Reprodução do artigo *Can ChatGPT Play the Role of a Teaching Assistant in an Introductory Programming Course?*

Helen Bento Cavalcanti
helen.cavalcanti@copin.ufcg.edu.br
Universidade Federal de Campina Grande
Campina Grande, Paraíba

1 INTRODUÇÃO

Com o advento e a ampla disponibilização de Grandes Modelos de Linguagem (LLMs), como o ChatGPT[3] da OpenAI, a comunidade de educação em computação tem investigado seu potencial transformador. Estudos demonstram que esses modelos já conseguem resolver problemas de programação introdutória com um desempenho comparável ao de um estudante médio, validando sua competência técnica no domínio. Consequentemente, o seu potencial como ferramenta pedagógica para apoiar assistentes de ensino e instrutores tem sido amplamente explorado. A literatura recente investiga sua aplicação em diversas tarefas[2] como o aprimoramento de mensagens de erro para facilitar a depuração de código, a geração de novos exercícios de programação e o fornecimento de feedback personalizado sobre as submissões dos alunos.

Nesse contexto, o artigo "Can ChatGPT Play the Role of a Teaching Assistant in an Introductory Programming Course?"de Anishka et al.[1] investiga o uso do ChatGPT para apoiar duas tarefas cruciais em cursos introdutórios de programação: a atribuição de notas e o fornecimento de feedback a submissões de código dos alunos. A motivação para o estudo original reside nos desafios logísticos de disciplinas com grande número de matrículas, onde a avaliação manual por múltiplos assistentes de ensino (AEs) pode levar a inconsistências e a um alto custo operacional. O artigo original concluiu que, embora o ChatGPT demonstre potencial, ele ainda enfrenta limitações significativas, especialmente na tarefa de atribuição de notas.

O presente trabalho consiste em uma reprodução de parte do estudo de Anishka et al., focando especificamente na tarefa de atribuição de notas por ser uma área com resultados que podem ser comparados mais diretamente. Esta reprodução busca reavaliar as conclusões originais utilizando um modelo mais recente e com *reasoning*(GPT-o4-mini) e um conjunto de dados diferente, construído a partir de submissões públicas da plataforma de programação competitiva Codeforces, uma vez que os dados originais não foram disponibilizados. Dessa forma, este artigo contribui para a validação dos achados anteriores em um novo contexto e com uma tecnologia mais avançada, aprofundando a compreensão sobre a viabilidade atual dos LLMs como ferramentas de avaliação automatizada.

2 OBJETIVO

Este trabalho de reprodução visa reavaliar e expandir as conclusões do estudo de Anishka et al. sobre a viabilidade do ChatGPT como assistente de ensino. O foco desta pesquisa recai especificamente sobre a capacidade de um LLM na tarefa de atribuição de notas a atividades de programação, uma área que, segundo o artigo original, ainda apresenta limitações significativas.

3 METODOLOGIA

A presente seção tem como objetivo descrever e comparar os delineamentos metodológicos do trabalho original e desta reprodução, explicitando suas principais semelhanças e, sobretudo, as diferenças que caracterizam este estudo.

3.1 METODOLOGIA ORIGINAL

O artigo que foi reproduzido utiliza um desenho de pesquisa experimental para determinar a viabilidade do ChatGPT como assistente de ensino em um curso de introdução à programação. A metodologia do estudo detalha uma série de experimentos para avaliar as capacidades do modelo em avaliar as submissões de código dos alunos e em fornecer feedback relevante. Para os experimentos, os pesquisadores usaram o modelo GPT-3.5 da OpenAI, optando pela versão API do modelo para facilitar a experimentação em larga escala.

3.1.1 ***Conjunto de Dados.*** Para a avaliação, o artigo original empregou um dataset proprietário, não disponível publicamente, que se diferenciava por incluir as notas atribuídas por AEs humanos para cada tarefa. O conjunto de dados totalizava 6783 submissões em Python, oriundas de três trabalhos de um curso de introdução à programação. Esses trabalhos eram compostos por múltiplas questões de dificuldades variadas, sendo que cada submissão individual correspondia à solução de um aluno para uma questão específica. Em contraste com muitas plataformas online existentes que avaliam as submissões de código estritamente com base na correção de entrada/saída, os AEs também consideram a funcionalidade parcial em suas avaliações.

