PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA E DE ARTES
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PUC
GOIÁS

DESENVOLVIMENTO DE IA PARA CORREÇÃO DE AVALIAÇÕES DE QUESTÕES SUBJETIVAS

DIOGO LOPES GOMES

GOIÂNIA
2026

DIOGO LOPES GOMES

DESENVOLVIMENTO DE IA PARA CORREÇÃO DE AVALIAÇÕES DE QUESTÕES SUBJETIVAS

Trabalho de Conclusão de Curso apresentado à Escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

Orientador(a):
Profª. Ma. Angélica da Silva Nunes

Banca examinadora:
Profº Fernando Gonçalves Abadia
Profº Rafael Leal Martins

GOIÂNIA
2026

DIOGO LOPES GOMES

DESENVOLVIMENTO DE IA PARA CORREÇÃO DE AVALIAÇÕES DE QUESTÕES SUBJETIVAS

Trabalho de Conclusão de Curso aprovado em sua forma final pela Escola Politécnica e de Artes, da Pontifícia Universidade Católica de Goiás, para obtenção do título de Bacharel em Ciência da Computação, em ____/____/_____.

Orientador(a): Profª. Ma. Angélica da Silva Nunes

Prof. Me. Fernando Gonçalves Abadia

Prof. Me. Rafael Leal Martins

GOIÂNIA

2026

DEDICATÓRIA

Dedico este trabalho primeiramente a Deus, por me guiar neste mundo e me conceder as forças necessárias para seguir em frente, independentemente dos obstáculos. Aos meus pais, Leonardo e Erika, pelo apoio incondicional e por tornarem este caminho possível. Aos meus avós por sempre apoiarem a mim e aos meus pais durante toda a formação. À minha namorada, Yasmin, por caminhar ao meu lado e me sustentar até nos momentos mais difíceis. À minha orientadora, Angélica, por sua paciência e por me manter firme em direção ao meu objetivo. Aos docentes, que partilharam comigo o seu conhecimento. E, em memória de minha avó, Carmen: um de seus maiores desejos era me ver formado, e hoje tenho a honra de cumprir essa promessa em sua memória.

AGRADECIMENTOS

Agradeço a Deus, pela força, sabedoria e discernimento concedidos durante toda a minha jornada acadêmica, permitindo-me superar os obstáculos com resiliência.

Aos meus pais, Leonardo e Erika, pelo suporte fundamental, pela dedicação e por viabilizarem a minha caminhada até a conclusão do ensino superior.

À minha companheira, Yasmin, pelo apoio contínuo, compreensão e incentivo constante nos momentos de maior dedicação a este trabalho.

À minha orientadora, Professora Angélica, pela excelente orientação, paciência e valiosas contribuições técnicas que enriqueceram o desenvolvimento deste pipeline de pesquisa.

Aos docentes da Escola Politécnica e de Artes, pelo conhecimento compartilhado e pelo rigor acadêmico que moldaram minha visão profissional e científica.

Aos colegas e amigos que compartilharam as angústias e as vitórias da vida universitária, tornando este percurso mais leve e enriquecedor.

"O coração pode ser fraco e às vezes pode ceder. Mas no fundo eu sei que há uma luz que nunca se apaga."

— Sora (Square Enix, Kingdom Hearts)

RESUMO

O presente trabalho desenvolve e avalia um sistema de Inteligência Artificial para correção automatizada de respostas discursivas de questões conceituais, utilizando técnicas de Processamento de Linguagem Natural e Aprendizado Profundo. O sistema emprega o modelo BERTimbau, variante do BERT pré-treinada em português brasileiro, combinado com uma camada classificadora *Multi-Layer Perceptron*, ajustada por *fine-tuning* para classificar respostas em três categorias: correta, parcialmente correta e incorreta. Para contornar a escassez de dados rotulados, adota-se uma arquitetura de Geração Aumentada por Recuperação local, baseada em vetorização TF-IDF e similaridade de cossenos, que seleciona trechos relevantes do livro-texto para orientar a geração de dados sintéticos via API do modelo Gemini. A *interface* do sistema foi desenvolvida com a biblioteca Streamlit, permitindo que o professor configure a questão, o gabarito e o material de referência, inicie o treinamento sob demanda e obtenha o boletim da turma em uma única sessão. Um ciclo de *Active Learning* permite que divergências corrigidas pelo docente sejam reinjetadas no conjunto de treinamento, promovendo a evolução contínua do modelo. Os experimentos demonstraram que o *pipeline* é tecnicamente viável, com acurácia próxima de 100% sobre dados sintéticos, distribuição de notas semanticamente coerente com o gabarito e tempo total de execução entre dois e cinco minutos em *hardware* de consumo. Conclui-se que a solução representa uma prova de conceito relevante para a automação do processo avaliativo, com potencial de reduzir o tempo dedicado pelo professor à correção de rotina e de ser adaptada a diferentes disciplinas sem necessidade de base de dados prévia.

Palavras-chave: Inteligência Artificial. Correção Automática. Processamento de Linguagem Natural. BERTimbau. *Active Learning*.

ABSTRACT

This study develops and evaluates an Artificial Intelligence system for the automated grading of short-answer conceptual questions, utilizing Natural Language Processing and Deep Learning techniques. The system employs the BERTimbau model—a BERT variant pre-trained on Brazilian Portuguese—combined with a Multi-Layer Perceptron classifier layer, fine-tuned to categorize answers into three classes: correct, partially correct, and incorrect. To overcome the scarcity of labeled data, a local Retrieval-Augmented Generation (RAG) architecture is adopted, based on TF-IDF vectorization and cosine similarity, which selects relevant excerpts from the textbook to guide the generation of synthetic data via the Gemini model API. The system's interface was developed using the Streamlit library, allowing instructors to configure the question, the answer key, and the reference material, initiate on-demand training, and obtain the class grade report within a single session. An Active Learning cycle enables instructor-corrected discrepancies to be reinjected into the training set, promoting the continuous evolution of the model. Experiments demonstrated that the pipeline is technically feasible, achieving an accuracy close to 100% on synthetic data, a grade distribution semantically coherent with the answer key, and a total execution time between two and five minutes on consumer *hardware*. It is concluded that the solution represents a relevant proof of concept for the automation of the assessment process, with the potential to reduce the time spent by teachers on routine grading and to be adapted to different disciplines without requiring a prior database.

Keywords: Artificial Intelligence. Automated Grading. Natural Language Processing. BERTimbau. Active Learning.

LISTA DE ABREVIATURAS

API	*Application Programming Interface*, *Interface* de Programação de Aplicativos
ASAG	*Automated Short Answer Grading*, Avaliação Automática de Respostas Curtas
BERT	*Bidirectional Encoder Representations from Transformers*, Representações de Codificador Bidirecional de Transformadores
BPTT	*Backpropagation Through Time*, Retropropagação Através do Tempo
CSV	*Comma-Separated Values*, Valores Separados por Vírgulas
DL	*Deep Learning*, Aprendizado Profundo
GPU	*Graphics Processing Unit*, Unidade de Processamento Gráfico
GRU	*Gated Recurrent Units*, Unidade Recorrente com Portão
IA	Inteligência Artificial
IDE	*Integrated Development Environment*, Ambiente de Desenvolvimento Integrado
JSON	*JavaScript Object Notation*, Notação de Objetos JavaScript
LLMs	*Large Language Models*, Grandes Modelos de Linguagem
LSTM	*Long Short-Term Memory*, Memória de Longo e Curto Prazo
ML	*Machine Learning*, Aprendizado de Máquina
MLM	*Masked Language Model*, Modelagem de Linguagem Mascarada
MLP	*Multi-Layer Perceptron*, Perceptron Multicamadas
NLTK	*Natural Language Toolkit*, Kit de Ferramentas de Linguagem Natural.
NSP	*Next Sentence Prediction*, Previsão da Próxima Sentença

PDF	*Portable Document Format*, Formato de Documento Portátil
PLN	Processamento de Linguagem Natural
RAG	*Retrieval-Augmented Generation*, Recuperação Aumentada por Geração
RNA	Redes Neurais Artificiais
RNNs	*Recurrent Neural Network*, Redes Neurais Recorrentes
TCC	Trabalho de Conclusão de Curso
TF-IDF	*Term Frequency – Inverse Document Frequency*, requência do Termo e Frequência Inversa nos Documentos
VSCode	*Visual Studio Code*

SUMÁRIO

1 INTRODUÇÃO

O avanço tecnológico tem exercido um impacto profundo em diversos setores da sociedade, e o campo pedagógico não é exceção, dada a rápida evolução da tecnologia que tem transformado a forma em como se aprende. Lima, Ferreira e Carvalho (2024), juntamente com Aldosari (2020), destacam que os discursos acerca de áreas da tecnologia na área de ensino, frequentemente otimistas, tendem a posicionar artefatos digitais como soluções promissoras para problemas historicamente enraizados e complexos. Essa crença no potencial da automação para livrar o trabalho humano de tarefas tediosas ou pesadas tem se renovado no campo da educação. Os autores afirmam que a educação é um campo fértil para o consumo de artefatos baseados em Inteligência Artificial (IA), vistos como a forma mais promissora de aumentar a eficiência (Costa Júnior et al., 2023; Freitas et al., 2024), como por exemplo, lidando com tarefas mais repetitivas e administrativas.

A IA, um ramo da Ciência da Computação, visa criar sistemas capazes de executar tarefas que exigem inteligência humana, como o raciocínio, o aprendizado e a resolução de problemas (Figueiredo et al., 2023; Ma et al., 2014; Costa Júnior et al., 2023; Silva, 2023).

A necessidade de ferramentas automatizadas intensificou-se, notavelmente após a pandemia de Covid-19, período em que a IA se consolidou como uma promissora aliada no processo educativo (Figueiredo et al., 2023; Seren; Ozcan, 2021). Neste contexto, o Processamento de Linguagem Natural (PLN), uma subárea da IA, tornou-se fundamental para o crescimento da área (Vicari, 2021). O PLN é essencial para o desenvolvimento de ferramentas como *chatbots* educacionais (a exemplo do ChatGPT), que têm ganhado força e espaço no campo da educação (Sant'Ana et al., 2023; Dores et al., 2021), sendo o ChatGPT um modelo que despertou grande discussão na sociedade e entre educadores. Esses sistemas, baseados em PLN, possuem uma aplicação na análise e interpretação de textos, ao processar e interpretar grandes volumes de dados, viabilizando sistemas que atuam no desenvolvimento de conteúdos, métodos de ensino e, de forma relevante para este estudo, na otimização da avaliação dos alunos (Chassignol et al., 2018; Freitas et al., 2024; Figueiredo et al., 2023).

Este Trabalho de Conclusão de Curso (TCC) se concentra na aplicação da IA no trabalho docente, explorando seu potencial para auxiliar na correção e análise de avaliações de questões curtas (questões conceituais). As questões curtas são um formato de avaliação que, tradicionalmente, exige uma análise humana detalhada e

subjetiva. Assim, o foco é investigar como os modelos de *Machine Learning*, Aprendizado de Máquina (ML) e *Deep Learning*, Aprendizado Profundo (DL) podem ser desenvolvidos para lidar com a complexidade semântica das respostas discursivas, proporcionando uma ferramenta de suporte ao professor.

A relevância deste estudo reside na necessidade de se otimizar o tempo gasto pelos docentes em tarefas administrativas rotineiras. O tempo gasto na correção de trabalhos é um desafio para os docentes. A IA, ao atuar na automação do processo avaliativo, pode aliviar esse peso de tarefas de correção de rotina, permitindo que os docentes dediquem mais tempo à instrução individual e ao suporte personalizado aos alunos. A automação dessas etapas pode otimizar o sistema de análise de desempenho (Figueiredo et al., 2023; Freitas et al., 2024) e resultar em um ensino mais eficiente (Costa Júnior et al., 2023).

Globalmente, a IA já é usada na educação, com iniciativas em diferentes países e níveis de ensino (Figueiredo et al., 2023; Cândido, 2017). Existem diversos casos de uso da IA em tarefas semelhantes ou correlatas à avaliação (Figueiredo et al., 2023). Plataformas de aprendizagem adaptativa, como a Knewton nos Estados Unidos e Geekie no Brasil, utilizam algoritmos para adaptar o conteúdo e a metodologia ao perfil do estudante (Arxer et al., 2017; Giraffa; Kohls-Santos, 2023; Figueiredo et al., 2023).

Em um contexto mais próximo da correção de texto, ferramentas de IA já são utilizadas em outros países para classificar testes de múltipla escolha e marcar erros padrões de ortografia e gramática em redações (Lee, 2019). O modelo ChatGPT, baseado em PLN, também pode auxiliar docentes na criação de respostas modelo para comparação com as respostas dos alunos, identificando lacunas no conhecimento e oferecendo suporte adicional e personalizado.

A IA permite o desenvolvimento de sistemas de avaliação mais sofisticados e abrangentes que, ao invés de focar apenas em respostas corretas, podem considerar processos de pensamento e habilidades socioemocionais. O potencial da IA para aprimorar o processo de ensino-aprendizagem é existente, mas sua aplicação na avaliação acadêmica ainda levanta questões que demandam investigação.

1.1 Justificativa

O presente trabalho justifica-se pela necessidade premente de otimizar o tempo e a eficiência do trabalho docente face aos desafios administrativos rotineiros, como a correção de avaliações. A correção de questões curtas (conceituais) e subjetivas, em particular, exige uma análise humana detalhada e está sujeita à subjetividade, tornando-se uma tarefa demorada.

A IA surge como uma aliada promissora para lidar com tarefas repetitivas e administrativas na educação. Contudo, a aplicação da IA na avaliação de trabalhos dos alunos ainda é uma área em desenvolvimento no contexto nacional. Conforme a pesquisa publicada na Agência Brasil, embora 56% dos professores no Brasil afirmem usar ferramentas de IA para fins educacionais, uma taxa superior à média dos países da OCDE, apenas 36% dos docentes utilizam a IA para avaliar ou corrigir o trabalho dos alunos. Este baixo índice (36%) aponta para uma lacuna na adoção tecnológica precisamente no domínio em que este TCC se foca: a automação da correção de avaliações.

Portanto, o desenvolvimento e a avaliação de um modelo de IA capaz de corrigir respostas curtas é fundamental para:

- Otimização do tempo do professor: a automação da correção pode aliviar o peso das tarefas de rotina, liberando os educadores para se concentrarem em atividades de maior valor agregado, como o suporte individualizado e o desenvolvimento de estratégias pedagógicas inovadoras;
- Padronização e objetividade das correções: a IA pode minimizar a subjetividade inerente aos métodos tradicionais, oferecendo avaliações mais objetivas e padronizadas, o que melhora a precisão dos resultados e a equidade no processo avaliativo;
- Possibilidade de *feedback* mais rápido aos alunos: a capacidade de fornecer *feedback* imediato é uma das grandes vantagens da IA, essencial para o aprendizado contínuo e eficaz. O *feedback* automatizado e personalizado permite que os alunos ajustem seu processo de aprendizagem de forma contínua.

1.2 Objetivos

1.2.1 Geral

Desenvolver e avaliar um modelo de Inteligência Artificial capaz de corrigir e analisar respostas curtas de avaliações conceituais, utilizando técnicas de PLN e DL.

1.2.2 Específicos

1. Pesquisar algoritmos de ML e DL para a análise semântica e classificação de texto, priorizando modelos de PLN como *Recurrent Neural Network*, Redes Neurais Recorrentes (RNNs) e *Transformers* (a exemplo do BERT);
2. Gerar e tratar um conjunto de dados sintéticos de respostas curtas por meio de IA Generativa (Gemini), orientada por gabarito oficial e trechos relevantes do livro-texto recuperados via arquitetura RAG local baseada em vetorização TF-IDF e similaridade de cossenos, viabilizando o treinamento supervisionado do modelo sem necessidade de base de dados prévia rotulada manualmente;
3. Treinar, testar e otimizar o modelo de IA selecionado, implementando-o em linguagem Python com o uso das bibliotecas PyTorch, Transformers (BERTimbau), Scikit-learn, spaCy e Streamlit;
4. Realizar testes utilizando de dados sintéticos gerados por IA para verificar a viabilidade da aplicação.

