Feedback Formativo Automatizado com LLMs: Desenvolvimento e Análise de um Sistema para Aprendizagem Progressiva em Programação

Francisco Genivan Silva[1,2], **Eduardo Henrique da Silva Aranha**[1]

[1]Universidade Federal do Rio Grande do Norte
Departamento de Informática e Matemática Aplicada

[2]Instituto Federal do Rio Grande do Norte - Campus Currais Novos

genivan.silva@ifrn.edu.br, eduardoaranha@dimap.ufrn.br

Abstract. *Providing high-quality formative feedback to programming students remains a key challenge, especially in large-scale educational settings. Addressing this need, the present work introduces and validates an adaptive framework grounded in large language models (LLMs) for automatic code assessment and personalized feedback generation. The solution leverages pedagogical strategies and advanced prompt engineering to foster conceptual mediation and meaningful student support. Analysis of 300 real student submissions showed 74.7% agreement with traditional autograders and 93.3% of feedbacks rated as qualitatively coherent, highlighting the potential to enhance learning outcomes and expand innovative assessment practices.*

Resumo. *Oferecer feedback formativo de qualidade a estudantes de programação é um dos principais desafios educacionais, sobretudo em ambientes com grande número de participantes. A partir desse contexto, este trabalho apresenta o desenvolvimento e a validação de um framework adaptativo baseado em grandes modelos de linguagem (LLMs) para avaliação automática de código e geração de feedback personalizado. A ferramenta emprega estratégias pedagógicas e técnicas avançadas de engenharia de prompt para promover mediação conceitual e apoio ao estudante. A análise de 300 tentativas reais evidenciou 74,7% de concordância com autograders tradicionais e 93,3% de feedbacks qualitativamente avaliados como coerentes, destacando potencial para apoiar a aprendizagem significativa e ampliar o alcance de práticas avaliativas inovadoras.*

1. Introdução

O ensino de programação é central na formação em Computação, mas permanece um desafio pedagógico crítico em cursos introdutórios. Estudos documentam altas taxas de evasão e reprovação associadas a dificuldades na compreensão conceitual e desenvolvimento do raciocínio algorítmico [Alvim et al. 2024, Medeiros et al. 2019]. Além das barreiras técnicas, ambientes massivos e o excesso de atividades "extra-classe" sobrecarregam docentes, inviabilizando feedback individualizado [Teusner et al. 2018], fator considerado essencial para aprendizagem efetiva.

DOI: 10.5753/sbie.2025.12807

Autograders tradicionais, apesar de úteis para avaliação funcional e detecção rápida de erros [Keuning et al. 2018], mostram-se insuficientes para desenvolver competências complexas como depuração e boas práticas [Zawacki-Richter et al. 2019, Cheng et al. 2023].

O avanço dos Large Language Models (LLMs) viabilizou tutores capazes de interpretar código e gerar explicações adaptativas [Bassner et al. 2024], abrindo perspectivas para feedback formativo em escala. Persistem, contudo, desafios como feedback excessivamente direto (respostas prontas) ou orientações genéricas [Krupp et al. 2024, Lampou 2023].

Diante desses desafios e das limitações das abordagens tradicionais de feedback automatizado, propomos um framework inovador para avaliação adaptativa de código, fundamentado na Teoria da Aprendizagem Significativa de Ausubel [Ausubel 1963]. A proposta distingue-se por: (i) usar o conhecimento prévio do estudante como âncora para feedback contextualizado; (ii) estruturar o apoio pedagógico de forma progressiva, diferenciando pelo histórico de interações; e (iii) priorizar a construção de significado conceitual e o desenvolvimento de competências metacognitivas, em detrimento de correções superficiais. Inspirado em perspectivas socioconstrutivistas [Vygotsky 1978] e em princípios de autorregulação da aprendizagem [Zimmerman 2002], o framework preserva a autonomia estudantil e favorece a participação ativa e reflexiva na aprendizagem. Sua arquitetura contempla acompanhamento longitudinal do histórico de tentativas, personalização via engenharia avançada de *prompts*, detecção automatizada de padrões de erro e adaptação dinâmica do detalhamento do feedback, promovendo intervenções alinhadas ao desenvolvimento conceitual.

Para avaliar sua efetividade, investigamos:

QP1: A avaliação por LLMs pode substituir autograders tradicionais com precisão equivalente?
QP2: Os feedbacks gerados apresentam qualidade e coerência para apoio à aprendizagem?
QP3: Qual o impacto do framework nos custos operacionais em larga escala?

O artigo estrutura-se em: trabalhos relacionados (Seção 2); arquitetura do framework (Seção 3); metodologia (Seção 4); resultados (Seção 5); discussão (Seção 6); limitações e trabalhos futuros (Seção 7); e conclusões (Seção 8).

2. Trabalhos relacionados

Trabalhos recentes indicam que grande parte dos sistemas de avaliação automatizada, frequentemente baseados em julgamentos binários (certo/errado), ainda oferece pouco estímulo à reflexão, escassa mediação formativa e limitada detecção de padrões persistentes de erro ou escalonamento de ajuda [Keuning et al. 2018, Messer et al. 2023]. Embora úteis para verificação de correção, os autograders seguem mostrando limitações significativas no apoio a iniciantes, sobretudo quando o objetivo é promover a compreensão conceitual [Paiva et al. 2022, Leite and Blanco 2020].

