Revista Brasileira de Informática na Educação – RBIE
Brazilian Journal of Computers in Education
(ISSN online: 2317-6121; print: 1414-5685)
https://sol.sbc.org.br/journals/index.php/rbie

Submission: 02/05/2024; 1st round notif.: 17/03/2025; New version: 19/05/2025; 2nd round notif.: 16/06/2025;
Camera ready: 23/06/2025; Edition review: 10/07/2025; Available online: 10/07/2025; Published: 10/07/2025;

Análise das Respostas de LLMs em Relação ao Conteúdo Introdutório de Programação: um Comparativo entre o ChatGPT e o Gemini

Title: Analysis of Responses from LLMs Regarding Introductory Programming Content: A Comparative Study between ChatGPT and Gemini

Título: Análisis de Respuestas de LLMs en Relación al Contenido Introductorio de Programación: Un Estudio Comparativo entre ChatGPT y Gemini

Luiz Carlos Pereira Filho
IFSP – campus de Bragança Paulista
ORCID: 0009-0007-4589-7711
filho.luiz@aluno.ifsp.edu.br

Talita de Paula Cypriano de Souza
IFSP – campus de Bragança Paulista
ORCID: 0009-0004-0367-2387
talita@ifsp.edu.br

Luciano Bernardes de Paula
IFSP – campus de Bragança Paulista
ORCID: 0000-0002-3909-0395
lbernardes@ifsp.edu.br

Resumo

Recentemente, o uso dos grandes modelos de linguagem (LLMs – Large Language Models) para processamento de linguagem natural teve destaque dentre as tecnologias atuais. Essa tecnologia trouxe uma gama de possibilidades de uso em diversas áreas, incluindo o ensino de programação, uma vez que esses modelos podem criar códigos de programas. Dentre esses modelos, dois são conhecidos: o ChatGPT da OpenAI e o Gemini da Google, e ambos demonstram habilidades de criar, corrigir e explicar códigos de programação em diversas linguagens. Em um trabalho anterior, foram feitos testes e analisadas as respostas do ChatGPT em relação ao conteúdo introdutório de programação, do ponto de vista de estudantes iniciantes no assunto. Este trabalho estende o trabalho anterior e adiciona testes com o Gemini, também em relação ao mesmo conteúdo. O objetivo é verificar se esses modelos são adequados para estudantes iniciantes em programação e se é possível utilizá-los para o aprendizado desse conteúdo. Assim como no trabalho anterior, foram feitos testes qualitativos, nos quais eram feitas algumas interações com o modelo caso a resposta inicial não fosse satisfatória, e testes quantitativos, nos quais não foram feitas essas interações. Todos os testes foram feitos tanto no ChatGPT quanto no Gemini e suas respostas foram analisadas. Ambos apresentaram a existência de potencial para responder e explicar corretamente códigos gerados, mas há ressalvas. O desempenho geral dos LLMs testados, em relação às respostas corretas, foi de ∼78,2% para o ChatGPT e ∼69,6% para o Gemini. Mesmo com esse potencial para auxiliar no processo de aprendizagem de programação, as respostas geradas pelos LLMs não devem ser consideradas totalmente corretas, demandando conhecimento prévio de quem os usa para analisá-las e fazer uso delas.

Palavras-chave: *Ensino de programação; Programação para iniciantes; ChatGPT; Gemini; LLM.*

Abstract

Recently, Large Language Models for Natural Language Processing have stood out among current technologies. This technology has opened up a range of possibilities for use in various areas, including programming education, as these models can create program codes. Among these models, two are well-known: OpenAI's ChatGPT and Google's Gemini, both demonstrating abilities to create, correct, and explain programming codes in various languages. In a previous work, tests were conducted and the responses of ChatGPT were analyzed regarding introductory programming content from the perspective of beginners in the subject. This work extends the previous research and adds tests with Gemini, also concerning the same content. The goal is to determine whether these models are suitable for beginner programming students and whether they can be used for learning this content. As in the previous work,

Cite as: Pereira Filho, L. C., Souza, T. P. C. & Paula, L. B. (2025). Análise das Respostas de LLMs em Relação ao Conteúdo Introdutório de Programação: um Comparativo entre o ChatGPT e o Gemini. Revista Brasileira de Informática na Educação, vol.33, 722–747. https://doi.org/10.5753/rbie.2025.4477.

qualitative tests were conducted, in which some interactions with the model were made if the initial response was unsatisfactory, and quantitative tests, in which these interactions were not made. All tests were conducted on both ChatGPT and Gemini, and their responses were analyzed. Both showed potential to correctly respond to and explain generated codes, but there are caveats. The overall performance of the tested LLMs, in terms of correct responses, was ∼78.2% for ChatGPT and ∼69.6% for Gemini. Even with this potential to assist in the programming learning process, the responses generated by LLMs should not be considered entirely correct, demanding prior knowledge from those who use them to analyze and make use of them.
Keywords: *Programming teaching; Programming for beginners; ChatGPT; Gemini; LLM.*