1.3 Metodologia

1.3.1 Natureza da pesquisa

Esta pesquisa se enquadra em uma abordagem mista, combinando elementos qualitativos e quantitativos. Em termos de objetivos, classifica-se como descritivo-experimental: descritiva na revisão do estado da arte sobre IA, PLN e suas aplicações na avaliação educacional; e experimental no desenvolvimento, implementação e avaliação iterativa de um protótipo funcional de correção automatizada. A dimensão experimental envolveu ciclos sucessivos de teste, identificação de falhas e refinamento arquitetural, resultando em métricas objetivas de

desempenho, como acurácia sobre o conjunto de dados sintéticos e tempo de execução do pipeline.

1.4 Procedimentos técnicos

A pesquisa combina dois procedimentos principais:

1. Pesquisa bibliográfica: para a fundamentação teórica, foi realizada uma revisão bibliográfica abrangente em artigos e obras que tratam da IA, ML, PLN e suas aplicações na educação e em sistemas de avaliação automatizada;
2. Pesquisa experimental: para o desenvolvimento e teste da solução proposta, foi adotada uma metodologia experimental que envolveu a implementação prática de um modelo de IA e a avaliação de seu desempenho predominantemente sobre um conjunto de dados sintéticos gerados via Gemini API com contexto RAG. A validação com respostas reais de alunos em larga escala, comparando estatisticamente as notas do sistema com as de docentes, permanece como principal direção de trabalho futuro

1.5 Etapas da pesquisa:

O fluxo de trabalho para o desenvolvimento deste projeto foi dividido nas seguintes etapas, alinhadas à lista de verificação de um projeto de ML:

1. Revisão bibliográfica (estado da arte): fundamentação teórica sobre IA, ML, DL e PLN, com foco nos algoritmos de classificação de texto relevantes;
2. Geração e preparação de dados sintéticos: geração automatizada de respostas simuladas de alunos por meio da API Gemini, orientada por gabarito oficial e trechos do livro-texto selecionados pelo módulo RAG local (TF-IDF + similaridade de cossenos); os dados resultantes foram balanceados em três classes (Correta, Parcial, Incorreta) e persistidos em CSV para alimentar o treinamento supervisionado. O Microsoft Forms foi utilizado exclusivamente na fase de inferência em lote, fornecendo as respostas reais dos alunos após o modelo já estar treinado;
3. Implementação do modelo: codificação da arquitetura de DL/RNN/Transformer em Python, utilizando as bibliotecas de PLN e DL selecionadas;
4. Treinamento e testes: treinamento do modelo com o conjunto de dados rotulado e realização de testes de validação para ajuste fino de hiperparâmetros e otimização;

5. Análise de resultados: avaliação comparativa da precisão do modelo em relação às correções humanas e análise da viabilidade da solução na otimização do processo avaliativo.

1.6 Resultados Esperados:

Ao final deste trabalho, espera-se alcançar:

- Um protótipo funcional de um sistema de IA capaz de classificar a correção conceitual de respostas curtas de avaliações, demonstrando a aplicação prática de modelos de DL para o PLN;
- Apresentação de insights e recomendações sobre como a IA pode ser integrada de maneira ética e eficiente na avaliação acadêmica, contribuindo para a melhoria da qualidade e da eficácia do ensino e apoiando a otimização do tempo do docente;

2 VISÃO GERAL DE IA

Este capítulo procura estabelecer a fundamentação teórica sobre IA apresentando seu percurso histórico e os conceitos tecnológicos essenciais que a definem, com foco nas subáreas para o PLN.

2.1 Introdução

A IA constitui um campo fundamental da Ciência da Computação dedicado ao desenvolvimento de sistemas capazes de realizar tarefas que, tipicamente, demandam capacidades intelectuais humanas, como raciocínio, aprendizado e resolução de problemas (Gomes, 2010; Silva, 2023). O propósito primordial da IA reside no estudo de como construir máquinas que possam executar atividades que são realizadas de forma superior pelos seres humanos (Rich; Knight, 1994 apud Vicari, 2018).

Conforme observado por Kai-Fu Lee (2019), o campo de pesquisa da IA, que historicamente existiu em laboratórios acadêmicos e filmes de ficção científica, experimentou avanços teóricos que resultaram em aplicações práticas no cotidiano. A IA alimenta muitos aplicativos e *websites*, e já está pronta para dirigir carros, gerenciar portfólios e, com grande potencial disruptivo, alterar profundamente o mercado de trabalho (Lee, 2019).

2.2 Histórico da IA

A Inteligência Artificial é um ramo da Ciência da Computação cujo objetivo é criar sistemas capazes de executar tarefas que exigem inteligência humana, como o aprendizado, o raciocínio e a resolução de problemas (Lima; Ferreira; Carvalho, 2024; Silva, 2023).

2.2.1 Evolução e Marcos Importantes

O campo da IA surgiu na década de 1950, período em que os pesquisadores começaram a explorar a ideia de construir máquinas que pudessem simular o intelecto humano (Barbosa; Bezzera, 2020). O termo IA foi formalmente criado em 1956, durante uma conferência realizada na Universidade de Dartmouth (Vicari, 2018).

Apesar do entusiasmo inicial, a trajetória da IA não foi constante. O campo enfrentou períodos de estagnação, conhecidos historicamente como "invernos da IA" (Feltrin, 2021), que

ocorreram por volta de 1980 a 1993, quando as aplicações existentes não conseguiam oferecer respostas adequadas, especialmente no diagnóstico médico e na compreensão da linguagem (Barbosa; Bezzera, 2020).

O ressurgimento e a aceleração da IA foi marcado por progressos nas décadas de 2000 e 2010, especialmente em áreas como o reconhecimento de imagens e a tradução automática (Barbosa; Bezzera, 2020). Um evento importante para a IA ocorreu em 2016 com a vitória do sistema AlphaGo, baseado em DL, sobre o campeão mundial do jogo Go, Lee Sedol (Lee, 2019). Esse evento impactou o cenário global devido, Lee (2019) utiliza analogias dinâmicas para descrever a natureza transformadora da IA motivada pelo *Big Data*. Para ele, o aumento de produtividade que a IA traz vai causar um impacto grande no mercado de trabalho.

2.3 Conceitos fundamentais

2.3.1 Inteligência Artificial

A IA, em sua conceituação mais ampla, pode ser definida como a capacidade de sistemas computacionais simularem o comportamento mental humano, como a habilidade de aprender, julgar e reagir a cenários não programados (Souza, 2008; Feltrin, 2021). A IA é um campo da Ciência da Computação voltado para a solução de problemas normalmente associados à cognição humana (Lima; Ferreira; Carvalho, 2024).

O conceito de IA foca na autonomia da máquina para gerar soluções, um processo que envolve a automação da vontade.

2.3.2 Aprendizado de Máquina (Machine Learning)

O ML é uma subárea da IA, é o estudo que capacita os computadores a aprenderem a partir de dados, sem serem explicitamente programados para cada passo (Géron, 2019; Zhang et al., 2021). Essa subárea permite que os sistemas de informação aprendam automaticamente padrões e relações a partir dos dados, ganhando conhecimento (Kühl et al., 2019; Brown; Miller, 2019).

O principal objetivo do ML é construir um modelo estatístico que aprenda a partir de uma amostra de dados de treinamento para fazer previsões ou tomar decisões (Géron, 2019; Feltrin, 2021). Dentre os tipos, o Aprendizado Supervisionado é o mais comum,

onde o desenvolvedor fornece as entradas e as saídas corretas (rótulos) para que a máquina possa identificar os padrões (Géron, 2019; Feltrin, 2021).

2.3.3 Redes Neurais Artificiais

As Redes Neurais Artificiais (RNA) são a tecnologia fundamental que possibilita o ML (Feltrin, 2021). Elas constituem modelos computacionais inspirados na estrutura e função do cérebro humano (Géron, 2019; Feltrin, 2021), e são compostas por unidades interconectadas chamadas "neurônios artificiais" ou *perceptrons* (Géron, 2019; Rumelhart; Hinton; Williams, 1986; Feltrin, 2021).

A arquitetura de uma rede neural envolve a comunicação entre camadas, onde os dados de entrada são ajustados por pesos e submetidos a funções de ativação (Géron, 2019; Feltrin, 2021). Esses pesos e funções matemáticas permitem ao sistema extrair padrões complexos, o que é crucial para tarefas como a classificação de texto (Géron, 2019).

2.3.4 Aprendizado Profundo (Deep Learning)

O DL é uma técnica avançada e um subconjunto do ML (Feltrin, 2021). O conceito de "profundo" refere-se ao uso de múltiplas camadas ocultas de processamento em uma rede neural (Géron, 2019; Feltrin, 2021), que permite aos modelos aprenderem representações de dados em vários níveis de abstração (Zhang et al., 2021). O ressurgimento do interesse em DL foi influenciado por artigos inovadores no início dos anos 2000, apresenando como treinar redes neurais profundas (Géron, 2019; Hinton et al., 2006).

O DL é um elemento central da inovação moderna (Feltrin, 2021). Essa capacidade de processamento multinível é importante para o PLN (Zhang et al., 2021), subárea da IA dedicada à compreensão da linguagem humana. Os modelos de DL, como os baseados na arquitetura *Transformer* (um tipo de DL), são treinados em grandes quantidades de dados (Silva, 2023) para prever palavras e gerar sequências de texto coerentes.

O ChatGPT, por exemplo, é um modelo de Aprendizado Profundo baseado em redes neurais. Embora a geração de conteúdo por meio do DL possua o potencial de aprimorar a eficiência, ela também apresenta desafios: o modelo, ocasionalmente, pode produzir respostas que não são coerentes ou precisas, o que é problemático em ambientes educacionais onde a veracidade é essencial (Tronco, 2023). Além disso, a falta de referências explícitas do ChatGPT,

que se baseia em vasta produção intelectual de terceiros, representa um risco de plágio e exige a verificação das informações em fontes confiáveis (Tronco, 2023; Lo, 2023).

3 ALGORITMOS DE MACHINE LEARNING

Este capítulo foca nos algoritmos relevantes para a proposta, dedicando-se à exploração dos modelos de ML mais pertinentes para o desenvolvimento de um sistema de correção e análise semântica de respostas curtas. A IA tem sido uma aliada promissora na educação, com potencial para otimizar a avaliação dos alunos. Este trabalho concentra-se na aplicação da IA no trabalho docente, mostrando como a tecnologia pode auxiliar na correção de questões curtas que, tradicionalmente, exigem análise humana detalhada.

3.1 Revisão de Algoritmos: Classificação de Texto e Análise Semântica

A automação da correção de avaliações de questões subjetivas e perguntas de resposta curta depende amplamente do PLN. O PLN é a subárea que envolve o uso de técnicas para criar sistemas que interagem e compreendem a linguagem falada e escrita (Giraffa; Kohls-Santos, 2023).

3.1.1 O Papel do ML e da Classificação

A categorização (previsão de classes) é uma das tarefas de aprendizado supervisionado mais comuns (Géron, 2019). A classificação de texto, em particular, é uma tarefa comum de PLN, que transforma uma sequência de texto de comprimento indefinido em uma categoria (Zhang et al., 2021). Este conceito é fundamental para o problema da correção, no qual as respostas dos alunos devem ser mapeadas para categorias (por exemplo, correta, incorreta). As aplicações em PLN utilizam modelos de DL, um subconjunto avançado do ML. O conceito de "profundo" refere-se ao uso de múltiplas camadas ocultas de processamento em uma rede neural, o que é essencial para lidar com a complexidade do PLN (Zhang et al., 2021).

3.1.2 Modelos Sensíveis ao Contexto para Análise Semântica

Para a análise semântica em línguas naturais, a compreensão do significado depende do contexto. Modelos mais antigos de incorporação de palavras (*word embedding*), como *word2vec* e *GloVe*, são todos independentes do contexto, o que dificulta o tratamento da polissemia (Zhang et al., 2021).

Para superar isso, modelos avançados fornecem representações de palavras sensíveis ao contexto. O *Bidirectional Encoder Representations from Transformers*, Representações de Codificador Bidirecional de Transformadores (BERT) é um modelo que codifica o contexto bidirecionalmente e combina essa capacidade com o fato de ser agnóstico à tarefa. O BERT utiliza um codificador de transformador pré-treinado, sendo a arquitetura *Transformer* baseada exclusivamente em mecanismos de atenção (Zhang et al., 2021).

3.2 Algoritmos Tradicionais para Classificação

3.2.1 Naive Bayes

O Naive Bayes é um algoritmo notavelmente claro que utiliza fundamentos probabilísticos para classificação (Zhang et al., 2021). A principal suposição subjacente a este classificador é que todos os recursos (ou seja, as palavras) são independentes uns dos outros para simplificar o cálculo. Por décadas, este classificador foi o padrão ouro para tarefas como detecção de *spam* (Zhang et al., 2021).

3.2.2 Random Forest

As Florestas Aleatórias (*Random Forests*) são um *ensemble* de Árvores de Decisão, sendo um dos algoritmos de Aprendizado de Máquina mais poderosos disponíveis (Géron, 2019). O algoritmo opera treinando muitas Árvores de Decisão em subconjuntos aleatórios das características. A introdução dessa aleatoriedade resulta em uma grande diversidade da árvore, trocando um viés ligeiramente maior por uma baixa variância (Géron, 2019).

3.3 Técnicas de Ensemble Learning

O *Ensemble Learning* é a técnica de combinar as previsões de um conjunto de previsores (*ensemble*), o que geralmente resulta em melhores previsões do que o melhor modelo individual (Géron, 2019).

3.3.1 Bagging (Bootstrap Aggregating)

O *Bagging* utiliza o mesmo algoritmo de treinamento em diferentes subconjuntos aleatórios do conjunto de treinamento (Géron, 2019). A amostragem é feita com substituição (*bootstrapping*). Este método resulta em melhores modelos, pois oferece uma variância reduzida. Uma Floresta Aleatória é um *ensemble* de Árvores de Decisão tipicamente treinado pelo método *bagging* (Géron, 2019).

3.3.2 Boosting e AdaBoost

O Boosting é um método *Ensemble* que combina vários aprendizes fracos em um forte (Géron, 2019). A ideia principal é treinar os previsores sequencialmente, no qual cada um tenta corrigir os erros cometidos por seu antecessor. Existem muitos métodos *boosting* disponíveis, sendo o AdaBoost (*Adaptive Boosting*) um dos mais populares (Géron, 2019).

3.4 Aplicações no Processamento de Linguagem Natural Avançado

Para o problema de correção de texto, a capacidade de realizar análise semântica é vital. O uso de modelos como *Transformers*/BERT é uma alternativa poderosa ou complementar aos algoritmos tradicionais, pois o BERT é capaz de fornecer representações sensíveis ao contexto (Zhang et al., 2021).

O BERT requer mudanças mínimas de arquitetura para uma ampla gama de tarefas de PLN. Por exemplo, em aplicações de classificação de texto único (como a classificação de respostas em "correta" ou "incorreta"), a representação do *token* especial “<cls>” (usado para classificação de sequência) é alimentada em um pequeno *Multi-Layer Perceptron*, Perceptron Multicamadas (MLP) (camadas totalmente conectadas/densas) para gerar o resultado (Zhang et al., 2021).

Durante o processo de ajuste fino (*fine-tuning*), que é o método recomendado para adaptar o BERT a uma tarefa *downstream*, os parâmetros das camadas extras são aprendidos do zero, enquanto todos os parâmetros no modelo BERT pré-treinado são ajustados (Zhang et al., 2021).

3.5 Redes Neurais Recorrentes: Arquiteturas para Sequências

As RNA são a tecnologia fundamental que possibilita o ML, sendo capazes de lidar com uma ampla variedade de tarefas, incluindo o PLN (Feltrin, 2021). As RNNs constituem uma classe de redes especificamente projetadas para trabalhar com sequências de comprimentos arbitrários, como frases, documentos ou amostras de áudio (Géron, 2019).

3.5.1 Definição e Mecanismo de Recorrência

As RNNs são definidas por usarem computação recorrente para estados ocultos, o que lhes permite explorar a estrutura sequencial em dados (Zhang et al., 2021). Diferentemente das redes neurais *feedforward*, as RNNs contêm conexões que apontam para trás (Géron, 2019).

O mecanismo essencial das RNNs é o estado oculto (Ht), que é capaz de capturar informações históricas da sequência até o intervalo de tempo corrente. Essa capacidade de reter o estado é fundamental para modelos de sequência (Zhang et al., 2021). Uma vantagem estrutural das RNNs é que o número de parâmetros do modelo não aumenta com o aumento do número de etapas de tempo (tamanho da sequência) (Zhang et al., 2021). O treinamento dessas redes utiliza uma estratégia chamada *Backpropagation Through Time*, Retropropagação Através do Tempo (BPTT) (Géron, 2019).