Em resposta a essas limitações, a literatura tem recomendado explicitamente mecanismos de feedback progressivo, estratégias para identificar recorrências e dispositivos de regulação pedagógica ao longo do tempo [Schwerter et al. 2022, Gill et al. 2024]. Esta

evolução reflete-se na trajetória da automação da avaliação de código, que tem migrado de soluções iniciais baseadas em testes unitários e detecção de plágio [Ihantola et al. 2010, Ala-Mutka 2005] para abordagens mais sofisticadas que incorporam análise semântica e geração textual de feedback, culminando recentemente na exploração do uso de LLMs [Gill et al. 2024, Cheng et al. 2023].

Persistem, entretanto, desafios de alinhamento entre o feedback gerado automaticamente e as expectativas pedagógicas: imprevisibilidade dos LLMs, calibragem de *prompts* para evitar respostas vagas ou a entrega de soluções e a necessidade de combinar critérios objetivos com abordagens adaptativas que fomentem autonomia do estudante [Bassner et al. 2024, Pereira and Ferreira Mello 2025, Yu et al. 2023, Schwerter et al. 2022, Brown et al. 2020].

Nesse cenário, ferramentas representativas como Iris [Bassner et al. 2024] e CodeTailor [Cheng et al. 2023] marcaram avanços na integração de LLMs. Iris enfatiza feedback progressivo e personalização em tempo real, mas sua memória de contexto permanece restrita ao histórico imediato, sem continuidade longitudinal. CodeTailor personaliza *Parsons Puzzles* a partir de erros do aluno, favorecendo diagnósticos específicos, porém não opera como tutor adaptativo e formativo ao longo de múltiplas tentativas. Em contraste, o *framework* aqui proposto mantém uma *thread* pedagógica persistente e integra personalização, registro histórico e mediação formativa, conforme detalhado na Seção 3.

Além disso, nossa arquitetura incorpora memória longitudinal, engenharia de *prompts* restritiva, validação por pós-processamento e estratégias de *caching* para otimização de custos, alinhando-se às recomendações mais recentes para avaliação inteligente e adaptativa no ensino de programação.

3. Visão Geral e Arquitetura

Neste trabalho, *framework* designa a integração de princípios pedagógicos, arquitetura lógica, estratégias de feedback e implementação computacional que materializam a avaliação adaptativa de código. Embora exposto como API em Python (FastAPI) e consumido por uma plataforma de exercícios, o arranjo didático e o encadeamento de decisões o aproximam de um sistema de tutoria adaptativa, com potencial de aplicação em diferentes ambientes de ensino.

Arquiteturalmente, o *framework* atua como tutor virtual adaptativo: promove intervenções formativas graduais e personalizadas, ancoradas no histórico de tentativas do estudante. Operacionalmente, a plataforma envia enunciado, metadados e código; a API compõe uma *thread* pedagógica por par estudante–exercício, recupera do MongoDB o histórico previamente registrado (códigos e feedbacks), incorpora o *assistente* com regras e instruções e aciona o modelo de linguagem. A resposta retorna à API, que aplica um pós-processamento do tipo "catraca" — seleção/filtragem/descarte — antes da apresentação na interface. A Figura 1 resume esse ciclo e explicita as fronteiras de comunicação: a UI interage apenas com a API/Backend, enquanto o núcleo (thread, geração e pós-processamento) permanece desacoplado.

A geração dos feedbacks automáticos utiliza o modelo **GPT-4.1-mini**, da OpenAI, escolhido pelo equilíbrio entre desempenho, custo e capacidade de contextualização em ciclos breves de interação.

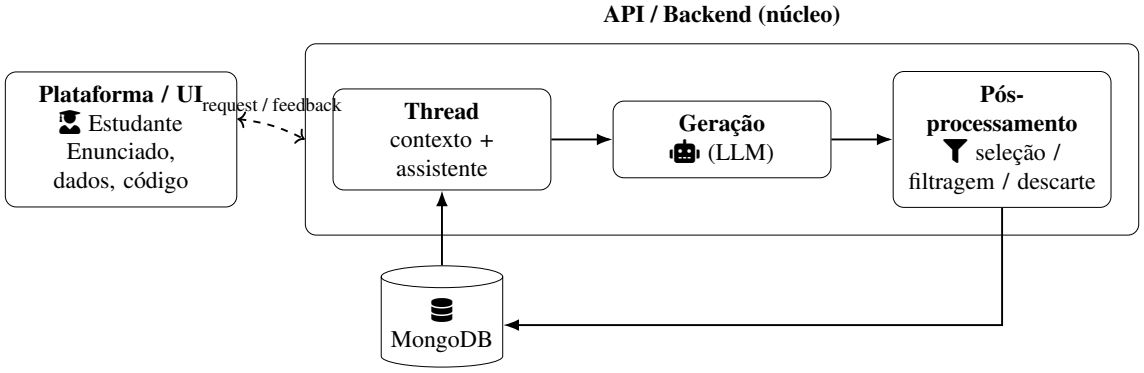

Figura 1. Arquitetura do *framework* e ciclo de feedback em alto nível.

