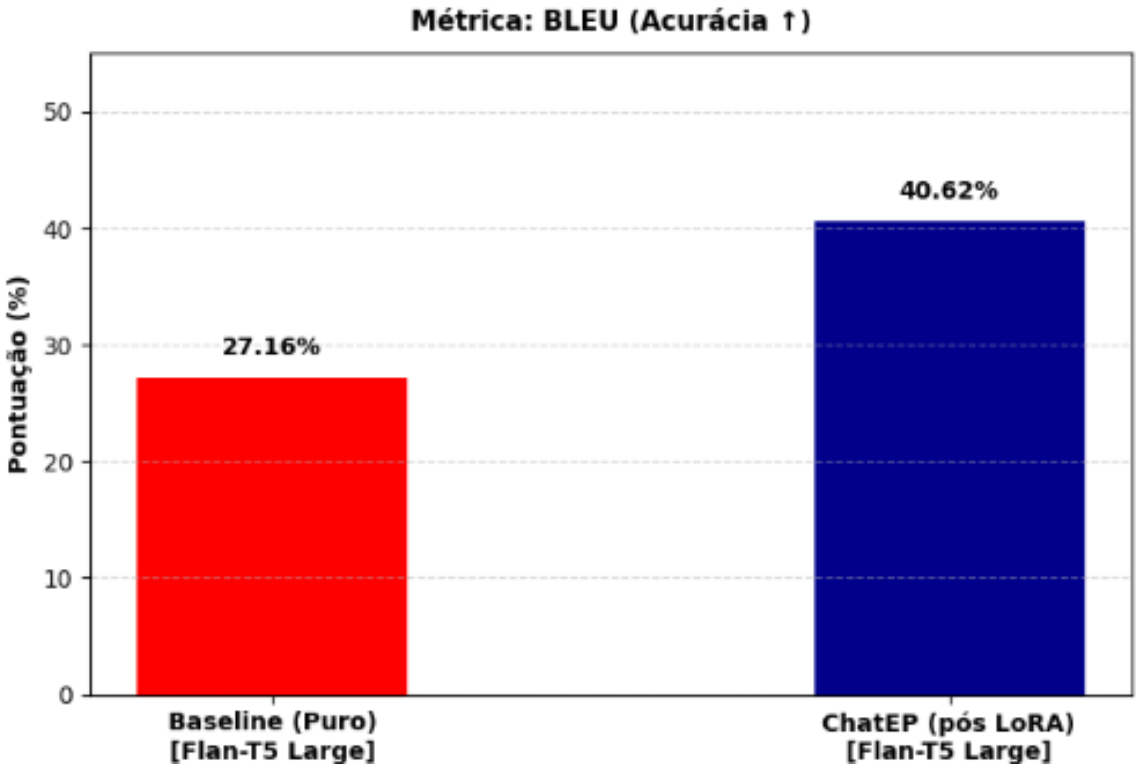

Figura 9. Avaliação experimental do modelo Flan-T5 Large através da métrica BLEU.

No caso do reconhecimento de fala com o *Whisper*, otimizado previamente por meio do treinamento com a base de dados pública *L2-Arctic* para garantir maior robustez na transcrição de fonemas, a telemetria do pipeline acústico foi avaliada por meio da métrica *WER*. O processo de avaliação experimental concentrou-se em quantificar a precisão da transcrição do áudio obtida pelo componente homologado, confrontando as predições geradas pelo microsserviço com os gabaritos textuais de referência para validar

a exatidão acústica do sistema.

Como pode ser observado no gráfico da Figura 10, o pipeline de áudio obteve uma taxa de erro *WER* de apenas 9,05%. Esse índice consolida o módulo dentro do nível de ”excelente reconhecimento”de voz, provando a robustez da engenharia de processamento de sinais implementada.