No entanto, as RNNs simples enfrentam o problema da instabilidade numérica que, em longas sequências, a memória das primeiras entradas desaparece gradualmente. Para mitigar esta limitação e os problemas de gradientes desaparecendo ou explodindo, foram desenvolvidas arquiteturas mais sofisticadas, conhecidas como RNNs controladas (*gated*) (Zhang et al., 2021).

Dentre estas, as arquiteturas *Long Short-Term Memory*, Memória de Longo e Curto Prazo (LSTM) e *Gated Recurrent Units*, Unidade Recorrente com Portão (GRU) são comuns (Zhang et al., 2021). A GRU é uma variante simplificada que, embora muitas vezes ofereça desempenho comparável ao da LSTM, é bem mais rápida para computar (Zhang et al., 2021). Tais modelos avançados suportam o bloqueio do estado oculto e utilizam portas (*gates*) para capturar dependências de longo prazo em sequências (Zhang et al., 2021).

3.5.2 Necessidade para o Programa de Correção de Questões

A aplicação de RNNs (ou suas variantes avançadas como LSTMs e *Transformers*) é necessária para o projeto de "Correção de avaliações de questões subjetivas" e "perguntas de

resposta curta (questão conceitual)" devido à complexidade da linguagem natural e à necessidade de análise de contexto e semântica.

O projeto se enquadra do PLN, a subárea da IA que visa criar sistemas que interagem e compreendem a linguagem escrita. O objetivo do sistema é analisar a semântica das respostas discursivas dos alunos para classificá-las quanto à sua correção conceitual.

A correção de texto complexo exige a compreensão do contexto para determinar o significado, uma vez que a compreensão do significado em línguas naturais é dependente dele:

- Modelos de ML tradicionais (como *Naive Bayes*) não possuem a capacidade de reter e processar o contexto de longo alcance necessário para interpretar nuances conceituais;
- O DL (do qual as RNNs fazem parte) é um elemento central da inovação moderna e permite que o modelo aprenda representações de dados em vários níveis de abstração, o que é essencial para lidar com a complexidade das respostas discursivas;

3.6 A Arquitetura *Transformer* e o Mecanismo de Atenção

A arquitetura *Transformer*, introduzida por Vaswani et al. (2017) no artigo “Attention Is All You Need”, representou uma ruptura paradigmática no campo do PLN ao eliminar a dependência de recorrência sequencial presente nas RNNs. Em vez de processar os *tokens* de forma linear, o *Transformer* processa toda a sequência de entrada em paralelo, utilizando exclusivamente mecanismos de atenção (“*attention mechanisms*”) para modelar as dependências entre os *tokens*, independentemente da distância que os separa na sequência (Zhang et al., 2021).

O componente central da arquitetura é a atenção multi-cabeça (*multi-head attention*), que permite ao modelo aprender simultaneamente múltiplas relações entre as palavras de uma sequência. Para cada cabeça de atenção, o mecanismo projeta os vetores de entrada em três matrizes distintas: *Query (Q), Key (K) e Value (V)*. A pontuação de atenção entre dois *tokens* é calculada pelo produto escalar de Q e K, normalizado pela raiz quadrada da dimensão do vetor, e aplicado a uma função *softmax*, que gera pesos normalizados usados para ponderar os vetores V (Vaswani et al., 2017). Essa operação pode ser expressa como: $Attention(Q,K,V) = softmax(QK^T/\sqrt{dk})V$, onde dk representa a dimensão das chaves.

O resultado prático desse mecanismo é que o modelo aprende durante o treinamento quais palavras são semanticamente relevantes para interpretar cada *token*. Em uma frase como “O banco da praça estava molhado”, o mecanismo de atenção associa o *token* “banco” aos *tokens* “praça” e “molhado”, inferindo que se trata do móvel e não da instituição financeira. Essa capacidade de desambiguação contextual é exatamente o requisito central do problema de correção de respostas discursivas (Zhang et al., 2021).

3.7 BERT: Pré-treinamento Bidirecional e *Fine-Tuning*

O BERT, proposto por Devlin et al. (2019), utiliza exclusivamente o codificador (*encoder*) da arquitetura *Transformer* para gerar representações de linguagem bidirecionais. Diferentemente de outros modelos que o antecederam, que liam o texto da esquerda para a direita ou combinavam representações unidirecionais, o BERT condicionou simultaneamente o contexto esquerdo e direito em todas as camadas do modelo, o que lhe confere uma compreensão mais profunda do significado de cada *token* (Devlin et al., 2019).

O pré-treinamento do BERT é realizado em dois objetivos não supervisionados: a *Masked Language Model*, Modelagem de Linguagem Mascarada (*MLM*), na qual uma parcela aleatória dos *tokens* da sequência de entrada é substituída pelo *token* especial [MASK] e o modelo é treinado para predizer os *tokens* originais a partir do contexto bidirecional; e a *Next Sentence Prediction*, Previsão da Próxima Sentença (NSP), na qual o modelo aprende a determinar se duas sentenças são sequencialmente adjacentes no corpus de origem (Devlin et al., 2019). Esses dois objetivos de pré-treinamento, aplicados a um *corpus* de grande escala (*Wikipedia* e *BooksCorpus*, totalizando cerca de 3,3 bilhões de palavras), conferem ao BERT um conhecimento linguístico geral que pode ser reutilizado em tarefas específicas.

O processo de *Fine-Tuning* consiste em adaptar o modelo pré-treinado a uma tarefa específica (*downstream task*) por meio do treinamento supervisionado com um conjunto de dados rotulado. Para tarefas de classificação de sequência, como a correção de respostas discursivas, utiliza-se a representação do *token* especial [CLS] (*Classification Token*), inserido no início de cada entrada. Esse vetor, após passar por

todas as 12 camadas de atenção do BERT-base, é alimentado em uma camada classificadora adicional para produzir a previsão final (Zhang et al., 2021). No presente trabalho, essa camada classificadora é implementada como uma MLP com arquitetura Linear (768, 64) → *ReLU* → *Dropout(0.2)* → *Linear(64, 3)*, resultando em três *logits* correspondentes às classes Incorreto, Parcial e Correto.

O *token* separador [SEP] é outro elemento nativo da arquitetura BERT com papel fundamental neste projeto. Originalmente utilizado para delimitar pares de sentenças nas tarefas de NSP durante o pré-treinamento, o [SEP] é reaproveitado neste sistema para sinalizar a fronteira semântica entre o enunciado da questão e a resposta do aluno. A concatenação "Pergunta [SEP] Resposta" garante que o modelo, ao processar o *token* [CLS], tenha acesso simultâneo ao contexto da pergunta e ao conteúdo da resposta, eliminando o que este trabalho denomina de "cegueira de contexto", o problema identificado no item 5.4, Teste 4 desse trabalho, em que o modelo avaliava respostas independentemente do enunciado ao qual pertenciam.

3.8 BERTimbau: Adaptação do BERT para o Português Brasileiro

O *BERTimbau* é um modelo de linguagem baseado na arquitetura BERT, pré-treinado especificamente sobre um *corpus* de português brasileiro de larga escala, composto por aproximadamente 2,68 bilhões de palavras extraídas de fontes como a Wikipedia em português e outros *corpora web* (Souza; Nogueira; Lotufo, 2020). Desenvolvido pelo grupo *NeuralMind* da Universidade Estadual de Campinas (Unicamp), o modelo é disponibilizado publicamente no *Hugging* Face *Hub* sob o identificador *neuralmind/bert-base-portuguese-cased*.

A escolha do *BERTimbau* em detrimento do BERT *multilingual* (mBERT) ou do BERT-*base* original em inglês justifica-se por três razões. Primeiro, o vocabulário do *tokenizador* foi construído a partir de texto em português, o que resulta em uma tokenização mais eficiente das palavras da língua, com menos fragmentação de *tokens* por palavra comparado ao *mBERT* (Souza; Nogueira; Lotufo, 2020). Segundo, os pesos pré-treinados já encapsulam padrões morfossintáticos e semânticos do português brasileiro, como a concordância nominal e verbal, a polissemia e as expressões idiomáticas, que são centrais para a interpretação das respostas dos alunos. Terceiro, os

resultados reportados em *benchmarks* de PLN em português, como o Análise de Semântica de Inferência (ASSIN 2), demonstram que o *BERTimbau* supera consistentemente o *mBERT* em tarefas de classificação e *entailment* no idioma (Souza; Nogueira; Lotufo, 2020).

No contexto deste projeto, o *BERTimbau* opera com a estratégia de congelamento de camadas (*layer freezing*): Durante o *fine-tuning*, os pesos das 12 camadas de codificação do BERT são congelados (*param.requires_grad = False*), e apenas os pesos da camada MLP classificadora são atualizados. Essa estratégia preserva o conhecimento linguístico geral adquirido no pré-treinamento e reduz substancialmente o número de parâmetros a otimizar, viabilizando o treinamento em *hardware* de consumo (CPU + AMD Radeon RX 7600) sem recorrer a infraestrutura de *Graphics Processing Unit*, Unidade de Processamento Gráfico (GPU) dedicada de alto custo, e convergindo em poucas épocas com um *dataset* sintético de apenas dezenas de exemplos por classe.

3.9 IA Generativa e o Modelo Gemini

Modelos de Inteligência Artificial Generativa são sistemas capazes de gerar conteúdo novo — texto, imagens, código ou áudio — a partir de uma instrução em linguagem natural (*prompt*). No domínio do texto, esses modelos são baseados em *Large Language Models*, Grandes Modelos de Linguagem (LLMs), treinados em *corpus* de escala *Internet* com bilhões de parâmetros. A capacidade generativa emerge do pré-treinamento em predição do próximo *token* em escala massiva, que leva o modelo a internalizar padrões de coerência, estilo e conhecimento factual de forma implícita (Zhang et al., 2021).

O Gemini é uma família de modelos multimodais de grande porte desenvolvida pelo Google DeepMind. No contexto deste trabalho, utilizou-se o modelo *gemini-2.0-flash* e, posteriormente, *gemini-2.5-flash*, acessados por meio da biblioteca *google-genai* para Python. A técnica de *Prompt Engineering* (Engenharia de Prompts) foi empregada para instruir o modelo a assumir o papel de um professor assistente e gerar, a partir de um trecho do livro-texto e do gabarito oficial, respostas sintéticas de alunos distribuídas equilibradamente em três níveis de qualidade: Correta (nota 2), parcialmente correta (nota 1) e incorreta (nota 0). Essa abordagem é denominada *Data Augmentation* por

geração sintética e constitui uma alternativa visável para contextos com escassez de dados rotulados, frequente em ambiente acadêmico.

3.10 Recuperação Aumentada por Geração

A arquitetura *Retrieval-Augmented Generation*, Recuperação Aumentada por Geração (RAG), proposta por Lewis et al. (2020), combina um componente de recuperação de informação com um modelo generativo, permitindo que o LLM fundamente suas respostas em documentos recuperados dinamicamente, em vez de depender exclusivamente do conhecimento implícito nos parâmetros do modelo. Essa abordagem reduz o problema de alucinações que seriam respostas factualmente incorretas, mas fluentes e permite atualizar o conhecimento acessível ao modelo sem necessidade de retreinamento (Lewis et al., 2020).

Neste projeto, implementou-se uma versão local e leve do RAG utilizando a vetorização *Term Frequency – Inverse Document Frequency*, requência do Termo e Frequência Inversa nos Documentos (TF-IDF) e a similaridade de cossenos, ambas fornecidas pelo Scikit-learn. O documento de referência (livro-texto) é segmentado em blocos de 5 páginas (chunks), cada um representado como um vetor TF-IDF. Diante de uma nova questão, o sistema calcula a similaridade de cossenos entre o vetor da questão concatenada ao gabarito e todos os vetores dos *chunks*, selecionando os três segmentos mais relevantes para compor o contexto enviado à *Application Programming Interface*, *Interface* de Programação de Aplicativos (API) do Gemini. Essa estratégia reduziu o consumo de *tokens* em aproximadamente 95% por consulta antes cerca de 297.000 *tokens* (envio do livro completo) para aproximadamente 11.000 *tokens* (envio apenas dos três *chunks* mais relevantes) assim eliminando os erros de cota (*429 RESOURCE_EXHAUSTED*) e os *timeouts* (*503 UNAVAILABLE*) que inviabilizavam o *pipeline* nas etapas anteriores.

Embora o RAG local baseado em TF-IDF seja menos sofisticado que implementações com *embeddings* neurais densos (como os gerados por modelos *sentence-transformers*), sua escolha se justifica pela eficiência computacional, pela ausência de dependência de serviços externos adicionais e pela adequação ao cenário

de domínio restrito deste projeto, no qual os *chunks* do livro-texto possuem vocabulário técnico especializado, favorecendo a recuperação baseada em frequência de termos.

4 PROCEDIMENTOS METODOLÓGICOS

Este capítulo descreve os passos, ferramentas e a arquitetura utilizados para o desenvolvimento do sistema automatizado de correção de questões discursivas, bem como o ambiente de desenvolvimento e as bibliotecas empregadas na construção do protótipo.

4.1 Visão geral

O sistema foi projetado sob uma arquitetura modular que engloba a extração de contexto de materiais didáticos, a geração de dados sintéticos por meio de Inteligência Artificial Generativa e o treinamento de um modelo de linguagem baseado em *Transformers* (BERT).

4.1.1 Recursos de Hardware

O desenvolvimento, a prototipação e a execução local dos treinamentos de redes neurais foram realizadas em um computador operando com o sistema Windows. A máquina é equipada com um processador AMD Ryzen 5 5600X (6-Core, 3.70 GHz), 32 GB de memória RAM (3200 MT/s), armazenamento de 1,35 TB e uma placa de vídeo dedicada AMD Radeon RX 7600 com 8 GB de VRAM.

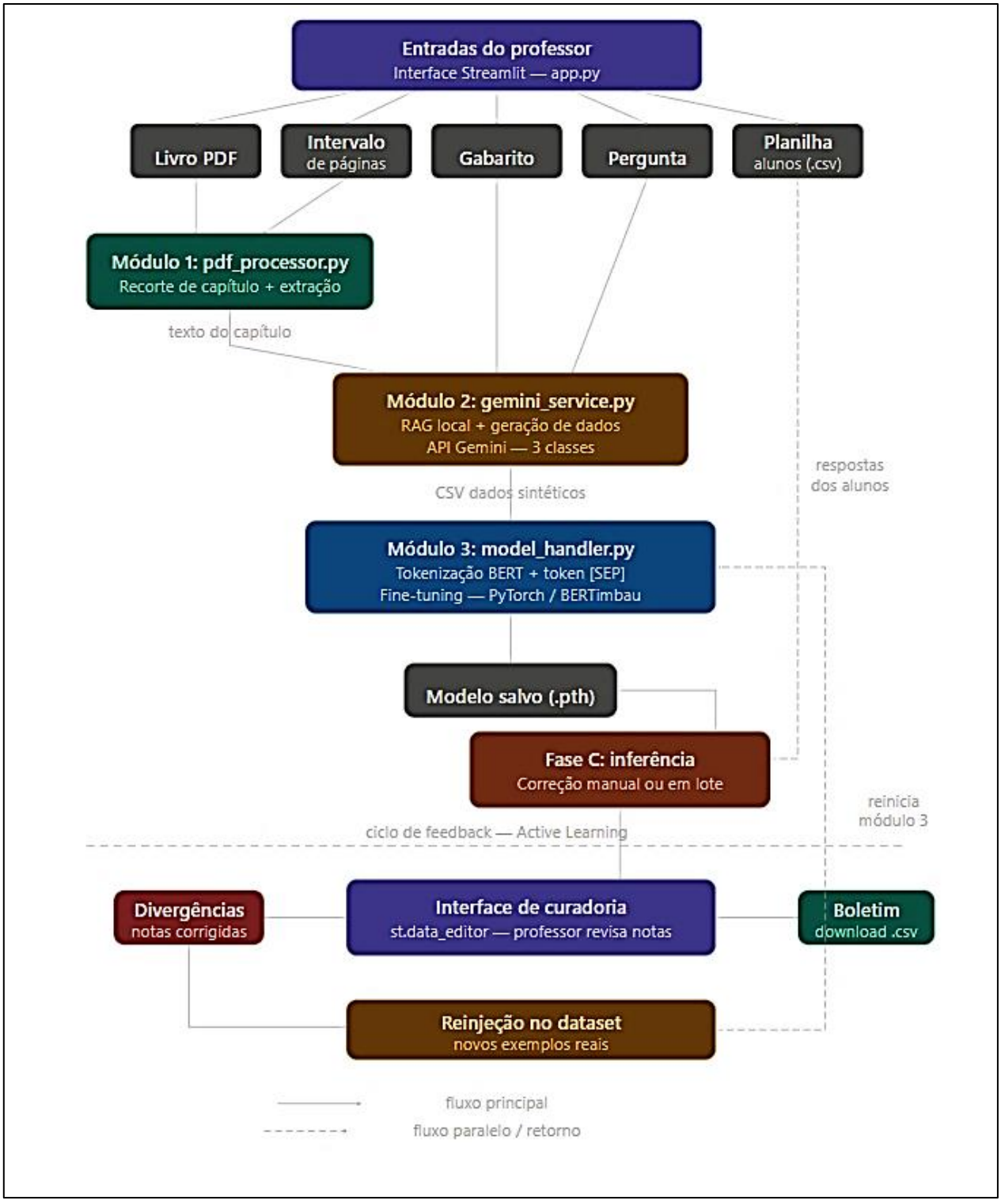
4.1.2 Recursos de Software

O ambiente de desenvolvimento baseou-se na linguagem de programação Python, escolhida por sua vasta adoção e robusto suporte no ecossistema de ML e PLN.