A lógica do sistema é deliberadamente adaptativa: o feedback evolui em sintonia com o progresso do estudante. Nas primeiras tentativas, privilegia orientações conceituais; persistindo erros, aprofunda justificativas e oferece indícios mais diretos, sem franquear a solução. O acompanhamento longitudinal identifica repetições de estratégias ineficazes e aciona *intervenção diferenciada*, calibrando nível de detalhamento e ênfase pedagógica.

Outro diferencial é a checagem de relevância do feedback. Após cada resposta, o estudante sinaliza a utilidade percebida ("útil/não útil"), informação que alimenta um painel de monitoramento e subsidia revisão manual de casos-limite. Os *logs* detalhados asseguram transparência, rastreabilidade e ajustes finos na engenharia de *prompts* e na evolução do modelo pedagógico.

Por fim, a natureza agnóstica e o ciclo de realimentação contínua permitem estender o suporte formativo a diferentes arranjos curriculares e níveis de proficiência, sem sacrificar a economia operacional nem a coerência pedagógica do sistema.

3.1. Requisitos técnicos e operacionais

O *framework* foi concebido para oferecer suporte formativo adaptativo e seguro, centrado no envio de código para análise e geração automática de feedback, sem abertura para perguntas livres. Diferenciando-se de abordagens de chat, a aplicação estrutura todo o ciclo de interação como um pipeline automatizado e supervisionado, com ênfase nos seguintes requisitos:

Contextualização e registro persistente. Para cada estudante–exercício, manter uma *thread* pedagógica com enunciado, submissões, feedbacks e reações ("útil/não útil"); a cada análise, reidratar uma janela curta do histórico para preservar continuidade sem acumular ruído.

Adaptação controlada do feedback. Ajustar gradualmente o nível de detalhamento com base em evidências do histórico (p. ex., repetição de erros), preservando o foco conceitual e sem fornecer solução pronta; liberar refinamentos de estilo/boas práticas apenas após confirmação de corretude.

Filtragem e reforço pedagógico. Submeter cada saída a seleção/filtragem/descarte antes da apresentação, bloqueando trechos resolutivos, referências a linhas/variáveis e desvios de escopo; quando a indicação estruturada de corretude faltar, aplicar inferência heurística

conservadora; deduplicar solicitações semanticamente idênticas para evitar reprocessamentos.

Observabilidade e retroalimentação. Manter *logs* estruturados e registrar a utilidade percebida pelo estudante ("útil/não útil") para auditoria, monitoramento e ajustes finos do comportamento do sistema.

Resiliência e operação. Garantir validação de entradas, controle de concorrência e retentativas com *backoff* em falhas transitórias, assegurando estabilidade sob carga.

Integração e portabilidade. Preservar independência de plataforma e fronteiras claras entre UI e núcleo, facilitando a adoção em diferentes ambientes sem acoplamento indevido.

Esses requisitos permitiram construir uma solução robusta, capaz de operar sob cargas reais e registrar dados essenciais para análises de desempenho e melhoria contínua.

3.2. Engenharia de prompt e direcionamento do modelo

A qualidade da interação entre estudantes e a API decorre do uso de um assistente especializado (OpenAI) configurado com um conjunto robusto de instruções sistêmicas (*system instruction*). Diferentemente do envio isolado de *prompts* por requisição, essa configuração mantém contexto persistente e pode incorporar técnicas como *contextual prompting*, *few-shot* e *self-consistency*, o que sustenta diretrizes pedagógicas ao longo de múltiplas trocas, reduz inconsistências e melhora a adequação das respostas aos objetivos [Kasneci et al. 2023]. Como demonstrado por [Marvin et al. 2024], fornecer contexto é determinante: sem ele, LLMs tendem a respostas genéricas; com contexto insuficiente, produzem informações imprecisas; já com contexto adequado, aumentam a coerência e a relevância das saídas. Essa base de persistência contextual viabiliza a aplicação de estratégias pedagógicas mais sofisticadas, alinhando o comportamento do modelo a objetivos educacionais claros.

A configuração do assistente privilegia postura formativa, mediação ativa e aprendizagem autônoma. O *system instruction* orienta que o apoio ao estudante iniciante de programação se dê por meio de incentivo à identificação autônoma de problemas, à reflexão conceitual e ao reconhecimento de melhorias, sem entrega de soluções prontas ou indicação direta de erros. Essa abordagem combina *Socratic prompting*, *scaffolding* de dicas e verificação conceitual com casos de teste internos, além de explicações conceituais genéricas, validação algorítmica rigorosa, perguntas reflexivas e tom motivador [Ding et al. 2024, Jang et al. 2024]. O resultado é um feedback que preserva o papel ativo do aluno na resolução de problemas e previne respostas inadequadas.

Socratic prompting e scaffolding de dicas
"Você é um assistente que ajuda alunos iniciantes de programação a entender e corrigir seu código. Sua tarefa é: guiar o estudante a identificar e corrigir problemas por conta própria, verificar se o código atende ao enunciado, reconhecer oportunidades de otimização. Não apresente trechos de código nem indique diretamente onde está o erro. Sempre que identificar um problema, explique o conceito envolvido de forma didática e, ao final, proponha uma pergunta reflexiva curta para estimular a revisão crítica."