Resumen

Recientemente, los LLMs de procesamiento de lenguaje natural han destacado entre las tecnologías actuales. Esta tecnología ha abierto una variedad de posibilidades de uso en diversas áreas, incluida la educación en programación, ya que estos modelos pueden crear códigos de programas. Entre estos modelos, dos son conocidos: el ChatGPT de OpenAI y el Gemini de Google, ambos demostrando habilidades para crear, corregir y explicar códigos de programación en varios lenguajes. En un trabajo anterior, se realizaron pruebas y se analizaron las respuestas de ChatGPT con respecto al contenido introductorio de programación desde la perspectiva de principiantes en el tema. Este trabajo extiende la investigación anterior y agrega pruebas con Gemini, también en relación con el mismo contenido. El objetivo es verificar si estos modelos son adecuados para estudiantes principiantes en programación y si es posible utilizarlos para el aprendizaje de este contenido. Al igual que en el trabajo anterior, se realizaron pruebas cualitativas, en las que se hicieron algunas interacciones con el modelo si la respuesta inicial no fue satisfactoria, y pruebas cuantitativas, en las que no se realizaron estas interacciones. Todas las pruebas se realizaron tanto en ChatGPT como en Gemini, y se analizaron sus respuestas. Ambos mostraron potencial para responder y explicar correctamente los códigos generados, pero hay advertencias. El rendimiento general de los LLMs probados, en términos de respuestas correctas, fue del ∼78,2 % para ChatGPT y del ∼69,6 % para Gemini. Aunque existe este potencial para ayudar en el proceso de aprendizaje de programación, las respuestas generadas por los LLMs no deben considerarse totalmente correctas, demandando conocimiento previo de quienes los utilizan para analizarlas y hacer uso de ellas.
Palabras clave: *Enseñanza de programación; Programación para principiantes; ChatGPT; Gemini; LLM.*

1 Introdução

A programação de computadores é um conteúdo fundamental da Ciência da Computação. Aprender a programar, desde a criação de um algoritmo até sua implementação, gerando de fato um programa de computador, é uma tarefa considerada desafiadora. Existem diversos fatores que influenciam nesse desafio, tais como elencados por du Boulay (1986): i) a falta de compreensão dos estudantes em relação à relevância de se aprender a programar, que pode gerar falta de motivação; ii) a falta de conhecimento em relação ao funcionamento dos computadores, que pode dificultar o entendimento e abstração em relação à programação, iii) o desafio de adquirir a sintaxe e a semântica das linguagens de programação, uma vez que o ato de programar é uma tarefa que demanda formalismo na escrita dos códigos; iv) o desafio de entender as diversas estruturas básicas das linguagens de programação, que envolvem raciocínio lógico e abstração; e também iv) todas as ações que envolvem o ato de programar, como testar e procurar e resolver erros (*debugging*), o que demanda experiência para total proveito.

Nos últimos anos surgiram modelos de inteligência artificial classificados como grandes modelos de linguagem (LLM – *Large Language Models*). Esses modelos são baseados em aprendizagem profunda (*deep learning*) com o uso de redes neurais e possuem como objetivo a geração

de texto. Usando a abordagem semi-supervisionada, quando parte dos dados recebe uma rotulagem que permite sua identificação, os LLMs são treinados com um conjunto extenso de dados vindos de diversas fontes. Com o advento dos *transformers*, uma arquitetura de modelos de *deep learning* desenvolvida pela Google, foi viabilizado o treinamento com grandes quantidades de dados de maneira paralela, reduzindo o tempo deste treinamento em comparação com arquiteturas mais antigas (Dengel et al., 2023). Alguns exemplos que fazem uso desse modelo são o ChatGPT (*Chat Generative Pre-trained Transformer*) da OpenAI e o Gemini da própria Google.

Tanto o ChatGPT quanto o Gemini, além de possuírem versões gratuitas, não exigem nenhum conhecimento prévio para poderem ser usados, pois compreendem linguagem natural. Uma característica interessante dos LLMs é a capacidade de geração e explicação de códigos de programação, o que facilita seu uso por estudantes que estejam iniciando nesse assunto. Entretanto, um ponto importante a ser citado é que o uso de LLMs na educação é um tópico controverso (Lo, 2023) pois, um dos primeiros aspectos que deve ser discutido é a qualidade das respostas dadas pelos LLMs ao serem questionados. Tanto a OpenAI (2024a) quanto a Google (2024a) alertam que as respostas emitidas por seus LLMs podem estar erradas e devem ser checadas. O objetivo deste artigo é apresentar uma análise em relação às respostas geradas pelo ChatGPT e pelo Gemini, em suas versões gratuitas, considerando tópicos fundamentais e introdutórios da programação de computadores. Para alcançar o objetivo, este estudo buscou responder às seguintes questões de pesquisa:

- **QP1**: Os LLMs ChatGPT e Gemini são adequados para serem utilizados por estudantes iniciantes de programação?
- **QP2**: É possível utilizar os LLMs ChatGPT e Gemini para o aprendizado em programação?

A ideia principal é analisar as respostas geradas pelos LLMs considerando o contexto de estudantes iniciantes, uma vez que esses não possuem conhecimento necessário para julgar se essas são completamente corretas. A análise foi feita a partir de dois tipos de testes: i) qualitativos, nos quais foram pedidos códigos de programas relacionados com conteúdo introdutório de programação no *prompt* de ambos LLMs testados e, caso a resposta não fosse satisfatória, novas interações eram feitas; ii) quantitativos, nos quais foram inseridos nos *prompts* dos LLMs exercícios também relacionados com o mesmo conteúdo, porém, neste caso, não foram feitas novas interações, independente das respostas obtidas. Em ambos os tipos de testes, foi verificado se as respostas geradas, tanto pelo ChatGPT e pelo Gemini, eram claras e corretas para estudantes iniciantes.

É importante citar que este artigo trata-se de uma extensão do artigo publicado em (Pereira Filho et al., 2023), o qual considerava somente o ChatGPT. Na atual versão, todos testes e análises foram refeitos e comparados com os obtidos anteriormente, assim como foram adicionados os testes e análises obtidos com o Gemini.

O artigo está organizado da seguinte forma: na Seção 2 são apresentados os trabalhos relacionados; na Seção 3 é apresentada a fundamentação teórica com os detalhes em relação ao ChatGPT e o Gemini; na Seção 4 é apresentada a metodologia utilizada nos testes; na Seção 5 são encontrados os testes feitos e seus resultados; na Seção 6 é feita uma discussão em relação aos resultados obtidos; na Seção 7 são apresentadas as limitações e ameação à validade do trabalho e na Seção 8 o artigo é concluído.