4.1.3 Diagrama de Blocos

A Figura 1 ilustra o fluxo de processamento contínuo (*pipeline*) da aplicação, estruturado de forma modular e sequencial. O processo inicia-se com as diretrizes fornecidas pelo docente através da *interface* visual no *Streamlit*. O Módulo 1 (*pdf_processor.py*) recebe o arquivo físico do Livro em formato *Portable Document Format*, Formato de Documento Portátil (PDF) juntamente com o parâmetro numérico do intervalo de páginas, sendo o responsável exclusivo por executar o recorte estrutural e a extração do texto puro do capítulo desejado.

Figura 1 Diagrama de blocos



Fonte: Elaborado pelo autor desse trabalho

Em seguida, o Módulo 2 (*gemini_service.py*) consome o texto extraído, combinando-o com a pergunta e o gabarito oficial. Este módulo emprega uma arquitetura RAG local associada à API do Gemini para gerar, de forma autônoma, um banco de dados em *Comma-Separated Values*, Valores Separados por Vírgulas (CSV) contendo respostas sintéticas balanceadas em três classes de pontuação. Este *dataset* sintético alimenta o Módulo 3 (*model_handler.py*), etapa onde ocorre a injeção do *token* de contexto *[SEP]* unindo a pergunta à resposta gerada. O Módulo 3 realiza o ajuste fino (*Fine-tuning*) do modelo de linguagem BERTimbau utilizando os tensores do PyTorch, consolidando o aprendizado no arquivo de pesos *.pth*.

Na fase de produção (Fase C: Inferência), a rede neural treinada cruza a planilha CSV contendo as respostas reais dos alunos com a matriz de conhecimento do modelo, gerando as notas. Por fim, a arquitetura destaca o ciclo de Aprendizado Ativo (*Active Learning*), representado na parte inferior do fluxograma da Figura 1. Nesta etapa, a *interface* de curadoria permite ao professor revisar o boletim e alterar notas incorretas atribuídas pela IA. As divergências corrigidas manualmente são capturadas pelo sistema e reinjetadas no *dataset* como novos exemplos reais, acionando um retreinamento no Módulo 3 e garantindo a evolução contínua da Inteligência Artificial.

4.2 Elementos da aplicação

Esta seção detalha as plataformas, modelos de Inteligência Artificial e pacotes de *software* específicos escolhidos para compor a solução tecnológica desenvolvida.

4.2.1 Visual Studio Code

O *Visual Studio Code* (VSCode) foi a *Integrated Development Environment*, Ambiente de Desenvolvimento Integrado (IDE) selecionada para a escrita do programa final devido problemas de armazenamento encontrados no ambiente inicialmente utilizado nesse trabalho, o *Google Cloud Shell*, depuração e organização do código-fonte. Sua escolha justifica-se pela arquitetura leve, alta customização por meio de extensões oficiais para Python e integração nativa com o terminal do sistema operacional. Tais características conferiram agilidade na navegação entre os múltiplos *scripts* modulares

do projeto e simplificaram o monitoramento dos recursos de *hardware* durante o treinamento da rede neural.

4.2.2 Gemini API

O modelo de Inteligência Artificial Generativa Gemini (do ecossistema Google), acessado via a biblioteca da API (*google-genai*), atuou como o motor de *Data Augmentation* (Aumento de Dados) da aplicação. Devido à dificuldade natural de se obter grandes volumes de dados reais de alunos previamente rotulados, utilizou-se a técnica de *Prompt Engineering* (Engenharia de Prompts). A API é programada para atuar como um "professor assistente": ela recebe o trecho do livro-texto e o gabarito oficial e retorna dezenas de respostas sintéticas, rigorosamente formatadas em *JavaScript Object Notation*, Notação de Objetos JavaScript (JSON). Estas respostas simulam o vocabulário de um aluno e são balanceadas nas três classes de correção (Nota 2, 1 e 0), fornecendo a base de conhecimento necessária para que a rede neural classificadora inicie seu treinamento.

4.2.3 Bibliotecas essenciais

A engenharia de *software* da aplicação dividiu as responsabilidades do sistema entre bibliotecas especializadas para garantir *performance* e modularidade:

4.2.3.1 Streamlit

A biblioteca *Streamlit* foi responsável por toda a camada visual (*Front-end*) e permitiu a construção da *interface* de usuário utilizando apenas a linguagem Python. Esta biblioteca gerenciou o *upload* de arquivos, o controle de estado da aplicação em tempo real e a renderização da tabela de dados interativa (*st.data_editor*), componente fundamental para a etapa de revisão humana e *Active Learning*.

4.2.3.2 PyTorch (ou torch)

A biblioteca *PyTorch* atuou como o núcleo computacional e matemático do processamento de *Deep Learning* (*Back-end*). Esta biblioteca de código aberto operou nos bastidores realizando os cálculos matriciais e a manipulação dos tensores multidimensionais exigidos pelo modelo BERT. O *PyTorch* foi o responsável por gerenciar o laço de treinamento da rede neural, instanciar o otimizador matemático (Adam), calcular a função de perda estatística para a classificação multiclasse (*CrossEntropyLoss*) e otimizar a atualização dos pesos sinápticos durante as épocas (*epochs*) de aprendizado (*Fine-tuning*).

4.2.3.3 Pandas

A biblioteca *Pandas* foi essencial para a manipulação estruturada de dados. Ela foi utilizada para criar, formatar e transitar os *DataFrames* entre os módulos geradores e classificadores, além de ser a ferramenta base para a leitura e exportação dos boletins acadêmicos em formato CSV.

4.2.3.4 PyMuPDF (ou fitz)

A biblioteca *PyMuPDF* foi empregada para a manipulação direta do arquivo de texto-base no formato (PDF) e permitiu a navegação e o recorte de capítulos específicos informados pelo usuário, extraindo o texto puro em segundos e descartando os resíduos de memória, evitando sobrecarga computacional.

4.2.3.5 Scikit-learn

A biblioteca *Scikit-learn* desempenhou um papel essencial nas etapas de prototipação, recuperação de informação e avaliação estatística. Inicialmente, a biblioteca forneceu os algoritmos matemáticos, como a vetorização TF-IDF e o cálculo de similaridade de cossenos, que viabilizaram a construção do módulo RAG local. Por

fim, o *Scikit-learn* foi a ferramenta empregada para a extração das métricas formais de validação do modelo final, incluindo o cálculo da acurácia.

4.3 Trechos essenciais do Código

Esta seção apresenta os trechos de código mais relevantes do sistema, organizados por módulo, evidenciando as decisões de implementação que viabilizaram o funcionamento do *pipeline* completo.

4.3.1 Arquitetura da classe ModeloBertMLP (model_handler.py, linhas 1–20)

A Figura 2 apresenta a definição da classe ModeloBertMLP, que constitui o núcleo classificador do sistema.

Figura 2 - Definição da classe ModeloBertMLP

app.py | pdf_processor.py | gemini_service.py | model_handler.py

TCC > Final > model_handler.py > treinar_bert

```
1   import torch
2   import torch.nn as nn
3   from transformers import BertTokenizer, BertModel
4   import pandas as pd
5   from torch.utils.data import DataLoader, TensorDataset
6
7   class ModeloBertMLP(nn.Module):
8       def __init__(self):
9           super(ModeloBertMLP, self).__init__()
10          self.bert = BertModel.from_pretrained('neuralmind/bert-base-portuguese-cased')
11          for param in self.bert.parameters():
12              param.requires_grad = False
13          self.mlp = nn.Sequential(
14              nn.Linear(768, 64), nn.ReLU(), nn.Dropout(0.2),
15              nn.Linear(64, 3)
16          )
17
18      def forward(self, input_ids, attention_mask):
19          saida = self.bert(input_ids=input_ids, attention_mask=attention_mask)
20          return self.mlp(saida.pooler_output)
21
```

Fonte: Capturado pelo autor desse trabalho

No método *__init__*, o modelo BERTimbau (*neuralmind/bert-base-portuguese-cased*) é carregado como extrator de features por meio da biblioteca *transformers*. Para

viabilizar o treinamento em *hardware* de consumo sem GPU dedicada ao *PyTorch*, os pesos do BERT são congelados na linha 11 (*param.requires_grad = False*), de modo que apenas a camada MLP acoplada é ajustada durante o *fine-tuning*. Isso significa que as 12 camadas de atenção do modelo pré-treinado não sofrem atualização de gradientes durante o treinamento. O BERT atua exclusivamente como um extrator de contexto estático, convertendo o texto de entrada em tensores. Esta decisão foi tomada para reduzir a carga de processamento e o consumo de memória VRAM.

Essa camada, definida nas linhas 13 a 16, é composta por uma sequência *nn.Sequential* com dois blocos lineares (*Linear (768, 64))* e *Linear (64, 3))*, uma função de ativação *ReLU* e uma camada de regularização *Dropout(0.2)*, resultando em três neurônios de saída correspondentes às classes: *Incorreto (0)*, *Parcial (1)* e *Correto (2)*.

O método *forward* recebe os tensores de entrada (*input_ids e attention_mask*) e retorna a saída da MLP a partir do vetor *pooler_output* do BERT, que representa semanticamente toda a sentença de entrada.

4.3.2 Função de treinamento com injeção de contexto via token [SEP] (model_handler.py, linhas 22–52)

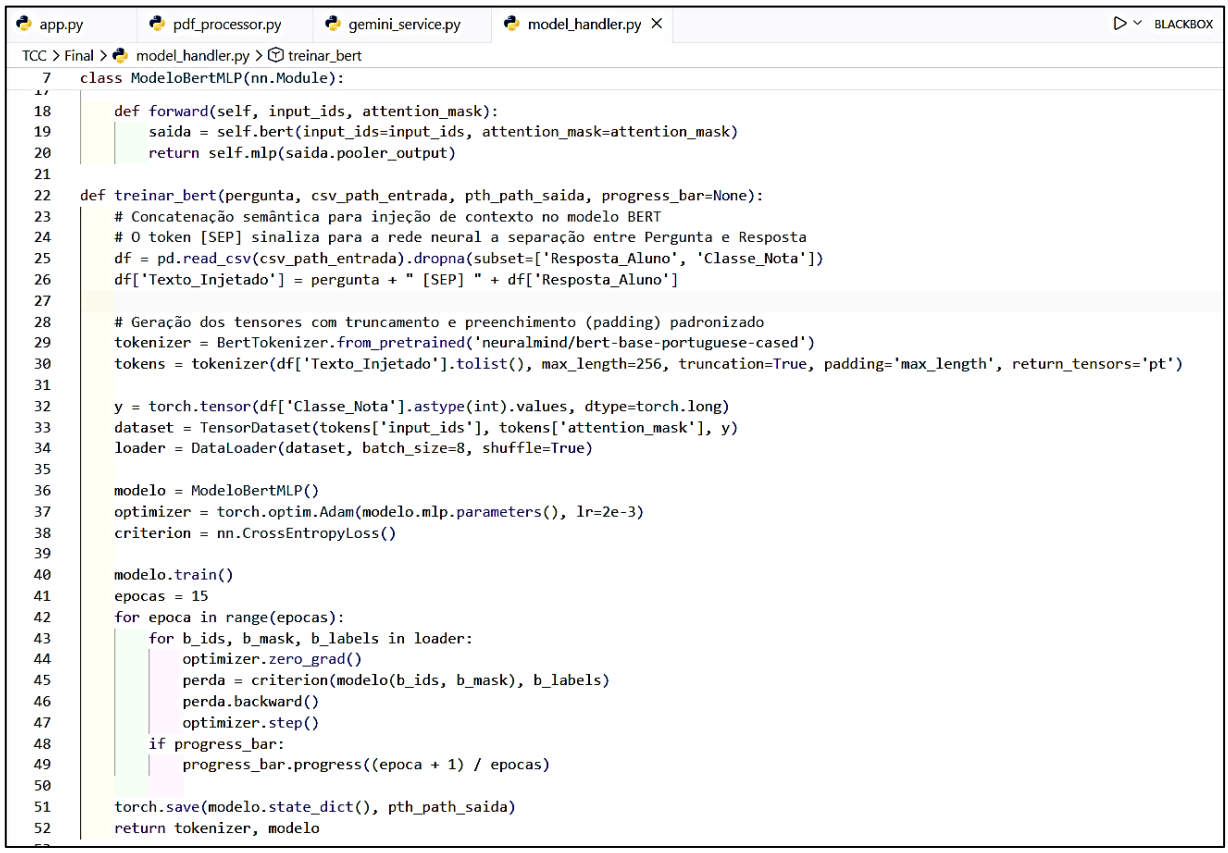
A Figura 3 exibe a função *treinar_bert*, responsável pelo ciclo completo de *fine-tuning*.

A linha 26 implementa a estratégia de injeção de contexto: antes da tokenização, o texto de entrada é construído como a concatenação da pergunta da prova com a resposta do aluno, separadas pelo *token* nativo [SEP] da arquitetura BERT. Essa decisão garante que o modelo aprenda a avaliar a resposta sempre em relação ao enunciado da questão, eliminando o problema de "cegueira de contexto" identificado no teste 7 descrito no item 5.7 deste trabalho.

Nas linhas 29 a 34, os textos concatenados são tokenizados com comprimento máximo de 256 *tokens* e os dados são encapsulados em um *TensorDataset* com *DataLoader* de tamanho de lote 8. O laço de treinamento (linhas 41 a 49) utiliza o otimizador Adam com taxa de aprendizado 2e-3 e a função de perda *CrossEntropyLoss*, adequada à classificação *multiclasse*.

A barra de progresso integrada ao *Streamlit* é atualizada a cada época via o parâmetro *progress_bar*, conferindo *feedback* visual ao professor durante o treinamento. Ao final, o modelo treinado é serializado em disco com *torch.save*.