3.1.2 ***Experimentos: Avaliação dos Envios de Código dos Alunos.*** No primeiro experimento, o estudo original investigou a performance do ChatGPT como avaliador de código de estudantes, com foco tanto na correção da funcionalidade quanto na qualidade da implementação. O procedimento consistiu em fornecer ao ChatGPT o enunciado da questão, a rubrica de avaliação e a submissão do aluno como entrada. O modelo foi então instruído a gerar duas pontuações: uma para a funcionalidade (variando de 0 a 2, para códigos incorretos, parcialmente corretos ou totalmente corretos) e outra para a qualidade, com base nos critérios de modularidade e nas métricas de complexidade de Halstead*. O *prompt* utilizado pode ser observado na Figura 1.

*As métricas de complexidade de Halstead são um conjunto de medidas quantitativas propostas por Maurice Halstead para avaliar a complexidade de um programa de software com base em suas propriedades léxicas, ou seja, no uso de operadores e operandos no código fonte. Essas métricas não dependem da lógica do controle de fluxo (como a complexidade ciclomática), mas sim da contagem de elementos do código. Elas buscam estimar aspectos como esforço de implementação, dificuldade e volume do software.

O estudo original conta com um segundo experimento que avaliou o potencial do ChatGPT para gerar feedback e sugestões de melhoria em submissões de código de alunos. Para isso, os pesquisadores selecionaram aleatoriamente cerca de 10 submissões de cada trabalho e solicitaram ao modelo que gerasse propostas de aprimoramento para cada uma delas. Este experimento foi omitido na reprodução em questão. A principal razão é a sua natureza subjetiva, que depende do julgamento humano, tornando a replicação direta mais complexa. Adicionalmente, sua execução exigiria um grande esforço de anotação e análise manual, tanto na classificação das sugestões por especialistas quanto na implementação e reavaliação de cada código modificado.

Prompt de Avaliação de Exercício

Context: You are a teaching assistant for an Introduction To Programming Course, and your task is to grade the student code submission using the provided rubric. The code is written in Python programming language.

Question: Integration through computation. Suppose the velocity of a rocket at a time t is given by:

$$f(t) = 2000 \ln \left[\frac{140000}{140000 - 2100t} \right] - 9.8t$$

Take as input starting time (a) and ending time (b), and find the distance covered by a rocket between time a and b. For computationally determining the distance, work with time increments (Δt) of 0.25 seconds. You can use the `math` module of Python for this problem.
(Hint: Compute the velocity at time t and $t + \Delta t$, take the average, and then compute the distance traveled in this Δt time duration. Start from a and keep computing in increments of Δt till b.)

Grading Instructions: Given this question you are supposed to mark the student code based on functionality. If the code is completely correct give 2 marks. If the code is partially correct give 1 mark. If the code is incorrect give 0 marks. If the code is empty give 0 marks. In addition to the functionality score also give a quality score on a scale of 1-5 based on Halstead metrics and code modularity. Do not correct the previous solution or solve the question just give the final score as output without any explanation or extra text.

Student Code: `{Student Code Goes Here}`

Output Format:

```
Functionality Score :
Halstead Metrics-based Quality Score :
Modularity-based Quality Score :
Overall Quality Score :
```

Figura 1: Estrutura do prompt para avaliação automatizada de exercícios de programação.

3.2 METODOLOGIA DA REPRODUÇÃO

Com o objetivo de detalhar o processo de reprodução do estudo, a presente subseção apresenta a metodologia utilizada. Com o intuito de facilitar futuras reproduções, o código e os dados utilizados estão disponibilizados em um repositório no GitHub[†].