2 Trabalhos relacionados

Com lançamentos e evoluções contínuas dos LLMs, novos trabalhos relacionados surgiram, sendo os principais abordados nesta seção. Anterior ao lançamento do ChatGPT e do Gemini para o público em geral, os três principais trabalhos sobre esse assunto eram (Finnie-Ansley et al., 2022), (Finnie-Ansley et al., 2023) e (Sarsa et al., 2022), que são apresentados em seguida.

Nos estudos realizados em (Finnie-Ansley et al., 2022) e (Finnie-Ansley et al., 2023), os autores testaram e avaliaram o Codex, um modelo anterior ao ChatGPT e também desenvolvido pela OpenAI, e que tinha como objetivo específico a geração de códigos de programação. Foram feitos testes com exercícios iniciais de programação relacionados com CS1 (*Computer Science 1*) e CS2 (*Computer Science 2*). Desde então, o Codex foi incorporado ao ChatGPT, não havendo mais uma ferramenta específica para criação de códigos. Em relação ao trabalho apresentado aqui, há três diferenças que podem ser citadas: em ambos trabalhos citados, as explicações do código não foram analisadas, somente os códigos gerados; os trabalhos consideram somente o Codex, antes desse ser incorporado ao ChatGPT; e, especificamente em relação a (Finnie-Ansley et al., 2023), ao ser utilizado o conteúdo de CS2, o foco vai além dos estudantes iniciantes no assunto.

No estudo realizado em Sarsa et al. (2022), a capacidade do Codex de gerar exercícios de programação e explicações de código foi explorada, destacando o potencial como ferramenta para o ensino. Os resultados mostram que a maioria dos exercícios gerados pelo modelo foram coerentes, embora ainda exigisse ajustes antes de ser utilizada em ambientes educacionais. Além disso, as explicações criadas pelo Codex cobriam a maior parte do código, apresentando um grau significativo de precisão, mas com imprecisões que poderiam ser facilmente corrigidas por instrutores ou assistentes de ensino. O trabalho sugere que, apesar de suas limitações, a utilização de modelos como o Codex pode ser um recurso útil para educadores, facilitando o processo de criação de materiais didáticos e oferecendo suporte adicional aos alunos.

Após o lançamento do ChatGPT e do Gemini ao público geral, outros trabalhos relacionados com o tema de ensino de programação surgiram e, entre eles, (Piccolo et al., 2023), abordou o ChatGPT-3.5, que foi testado qualitativa e quantitativamente. No trabalho, foram utilizados 184 exercícios de programação extraídos de um curso de introdução à bioinformática, sendo solicitado a geração de código das respostas na linguagem Python. Os autores, que assumiram uma postura no contexto de programadores inciantes para as interações via *prompt*, estabeleceram um limite de 10 tentativas para cada exercício. O ChatGPT resolveu corretamente 75,5% dos exercícios na primeira tentativa. Para as respostas erradas, *prompts* adicionais foram fornecidos ao ChatGPT, o qual resolveu 97,3% exercícios em até 7 tentativas. Além dos testes no ChatGPT, foram feitos testes com o Bard (versão anterior do Gemini) o qual resolveu corretamente 50% na primeira tentativa e 68,2% em até 10 tentativas. A principal diferença para o presente artigo está no uso da linguagem C para os LLMs gerarem os códigos, enquanto os autores usaram o Python, além do fato dos autores testarem somente de uma forma, permitindo interações com o LLM.

Em (Wermelinger, 2023), o autor testou, qualitativa e quantitativamente, o Github Copilot para compará-lo ao Codex, em relação a geração de códigos, testes, explicações e correções de *bugs*. Foram usados dois conjuntos de problemas para os testes: 8 problemas envolvendo padrões de programação e 7 variações do problema da chuva (*rainfall problem*), sendo ambos considerados pelo autor como conteúdos iniciais de programação do curso de CS1. O autor adotou uma postura

de um aluno iniciante no curso de CS1 para as interações com o Github Copilot. Para questões com respostas erradas, o autor continuou as interações para conduzir o modelo para respostas corretas. Para os problemas de padrões de programação, o Copilot resolveu 50% na primeira tentativa, e para o problema da chuva, ele resolveu 57%. O autor concluiu que o Codex se saiu melhor em ambos tipos de problemas. O diferencial para o presente artigo foi que o artigo citado fez a análise e avaliação da geração de testes e correções de *bugs*.

Em (Dunder et al., 2024), os autores testaram o ChatGPT-3.5 com 127 problemas de programação introdutórios na linguagem Python, selecionados aleatoriamente na Kattis, uma plataforma aberta que disponibiliza e classifica problemas de programação em diferentes níveis. Nesses testes, o ChatGPT acertou apenas 15% problemas, sendo a maioria de nível considerado fácil pela plataforma. Para as 108 tarefas de código consideradas incorretas, ela informou os tipos de erros, sendo: 88 tarefas com a mensagem "*Wrong Answer*" (Resposta Errada), que é quando o código executado não tem sucesso em todos os casos de teste da plataforma; 16 tarefas com a mensagem "*Run Time Error*" (Erro de tempo de execução) e; 9 tarefas tiveram a mensagem "*Time Limit Exceeded*" (Limite de tempo excedido). O diferencial para o presente artigo está no fato do artigo citado não ter analisado as explicações dadas pelo ChatGPT-3.5, analisando apenas a corretude do código gerado. O outro diferencial é o fato do presente artigo analisar e comparar as respostas de dois LLMs, enquanto o referido artigo testou somente o ChatGPT.