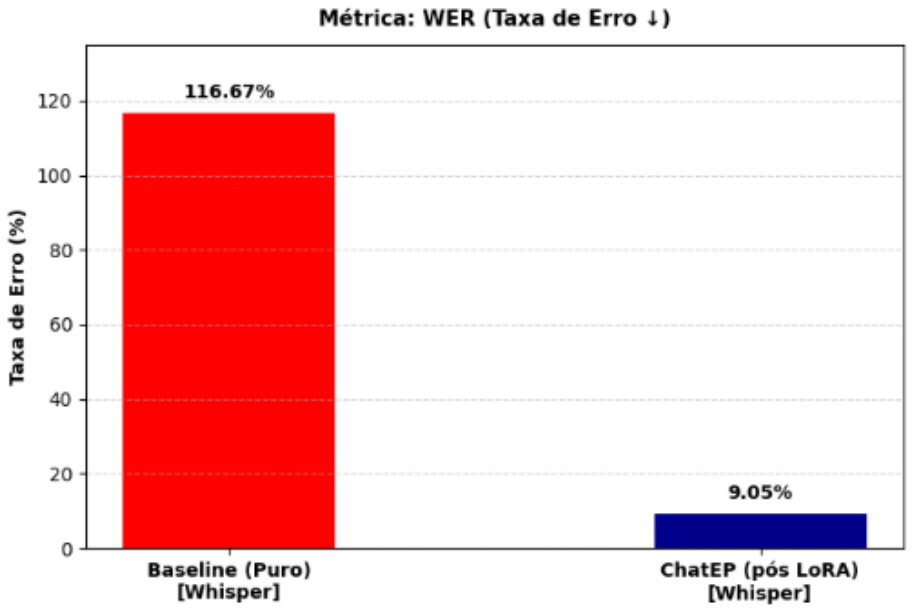

Figura 10. Avaliação experimental do modelo Whisper através da métrica WER.

Por fim, o agente conversacional baseado no modelo *TinyLlama-1.1B-Chat*, cujo alinhamento prévio foi realizado por meio do conjunto de dados *conversation.json* desenvolvido pelo autor com instâncias em turnos de diálogo contínuo, foi avaliado por meio da métrica *BERTScore*. O processo de avaliação experimental concentrou-se na validação de similaridade semântica do tutor artificial, confrontando as respostas contextuais geradas pelo componente homologado com o histórico e os gabaritos de referência para garantir a coesão necessária durante as sessões de tutoria ativa.

O tutor conversacional marcou 0,8659 no teste de *BERTScore*. O comportamento dessa evolução fina é detalhado na Figura 11, comprovando uma ”boa similaridade semântica”e garantindo que os diálogos mantenham a coesão necessária durante as sessões de tutoria ativa.

Cruzando os dados gerados na telemetria com as restrições de infraestrutura do sistema, a Tabela 4 apresenta o mapeamento de conformidade dos componentes do ecossistema, onde as setas indicam a orientação dos indicadores, ↑ para métricas em que maiores valores representam melhor desempenho e ↓ para o oposto. Os resultados experimentais funcionam como testes de aceitação de *software*, homologando que o ecossistema atende com segurança e estabilidade os limites operacionais exigidos pelas tarefas de processamento de áudio e linguagem natural.