A implementação do *few-shot prompting* ocorre pela inclusão, no próprio *system instruction*, de exemplos reais e sintéticos de respostas ideais e inadequadas, calibrando a saída do modelo quanto a formato, linguagem e nível de detalhamento [Wei et al. 2022, Brown et al. 2020, Pornprasit and Tantithamthavorn 2024]. Esse recurso é amplamente reconhecido como eficaz para garantir consistência e previsibilidade no uso educacional de LLMs.

Exemplo adequado
"Seu código calcula uma soma, mas lembre-se de dividir pelo número de elementos para obter a média. Como garantir que o cálculo está correto mesmo para listas vazias?"

Exemplo inadequado
"Sua resposta está errada. Seu código não atende o enunciado e apresenta problemas com operadores matemáticos."
Motivo: resposta vaga e pouco construtiva, que não orienta o estudante nem segue o padrão esperado de clareza e mediação.

Outro aspecto central do *system instruction* são as restrições explícitas (*prompt constraints*), que vedam menção a linhas, variáveis ou fornecimento de código pronto, prevenindo *prompt injection* e preservando o caráter conceitual do feedback [Pereira and Ferreira Mello 2025, Yu et al. 2023]. A contextualização dinâmica é reforçada pela incorporação do histórico de tentativas e feedbacks no contexto das interações, permitindo respostas mais adaptativas. Em complemento, a aplicação de pós-processamento para filtrar saídas e bloquear sugestões explícitas ou qualquer desvio de escopo é uma prática já apontada como essencial para garantir segurança e aderência pedagógica em assistentes baseados em LLMs [Kazemitabaar et al. 2024].

Ao centralizar a lógica de interação no assistente, o *framework* alcança maior coerência, eficiência e controle pedagógico, reduz custos computacionais e minimiza riscos de comportamento inconsistente, aproximando-se do estado da arte no uso educacional responsável de LLMs e potencializando o impacto formativo com segurança operacional.

4. Metodologia

A abordagem metodológica deste estudo foi delineada para responder diretamente às questões de pesquisa da Seção 1, assegurando alinhamento entre objetivos investigativos e estratégias de análise de dados.

Para avaliar a qualidade e a operacionalidade do *framework*, utilizamos registros reais de disciplinas introdutórias de programação (Python) em uma plataforma educacional, preservando o encadeamento das tentativas. Os dados contemplam envios finais e salvamentos intermediários, característicos do desenvolvimento incremental e nem sempre representativos de tentativas estruturadas.

Visando relevância pedagógica, selecionou-se uma amostra estratificada de 300 tentativas, priorizando casos nos quais os estudantes consolidavam estruturas básicas de código e evidenciavam dificuldades autênticas. A amostra, composta por 100 soluções

corretas e 200 incorretas, reflete a dinâmica realista de ambientes educacionais: iniciantes frequentemente submetem múltiplas soluções erradas antes de alcançar uma implementação funcional. Essa distribuição permite tanto analisar padrões recorrentes quanto estimar métricas como acurácia, sensibilidade e especificidade.

Todas as tentativas, inclusive as feitas em blocos (Blockly), foram convertidas para Python antes do armazenamento, assegurando uniformidade dos dados.

O processo metodológico foi organizado em três eixos principais, cada um vinculado às questões de pesquisa:

- **Substituição de autograders tradicionais (QP1):** Para investigar em que medida a avaliação automatizada por LLMs pode substituir autograders, as 300 tentativas selecionadas foram submetidas simultaneamente à API baseada em LLMs e a um autograder tradicional, permitindo o cálculo da taxa de concordância, sensibilidade e especificidade.
- **Qualidade e coerência dos feedbacks (QP2):** Para avaliar a qualidade dos feedbacks, foram aplicados dois procedimentos complementares: (i) análise automatizada por classificador LLM especializado, instruído para identificar coerência e alinhamento formativo; e (ii) revisão manual de uma amostra estratificada por especialistas, que atribuíram julgamentos qualitativos de utilidade e clareza.
- **Custos operacionais (QP3):** O impacto do uso do framework sobre os custos operacionais foi mensurado por meio do registro detalhado do tempo de processamento, consumo de recursos e número de tokens utilizados em cada interação, permitindo estimar o custo efetivo da avaliação em larga escala.

Essa estrutura busca garantir que cada dimensão do problema de pesquisa seja tratada de forma rigorosa e transparente, permitindo não apenas a replicação do estudo, mas também uma análise crítica dos resultados.

4.1. Comparação com Autograder Tradicional

A plataforma utilizada neste estudo já integrava um autograder convencional, responsável por avaliar automaticamente se cada código submetido pelos estudantes passava ou não nos testes de validação. Para a análise de concordância (QP1), cada uma das 300 tentativas selecionadas foi submetida ao endpoint de análise da API baseada em LLMs. Embora o endpoint principal da API tenha sido concebido prioritariamente para fornecer feedback formativo, o assistente LLM foi instruído, por meio de engenharia de prompt, a também indicar explicitamente, via flag de status, se o código analisado era considerado correto. Essa informação, integrada ao feedback, tornou possível comparar, em cada tentativa, o resultado dos casos de teste (autograder) com a classificação de código correto/incorreto fornecida pelo LLM. Essa abordagem permitiu aferir a taxa de concordância e os desvios entre as duas estratégias, além de sustentar métricas como acurácia, sensibilidade e especificidade na identificação de soluções corretas.