Figura 3 – Função treinar_bert



Fonte: Capturado pelo autor desse trabalho

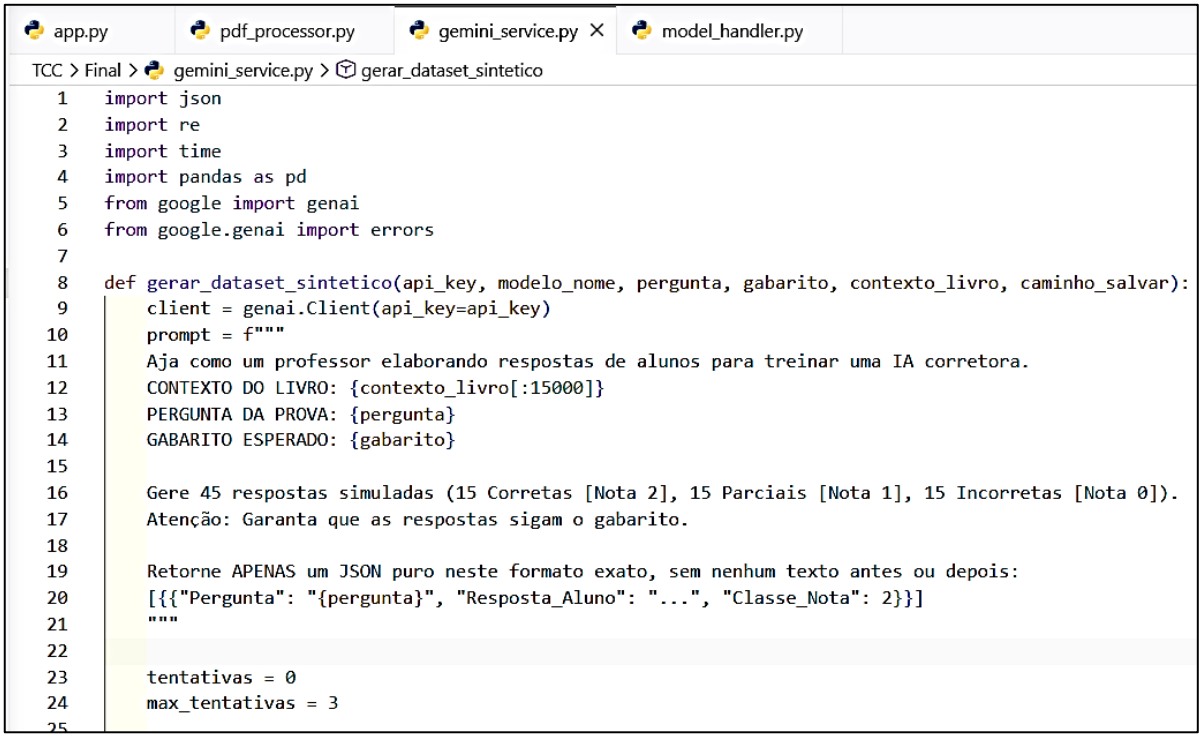
4.3.3 Engenharia de prompt e estrutura da geração de dados sintéticos (gemini_service.py, linhas 1–24)

A Figura 4 mostra o início da função *gerar_dataset_sintetico*, que opera como o motor de *Data Augmentation* do sistema.

O *prompt* construído nas linhas 10 a 22 instrui o modelo Gemini a assumir o papel de um professor assistente, recebendo como variáveis dinâmicas o trecho do livro-texto (limitado a 15.000 caracteres, já filtrado pelo módulo RAG), a pergunta da prova e o gabarito esperado.

A instrução solicita explicitamente a geração de 45 respostas balanceadas entre as três classes (15 por classe), e determina que a saída deve ser exclusivamente um (JSON) puro no formato *[{"Pergunta": "...", "Resposta_Aluno": "...", "Classe_Nota": 2}]*, sem nenhum texto antes ou depois, prevenindo erros de decodificação.

Figura 4 – Função gerar_dataset_sintetico



Fonte: Capturado pelo autor desse trabalho

O contador de tentativas inicializado na linha 23 prepara o mecanismo de tolerância a falhas descrito na Figura 5.

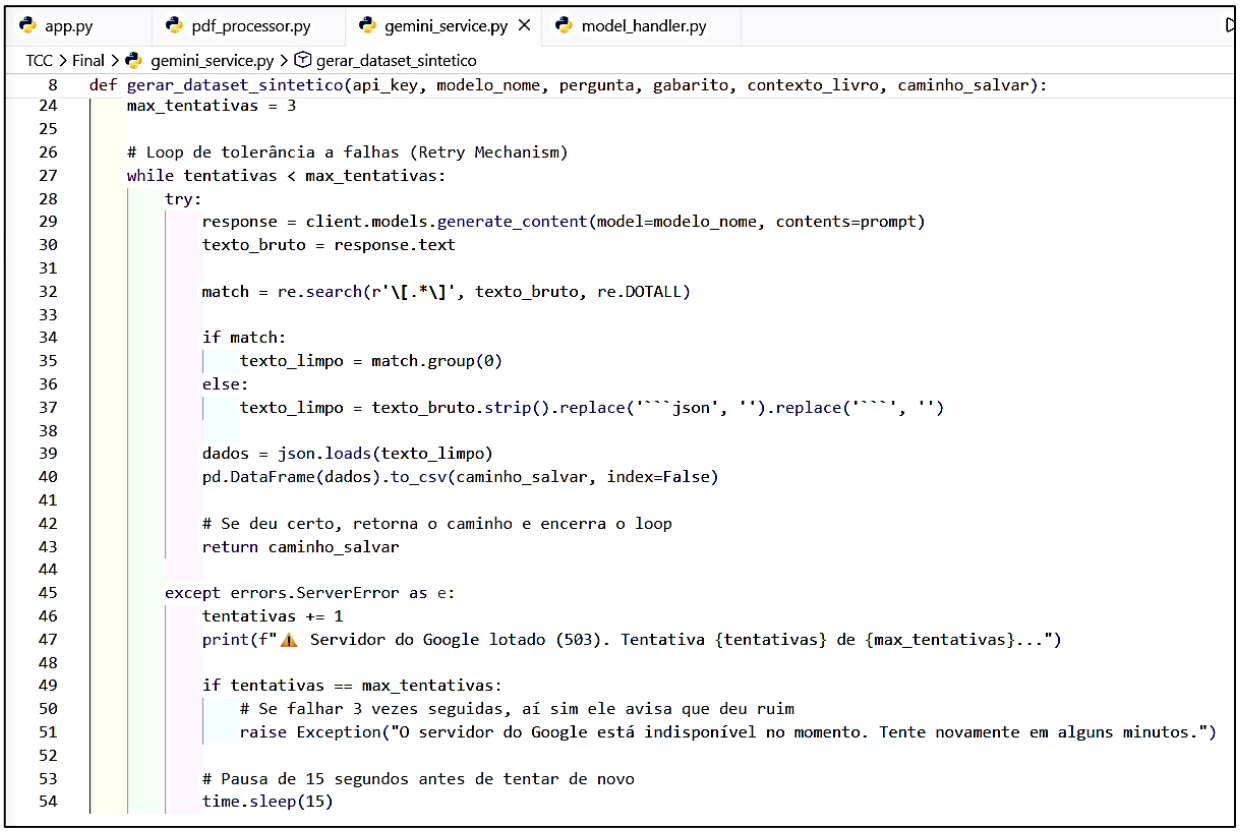
4.3.4 Mecanismo de tolerância a falhas e extração de JSON via Regex (gemini_service.py, linhas 24–54)

A Figura 5 apresenta a continuação da função *gerar_dataset_sintetico*, com destaque para dois mecanismos de resiliência implementados após os erros encontrados nos testes de integração.

O laço *while* (linha 27) implementa o padrão *Retry*: Em caso de erro *503 UNAVAILABLE* da *API*, o sistema aguarda 15 segundos (linha 54) antes de realizar uma nova tentativa, até o limite de três.

Na linha 32, a Expressão Regular *re.search(r'\[.*\]', texto_bruto, re.DOTALL)* extrai o bloco *JSON* da resposta independentemente de textos periféricos gerados pelo modelo — solução para o *JSONDecodeError* identificado no Teste 8.

Figura 5 – Trecho de função para tolerância a falhas



Fonte: Capturado pelo autor desse trabalho

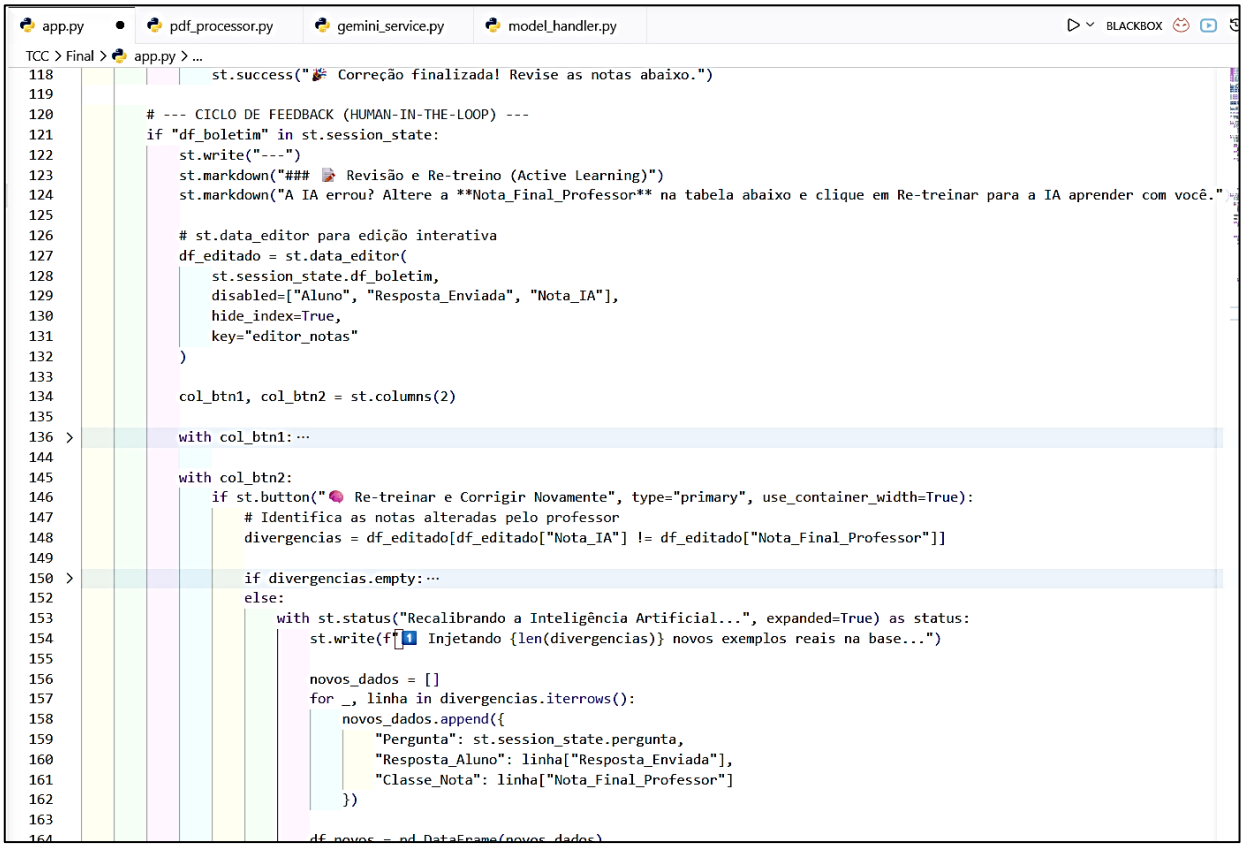
O *fallback* nas linhas 36 e 37 realiza a limpeza manual removendo marcadores de bloco de código *Markdown (```json e ```),* caso a *regex* não encontre um *array*. Somente após a extração bem-sucedida o JSON é *parseado* e convertido em *DataFrame* Pandas, que é salvo como *CSV* na linha 40.

4.3.5 Ciclo de Active Learning com st.data_editor (app.py, linhas 118–164)

A Figura 6 ilustra a implementação do ciclo de *Human-in-the-Loop* no módulo principal da aplicação.

Após a exibição do boletim, a linha 127 renderiza um componente *st.data_editor* com as colunas *Aluno, Resposta_Enviada e Nota_IA* bloqueadas para edição (*disabled*), enquanto a coluna *Nota_Final_Professor* permanece editável, permitindo que o docente corrija eventuais divergências diretamente na *interface*.

Figura 6 – Trecho de app.py, ciclo de *Active Learning*



Fonte: Capturado pelo autor desse trabalho

O botão "Re-treinar e Corrigir Novamente" (linha 146) aciona a linha 148, que filtra as linhas onde *Nota_IA* difere de *Nota_Final_Professor*.

O laço *iterrows* nas linhas 157 a 162 constrói uma lista de novos exemplos reais com a pergunta, a resposta do aluno e a nota corrigida pelo professor, que são então

injetados de volta no *dataset* de treinamento, reiniciando o *fine-tuning* com dados autênticos de sala de aula.

4.3.6 Extração e recorte de capítulo do livro-texto (pdf_processor.py, linhas 1–27)

A Figura 7 exibe o módulo pdf_processor.py, responsável pela extração cirúrgica do conteúdo do livro-texto.

Figura 7 – Função processar_capitulo

app.py ● | pdf_processor.py × | gemini_service.py | model_handler.py

TCC > Final > pdf_processor.py > deletar_arquivos_questao

```python
import fitz  # PyMuPDF
import os
import shutil

def processar_capitulo(upload_file, intervalo_paginas, pasta_destino):
    temp_path = os.path.join(pasta_destino, "livro_completo_temp.pdf")
    output_path = os.path.join(pasta_destino, "capitulo_cortado.pdf")

    with open(temp_path, "wb") as f:
        f.write(upload_file.getbuffer())

    doc = fitz.open(temp_path)
    try:
        inicio, fim = map(int, intervalo_paginas.split("-"))
        doc.select(range(inicio-1, min(fim, doc.page_count)))
        doc.save(output_path)

        texto_extraido = ""
        for page in doc:
            texto_extraido += page.get_text("text") + "\n"

        doc.close()
        os.remove(temp_path)
        return texto_extraido, output_path
    except Exception as e:
        if doc: doc.close()
        return None, None

```

Fonte: Capturado pelo autor desse trabalho

A função *processar_capitulo* recebe o arquivo PDF enviado pelo professor via *interface Streamlit*, salva-o temporariamente em disco (linha 9) e abre-o com a biblioteca *fitz* (*PyMuPDF*) na linha 12.

O intervalo de páginas informado pelo professor (ex.: "126-150") é *parseado* na linha 14 com *map(int, ...)* e utilizado para selecionar apenas o subconjunto de páginas relevante via *doc.select* (linha 15), descartando todo o restante do documento.

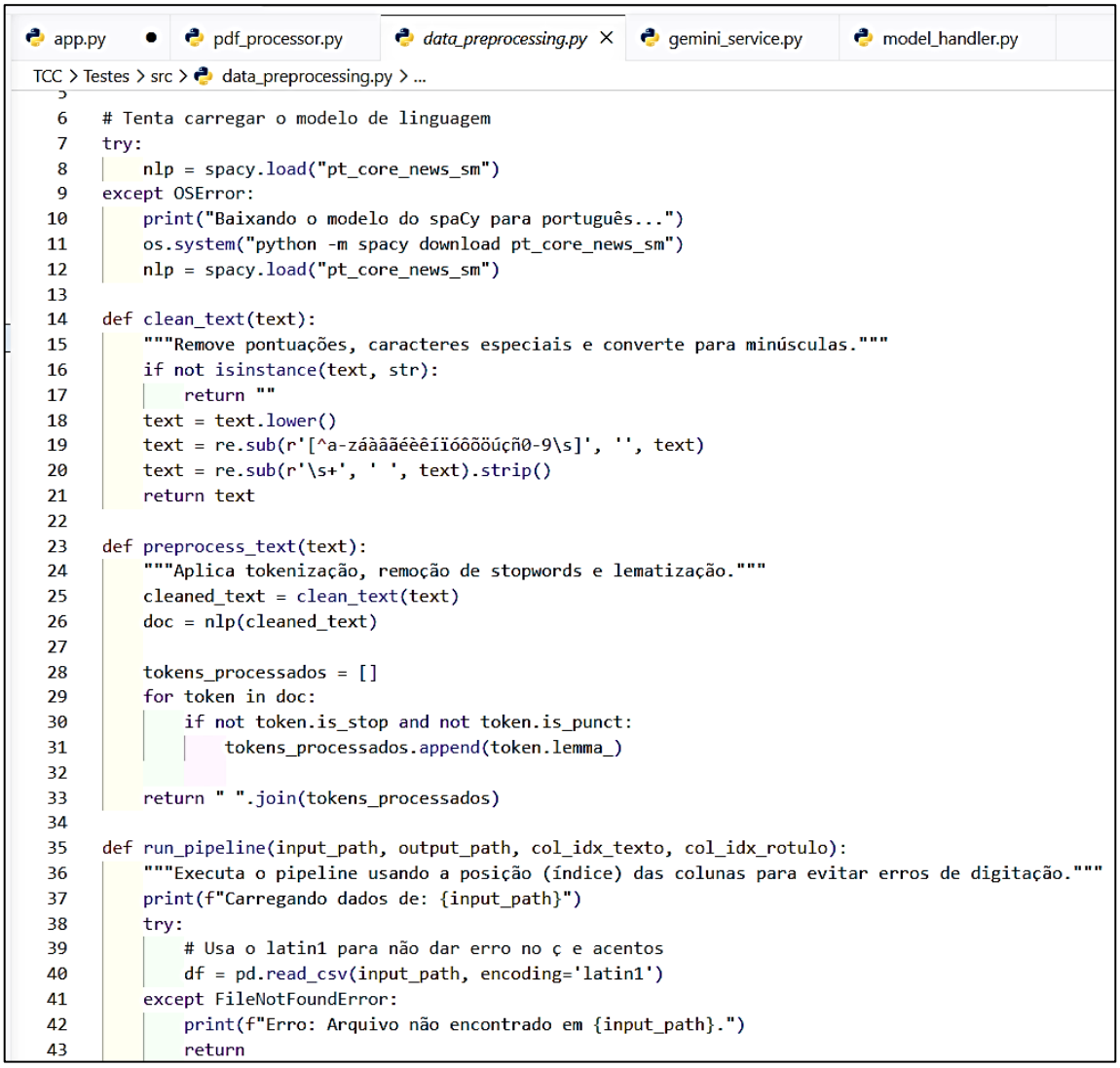
O texto puro é extraído página a página no laço das linhas 19 e 20 com *page.get_text("text"),* e o arquivo temporário é removido da memória na linha 23 para evitar sobrecarga.