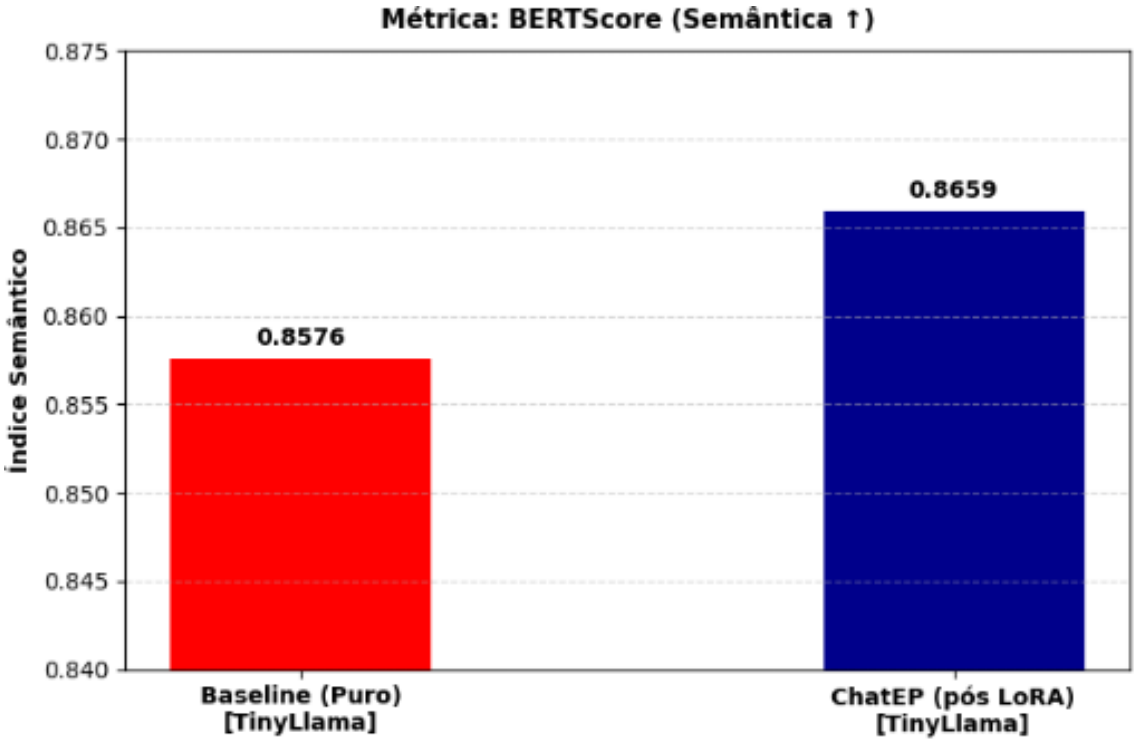

Figura 11. Avaliação experimental do modelo TinyLlama através da métrica BERTScore.

Componente / Modelo	Pipeline Core	Indicador Técnico	Desempenho	Status RNF
Flan-T5 Large	Hugging Face	BLEU ↑	40,62%	Homologado
Whisper	Hugging Face	WER ↓	9,05%	Homologado
TinyLlama-1.1B	Hugging Face	BERTScore ↑	0,8659	Homologado

Tabela 4. Matriz de homologação arquitetural e validação de limites dos modelos.

Para complementar a validação analítica, o sistema foi submetido a testes experimentais de aceitação em ambiente de homologação, simulando cenários reais de carga com um grupo de 5 estudantes de inglês com diferentes níveis de proficiência (do A2 ao C1) operando em diferentes níveis de concorrência. O objetivo foi validar a robustez da arquitetura multitarefa frente aos fluxos de *listening, speaking, writing* e *reading*. Após a utilização contínua da plataforma por um período de 2 a 3 dias, os participantes responderam a um questionário estruturado em escala do tipo Likert de 5 pontos, variando de ”Discordo totalmente”a ”Concordo totalmente”, cujas opções foram mapeadas numericamente de 1 a 5. A validação de cada item foi determinada pela média das pontuações obtidas, em que valores mais próximos do teto (5,0) indicam alta aceitação e utilidade da funcionalidade analisada.

Os relatórios de inspeção técnica apontaram alta resiliência na integridade das requisições e estabilidade na persistência dos dados no *backend*. Pelo prisma de requisitos funcionais e metodológicos, a arquitetura de *feedbacks* síncronos e o fluxo orientado a tarefas pautadas pela *TBLT* atenderam plenamente aos critérios de aceitação. Contudo, os testes evidenciaram um gargalo de infraestrutura associado às restrições de processa-

mento do servidor em nuvem, gerando latência síncrona na entrega das respostas. Como evolução técnica para o *backlog* futuro, registraram-se necessidades de otimização desse tempo de resposta, seja via implementação de filas assíncronas ou técnicas de *caching*, e a modularização de novas *features* parametrizadas por nível de perfil do usuário.

6. Conclusão

Este trabalho consolidou o ciclo completo de engenharia, desenvolvimento e homologação da plataforma ChatEP, uma aplicação *web* projetada para atuar como um ecossistema autônomo no suporte ao aprendizado da língua inglesa através das competências de *listening, speaking, writing* e *reading*. O projeto demonstrou a viabilidade prática de acoplar a metodologia pedagógica *TBLT* a uma arquitetura distribuída baseada em microsserviços. Essa abordagem de *design* de *software* provou-se altamente escalável, garantindo o isolamento de escopo no *backend* e viabilizando a orquestração harmônica de modelos heterogêneos de Inteligência Artificial especializados a partir do ecossistema *Hugging Face*.

A validação quantitativa dos componentes baseados em IA, com modelos base o *Flan-T5 Large*, *Whisper* e o *TinyLlama-1.1B-Chat* e o processo após a customização via *LoRA*, foi avaliado por meio de métricas de telemetria de dados (*BLEU, WER* e *BERTScore*). Os dados experimentais coletados nas esteiras de teste homologaram o desempenho do sistema, atestando que os *pipelines* computacionais atingiram os critérios de aceitação necessários para operar estavelmente dentro das regras de negócio propostas pela aplicação.

A condução do estudo piloto atuou como uma etapa crucial de garantia de qualidade (*Quality Assurance*), permitindo expor o comportamento do *software* sob condições reais de uso. Os relatórios de inspeção técnica gerados a partir do *feedback* dos usuários permitiram mapear empiricamente os *trade-offs* de infraestrutura, convertendo dados subjetivos de usabilidade em métricas objetivas de latência e limitações de processamento em ambiente de nuvem.