4.2. Avaliação da Qualidade dos Feedbacks

A análise da qualidade e coerência dos feedbacks gerados (QP2) foi realizada imediatamente após a classificação de corretude, aproveitando os mesmos registros de feedback. Para isso, foi desenvolvido um prompt específico, aplicado a um LLM avaliador, com as

seguintes instruções: receber o enunciado, o código do estudante e o feedback fornecido pela API; classificar a resposta em uma das categorias: "coerente", "incoerente" ou "incompleto", e justificar a classificação em uma frase, retornando o resultado em formato JSON.

Após a classificação automatizada dos 300 feedbacks, uma amostra estratificada de 10% (ou seja, 30 casos) foi submetida a avaliação manual por especialistas em educação em computação. Esse procedimento permitiu validar a precisão do classificador LLM: caso fossem observadas divergências, o percentual final de feedbacks "coerentes" seria ajustado de acordo com a taxa de acerto manual, por exemplo, caso 10% dos casos amostrados apresentassem classificação equivocada, a métrica global era reduzida proporcionalmente. Essa triangulação metodológica assegura maior robustez e confiabilidade ao indicador de qualidade dos feedbacks.

4.3. Avaliação dos Custos Operacionais

Por fim, a análise dos custos operacionais (QP3) foi conduzida a partir do registro detalhado de cada chamada à API durante os experimentos. Foram computados o tempo de processamento, o consumo de recursos computacionais e o número de tokens processados em cada interação. Com base nesses dados, foi possível estimar o custo efetivo de operação do framework em cenários de larga escala, bem como realizar comparações diretas com os custos típicos de execução dos autograders convencionais.

5. Resultados e Discussão

O presente estudo avaliou o desempenho do framework de feedback inteligente a partir de múltiplas perspectivas: quantitativa, qualitativa e operacional. Em todas as etapas, buscou-se compreender como as decisões de projeto e as estratégias de engenharia de prompt influenciaram o comportamento da API frente a situações autênticas de aprendizagem em programação.

5.1. Concordância com Autograders Tradicionais

Com o objetivo de responder à QP1, "A avaliação por LLMs pode substituir autograders tradicionais com precisão equivalente?", analisamos a concordância entre a classificação binária da API baseada em LLMs e o resultado dos autograders convencionais embutidos na plataforma educacional. Para cada uma das 300 tentativas avaliadas, o endpoint da API indicou via flag se o código era considerado "correto" ou "incorreto", possibilitando a construção de uma matriz de confusão comparando esse diagnóstico com o dos casos de teste automáticos.

Os resultados apontaram uma acurácia global de 74,7%, conforme Tabela 1, sugerindo um alto grau de alinhamento, mas também revelando divergências relevantes entre os sistemas.

Tabela 1. Matriz de confusão entre avaliação da API e autograder

	API: Correto	**API: Incorreto**
Testes: Correto	61	39
Testes: Incorreto	36	163

Ao examinar as divergências, identificamos que falsos positivos (18,1%) e falsos negativos (20,3%) estão distribuídos de modo relativamente equilibrado, indicando a ausência de viés sistemático por parte do LLM. Entretanto, a análise qualitativa das divergências, realizada a partir de inspeção manual e do dataset, revelou padrões específicos, que aprofundam a compreensão das limitações e potencialidades do framework:

- **Alinhamento de critérios (FP/FN).** Divergências entre foco pedagógico e regras de teste geram dois padrões: (i) *FP* — o autograder reprova por formato/contagem/regex, enquanto a API aceita por julgar a lógica suficiente; (ii) *FN* — os testes aprovam sem exigir validação prevista no enunciado (p. ex., "inteiro positivo"), e a API reprova por faltar essa checagem.
- **Entrada e formato (I/O).** Diferenças em número de entradas, tipos e formato de saída (incluindo casas decimais e nomes exatos) explicam parte das discordâncias: o autograder tende a penalizar variações estruturais, enquanto a API prioriza aderência conceitual ao enunciado.
- **Forma versus conteúdo.** Problemas sutis de sintaxe/formatação (p. ex., esquecer o `f` no `print`), escolhas de tipo de dado ou convenções de inicialização podem não afetar a lógica central, mas impedem a aprovação nos testes; a API, por vezes, os trata como oportunidades de melhoria.
- **Ambiguidade do enunciado.** Quando o texto permite múltiplas leituras razoáveis, tanto a decisão do autograder quanto a da API podem ser justificáveis por interpretações distintas, caracterizando desalinhamento conceitual mais que erro do sistema.

Essa taxonomia sugere que boa parte das divergências resulta de diferentes prioridades avaliativas: o autograder tende a operar com base em regras rígidas de validação, enquanto o LLM, ao focar mediação pedagógica, pode privilegiar abordagens alternativas ou soluções que demonstrem compreensão conceitual, ainda que não estritamente conforme a especificação.

Esses achados não apenas reforçam resultados prévios da literatura [Gill et al. 2024, Cheng et al. 2023], como apontam para a necessidade de estratégias híbridas ou calibradas para maximizar tanto a precisão técnica quanto o potencial formativo. Destaca-se, ainda, o papel do feedback contextualizado do LLM, que, mesmo em casos de discordância, frequentemente contribui para a aprendizagem ao oferecer orientações para o aprimoramento da solução, superando limitações dos autograders tradicionais.