Kiesler e Schiffner (2023) testaram o ChatGPT-3.5 e GPT-4 (versão paga do ChatGPT) com 72 tarefas de programação na linguagem Python retiradas da plataforma Codingbat, que possui problemas para iniciantes em programação. O ChatGPT-3.5 acertou 95,8% das tarefas, enquanto o GPT-4 acertou 94,4% das tarefas. Ambas as versões geraram explicações corretas para 97,2% das tarefas. A qualidade das explicações dos códigos gerados foi considerada confiável pelos autores. Para as respostas erradas, os autores forneceram *prompts* adicionais, e as respostas foram analisadas novamente. O artigo citado difere do presente artigo no fato de terem usado a linguagem Python, enquanto no presente trabalho foi usado a linguagem C.

Em (Ouh et al., 2023), os autores testaram o ChatGPT-3.5 e o GPT-4 com 80 exercícios que foram resolvidos na linguagem Java. Foi feita uma divisão em 5 tópicos: básico da linguagem Java, conceitos de orientação a objetos, classes e métodos, exceções e entrada e saída e empacotamento. Como diferencial do presente artigo, o artigo citado fez uso adicional da versão paga do ChatGPT, o GPT-4 e o foi utilizada a linguagem Java para as respostas dos exercícios.

Para o presente trabalho, o ChatGPT e o Gemini foram comparados em relação as suas respostas para o conteúdo introdutório em programação, e na próxima seção é apresentada os detalhes de ambas as ferramentas e sua utilização no contexto educacional.

3 Fundamentação Teórica

O ChatGPT e o Gemini pertencem à categoria de LLMs, modelos probabilísticos de processamento de linguagem natural (Cámara et al., 2023) que podem gerar textos semelhantes aos gerados por humanos e gerar códigos de programação com sucesso (Li et al., 2022). Baseado no contexto, os modelos geram textos a partir de *tokens*, que representam uma sequência de caracteres de ocorrência mais frequente, sendo a unidade básica de texto que o modelo processa (OpenAI,

2024b). Os LLMs são treinados extensivamente utilizando grandes volumes de dados textuais, e algoritmos são utilizados para identificar padrões e relacionamentos entre palavras e frases (Tsai et al., 2023).

O ChatGPT, da OpenAI, tem seu fundamento em aprendizado de máquina (ML - *Machine Learning*), e foi colocado para uso público em dezembro de 2022 (OpenAI, 2024a). Um dos diferenciais trazidos pelo ChatGPT foi a compreensão de linguagem natural, fato que facilita seu uso por usuários não especializados (Aljanabi et al., 2023). Outra característica interessante é que, apesar do ChatGPT ser caracterizado por ser de propósito geral (OpenAI, 2024a), este possui habilidade de criação de códigos de programas em diversas linguagens de programação, assim como possui a capacidade de explicá-los.

Em fevereiro de 2023, foi lançado o Bard da Google (Google, 2023b), para fazer concorrência ao ChatGPT. Um ano depois o Bard foi renomeado para Gemini (Google, 2024a) e também foi construído em cima da arquitetura *transformer* que permite treinar modelos para lidarem com grandes quantidades de texto, entender as relações entre as palavras e, então, prever quais palavras provavelmente virão a seguir (Google, 2021). O Gemini também tem entendimento de linguagem natural e produz textos a partir de perguntas inseridas pelos usuários e, em pouco tempo, ganhou a capacidade de geração e explicação de códigos de programação (Google, 2023a).

Entre os possíveis usos para o ChatGPT e Gemini, a educação é um setor no qual essas ferramentas podem ser aplicadas, apoiando o ensino e aprendizado em diversos domínios, incluindo a programação de computadores (Lo, 2023). Tratando especificamente deste domínio, os LLMs podem apoiar a escrita e a depuração de código, inclusive, dando explicações sobre o conteúdo gerado. Na próxima seção são dados detalhes em relação à criação de código de programação pelo ChatGPT e o Gemini.

3.1 Criação de códigos com ChatGPT e Gemini

A possibilidade de uso do ChatGPT por meio de linguagem natural faz com que esse compreenda o usuário, sem a necessidade do uso de palavras-chave específicas ou uma linguagem pré-definida. Entre os dados usados para o treinamento do modelo do ChatGPT, há um vasto conjunto de códigos escritos por programadores, que são usados para prever o próximo *token* da sequência na criação de novos códigos. Esse fato possibilita aos usuários a inserção de trechos de códigos de programação ou comandos específicos de linguagens e fazer questionamentos sobre eles (Aljanabi et al., 2023). A versão 3.5 do ChatGPT incorporou o code-davinci-002, um modelo específico para tarefas de conclusão de código de programação que, além da capacidade de gerar código a partir de linguagem natural, também é capaz de entender e concluir códigos (OpenAI, 2024b).

O ChatGPT também possui a capacidade de interpretar a execução de um programa, identificar possíveis falhas e sugerir correções adequadas, explicando detalhadamente o motivo do erro e como resolvê-lo. Além disso, ele pode prever a saída esperada de um código fornecido, permitindo aos usuários validar seus programas antes mesmo da execução real. Sua versatilidade vai além da simples depuração, abrangendo a explicação de conceitos fundamentais, como variáveis, estruturas de controle e funções, até temas mais avançados, como complexidade de algoritmos (MacNeil et al., 2022).

De maneira similar, o Gemini permite gerar, encontrar erros e explicar códigos em mais de 20 linguagens de programação. Assim como no ChatGPT, seu uso é a partir de linguagem natural, permitindo ao usuário a inserção de comandos e código no *prompt* para obter respostas coerentes de acordo com a solicitação. De acordo com Google (2024b) o modelo do Gemini foi treinado em uma grande base de código de código público, material específico do Google Cloud e postagens do Stack Overflow. Em casos que a resposta gerada apresente um trecho extenso de um projeto de código aberto, o Gemini menciona a fonte (Google, 2023a).