Como direcionamentos para o *backlog* técnico e mitigação de débitos arquiteturais, propõe-se o desenvolvimento de estratégias de otimização arquitetural focadas na redução da latência (*delay*) observada nas respostas do sistema. No escopo funcional e de engenharia de dados, o planejamento futuro prevê a estruturação de exercícios personalizados por nível de proficiência do usuário. Por fim, projeta-se a implementação de mecanismos dinâmicos para a coleta e classificação contínua desses níveis de usuário, refinando a adaptabilidade e a modularidade da plataforma.

Referências

Ahmad, Y., Hussain, S., Mumtaz, U., and Riaz, S. (2025). Investigating the influence of duolingo, elsa speak, and hello english mobile applications on students' attitudes and speaking skills. *Review Journal of Social Psychology & Social Works*, 3(1).

Bahari, A., Han, F., and Strzelecki, A. (2025). Integrating call and aiall for an interactive pedagogical model of language learning. *Educational Technology Society*, 30:14305–14333.

Bezerra, R. L. and Medeiros, I. (2026). Desenvolvimento de ecossistema web para prática ativa de inglês: Refinamento linguístico e autodidatismo assistidos por inteligência

artificial. In *Anais do II Workshop de Inovação, Desenvolvimento, Educação e Inclusão com Ações Maker (IDEIA - CSBC 2026)*. SBC. No prelo.

Bogner, J., Fritzsch, J., Wagner, S., and Zimmermann, A. (2021). Industry practices and challenges for the evolvability assurance of microservices. *Empirical Software Engineering*, 26(5):104.

British Council (2014). Learning english in brazil: Understanding the aims and expectations of the brazilian emerging middle classes. *British Council Research Reports*. Accessed: 2026-02-09.

Choi, G., Shin, H., Shin, C., Oh, E. J., and Kwak, N. (2026). Software–hardware co-design for real-time speech translation on resource-constrained npus. *IEEE Access*, 14:48901–48917.

Coşkun, Z. and Solmaz, O. (2024). Exploring high school students' acceptance and use of mobile applications for english language learning. *Dil Eğitimi ve Araştırmaları Dergisi*, 10(2):636–661.

Dhilleswararao, P., Boppu, S., Manikandan, M. S., and Cenkeramaddi, L. R. (2022). Efficient hardware architectures for accelerating deep neural networks: Survey. *IEEE Access*, 10:131788–131828.

Islam, M. R., Dhar, N., Deng, B., Nguyen, T. N., He, S., and Suo, K. (2024). Characterizing and understanding the performance of small language models on edge devices. In *2024 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, pages 1–10.

Jurafsky, D. and Martin, J. H. (2026). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. Stanford University, 3 edition. Draft of January 6, 2026.

Kalimullah, M. and Tabassum, F. (2025). Technology-supported english language learning: Effectiveness and limitations. *International Journal of Social Impact*, 10(1):281–286.

Kalra, M. and Sharma, V. (2025). Impact of low rank adaptation with parameter efficient fine-tuning techniques on transformer models. In *2025 6th International Conference for Emerging Technology (INCET)*, pages 1–6.

Li, S. (2021). Task-based language teaching based on computer-assisted language learning. In *2021 2nd International Conference on Education, Knowledge and Information Management (ICEKIM)*, pages 863–866.

Mohd Sapuan, W., Hassan, H., Rostam, N. L. S., and Azmimurad, A. (2024). The impacts of language learning apps on students' perceptions of language proficiency. *International Journal of Research and Innovation in Social Science*, VIII:3447–3455.

Olajide-Owoyomi, F. (2025). English proficiency barriers in brazil: Communication challenges among international students and professionals. *Journal of International Students*, 15(12):177–192.

Osipov, I. V., Prasikova, A. Y., and Volinsky, A. A. (2015). Monetization as a motivator for the freemium educational platform growth. *arXiv preprint arXiv:1501.04157*.

Saadany, H. and Orăsan, C. (2022). Bleu, meteor, bertscore: Evaluation of metrics performance in assessing critical translation errors in sentiment-oriented text. In *Proceedings of the Conference*, United Kingdom. University of Wolverhampton and University of Surrey.

Sai, A. B., Mohankumar, A. K., and Khapra, M. M. (2020). A survey of evaluation metrics used for nlg systems. *arXiv preprint arXiv:2008.12009*.

Verdecchia, R., Kruchten, P., Lago, P., and Malavolta, I. (2021). Building and evaluating a theory of architectural technical debt in software-intensive systems. *Journal of Systems and Software*, 176:110925.

Wang, X., Hamat, A. B., and Shi, N. (2024). Designing a pedagogical framework for mobile-assisted language learning. *Heliyon*, 10(7):e28102.