3.2.1 ***Conjunto de Dados.*** Dada a indisponibilidade do *dataset* original, a metodologia desta reprodução baseou-se em um novo conjunto de dados[‡], coletado de submissões públicas na plataforma Codeforces[§], um conhecido ambiente online para competições de programação. Para se alinhar ao foco do estudo original em introdução à programação, foram escolhidos 15 problemas da menor faixa de dificuldade da plataforma (tag 800). De cada um dos 15 problemas, foram coletadas 10 submissões, totalizando 150 amostras para análise. O conjunto de dados está balanceado, visto que para cada problema, foram selecionadas cinco submissões julgadas como "*Accepted*"(corretas) e cinco como "*Wrong Answer*"(incorretas). Uma diferença fundamental na metodologia de avaliação reside na escala de notas. Enquanto esta reprodução adotou uma classificação binária, considerando as submissões apenas como corretas ou incorretas , o estudo original empregou um sistema multiclasse que incluía a possibilidade de crédito parcial para respostas não totalmente corretas.

3.2.2 ***Experimentos: Avaliação dos Envios de Código dos Alunos.*** Uma vez que o código-fonte do estudo original não foi disponibilizado , um novo *Jupyter notebook* de análise foi implementado para esta reprodução. Este novo código buscou seguir a metodologia descrita no artigo original, mas incluiu três adaptações principais: o uso do novo dataset gerado a partir do Codeforces, pequenas modificações no prompt enviado ao ChatGPT para adequá-lo ao novo contexto experimental (como pode ser observado na Figura 2), além do GPT 3.5, usar também o GPT o4-mini.

Essa decisão visou não apenas replicar o experimento, mas também estendê-lo, permitindo uma comparação entre o desempenho de diferentes gerações de modelos e investigando se uma capacidade de *reasoning* aprimorada resultaria em uma avaliação de código mais precisa.

[†]https://github.com/helenbc/Article-Reproduction
[‡]https://www.kaggle.com/datasets/immortal3/codeforces-dataset
[§]https://codeforces.com/

Prompt de Avaliação de Exercício na Reprodução

Context: You are a teaching assistant for an Introduction To Programming Course, and your task is to grade the student code submission using the provided rubric. The code is written in Python programming language.

Question: *{Problem Statement Goes Here}*

Grading Instructions: If the code is completely correct give 1 marks. If the code is incorrect give 0 marks. If the code is empty give 0 marks. Do not return the distribution of marks. Just return the final marks. In addition to the functionality score also give a quality score on a scale of 1-5 based on Halstead metrics and code modularity. Do not correct the previous solution or solve the question just give the final score as output without any explanation or extra text. Follow all the instructions carefully.

Student Code: `{Student Code Goes Here}`

Output Format:

```
Functionality Score :
Overall Quality Score :
```

Figura 2: Estrutura do prompt para avaliação de exercícios de programação na Reprodução.

A Tabela 1 resume as principais diferenças e semelhanças entre os estudos.

4 RESULTADOS

Esta seção detalha os resultados da análise quantitativa realizada sobre as submissões de código. A análise está dividida em quatro partes: as carcterísticas do dataset, a distribuição das métricas de qualidade do código, o desempenho do modelo GPT-3.5 na tarefa de avaliação e, por fim, uma análise comparativa com o modelo GPT-o4-mini.

4.1 CARACTERÍSTICAS DO DATASET

O dataset final consistiu em 147 submissões únicas de código Python após verificação de duplicatas por *id* da submissão. A análise assegurou que não exista submissões duplicadas no conjunto de dados, garantindo a integridade da análise estatística. As submissões abrangeram 15 problemas distintos de programação competitiva.