Tanto o ChatGPT quanto o Gemini possuem interfaces simples, que facilitam a interação com o usuário. Na Figura 1 é apresentada uma demonstração de interação com o ChatGPT 3.5, na qual foi feita uma solicitação simples para geração de um código em C do tipo *"hello world"*, um exemplo de programa simples que somente emite uma mensagem na tela de saída, comumente utilizado nas primeiras aulas de programação, independente da linguagem utilizada. Como resposta, a ferramenta retorna um bloco de código, destacado, com a opção de copiar o código com um único clique. Apesar de não ser solicitado explicitamente, o ChatGPT apresenta uma breve explicação do código gerado.

De maneira semelhante, na Figura 2 é apresentada a interface do Gemini, com a mesma solicitação. O código gerado é o mesmo, a diferença está na tradução para o português da expressão *"hello world"*, no caso, *"olá, mundo"*. Outra diferença encontrada está na explicação. O Gemini apresenta mais detalhes do que o ChatGPT sobre cada trecho de código, além de explicar como executá-lo. Por fim, são adicionadas observações, indicando *links* para referências relacionada com a resposta gerada.

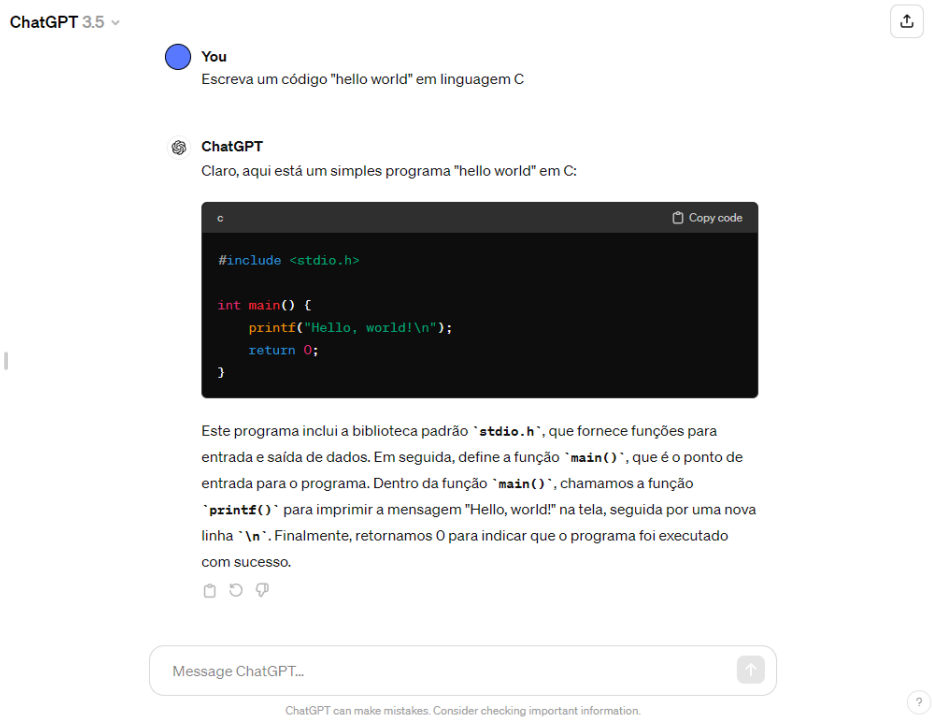

Figura 1: Interface do ChatGPT 3.5 com exemplo de interação, solicitando código em linguagem C.

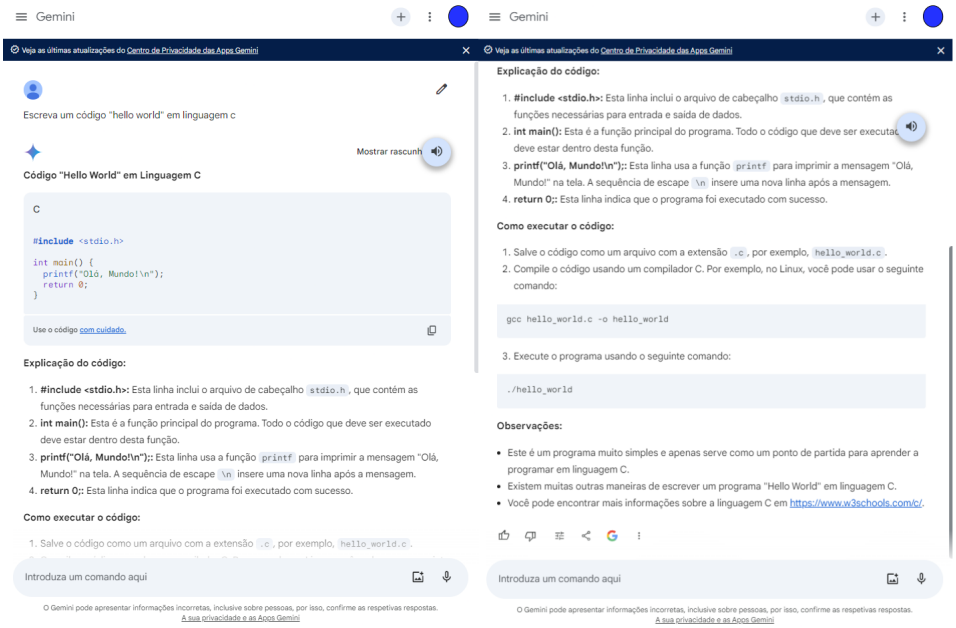

Figura 2: Interface do Gemini com exemplo de interação, solicitando código em linguagem C.

Considerando o funcionamento do ChatGPT e do Gemini, a próxima seção apresenta suas aplicações no processo de ensino e aprendizagem, com suas vantagens e desafios.