Em caso de exceção, a função retorna *None*, *None* (linha 27), permitindo tratamento de erro na camada superior sem interromper a aplicação.

4.3.7 Pré-processamento de texto com spaCy (data_preprocessing.py, linhas 6–43)

A Figura 8 apresenta o *script data_preprocessing.py*, desenvolvido na fase de testes iniciais (Teste 1) e utilizado nas iterações executadas no *Google Cloud Shell*.

Figura 8 – Trecho de *data_preprocessing.py*, função de remoção de caracteres especiais



Fonte: Capturado pelo autor desse trabalho

A função *clean_text* (linha 14) realiza a normalização do texto: conversão para minúsculas, remoção de caracteres especiais por meio de Expressão Regular que preserva letras acentuadas do português (linha 19) e eliminação de espaços redundantes.

A função *preprocess_text* (linha 23) aplica o pipeline completo de *PLN* do *spaCy*: tokenização, remoção de *stopwords (token.is_stop)* e lematização (*token.lemma_),* retornando os *tokens* processados reunidos em uma *string*.

A função *run_pipeline* (linha 35), que orquestra o fluxo completo, utiliza *pd.read_csv* com *encoding='latin1'* (linha 40) — correção aplicada após o *UnicodeDecodeError* identificado no Teste 1 — e acessa as colunas do *CSV* exportado pelo *Microsoft Forms* por índice numérico em vez de nome, contornando o *KeyError* causado pelo travessão longo no cabeçalho original.

5 TESTES E RESULTADOS

Este capítulo apresenta os experimentos realizados ao longo do ciclo de vida do desenvolvimento do sistema, desde as primeiras execuções isoladas de scripts em ambiente de terminal até a consolidação de uma aplicação web unificada. Os testes foram conduzidos de forma iterativa e incremental, seguindo uma abordagem empírica na qual cada etapa revelou limitações técnicas que motivaram evoluções arquiteturais subsequentes.

A metodologia adotada permitiu validar progressivamente cada componente do pipeline — o pré-processamento de dados, a geração de dados sintéticos via API, o treinamento do modelo *BERTimbau* e a interface de correção — antes de integrá-los em uma solução coesa. Para cada teste são descritos o objetivo perseguido, os resultados efetivamente alcançados e a avaliação crítica desses resultados, incluindo os erros encontrados, as correções aplicadas e as decisões de projeto delas decorrentes.

5.1 Teste 1 — Pré-processamento de Dados (Google Cloud Shell)

O objetivo desse teste foi testar o *script data_preprocessing.py*, já descrito no item 4.3.7 desse trabalho, para limpar e tokenizar as respostas dos alunos exportadas do Microsoft Forms e preparar os dados para o treinamento.

Após esse tetes constatou-se que, apesar do *script* ter sido executado sem erros de sintaxe, houve três erros em sequência nos módulos:

- *ModuleNotFoundError*: no *module named 'tkinter'* (ambiente sem interface gráfica);
- *UnicodeDecodeError*: ao tentar ler o arquivo no formato CSV com caracteres especiais do português, como vogais acentuadas e o caractere “ç”;
- *KeyError*: ao tentar acessar a coluna de pontos pelo nome exato (o travessão longo – causou incompatibilidade).

Por meio desse teste, verificou-se que não houve dados processados com sucesso. As falhas revelaram que o ambiente Cloud Shell é incompatível com *tkinter*, que o CSV exportado pelo Forms usa codificação *latin1* e não UTF-8, e que os nomes de colunas com caracteres especiais são frágeis. Três correções foram aplicadas no Cloud

Shell: a substituição de *tkinter* por *input()* no terminal, a adição de *encoding='latin1'* na leitura do CSV e o acesso às colunas por índice numérico em vez de nome.

5.2 Teste 2 — Primeiro treinamento do BERTimbau (Google Cloud Shell)

O objetivo desse teste foi executar o *script bert_model.py* com *PyTorch* e o modelo *BERTimbau* (*neuralmind/bert-base-portuguese-cased*) sobre os dados reais processados, validando se a arquitetura BERT + MLP seria capaz de superar a linha de base.

Após esse teste constatou-se que houve vários erros em sequência antes do sucesso:

- *ImportError: cannot import name 'TFBertModel'* (incompatibilidade do TensorFlow com a versão atual da biblioteca *transformers*);
- Erro de espaço: tentativa de instalação do PyTorch no ambiente do Google Cloud Shell resultou em erro de disco cheio (*No space left on device*).

Por meio desse teste, verificou-se que, após diversas tentativas para contornar a limitação de espaço de disco do Google Cloud Shell, seria mais viável realizar a migração do projeto para o VSCode para evitar esse problema.

5.3 Teste 3 — Treinamento com dados sintéticos e migração para o VSCode

O objetivo desse teste foi migrar o projeto para o VSCode e gerar 50 respostas sintéticas com o script *gerar_dados.py*, para realizar um primeiro teste, pré-processar e treinar o BERT com uma base maior, aumentando a confiabilidade das métricas.

Após esse teste constatou-se o primeiro erro ao gerar o CSV:

- *UnicodeEncodeError*: causado pelo travessão (–) no nome da coluna ao salvar com *encoding='latin1'*.

Após corrigir para traço simples (-), o CSV foi gerado com sucesso. O treinamento foi testado com 20 épocas mostrou progressão clara: acurácia travada em 60% nas primeiras épocas, saltando para 90% na época 7 e atingindo uma acurácia bem próxima de 100% com base nos dados sintéticos nas últimas épocas. Posteriormente foram feitos

testes com quantidades diferentes de épocas e foi notado que o resultado de 15 épocas era bem semelhante ao de 20 e foi preferido por tornar a execução mais ágil.

Por meio desse teste, verificou-se que a acurácia próxima de 100% sobre dados sintéticos indica que o modelo aprendeu os padrões das respostas geradas programaticamente. A função de *Loss* chegou a um valor abaixo de 0,1. Importante notar que esse resultado é otimista por se tratar de dados sintéticos — a validação com dados reais seria necessária nas etapas seguintes.

5.4 Teste 4 — *Script* de treinamento final e modo ao vivo

O objetivo desse teste foi usar o script *train_evaluate.py* para treinar o modelo, salvá-lo em disco como *modelo_bert_correcao.pth* e ativar o "Modo de Correção ao Vivo" no terminal.

Após esse teste constatou-se que o treinamento foi realizado com sucesso. No terminal aparecia um campo para inserir a resposta dos alunos e verificar se estariam sendo avaliados de forma correta ou incorreta pela inteligência artificial.

Por meio desse teste, verificou-se que, mesmo a acurácia sendo menor do que 0,1% na avaliação feita pelo algoritmo de cálculo de *Loss*, houve erros na correção da Inteligência Artificial. Respostas que deveriam ser consideradas incorretas eram avaliadas como corretas por terem muitas palavras semelhantes à resposta correta, mesmo permanecendo semanticamente erradas.

5.5 Teste 5 — Integração com API do Gemini

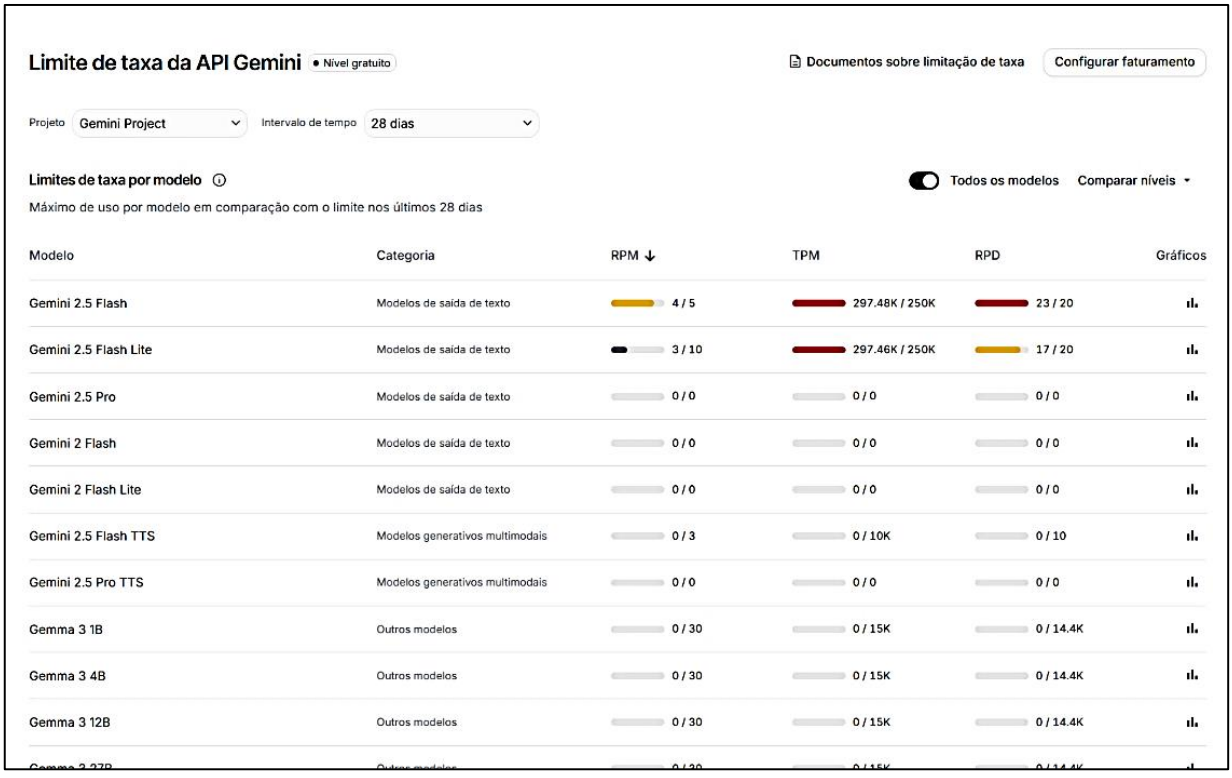
O objetivo desse teste foi integrar a *API* do *Gemini* para geração de dados sintéticos orientada a gabarito e livro-texto, e atualizar a arquitetura *BERT* para classificação multiclasse (0 = Incorreto, 1 = Parcial, 2 = Correto).

Após esse teste constatou-se erros no pré-processamento por divergência de número de colunas entre o CSV do *Forms* e o CSV gerado pelo *Gemini*. Após algumas validações, confirmou-se que não era mais necessário o pré-processamento das respostas geradas pelo *Gemini*, pois já vinham limpas para o treinamento da IA. Foram feitos testes com o modelo *gemini-2.0-flash* e, posteriormente, *gemini-2.5-flash* para

melhor desempenho. Foi necessário o fatiamento do livro utilizado para testes Tanenbaum (2021) em 231 partes de 5 páginas, devido às limitações no envio do arquivo para o *Gemini*. O fatiamento funcionou corretamente via biblioteca *PyMuPDF* e os dados sintéticos foram gerados com sucesso para ambas as questões cadastradas.

Por meio desse teste, verificou-se que o *pipeline* de geração de dados com Gemini foi validado. Entretanto o consumo excessivo de *tokens* da API do *Gemini* gratuita (*429 RESOURCE_EXHAUSTED*) motivou a implementação da arquitetura RAG para diminuir o uso de *tokens*. Os dados gerados passaram a ter uma estrutura de 3 classes, e a arquitetura BERT foi atualizada para usar *CrossEntropyLoss* com 3 neurônios de saída para a avaliação de correto, parcialmente correto e incorreto.

Figura 9 – Site da API Gemini para verificação do consumo de *tokens*



Fonte: Captura de tela do site Google AI Studio (2026)

5.6 Teste 6 — Implementação do RAG local e geração estável de dados

O objetivo desse teste foi substituir o envio do PDF completo para a API por um sistema de recuperação local (RAG com TF-IDF), eliminando erros de *timeout* e cota da API gratuita.

Após esse teste constatou-se que o *script RAG* indexou o livro Tanenbaum (2021) em 231 blocos de texto (5 páginas por *chunk*). Para a Questão 1 “Qual a finalidade da camada de enlace de dados do modelo OSI?”, o RAG identificou corretamente as páginas mais relevantes [126 a 130], [131 a 135], [136 a 140]. A questão foi processada com sucesso e o *dataset* final foi gerado com 62 exemplos salvos em *data/raw/Respostas_Treino_Geradas.csv*.

Por meio desse teste, verificou-se que o RAG local resolveu os problemas de *tokens* em aproximadamente 95% por consulta (caindo de 297k *tokens* para cerca de 11k por execução). Também foi implementado um algoritmo com tentativas múltiplas de conexão com o servidor da API para evitar o erro de *timeout (503).* O tempo de resposta por questão caiu para segundos ao invés de minutos, devido a diminuição de informações enviadas para a geração de dados do Gemini. O *dataset* gerado continha exemplos pareados com Pergunta + Resposta + Classe, permitindo o treinamento. A implementação foi bem-sucedida e o *pipeline* se tornou estável.

5.7 Teste 7 — Treinamento com Injeção de Contexto e Correção em Lote

O objetivo desse teste foi treinar o modelo BERT com o *dataset* gerado pelo *RAG*, usando a concatenação Pergunta+Resposta para que o modelo aprenda a associar a resposta ao contexto da pergunta, validando também a correção em lote via *CSV*.

Após esse teste constatou-se que o treinamento com 50 épocas convergiu com sucesso. O módulo de correção em lote foi testado com arquivos *CSV* exportados do *Microsoft Forms* e o modelo foi capaz de diferenciar respostas da mesma turma para perguntas diferentes. O *Active Learning* (via *st.data_editor*) permitiu que divergências entre a nota da IA e a nota do professor fossem capturadas e recolocadas no *dataset* para re-treinamento imediato.

Por meio desse teste, verificou-se que a injeção do *token* resolveu o problema de "cegueira de contexto" — sem ela, o modelo avaliava apenas o texto da resposta sem considerar a qual pergunta ela pertencia. O sistema de *Active Learning* mostrou resultados promissores ao adaptar o modelo ao vocabulário real dos alunos, sem depender exclusivamente de dados sintéticos.

5.8 Teste 8 — Versão Final: Aplicação Web com Streamlit

O objetivo desse teste foi unificar todo o pipeline (extração de PDF, geração via Gemini, treinamento do BERT e correção) em uma única aplicação *web* acessível pelo navegador, com *interface* para o professor preencher questão, gabarito e realizar o *upload* do livro e capítulo, além de acessar os módulos de correção manual e em lote.

Após esse teste constatou-se que a aplicação foi modularizada em 4 arquivos (*app.py*, *pdf_processor.py*, *gemini_service.py*, *model_handler.py*) com as pastas *data/* e *models/*. O *Streamlit* subiu com sucesso. Contudo, alguns erros finais surgiram e foram solucionados em sequência:

- *ImportError: cannot import name 'BertTokenizer'* (solucionado forçando a reinstalação com *pip install --upgrade --force-reinstall transformers tokenizers)*;
- *ModuleNotFoundError*: *No module named 'torchvision'* (avisos inofensivos do *file watcher do Streamlit*, solucionados com *pip install torchvision)*;
- *JSONDecodeError*: *Extra data* (solucionado com a extração via *Regex re.search(r'$$.*$$', texto_bruto, re.DOTALL))*;
- *503 UNAVAILABLE* (solucionado com mecanismo de *Retry* automático usando *time.sleep(15)* entre tentativas).