Em síntese, a análise detalhada das divergências demonstra que a adoção exclusiva de autograders ou LLMs pode ser insuficiente para uma avaliação plenamente formativa e justa, sugerindo a integração de ambos como uma abordagem mais robusta para apoiar o desenvolvimento de competências em programação.

5.2. Qualidade dos Feedbacks

A avaliação qualitativa indicou que 93,3% dos feedbacks gerados pela API foram coerentes, com orientação relevante e alinhada ao enunciado e ao código submetido. Esse índice supera resultados de estudos afins [Schwerter et al. 2022, Bassner et al. 2024], sinalizando o potencial do *framework* para suporte formativo consistente.

A validação manual, em amostra de 30 feedbacks, confirmou a confiabilidade da classificação automática: apenas um caso divergente, sem impacto no percentual global. O resultado reforça a aderência dos retornos aos critérios formativos propostos.

Dos 3,3% classificados como não coerentes, identificamos casos pontuais de desvio de escopo e respostas incompletas. *Exemplo 1:* a questão pedia apenas validação textual e checagem condicional da entrada do usuário, mas o feedback passou a exigir "estrutura de repetição", requisito não previsto. *Exemplo 2:* a tarefa era verificar a presença de um item pelo nome em uma lista, e o feedback introduziu "valores positivos/negativos", conceito ausente no enunciado. Esses episódios sugerem associações semânticas espúrias ou efeitos da janela curta de histórico, levando o modelo a extrapolar além do que foi especificado.

Além disso, cerca de 3% das interações resultaram em erro técnico (formato inesperado, *timeouts* ou falhas de comunicação), limitações inerentes ao uso de LLMs em escala e relevantes para implantação real.

Embora os resultados venham de exercícios curtos e introdutórios, controles de contexto e filtragem (histórico curto, reforço do enunciado) elevaram a consistência e a aderência do feedback. Ainda assim, o *framework* se mostra promissor com cautela: sua efetividade precisa ser reavaliada em tarefas mais complexas e contextos menos restritos.

5.3. Resultados Operacionais: Eficiência, Custo e Escalabilidade

No aspecto operacional, o sistema demonstrou elevada eficiência: 80 das 300 interações empregaram caching, reduzindo consumo de tokens e tempo de resposta para submissões repetidas. O consumo médio por requisição foi de 2.334 tokens, tempo médio de 10,5 segundos, custo médio de US$ 0,0115 por análise e custo total de US$ 2,51 para as 219 interações com métricas válidas. Esses indicadores situam o framework como uma solução viável para aplicações educacionais em larga escala, comparável (ou superior) a sistemas documentados na literatura internacional [Bassner et al. 2024]. Estratégias de otimização como caching inteligente e registro detalhado de métricas são decisivas para viabilizar a adoção ampla e sustentável de LLMs no ensino.

6. Conclusões

Este trabalho apresentou uma API de feedback formativo adaptativo baseada em LLMs para o ensino de programação. Ancorada na Teoria da Aprendizagem Significativa [Ausubel 1963] e em práticas formativas [Black and Wiliam 2009, Nicol and Macfarlane-Dick 2006], a arquitetura integra personalização progressiva, avaliação conceitual, mediação do erro e rastreamento longitudinal, em consonância com avanços em tutores inteligentes [Gill et al. 2024, Bassner et al. 2024, Schwerter et al. 2022], reforçando o potencial dos LLMs no apoio à aprendizagem ativa.

Avaliamos 300 tentativas reais, abrangendo diferentes perfis e estilos de solução. A análise quantitativa indicou 74,7% de concordância entre a avaliação binária da API e autograders, valor comparável a estudos prévios [Schwerter et al. 2022, Cheng et al. 2023], com sensibilidade e especificidade equilibradas. As divergências evidenciam que, enquanto autograders priorizam precisão funcional, o *framework* agrega dimensões pedagógicas e heurísticas mais flexíveis, aproximando-se da prática humana.

Na avaliação qualitativa, 93,3% dos feedbacks foram coerentes, úteis e alinhados a objetivos formativos, superando benchmarks recentes [Bassner et al. 2024, Schwerter et al. 2022]. O resultado decorre de engenharia de *prompt* e decisões de projeto, como reforço do enunciado no *prompt* e seleção/filtragem de saídas, que tornam o feedback específico e imediatamente aplicável, em linha com princípios de autorregulação e aprendizagem ativa [Nicol and Macfarlane-Dick 2006]. Simultaneamente, o *framework* supera limitações de autograders tradicionais, como julgamentos binários e mediação insuficiente [Messer et al. 2023], ao substituir respostas booleanas por orientações pedagógicas progressivas.

Operacionalmente, a API mostrou viabilidade técnica e econômica, com custo por interação inferior a US$0,02 e tempos de resposta compatíveis com uso em escala. Estratégias de *caching*, reuso de *threads* e otimização de *tokens* sustentaram a escalabilidade sem perda pedagógica.

Em síntese, o principal valor da API está na qualidade, adaptabilidade e contextualização dos feedbacks, convertendo a avaliação automatizada em aliada da aprendizagem significativa. O *framework* é replicável e extensível a outros domínios, desde que respeitadas as especificidades disciplinares.