4.2 DISTRIBUIÇÃO DAS MÉTRICAS DE QUALIDADE

Para cada uma das submissões de código, foram calculadas as métricas de esforço de Halstead e modularidade utilizando a biblioteca *Radon*[¶]. A análise da distribuição dessas métricas revelou:

- **Modularidade** A distribuição dos scores de modularidade, em uma escala de 0 a 100, concentra-se majoritariamente na faixa entre 60 e 80, com uma média em torno de 68,49, como pode-se observar na Figura 3.
- **Esforço de Halstead** A distribuição desta métrica é fortemente assimétrica à direita, indicando que a maioria dos códigos possui baixa complexidade, com poucos casos apresentando valores de esforço muito elevados (um outlier), como observado na Figura 4.
- **Relação entre Métricas** Uma análise de regressão entre o esforço de Halstead e a modularidade (após a remoção de um outlier de esforço superior a 9000) indicou um coeficiente de correlação de −0,47, sugerindo uma relação negativa moderada entre as duas variáveis. Como visto nas Figuras 5 e 6.

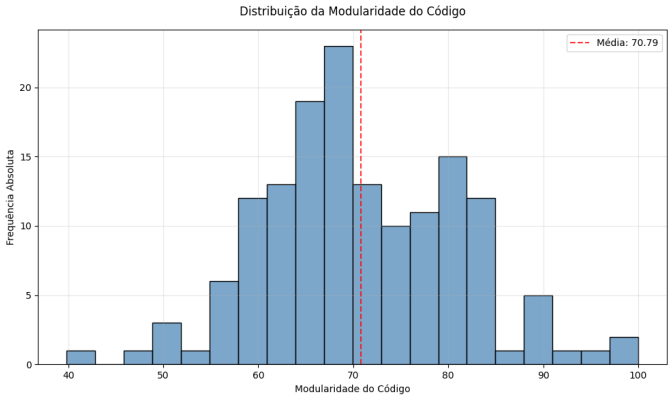

Figura 3: Distribuição da Modularidade do Código.

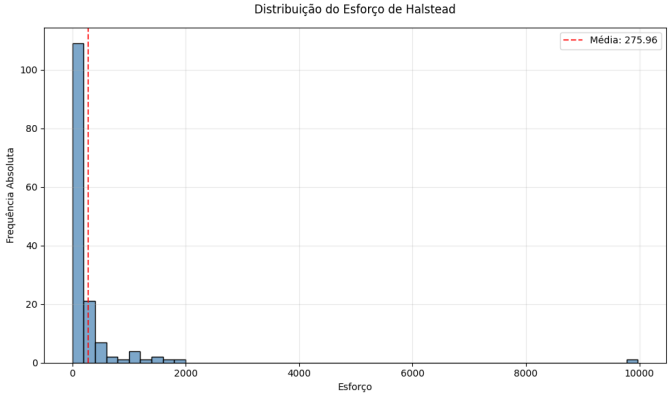

Figura 4: Distribuição do Esforço de Halstead.

4.3 AVALIAÇÃO DO DESEMPENHO DO GPT-3.5

O modelo GPT-3.5 foi utilizado para gerar uma nota de funcionalidade (0 para incorreto, 1 para correto) e uma nota de qualidade (em uma escala de 1 a 5) para cada submissão. Esta abordagem difere do estudo original, que utilizou uma escala multiclasse. O veredito da plataforma *Codeforces* serviu como *ground truth* na reprodução,

[¶]https://pypi.org/project/radon/

Tabela 1: Síntese Comparativa das Metodologias

Aspecto	Artigo Original	Reprodução
	Diferenças	
Fonte dos Dados	Dataset proprietário (curso)	Dataset público (Codeforces)
Avaliação (Ground Truth)	Multiclasse (0, 1, 2) por AEs	Binária (0, 1) baseado no Codeforces
Modelo de LLM	GPT-3.5	GPT-3.5 e GPT-o4-mini
Análise Estatística	Correlação (Pearson), desvios	Correlação (Spearman), bootstrap, matriz de confusão
	Semelhanças	
Linguagem	Python	Python
Avaliação de Qualidade	Nota (1-5) vs. métricas Radon	Nota (1-5) vs. métricas Radon
Estratégia de Prompt	Prompt detalhado	Prompt adaptado do original