Por meio desse teste, verificou-se que a versão final consolidou todas as funcionalidades: configuração da questão com gabarito e livro, geração de dados sintéticos contextualizados, treinamento sob demanda com barra de progresso visual, correção manual em tempo real, correção em lote com *download* do boletim, revisão pelo professor via tabela editável com re-treinamento via *Active Learning*, e o botão "Nova Questão" capaz de limpar todos os arquivos temporários.

5.9 Comentário de Resultados

Esta seção apresenta o comportamento observado do sistema durante sua execução real, ilustrando cada etapa do fluxo de uso por meio de capturas de tela obtidas em ambiente de desenvolvimento local (*localhost:8501*). As Figuras 10, 11, 12 e 13 mostram a progressão do professor desde a configuração inicial da questão até a obtenção do boletim final, passando pelo ciclo de *Active Learning*.

5.9.1 Tela Inicial: Preparação da Correção

A Figura 10 apresenta a tela inicial da aplicação em seu estado vazio, imediatamente após a inicialização pelo comando *python -m streamlit run app.py*. O painel lateral esquerdo exibe o botão “Nova Questão (Resetar)”, responsável por limpar todos os arquivos temporários da sessão anterior e reiniciar o sistema para uma nova configuração. A área principal apresenta o formulário de preparação da correção, composto por quatro campos: enunciado da questão, gabarito oficial, *upload* do livro-texto em PDF e intervalo de páginas correspondente ao capítulo relevante. O botão “Confirmar e Iniciar Treinamento” aciona o *pipeline* completo após o preenchimento de todos os campos.

Figura 10 - Tela inicial: formulário “Preparação da Correção” vazio

Fonte: Capturado pelo autor desse trabalho

5.9.2 Execução do Pipeline: Treinamento em Andamento

A Figura 11 ilustra o sistema após o professor preencher todos os campos e acionar o botão “Confirmar e Iniciar Treinamento”. O formulário exibe a questão de teste utilizada — “Qual a finalidade da camada de enlace de dados do modelo OSI?” — com seu gabarito e o livro Tanenbaum (2021) carregado (32,3 MB), no intervalo de páginas 284–365. Na parte inferior, o componente de *status* exibe o progresso em três etapas sequenciais: 1/3 Recortando o PDF (Módulo 1), 2/3 Gerando respostas sintéticas via Gemini (Módulo 2) e 3/3 Treinando a Rede Neural BERT (Módulo 3), acompanhado por uma barra de progresso azul que avança proporcional às épocas de treinamento concluídas. Essa retroalimentação visual foi implementada por meio do componente *st.progress* do *Streamlit*, integrado ao laço de treinamento do *model_handler.py*.

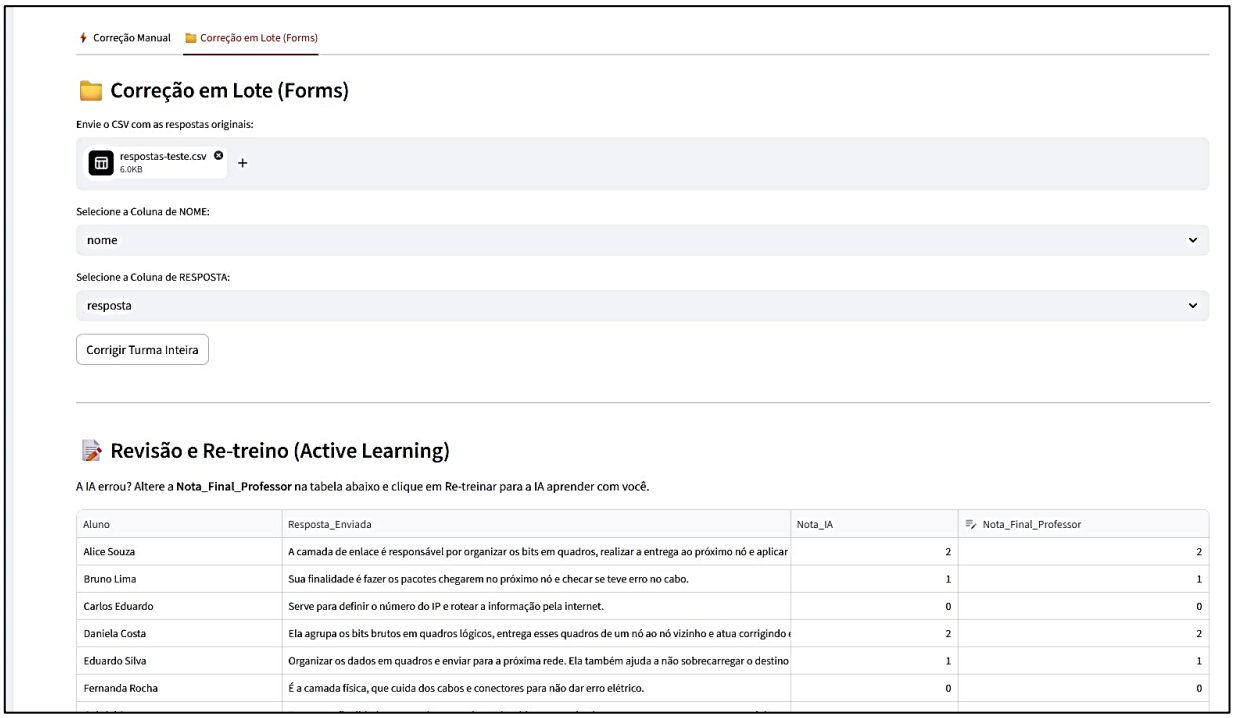
Figura 11 - Pipeline em execução: três etapas de processamento com barra de progresso

Fonte: Capturado pelo autor desse trabalho

5.9.3 Correção em Lote via CSV

A Figura 12 apresenta a aba “Correção em Lote (Forms)”, acessível após a conclusão do treinamento. O professor realiza o *upload* da planilha exportada pelo Microsoft Forms (respostas-teste.csv, 6,0 KB), seleciona por meio de menus suspensos a coluna correspondente ao nome do aluno (“nome”) e a coluna contendo as respostas (“resposta”), e clica em “Corrigir Turma Inteira”. Após o processamento, o sistema exibe o boletim gerado pela IA na seção “Revisão e Re-treino (*Active Learning*)”. A tabela apresenta, para cada aluno, a resposta enviada e a nota atribuída pela rede neural (Nota_IA). Observa-se que as notas variam entre 0 (Incorreto), 1 (Parcial) e 2 (Correto), distribuindo os alunos em três níveis de desempenho. A coluna “Nota_Final_Professor” permanece editável para a etapa de curadoria.

Figura 12 - Correção em lote: *upload* da planilha, seleção de colunas e boletim gerado



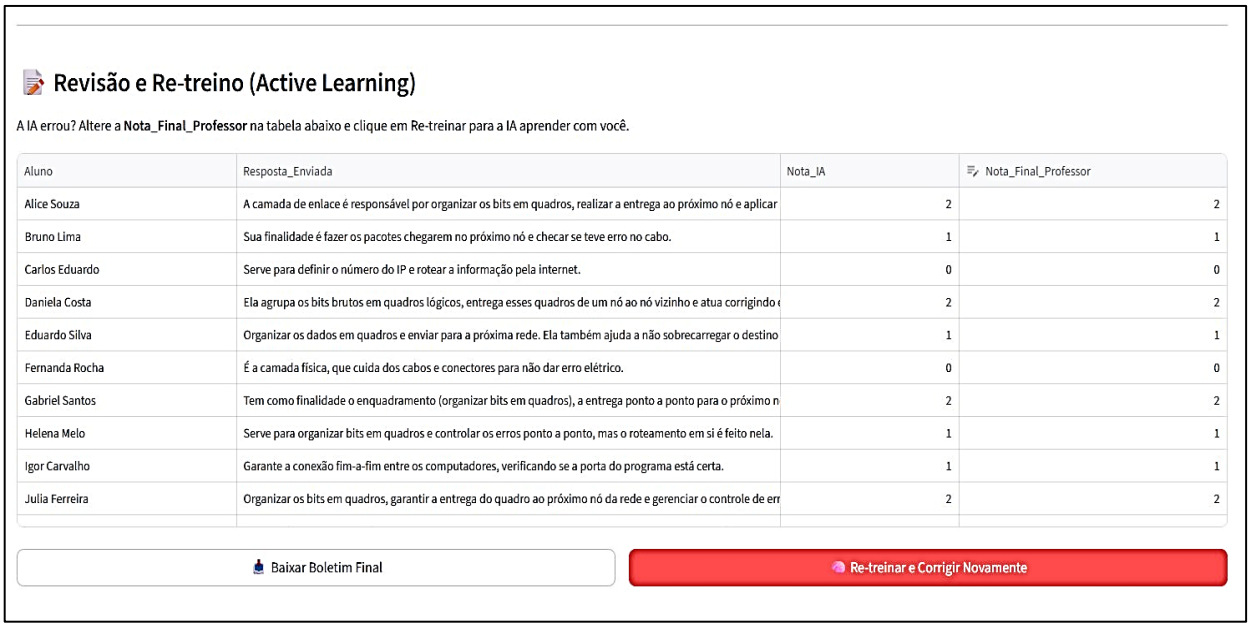
Aluno	Resposta_Enviada	Nota_IA	Nota_Final_Professor
Alice Souza	A camada de enlace é responsável por organizar os bits em quadros, realizar a entrega ao próximo nó e aplicar	2	2
Bruno Lima	Sua finalidade é fazer os pacotes chegarem no próximo nó e checar se teve erro no cabo.	1	1
Carlos Eduardo	Serve para definir o número do IP e rotear a informação pela internet.	0	0
Daniela Costa	Ela agrupa os bits brutos em quadros lógicos, entrega esses quadros de um nó ao nó vizinho e atua corrigindo	2	2
Eduardo Silva	Organizar os dados em quadros e enviar para a próxima rede. Ela também ajuda a não sobrecarregar o destino	1	1
Fernanda Rocha	É a camada física, que cuida dos cabos e conectores para não dar erro elétrico.	0	0

Fonte: Capturado pelo autor desse trabalho

5.9.4 Tabela de Revisão e Ciclo de Active Learning

A Figura 13 detalha a tabela de revisão renderizada pelo componente *st.data_editor* do *Streamlit*. A tabela exibe dez alunos da turma de teste, com suas respectivas respostas e notas atribuídas pela IA. Observa-se que o modelo atribuiu nota 2 (Correto) para Alice Souza, cujo texto aborda explicitamente a organização de *bits* em quadros, a entrega ao próximo nó e o controle de erros; nota 1 (Parcial) para Bruno Lima, que menciona apenas parte das funções da camada; e nota 0 (Incorreto) para Carlos Eduardo, que descreve funções da camada de rede em vez da camada de enlace. Esse padrão de distribuição é coerente com o gabarito configurado, demonstrando que o modelo aprendeu a distinguir respostas semanticamente corretas das parciais e das confundidas com outras camadas do modelo OSI. Os botões “Baixar Boletim Final” e “Re-treinar e Corrigir Novamente” encerram o ciclo: o primeiro exporta o CSV final; o segundo aciona o *reinjection* das divergências corrigidas pelo professor e inicia um novo ciclo de *fine-tuning*.

Figura 13 Tabela de revisão do *Active Learning*: notas da IA e coluna editável pelo professor



Fonte: Elaborado pelo autor desse trabalho

5.9.5 Análise Geral dos Resultados

Os resultados observados nas capturas de tela mostram que o sistema cumpre seu objetivo central: oferecer ao professor uma ferramenta funcional que automatiza a correção de respostas discursivas de forma contextualizada, sem exigir do usuário qualquer conhecimento técnico em aprendizado de máquina. O fluxo de interação é linear e intuitivo — formulário, espera pelo treinamento, *upload* das respostas, obtenção do boletim — e o tempo total de execução, incluindo a geração de dados sintéticos e o treinamento por 15 épocas, ficou na ordem de 2 a 5 minutos em *hardware* de consumo.

O Teste 3 comparou treinamentos com 20 e 15 épocas sobre 50 respostas sintéticas. Ambas as configurações atingiram acurácia próxima de 100% no conjunto de dados sintéticos, com Loss final abaixo de 0,1. A diferença de desempenho entre as duas configurações foi negligenciável, ao passo que 15 épocas reduziram o tempo de treinamento em aproximadamente 25%, resultando em uma experiência mais ágil para o professor na interface Streamlit. Essa foi a razão pela qual 15 épocas foram adotadas como padrão a partir do Teste 4. Foi feito um teste com 50 épocas, mas foi visto como inviável para a aplicação devido ao tempo de espera da execução ser maior do que 1 hora, resultados podem ser vistos na Tabela 1.

A distribuição das notas atribuídas pelo modelo para a turma de teste (questão sobre a camada de enlace do modelo OSI) mostrou-se semanticamente coerente: respostas que citavam as três funções centrais do gabarito (organização em quadros, entrega ao próximo nó, controle de erros) receberam nota 2; respostas que mencionavam apenas uma ou duas funções, ou que incluíam imprecisões, receberam nota 1; e respostas que confundiam a camada de enlace com a camada de rede ou a camada física receberam nota 0. Esse padrão indica que a combinação entre a geração de dados sintéticos orientada ao gabarito e o *fine-tuning* do BERTimbau com injeção de contexto via *token* [SEP] foi suficiente para que o modelo internalizasse os critérios de avaliação da questão específica configurada.

Importante destacar, contudo, que os resultados observados constituem uma prova de conceito e não uma validação estatística formal. A turma de teste utilizada foi composta por dados sintéticos gerados pelo próprio Gemini, o que pode introduzir um

viés de confirmação nos resultados. A validação com respostas reais de alunos, comparando a nota do sistema com a nota atribuída manualmente por docentes, é apontada como a principal direção de trabalho futuro para consolidar a relevância acadêmica da ferramenta.

Tabela 1 – Comparação dos dados treinados em épocas

Épocas	Dados sintéticos	Acurácia final (sintético)	Loss final	Tempo aprox.
20	50 respostas (Teste 3)	~100%	< 0,1	~4–5 min
15 ★	45 respostas (Testes 5–8)	~100%	< 0,1	~2–3 min ✓
50	45 respostas (Teste 7)	~100%	< 0,05	~7–8 min

★ valor adotado para execução do treinamento

Fonte: Elaborado pelo autor desse trabalho

6 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo o desenvolvimento e a avaliação de um sistema de Inteligência Artificial capaz de corrigir respostas discursivas de questões conceituais, utilizando técnicas de PLN e DL. A trajetória percorrida ao longo do desenvolvimento mostrou que o problema proposto é tecnicamente viável, porém envolve uma cadeia de desafios de engenharia que vão além da simples aplicação de um modelo pré-treinado.

Os objetivos iniciais foram alcançados em sua essência. O protótipo funcional desenvolvido é capaz de classificar respostas discursivas em três categorias — correta, parcialmente correta e incorreta — utilizando o modelo BERTimbau com uma camada classificadora MLP acoplada, treinada sob demanda para cada questão específica.

A metodologia adotada mostrou-se adequada ao problema: a combinação de *Prompt Engineering* para geração de dados sintéticos, RAG local para recuperação de contexto do livro-texto e injeção do *token* [SEP] para o aprendizado contextualizado produziu um pipeline estável e reproduzível.

Os resultados obtidos são coerentes com o esperado para um protótipo acadêmico: a acurácia próxima de 100% verificada sobre dados sintéticos representa uma prova de conceito bem-sucedida da arquitetura, ainda que a validação sobre respostas reais de alunos em larga escala permaneça como trabalho futuro.

As principais dificuldades encontradas no desenvolvimento foram de ordem infraestrutural e de integração. O ambiente Google Cloud Shell revelou-se inadequado para o treinamento de modelos de DL por limitações de armazenamento em disco, exigindo a migração do projeto para um ambiente local. A integração com a API do Gemini apresentou sucessivos obstáculos — erros de cota de *tokens*, *timeouts* e inconsistências no formato de saída — que foram solucionados progressivamente por meio da implementação do RAG local, do mecanismo de *retry* automático e da extração de JSON por expressões regulares. Cada uma dessas dificuldades gerou uma correção de rota que aprimorou a robustez do sistema final.