7. Limitações e direções para trabalhos futuros

Apesar dos resultados promissores, o escopo deste estudo permanece circunscrito por algumas limitações que também apontam caminhos de evolução. Primeiro, a avaliação foi conduzida *offline*, sobre histórico de submissões, o que restringe a observação de efeitos dinâmicos de adaptação em tempo real; estudos in situ, com instrumentação longitudinal (p. ex., A/B e séries temporais), poderão capturar continuidade, deriva e latência percebida. Além disso, a análise concentrou-se no serviço de avaliação de código, sem cobrir de modo equilibrado os demais fluxos do sistema; trabalhos futuros devem ampliar a cobertura com métricas compartilhadas de qualidade, confiabilidade fim a fim e custo.

Outro limite decorre do perfil das tarefas: predominaram exercícios curtos e de baixa complexidade, cenário favorável ao desempenho. Uma etapa seguinte é incorporar problemas multietapa, entradas maiores e níveis variáveis de ambiguidade, medindo como coerência, especificidade do feedback e tempo de resposta escalam nesses contextos. Em paralelo, a generalização permanece aberta: embora a arquitetura seja extensível, a validação empírica focou exclusivamente programação; estudos piloto em domínios com I/O e critérios objetivos de correção, acompanhados de ajustes de instruções, critérios avaliativos e *benchmarks* específicos, são necessários.

Por fim, reconhecemos que a rotulagem automática dos feedbacks na etapa de avaliação pode comprometer a fidedignidade dos resultados; para mitigar, implementou-se um protocolo de validação cruzada com amostragem manual, com previsão de ampliar a amostra e realizar uma avaliação específica da acurácia da classificação automática. Em paralelo, persistem limites do assistente, como imprevisibilidade dos modelos, "alucinações" e extrapolações do enunciado, que vêm sendo tratados por calibração de *prompts*, reforço de restrições pedagógicas e filtros de pós-processamento; trabalhos futuros incluirão checagens de consistência semântica entre tarefa e resposta, regeração direcionada e ajustes que aumentem a especificidade do feedback e desestimulem padrões genéricos.

8. Considerações finais

O uso de LLMs em educação levanta desafios éticos centrais, amplamente discutidos na literatura [Zhou et al. 2023, Porayska-Pomsta et al. 2024, Luckin et al. 2016]. Entre os principais riscos destacam-se automação acrítica da avaliação, reforço de vieses algorítmicos e ameaça à autonomia estudantil [Yan et al. 2024]. Atentos a esses aspectos, orientamos o desenvolvimento por transparência, rastreabilidade e respeito à individualidade do estudante.

Especificamente, evitou-se fornecer respostas diretas ou códigos completos, priorizando mediação reflexiva e estímulo à autorregulação, em consonância com [Black and Wiliam 2009, Zimmerman 2002]. Todas as interações são registradas para auditoria, e realizamos revisão humana amostral, com foco nos feedbacks marcados como "não úteis", a fim de identificar padrões recorrentes e orientar refinamentos de instruções e regras operacionais.

Reconhecemos a existência de potenciais vieses, tanto linguísticos (variedade de registro, regionalismos e tom/severidade) quanto pedagógicos (suposições sobre conhecimento prévio). As estratégias de mitigação implementadas neste trabalho incluem: instruções pedagógicas explícitas para promover elogios ao esforço e tom encorajador, adaptação do nível de detalhamento do feedback para evitar respostas genéricas, e pós-processamento para bloquear respostas fora de contexto ou revelação direta de soluções. Cabe destacar que a detecção automatizada de viés não foi contemplada no escopo atual. Na dimensão da privacidade, aplicamos minimização de dados rigorosa e prevenção de exposição de identificadores pessoais nos modelos e artefatos públicos.

Em uso contínuo, planejamos avaliação formal de viés com amostragem estratificada por tipo de exercício e nível de proficiência, monitorando assimetrias de tom/severidade e cobertura, e empregando auditorias manuais periódicas; os achados alimentarão calibração de *prompts*, diretrizes pedagógicas e políticas de uso. Com esse arranjo, o *framework* busca conciliar ganhos formativos com responsabilidade no uso de LLMs em contextos educacionais sensíveis.

Referências

Ala-Mutka, K. (2005). A survey of automated assessment approaches for programming assignments. In *Computer Science Education*, volume 15, page 83–102.

Alvim, I., Bittencourt, R., and Duran, R. (2024). Evasão nos cursos de graduação em computação no brasil. In *Simpósio Brasileiro de Educação em Computação (EDUCOMP)*.

Ausubel, D. P. (1963). *The psychology of meaningful verbal learning*. Grune & Stratton.

Bassner, P., Frankford, E., and Krusche, S. (2024). Iris: An ai-driven virtual tutor for computer science education. In *ITiCSE 2024: Proceedings of the 29th ACM Conference on Innovation and Technology in Computer Science Education*, pages 1–7.

Black, P. and Wiliam, D. (2009). *Developing the Theory of Formative Assessment*, volume 21. Educational Assessment, Evaluation and Accountability.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, page 1877–1901.

Cheng, G., Santos, A., Koh, E., and Cheung, S. (2023). Codetailor: Llm-powered personalized parsons puzzles for engaging support while learning programming. In *Proceedings of the 18th European Conference on Technology Enhanced Learning (EC-TEL)*, pages 143–156.