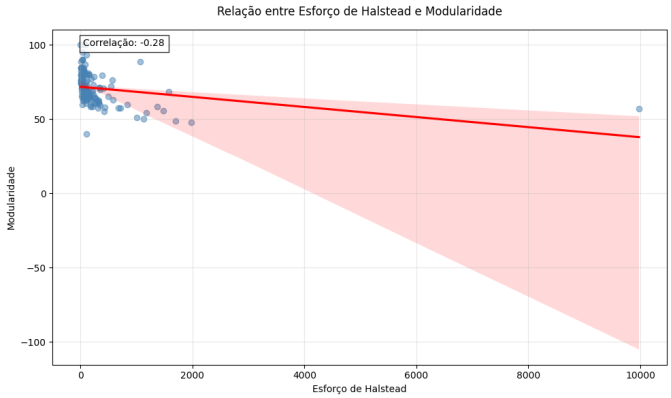

Figura 5: Relação entre Esforço de Halstead e Modularidade

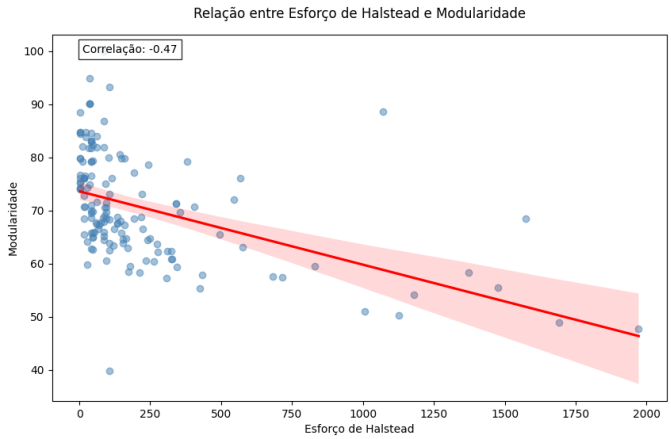

Figura 6: Relação entre Esforço de Halstead e Modularidade sem *Outlier*.

contendo uma coluna binária com distribuição balanceada de 75 respostas corretas e 75 incorretas.

- **Acurácia e Matriz de Confusão** A acurácia geral do GPT-3.5 na atribuição de notas de funcionalidade foi de 49,66%, com variações significativas por problema (entre 30% e 66,7%). O estudo original não calculou uma métrica de acurácia, mas reportou desvios médios e máximos elevados entre as notas do modelo e as dos AEs, o que também indica uma baixa concordância.
  A análise da matriz de confusão (Figura 7) nesta reprodução revelou um viés do modelo em classificar as submissões como corretas. O modelo gerou um número elevado de falsos positivos (58) em comparação com os falsos negativos (16). Este tipo de análise de erro não foi apresentado no artigo original, mas corrobora a conclusão de que o modelo não é totalmente confiável para a tarefa.