6.1 Comparação com trabalhos semelhantes

Sistemas de correção automática de respostas discursivas, conhecidos na literatura como *Automated Short Answer Grading*, Avaliação Automática de Respostas Curtas (ASAG), têm sido explorados academicamente com diferentes abordagens. Outros trabalhos frequentemente dependem de grandes conjuntos de dados manualmente rotulados para o treinamento, o que representa uma barreira prática para adoção em contextos institucionais específicos (BURROWS; GUREVYCH; STEIN, 2015).

A principal diferença deste trabalho em relação a abordagens de outros trabalhos reside no mecanismo de geração de dados sintéticos orientada a gabarito e livro-texto, que elimina a necessidade de uma base de dados prévia, tornando o sistema adaptável a qualquer disciplina e professor sem etapas manuais de rotulagem.

6.2 Limitações da solução

O sistema apresenta limitações que devem ser reconhecidas. A qualidade das respostas sintéticas geradas pelo Gemini é diretamente dependente da qualidade do gabarito fornecido pelo professor e da riqueza do trecho do livro utilizado como contexto: gabaritos vagos tendem a produzir dados de treinamento pouco discriminativos. Além disso, o modelo treinado sobre dados sintéticos pode não capturar a diversidade vocabular real dos alunos, especialmente erros de digitação e construções coloquiais, problema mitigado, mas não eliminado, pelo mecanismo de *Active Learning*. A dependência de conexão com a *Internet* para a etapa de geração de dados via API é outra restrição operacional relevante. E a dificuldade da correção de mais uma questão de cada vez pela limitação do modelo BERT.

Por fim, um aspecto a ser considerado no uso contínuo da ferramenta é o efeito da alimentação retroativa (*feedback loop*) gerado pelo mecanismo de *Active Learning*. Embora a reinjeção das notas corrigidas pelo professor seja o motor para a evolução do sistema, esse processo cria uma dependência direta da precisão humana. Se o docente realizar curadorias apressadas, inconsistentes ou com vieses de avaliação, a alimentação retroativa forçará o modelo a internalizar e reproduzir esses mesmos erros nas correções futuras. Portanto, a integridade da base de conhecimento do sistema a

longo prazo exige um rigor constante na revisão manual, evidenciando que a Inteligência Artificial não substitui o critério pedagógico humano, mas atua como um reflexo de sua qualidade.

6.3 Importância e contribuições

Os resultados obtidos mostram que é possível construir, com ferramentas de código aberto e *hardware* acessível, um sistema de suporte à correção de avaliações que pode reduzir o tempo dedicado pelo professor a essa tarefa administrativa. A arquitetura modular desenvolvida — dividida entre os módulos de processamento de PDF, geração de dados, treinamento de rede neural e interface *web* constitui uma contribuição prática e replicável para a comunidade acadêmica. O ciclo de *Active Learning* implementado representa, adicionalmente, uma contribuição conceitual relevante: ao incorporar as correções do professor como novos dados de treinamento, o sistema evolui continuamente, aproximando-se progressivamente do padrão avaliativo do docente responsável.

6.4 Sugestões de trabalhos futuros

Com base nas limitações identificadas e nas possibilidades abertas por este protótipo, três direções de investigação se mostram promissoras.

1. A validação do sistema com dados reais de avaliações aplicadas em turmas regulares, comparando estatisticamente as notas atribuídas pela IA com as notas dos professores para medir a concordância e identificar os tipos de resposta em que o modelo mais erra;
2. A substituição do *BERTimbau* por modelos de linguagem de maior porte, como variantes do GPT ou modelos da família *Llama* adaptados ao português, avaliando o impacto no desempenho classificatório em cenários de poucos dados de treinamento;
3. A extensão do sistema para suportar múltiplas questões em uma única sessão de treinamento, permitindo que o professor configure uma prova completa e obtenha o boletim de toda a turma em uma única execução, além da

exploração de mecanismos de explicabilidade que apresentem ao professor não apenas a nota sugerida, mas os trechos da resposta do aluno que mais influenciaram a decisão do modelo.

REFERÊNCIAS

ALDOSARI, Share Aiyed. The future of higher education in the light of artificial intelligence transformations. **International Journal of Higher Education**, v. 9, n. 3, p. 145-151, 2020.

ARXER, E. A.; CRUZ, J. A. S.; CUNHA, A. K.; BIZELLI, J. L. Plataforma Geekie: uma opção para avaliar e mapear conhecimentos dos alunos. **Tecnologia Educacional** [on line], Rio de Janeiro, n. 216, p. 118-126, 2017.ISSN: 0102-5503. Disponível em: http://abt-br.org.br/wp-content/uploads/2017/08/216.pdf#page=118. Acesso em: 28 set. 2025.

BARBOSA, X.; BEZERRA, R**.** Breve Introdução à história da Inteligência artificial. **Jamaxi**, v. 4, n. 2, 2020.

BROWN, A.; MILLER, B. Machine Learning: What It Is and Why It Matters. **Harvard Data Science Review**, v. 1, n. 1, 2019. Disponível em: https://hdsr.mitpress.mit.edu/. Acesso em: 10 set. 2025.

BURROWS, Steven; GUREVYCH, Iryna; STEIN, Benno. **The eras and trends of automatic short answer grading.** International Journal of Artificial Intelligence in Education, v. 25, n. 1, p. 60–117, 2015. Disponível em: https://doi.org/10.1007/s40593-014-0026-8. Acesso em: 10 mar. 2026.

CHASSIGNOL, Maud et al. Artificial Intelligence trends in education: a narrative overview. **Procedia Computer Science**, v. 136, p. 16-24, 2018.

CHEN, B.; LIU, H.; ZHANG, J. Integrating artificial intelligence into educational technology research and development. **Educational Technology Research and Development**, v. 68, p. 1–14, 2020.

COSTA JÚNIOR, J. F. et al. O futuro da aprendizagem com a inteligência artificial aplicada à educação 4.0. **Revista Científica Multidisciplinar**, V. 07, N.14, p. 1-28, Jul./Dez. 2023.

CÂNDIDO, P. L. de O.; CASILLO, L. A. **A importância do ensino da programação nas escolas.** Trabalho de Conclusão de Curso (Licenciatura em Computação) - Núcleo de Educação a Distância (NEaD), Universidade Federal Rural do SemiÁrido (Ufersa), Mossoró, RN, 2017. Disponível em: https://repositorio.ufersa.edu.br/handle/prefix/8375. Acesso em 2 set. 2025.

DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**. In: Proceedings of NAACL-HLT 2019. Minneapolis: Association for Computational Linguistics, 2019. p. 4171–4186. Disponível em: https://arxiv.org/abs/1810.04805. Acesso em: 10 mar. 2026.

DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**. In: Proceedings of NAACL-HLT 2019. Minneapolis: Association for Computational Linguistics, 2019. p. 4171–4186. Disponível em: https://arxiv.org/abs/1810.04805. Acesso em: 10 mar. 2026.

DORES, A. R. das . et al. Aplicação da IA na educação: proposta de um projeto ou utilização de chatbot como sistema de tutorial aplicado em um AVA. **Revista InovaEduc**, n. 7, p. 1–16, 2021.

FELTRIN, F. **Python Total**: Deep Learning e Visão Computacional. [S.l.]: Fernando Feltrin, 2021

FIGUEIREDO, L. de O. et al. Desafios e impactos do uso da Inteligência Artificial na educação. **Revista Educação Online**, Rio de Janeiro, v. 18, n.44, p. 1-22, set.-dez. 2023.

FREITAS, Clayton Alencar de; PEREIRA, Leonardo Gonçalves; NASCIMENTO, Fernanda Machado do; ALBUQUERQUE, Maria Analice de Araujo; ARAUJO, Maria Ila de. Impacto da inteligência artificial na avaliação acadêmica: transformando métodos tradicionais de avaliação no ensino superior. **Revista Ibero-Americana de Humanidades, Ciências e Educação**, *[S. l.]*, v. 11, n. 1, p. 2736–2752, 2025. DOI: 10.51891/rease.v11i1.18011. Disponível em: https://periodicorease.pro.br/rease/article/view/18011. Acesso em: 19 out. 2025.

GIRAFFA, Lucia; KOHLS-SANTOS, Pricila. Inteligência Artificial e Educação: conceitos, aplicações e implicações no fazer docente. **Educação em Análise**, Londrina, v. 8, n. 1, p. 116–134, 2023. DOI: 10.5433/1984-7939.2023v8n1p116. Disponível em: https://ojs.uel.br/revistas/uel/index.php/educanalise/article/view/48127. Acesso em: 1 nov. 2025.

GOMES, D**.** Inteligência Artificial: conceitos e aplicações. **Revista Olhar Científico**, v. 1, n. 2, p. 234-246, ago./dez. 2010. Disponível em: https://www.professores.uff.br/screspo/wp-content/uploads/sites/127/2017/09/ia_intro.pdf. Acesso em: 2 nov. 2025.

GOOGLE. **Google AI Studio**. Disponível em: https://aistudio.google.com/. Acesso em: 10 mar. 2026.

GÉRON, A. **Mãos à Obra**: Aprendizado de Máquina com Scikit-Learn & TensorFlow: Conceitos, Ferramentas e Técnicas para a Construção de Sistemas Inteligentes. Tradução Rafael Contatori. Rio de Janeiro: Alta Books, 2019.

HINTON, G. E.; OSINDERO, S.; THE, Y-W. A Fast Learning Algorithm for Deep Belief Nets. **Neural Computation**, v. 18, n. 7, p. 1527-54, 2006.

KÜHL, N.; GOUTIER, M.; HIRT, R.; SATZGER, G. **Machine Learning in Artificial Intelligence: Towards a Common Understanding**. In: Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS 2019), Honolulu, HI, USA, Jan. 8-11, 2019.

LEE, Kai-Fu**. Inteligência artificial**: como os robôs estão mudando o mundo, a forma como amamos, nos relacionamos, trabalhamos e vivemos. Tradução Marcelo Barbão. Rio de Janeiro: Globo Livros, 2019.

LEWIS, Patrick et al. **Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks**. Advances in Neural Information Processing Systems (NeurIPS), v. 33, p. 9459–9474, 2020. Disponível em: https://arxiv.org/abs/2005.11401. Acesso em: 10 mar. 2026.

LIMA, G. de M.; FERREIRA, G. M. dos S.; CARVALHO, J. de S. Automação na educação: caminhos da discussão sobre a inteligência artificial. **Educ. Pesqui**., São Paulo, v. 50, e273857, 2024.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning Representations by Back-propagating Errors. **Nature**, v. 323, n. 6088, p. 533-536, 1986.

SANT´ANA, F. P.; SANT´ANA, I. P.; SANT´ANA, C. de C. Uma utilização do Chat GPT no ensino. **Com a Palavra, o Professor**, v. 8, n. 20, p. 74–86, 2023.

SEREN, M.; OZCAN, Z. E. Post pandemic education: distance education to artificial intelligence based education. **International Journal of Curriculum and Instruction**, [S. l.], v. 13, n. 1, p. 212-225, 2021.

SILVA, I. **Inteligência Artificial 101**: Explorando a Inteligência Artificial: definições, vantagens, desafios e perspectivas. [S. l.: s. n.], 2023. E-book.

SILVA, V. L. da. **Ética e responsabilidade na era da Inteligência Artificial**: aprendizagem digital no Chat GPT. Monografia (Especialização em Mídia e Educação) - Universidade Aberta do Brasil, Campus São Borja (UFP/UAB), São Borja/RS, 2023.

SOUZA, Fábio; NOGUEIRA, Rodrigo; LOTUFO, Roberto. **BERTimbau: Pretrained BERT Models for Brazilian Portuguese**. In: CERRI, R.; PRATI, R. C. (Ed.). Intelligent Systems. BRACIS 2020. Lecture Notes in Computer Science, v. 12319. Cham: Springer, 2020. p. 403–417. Disponível em: https://dl.acm.org/doi/10.1007/978-3-030-61377-8_28. Acesso em: 10 mar. 2026.

SOUZA, Fábio; NOGUEIRA, Rodrigo; LOTUFO, Roberto. **BERTimbau: Pretrained BERT Models for Brazilian Portuguese**. In: CERRI, R.; PRATI, R. C. (Ed.). Intelligent Systems. BRACIS 2020. Lecture Notes in Computer Science, v. 12319. Cham: Springer, 2020. p. 403–417. Disponível em: https://dl.acm.org/doi/10.1007/978-3-030-61377-8_28. Acesso em: 10 mar. 2026.

SOUZA, S. I. N. de. Responsabilidade civil e a inteligência artificial nos contratos eletrônicos na sociedade da informação. **Revista dos Tribunais**, São Paulo, v. 877, ano 97, p. 33-34, 2008.

SUNAGA, **A. 4 usos da IA na educação 5.0**. 2023a. Disponível em: https://alexsandrosunaga.com.br/2023/05/19/4-usos-da-ia-na-educacao/. Acesso em: 01 jun. 2023

TANENBAUM, A. S.; FEAMSTER, N.; WETHERALL, D. J. **Redes de computadores**. 6. ed. São Paulo: Pearson, 2021.

TOKARNIA, Mariana. Professores no Brasil usam mais IA que média dos países da OCDE. **Agência Brasil**, Rio de Janeiro, 6 out. 2025. Disponível em: https://agenciabrasil.ebc.com.br/educacao/noticia/2025-10/professores-no-brasil-usam-mais-ia-que-media-dos-paises-da-ocde#:~:text=Professores%20no%20Brasil%20usam%20mais,pa%C3%ADses%20da%20OCDE%20%7C%20Ag%C3%AAncia%20Brasil. Acesso em: 4 nov. 2025.

TRONCO, G. B**.** ChatGPT impacta rotinas na pesquisa e na educação e levanta questionamentos sobre veracidade e metodologias de avaliação. **Jornal da Universidade**, Porto Alegre, 13 de abril de 2023.

VICARI, R. M. **Influências das Tecnologias da Inteligência Artificial no ensino**. *Estudos Avançados*, São Paulo, v. 35, n. 101, p. 73-84, 2021.

ZHANG, Aston et al. **Dive into Deep Learning**. Cambridge: Cambridge University Press, 2021. Disponível em: https://d2l.ai. Acesso em: 18 ago. 2025.

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
GABINETE DO REITOR

Av. Universitária, 1069 ● Setor Universitário
Caixa Postal 86 ● CEP 74605-010
Goiânia ● Goiás ● Brasil
Fone: (62) 3946.1000
www.pucgoias.edu.br ● reitoria@pucgoias.edu.br

RESOLUÇÃO n° 038/2020 – CEPE

ANEXO I

APÊNDICE ao TCC

Termo de autorização de publicação de produção acadêmica

O(A) estudante Diogo Lopes Gomes
do Curso de Ciência da Computação, matrícula 2021.2.0028.0008-5,
telefone: _____ e-mail 20212002800085@pucgo.edu.br, na qualidade de titular dos direitos autorais, em consonância com a Lei nº 9.610/98 (Lei dos Direitos do autor), autoriza a Pontifícia Universidade Católica de Goiás (PUC Goiás) a disponibilizar o Trabalho de Conclusão de Curso intitulado Desenvolvimento de IA para correção de avaliações de questões subjetivas
_____, gratuitamente, sem ressarcimento dos direitos autorais, por 5 (cinco) anos, conforme permissões do documento, em meio eletrônico, na rede mundial de computadores, no formato especificado (Texto (PDF); Imagem (GIF ou JPEG); Som (WAVE, MPEG, AIFF, SND); Vídeo (MPEG, MWV, AVI, QT); outros, específicos da área; para fins de leitura e/ou impressão pela internet, a título de divulgação da produção científica gerada nos cursos de graduação da PUC Goiás.

Goiânia, 17 de junho de 2026.

Assinatura do(s) autor(es): _____

birdID
Documento assinado digitalmente por:
DIOGO LOPES GOMES
701.437.551-29
2026-06-17T07:56:26.307327

Nome completo do autor: Diogo Lopes Gomes

Assinatura do professor-orientador: _____

gov.br
Documento assinado digitalmente
ANGELICA DA SILVA NUNES
Data: 17/06/2026 07:54:45-0300
Verifique em https://validar.iti.gov.br

Nome completo do professor-orientador: Angélica da Silva Nunes