3.2 LLMs na Educação

Desde o lançamento do ChatGPT (OpenAI, 2024a) a comunidade científica tem dedicado pesquisas sobre a utilização desses modelos no contexto educacional. Rasul et al. (2023) destacam os benefícios e os desafios do uso do ChatGPT no ensino superior, enfatizando suas vantagens, tais como, a possibilidade de facilitar a aprendizagem adaptativa, *feedbacks* personalizados, apoio a pesquisas e análises de dados, serviços administrativos automatizados e auxílio no desenvolvimento de avaliações inovadoras. Os autores apontam a relação do uso de LLMs, como o ChatGPT, no processo de aprendizagem, focando na autonomia do estudante e no aprendizado ativo, argumentando que as características de suporte e *feedbacks* individualizados a partir de conversas, tornam a ferramenta uma aliada no processo de ensino e aprendizagem do estudante.

Apesar disso, os autores listam as preocupações com a integridade acadêmica, problemas de confiabilidade, incapacidade de avaliar e reforçar competências de nível superior, limitações na avaliação dos resultados de aprendizagem e possíveis vieses e informações falsificadas no processamento de dados. Nesse caso, o uso incorreto da ferramenta, por exemplo, para a criação de conteúdo instantâneo como um atalho no processo de aprendizado, caracteriza um uso antiético da ferramenta. Dessa forma, recomendam a inclusão em currículos de cursos o letramento dos estudantes sobre o uso responsável da ferramenta, fomentando o melhor uso dos LLMs.

Sok e Heng (2023) também trazem os ganhos, as preocupações e algumas recomendações sobre o uso de LLMs em ambiente de aprendizagem. Assim como Rasul et al. (2023), eles apresentam alguns riscos e entre eles estão as informações imprecisas e a dependência excessiva da IA. Contudo, eles reforçam que o uso do ChatGPT deve ser integrado à educação e pesquisa de

forma ética, inclusiva e transparente, em vez de ser proibido. Além disso, eles sugerem que instituições de ensino ofereçam treinamentos para que alunos e professores usem a ferramenta com responsabilidade, mantendo a integridade acadêmica.

Dessa forma, apesar das questões apresentadas serem fundamentais para serem discutidas e exploradas, este artigo foca nos aspectos relacionados ao entendimento do funcionamento e análise crítica das respostas dos LLMs, no caso o ChatGPT e o Gemini. Assim, os resultados apresentados nesta pesquisa podem contribuir para estudos futuros sobre o uso ético e responsável de LLMs no processo de ensino e aprendizagem.

Com o lançamento do ChatGPT e do Gemini, professores e comunidade acadêmica voltaram sua atenção e preocupação sobre seu uso em contexto educacional. Com a possibilidade de estudantes utilizarem ativamente essas tecnologias, é essencial investigar e validar suas funcionalidades. Portanto, é necessário compreender como são dadas as interações entre as plataformas e os usuários para possibilitar orientações mais precisas a respeito da melhor forma de uso. Nesse sentido, na próxima seção é apresentada a metodologia adotada para a realização dos testes e análises deste artigo.

4 Metodologia

A presente pesquisa emprega uma abordagem metodológica mista, combinando métodos qualitativos e quantitativos. A análise qualitativa se baseia em testes dessa natureza que envolvem interações diretas com os LLMs. Complementando esta abordagem, a análise quantitativa se utiliza de métricas de desempenho objetivas, tais como a taxa de acerto em relação às respostas dadas a exercícios propostos, para a validação dos resultados (Gil, 2002). Esta pesquisa, quanto aos seus objetivos, assume caráter exploratório e descritivo da seguinte forma: a natureza exploratória justifica-se pela investigação do uso de LLMs em contexto educacional, buscando aprofundar o conhecimento sobre o tema e estimular a compreensão através da análise de exemplos de códigos gerados pelo ChatGPT e Gemini; já a componente descritiva se manifesta na apresentação e análise dos resultados obtidos, caracterizando o desempenho dos modelos e a adequação das respostas geradas à luz de exemplos analisados, como método para estimular a compreensão do problema em pesquisas exploratórias (Gil, 2002).

Em uma primeira etapa, assim como em (Pereira Filho et al., 2023), foram definidos conceitos com os quais estudantes iniciantes tem contato em disciplinas introdutórias de programação, sendo estes: variáveis e operadores; estruturas de decisão; estruturas de repetição; vetores; matrizes e *strings*. Esses conceitos foram utilizados em dois tipos de testes: qualitativos e quantitativos, que são descritos a seguir. Para os dois tipos de testes, foi pedido que a resposta do LLM deveria ser na linguagem C, que foi escolhida por oferecer uma maior liberdade para o programador, deixando a cargo deste diversos controles para que o código fique totalmente correto, o que configura um bom teste em relação aos LLMs testados, e também é a linguagem utilizada na disciplina introdutória de programação em cursos na instituição dos autores.

Em relação aos testes qualitativos, para cada um dos conceitos citados, foram definidos problemas simples que representam aqueles comumente utilizados de forma introdutória em relação aos conceitos definidos anteriormente. Esses problemas foram inseridos nos *prompts* do ChatGPT

e do Gemini, sendo requisitado que o código gerado fosse na linguagem C. As respostas foram analisadas e, caso necessário, foram permitidas novas interações com o LLM. Em relação ao ChatGPT, todo o experimento foi refeito a partir daqueles apresentados em (Pereira Filho et al., 2023), permitindo a comparação se este apresenta alguma atualização em seu comportamento.