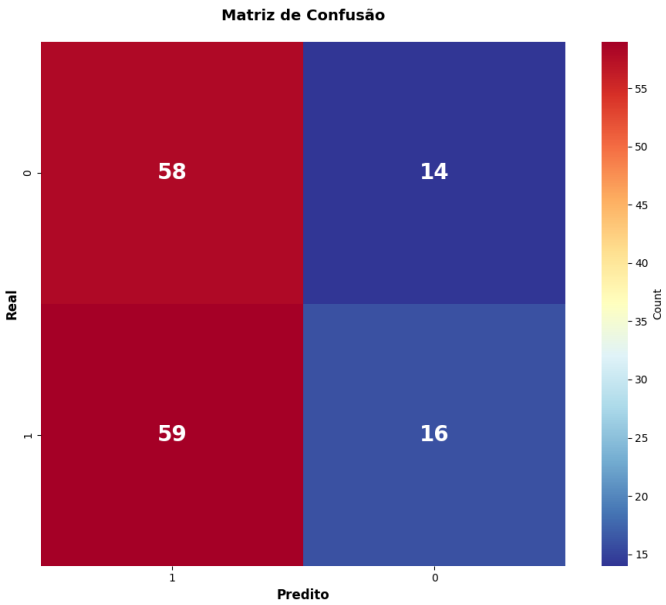

Figura 7: Matriz de Confusão GPT-3.5

- **Análise de Correlação Comparativa** Uma divergência notável em relação ao estudo original foi encontrada na análise de correlação da funcionalidade. Nesta reprodução, não houve correlação estatisticamente significativa entre a nota de funcionalidade do GPT-3.5 e o veredito real ($r \approx -0{,}02$) (Figura 10). Em contraste, o artigo de Anishka et al.[1] encontrou uma correlação positiva, ainda que de fraca a moderada (coeficientes entre 0,14 e 0,59), com as notas dos AEs. Essa diferença pode ser atribuída às distintas naturezas dos *datasets* e dos *ground truths* (veredito binário de plataforma vs. avaliação humana com crédito parcial).
  No entanto, em relação à nota de qualidade, os resultados foram consistentes. Assim como no artigo original, que não encontrou correlação significativa entre a nota de qualidade e as métricas do Radon, esta reprodução também obteve um resultado similar, com um coeficiente de correlação próximo de zero.

4.4 ANÁLISE COMPARATIVA COM O GPT-O4-MINI

O experimento foi repetido utilizando o modelo GPT-o4-mini-2025-04-16 para fins de comparação.

- **Acurácia Comparativa** O modelo GPT-o4-mini alcançou uma acurácia de 53,06%, um resultado ligeiramente superior ao do GPT-3.5 8. A matriz de confusão (Figura 9) para este modelo revelou 33 falsos positivos e 36 falsos negativos, indicando um comportamento mais equilibrado em comparação com o GPT-3.5.

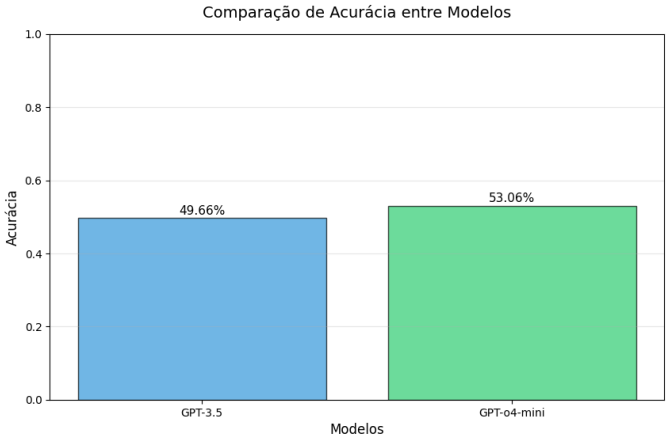

Figura 8: Comparação de Acurácia entre Modelos

- **Correlação Comparativa** O gráfico de barras comparativo das correlações (Figura 10) mostra que ambos os modelos apresentaram um desempenho similar na correlação entre as notas de qualidade e a modularidade (próximo de zero) e na correlação entre as notas de funcionalidade e o veredito (também próximo de zero). A correlação negativa moderada entre o esforço de Halstead e a modularidade foi consistente em ambas as análises.