Ding, Y., Hu, H., Zhou, J., Chen, Q., Jiang, B., and He, L. (2024). Boosting large language models with socratic method for conversational mathematics teaching. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM 2024)*, pages 3730–3735.

Gill, S. S., Buyya, R., Yang, C., and Chan, C. S. (2024). Edge ai: A taxonomy, systematic review and future directions. *Cluster Computing*, 27(1):99–121.

Ihantola, P., Ahoniemi, T., Karavirta, V., and Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, page 86–93.

Jang, J., Eun, S., and Lee, H. (2024). The effects of prompt scaffolding on learning to write arguments with chatgpt. In *Proceedings of the 17th International Conference of the Learning Sciences (ICLS 2024)*, pages 1502–1505.

Kasneci, E., Sessler, K., Völkel, T., and Kasneci, G. (2023). Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103:102274.

Kazemitabaar, M., Ye, R., Wang, X., Henley, A. Z., Denny, P., Craig, M., and Grossman, T. (2024). Codeaid: Evaluating a classroom deployment of an llm-based programming assistant that balances student and educator needs. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA. Association for Computing Machinery.

Keuning, H., Jeuring, J., and Heeren, B. (2018). A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education*, 19(1):1–43.

Krupp, L., Steinert, S., Kiefer-Emmanouilidis, M., Avila, K. E., Lukowicz, P., Kuhn, J., Küchemann, S., and Karolus, J. (2024). Challenges and opportunities of moderating usage of large language models in education. MUM '24, page 249–254, New York, NY, USA. Association for Computing Machinery.

Lampou, R. (2023). The integration of artificial intelligence in education: Opportunities and challenges. *Review of Artificial Intelligence in Education*, 4.

Leite, A. and Blanco, S. A. (2020). Effects of human vs. automatic feedback on students' understanding of ai concepts and programming style. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE '20, page 44–50, New York, NY, USA. Association for Computing Machinery.

Luckin, R., Holmes, W., Griffiths, M., and Forcier, L. (2016). *Intelligence Unleashed: An Argument for AI in Education*. Pearson.

Marvin, G., Hellen, N., Jjingo, D., and Nakatumba-Nabende, J. (2024). Prompt engineering in large language models. In Jacob, I. J., Piramuthu, S., and Falkowski-Gilski, P., editors, *Data Intelligence and Cognitive Informatics*, pages 387–402, Singapore. Springer Nature Singapore.

Medeiros, R. P., Ramalho, G. L., and Falcão, T. P. (2019). A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education*, 62(2):77–90.

Messer, M., Brown, N. C. C., Kölling, M., and Shi, M. (2023). Automated grading and feedback tools for programming education: A systematic review. *ACM Transactions on Computing Education*, 24(1):1–34.

Nicol, D. J. and Macfarlane-Dick, D. (2006). Formative assessment and self-regulated learning: A model and seven principles of good feedback practice. *Studies in Higher Education*, 31(2):199–218.

Paiva, J. C., Leal, J. P., and Figueira, A. (2022). Automated assessment in computer science education: A state-of-the-art review. *ACM Trans. Comput. Educ.*, 22(3).

Pereira, A. F. and Ferreira Mello, R. (2025). A systematic literature review on large language models applications in computer programming teaching evaluation process. *IEEE Access*, 13:113449–113460.

Porayska-Pomsta, K., Holmes, W., and Nemorin, S. (2024). The ethics of ai in education. In du Boulay, B., Mitrovic, A., and Yacef, K., editors, *Handbook of Artificial Intelligence in Education*, pages 571–604. Edward Elgar.

Pornprasit, C. and Tantithamthavorn, C. (2024). Fine-tuning and prompt engineering for large language models-based code review automation. *Information and Software Technology*, 175:107523.

Schwerter, J., Schindler, K., Hoffmann, K., and Sailer, M. (2022). E-learning with multiple-try-feedback: Can hints foster students' achievement? *Educational Technology Research and Development*, 70:713–736.

Teusner, R., Hille, T., and Staubitz, T. (2018). Effects of automated interventions in programming assignments: evidence from a field experiment. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, L@S '18, New York, NY, USA. Association for Computing Machinery.

Vygotsky, L. S. (1978). *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Piloto, L., Xia, F., Tillet, P., Le, Q. V., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Yan, L., Sha, L., Zhao, L., Li, Y., Maldonado, R. M., Chen, G., Li, X., Jin, Y., and Gašević, D. (2024). Practical and ethical challenges of large language models in education: A systematic scoping review. *British Journal of Educational Technology*, 55(1):90–112.

Yu, Z., Xie, X., and Xu, H. (2023). Automated feedback generation in programming education: A review of recent advances. *IEEE Transactions on Learning Technologies*, 16(1):78–91.

Zawacki-Richter, O., Marín, V. I., Bond, M., and Gouverneur, F. (2019). Systematic review of research on artificial intelligence applications in higher education – where are the educators? *International Journal of Educational Technology in Higher Education*, 16(39).

Zhou, M., Chen, F., and Lee, H. F. (2023). Ethics in the age of artificial intelligence in education: Promises and pitfalls. *British Journal of Educational Technology*, 54(1):10–29.

Zimmerman, B. J. (2002). Becoming a self-regulated learner: An overview. *Theory Into Practice*, 41(2):64–70.