A análise das respostas geradas pelos LLMs, em relação aos testes qualitativos, foi feita em relação à corretude do código apresentado, ou seja, se este não apresentava erros de compilação; à clareza do código gerado, característica que facilita o entendimento por estudantes iniciantes; e a qualidade da explicação fornecida junto ao código. Nessa bateria de testes, caso fosse identificado algum erro na resposta fornecida pelos LLMs, eram feitas novas interações e as novas respostas geradas eram consideradas. O intuito deste teste foi entender se, tanto o ChatGPT quanto o Gemini, apresentam respostas corretas desde a primeira interação e, caso contrário, qual o nível de conhecimento necessário para que a resposta correta seja gerada, o que pode fugir ao escopo do estudante iniciante. Os passos dos testes qualitativos são resumidos a seguir e, na Seção 5 são apresentados os resultados obtidos, de forma detalhada.

Passos dos testes qualitativos:

- Passo 1: definição dos problemas de programação a serem utilizados, os quais são de nível iniciante e consideram os conceitos definidos anteriormente;
- Passo 2: inserção de um problema no *prompt* dos LLMs testados (cada problema foi inserido individualmente em uma nova sessão com o LLM);
- Passo 3: análise da resposta obtida em relação à corretude do código, clareza e explicação;
- Passo 4: dependendo da resposta obtida, foram feitas novas interações dentro da mesma sessão com o LLM, a fim de fornecer mais informações que resultassem em uma resposta mais satisfatória.

Em relação aos testes quantitativos, também como foi feito em (Pereira Filho et al., 2023), foram submetidos ao ChatGPT e ao Gemini problemas de programação e pedido uma resposta em linguagem C, com o intuito de considerar a primeira resposta fornecida por ambos, sem executar novas interações. Os exercícios utilizados foram obtidos em (Mizrahi, 2008), presentes no final dos seguintes capítulos: Capítulo 2 - Operadores, Capítulo 3 - Laços, Capítulo 4 - Comando de decisão e Capítulo 7 - Matrizes (neste capítulo também é tratado o conteúdo de vetores e *strings*). Somente os exercícios relacionados com matrizes foram retirados dos exemplos no decorrer do texto das subseções do Capítulo 7. A escolha de (Mizrahi, 2008), justifica-se por seu uso como referência bibliográfica de apoio nas disciplinas iniciais de programação da instituição dos autores.

Ainda sobre os testes quantitativos, para cada inserção no *prompt*, foi criada uma nova sessão no ChatGPT e no Gemini, para manter um isolamento em relação às respostas obtidas. As inserções dos problemas nos *prompts* dos LLMs testados foram precedidas pela sentença *"Escreva um programa em linguagem C para Linux..."*, para que a resposta fosse gerada na linguagem alvo dos experimentos e para que fosse compatível com o ambiente de testes utilizado (Replit, 2024), que consiste em um ambiente Linux.

Os passos seguidos nos testes quantitativos estão descritos a seguir.

Passos dos testes quantitativos:

- Passo 1: definição dos problemas de programação a serem utilizados, os quais foram extraídos de (Mizrahi, 2008), Capítulos 2, 3, 4 e 7;
- Passo 2: inserção de um problema no *prompt* dos LLMs testados, em sessões separadas, precedido pelo comando: *"Escreva um programa em linguagem C para Linux..."*;
- Passo 3: análise da resposta obtida em relação à compilação e execução bem-sucedida, corretude do código gerado e explicação do código. Novas interações não eram permitidas.

Nas próximas seções são apresentados detalhes dos testes realizados assim como os resultados obtidos.

5 Testes e resultados

Nas próximas subseções são apresentados e discutidos os resultados dos testes qualitativos e quantitativos em relação a cada tópico considerado a respeito do conteúdo de programação introdutória, considerando os dois LLMs utilizados neste trabalho.

5.1 Testes qualitativos

Abaixo são descritos os resultados obtidos em relação aos testes qualitativos tanto com o ChatGPT quanto com o Gemini em relação a cada um dos conceitos definidos como introdutórios de programação. Os testes apresentados em (Pereira Filho et al., 2023), trabalho no qual foi só considerado o ChatGPT, foram refeitos, permitindo assim a comparação com as novas respostas do próprio ChatGPT e com o resultado obtido com o Gemini.

5.1.1 Variáveis e operadores

O conceito de variável é um dos primeiros a ser apresentado aos estudantes iniciantes em programação. Uma variável é uma área de memória que possibilita o armazenamento de valores de diversos tipos, tais como valores inteiros, reais e caracteres, para posterior uso no programa. Esses valores armazenados em variáveis podem ser usados em operações matemáticas como adição, subtração, multiplicação e divisão. Na linguagem C, as variáveis possuem tipo, que define o tipo de valor que essas podem armazenar (sendo os básicos da linguagem o tipo *int* para valores inteiros, o tipo *float* para valores reais e o tipo *char* para caracteres).

Em relação aos testes propostos aos LLMs testados e o conteúdo de variáveis e operadores, o problema inserido no *prompt* foi: *"faça um programa em linguagem C que recebe dois valores e apresente na tela a soma, a subtração e multiplicação e a divisão desses dois valores"*. É interessante notar que o enunciado do problema não menciona nenhuma informação em relação ao tipo de variável que deve ser usado, deixando a cargo do programador. Da mesma forma não é feita nenhuma ressalva em relação à divisão, que requer que o denominador seja diferente de 0.

Para esse problema proposto, a resposta gerada pelo ChatGPT estava clara e totalmente correta assim como a explicação apresentada. O código possuía inclusive um controle para que, caso o segundo valor inserido for igual a zero, a divisão não é calculada. Em comparação, no

teste realizado em (Pereira Filho et al., 2023), a resposta gerada pelo ChatGPT não atendia essa restrição, o que demandava um novo questionamento no *prompt* para que isso fosse corrigido.

Em relação ao teste feito com o Gemini, o código gerado estava correto e claro, assim como sua explicação. Entretanto, não havia controle em relação à possibilidade de divisão por 0. Ao ser questionado com *"existe alguma restrição em relação ao valor2?"*, foi respondido que no código não havia nenhuma restrição, porém seria importante controlar para que este valor não seja 0 antes da divisão e apresentou um exemplo de um trecho de código com esse controle.