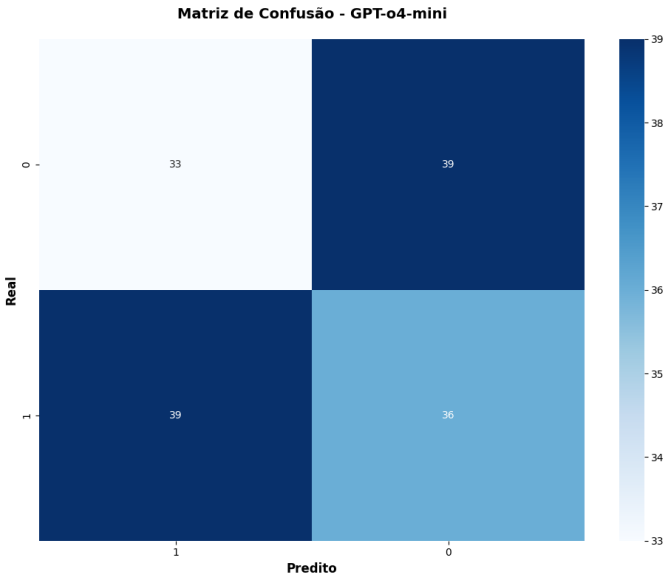

Figura 9: Matriz de Confusão - GPT-o4-mini

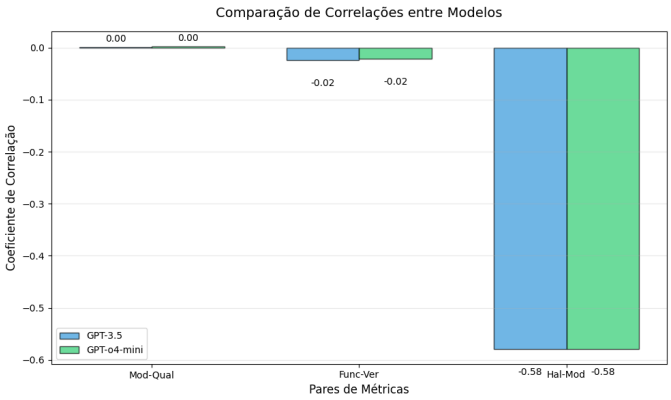

Figura 10: Comparação de Correlações entre Modelos

5 CONCLUSÃO

Os resultados encontrados nesta reprodução corroboram a conclusão do estudo original: os LLMs, mesmo em versões mais recentes como o GPT-o4-mini, ainda não demonstram a confiabilidade necessária para a tarefa de atribuição autônoma de notas em exercícios de programação. O modelo alcançou uma acurácia de apenas 53,06%, um desempenho pouco superior ao acaso em um cenário de classificação binária. A análise da matriz de confusão revelou uma distribuição de erros significativa, com 33 falsos positivos e 36 falsos negativos, indicando que o modelo ainda comete um número elevado de erros de classificação. Adicionalmente, confirmou-se a ausência de correlação estatisticamente significativa entre as notas de funcionalidade e qualidade atribuídas pelo modelo e as métricas objetivas de referência, reforçando a dificuldade do LLM em capturar nuances de correção e boas práticas de codificação.

Em comparação com o trabalho original, este estudo reforça a tese de que a avaliação de código por LLMs é uma tarefa complexa, mas também introduz novas perspectivas por meio de uma análise estatística diferente. Uma similaridade chave foi a incapacidade de ambos os modelos em gerar notas de qualidade que se alinhassem com métricas de software objetivas. Uma divergência notável, no entanto, foi a ausência total de correlação na avaliação de funcionalidade, enquanto o estudo original encontrou uma correlação positiva fraca, o que pode ser atribuído a diferenças metodológicas como o *dataset* (*Codeforces* vs. atividades de curso) e o sistema de notas (binário vs. multiclasse com crédito parcial). Reconhecem-se as limitações desta reprodução, como o tamanho reduzido do *dataset* e a ausência de crédito parcial. Trabalhos futuros podem explorar o impacto de diferentes estratégias de *prompt engineering*, utilizar *datasets* mais amplos e comparar o desempenho de uma gama maior de modelos de LLMs para aprofundar a compreensão sobre seu potencial e suas limitações no contexto educacional.

REFERÊNCIAS

[1] Anishka, Atharva Mehta, Nipun Gupta, Aarav Balachandran, Dhruv Kumar, and Pankaj Jalote. Can chatgpt play the role of a teaching assistant in an introductory programming course?, 2024.

[2] Bruno Pereira Cipriano and Pedro Alves. Gpt-3 vs object oriented programming assignments: An experience report. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2023, page 61–67, New York, NY, USA, 2023. Association for Computing Machinery.

[3] OpenAI. Chatgpt, 2025. Accessed: June 2025.