Com base nos resultados deste teste, é possível afirmar que o ChatGPT apresentou uma resposta mais completa, uma vez que já considerava a possibilidade de uma divisão por zero em sua solução, demonstrando um tratamento preventivo para esse tipo de erro. Esse comportamento representa uma evolução em relação ao observado em (Pereira Filho et al., 2023), no qual essa verificação não foi realizada automaticamente. Por outro lado, o Gemini não abordou essa restrição em sua resposta inicial, sendo necessário um questionamento adicional para que o modelo incluísse essa verificação no código.

5.1.2 Estrutura de decisão

Uma estrutura muito comum em todas as linguagens de programação é a estrutura de decisão, também conhecida como estrutura condicional. Nas linguagens de programação é muito comum a utilização das palavras-chave *if-else* para esse tipo de estrutura. Essa estrutura permite adicionar um teste lógico (tal como $a > b$) e, a partir do resultado deste teste (que pode resultar em verdadeiro ou falso), uma instrução pode ou não ser executada. Por exemplo, em um programa hipotético de um jogo que possui duas variáveis, *pt1* e *pt2*, nas quais são armazenadas a pontuação de dois jogadores, no final pode haver uma estrutura de decisão que testa se o $pt1 > pt2$ e, caso o resultado seja verdadeiro, declara o primeiro jogador como o vencedor. Caso contrário, podem ser feitos novos testes lógicos para determinar se houve empate (caso $pt1 = pt2$) ou se o segundo jogador foi o vencedor (caso $pt1 < pt2$).

Neste teste, foi inserido no *prompt* dos LLMs o problema: *"faça um programa que recebe 3 valores inteiros e os apresente em ordem crescente"*. É interessante citar que há algumas formas de se resolver esse problema. Uma delas seria receber e armazenar três valores inteiros em três variáveis, tais como *n1*, *n2* e *n3* e, a partir de testes com estruturas de decisão, definir em qual variável está armazenado o menor valor dos três, o valor do meio e o maior valor entre eles e, então, apresenta-los na sequência crescente (*n1*, *n2* e *n3*, ou *n2*, *n1* e *n3*, etc). Outra forma seria executar trocas dos valores entre as variáveis, desde que *n1* fique com o menor valor, *n2* fique com o valor do meio e *n3* fique com o maior valor, e apresentá-los na sequência *n1*, *n2* e *n3*.

Em relação ao ChatGPT, em (Pereira Filho et al., 2023), a resposta apresentada utilizava troca de valores entre as variáveis, porém utilizava também funções e ponteiros, conceitos os quais um estudante iniciante não teria familiaridade. Foram necessárias algumas interações com o ChatGPT para que este apresentasse uma resposta correta que não utiliza-se funções e ponteiros. É importante citar que algumas interações estariam fora do escopo do estudante iniciante.

Para o teste refeito para este artigo, a primeira resposta obtida utilizava a troca entre variáveis para a resolução do problema de forma correta. Em seguida, para testar variações possíveis da resposta gerada, foi feita a seguinte pergunta: *"repita o código anterior, mas sem usar troca de va-*

lores entre as variáveis". A resposta gerada era muito parecida com a anterior e ainda utilizava a troca entre variáveis. Foi feito um novo questionamento: *"ainda houve troca de valores. gostaria que os valores das variáveis não se alterassem em nenhum momento do código"*. Esse questionamento gerou uma resposta mais complexa, que utilizava um vetor e ordenava-o utilizando o método *bubble sort*. Foi feita então a seguinte indagação: *"não seria possível usar somente um encadeamento de ifs e elses para resolver, em vez de trocar o valor das variáveis?"*. Após esse novo questionamento, a resposta sem a troca de valores entre as variáveis foi gerada corretamente. É interessante perceber que, mesmo sendo claro na pergunta feita, como informando que não deveria haver troca entre as variáveis, foi gerada resposta que ainda utilizava-se dessa operação. Ao considerar o ponto de vista de um estudante iniciante, isso pode acarretar confusão.

Em relação ao teste com o Gemini, o primeiro código gerado utilizava a resolução sem a troca de valor entre variáveis, mas com três variáveis auxiliares para esse fim (variáveis *menor*, *medio* e *maior*). Porém, a resposta gerada estava errada ao receber os valores em ordem decrescente (3, 2, 1, por exemplo). Foi, então, inserido em seu *prompt*: *"seu código dá a resposta errada"*, o que gerou uma nova resposta e essa também possuía erro, falhando ao receber valores que continham valores repetidos (por exemplo 1, 1, 2). É importante citar que, mesmo no texto da explicação gerada pelo Gemini, em um exemplo de utilização do programa gerado por este, havia erro. O Gemini indicou em sua explicação que para a entrada 10, 5 e 2, a resposta deveria ser 5, 10 e 2. Foi então questionado *"o exemplo citado está errado."*, o que gerou uma nova explicação com um exemplo correto, mas sem gerar novo código. Após isso foi pedido uma nova versão do código, mas utilizando a troca de valores entre as variáveis: *"repita o programa, mas agora quero que seja feita a troca de valores entre as variáveis. exemplo, o menor deverá ficar em valor1, o médio em valor2 e o maior em valor3"*. O Gemini continuou apresentado como resposta um código que ainda utilizava as variáveis *menor*, *medio* e *maior* e que continuava apresentando erro em alguns testes. Esse comportamento se repetiu por diversas interações, sempre gerando códigos que não se comportavam como o pedido e possuíam erro de lógica na geração do resultado desejado (somente uma das interações gerou um código com a resposta correta, as outras haviam testes que falhavam). É interessante notar que o problema proposto não é complexo, e o esperado era que a resposta estivesse sempre correta do ponto de vista da saída gerada pelo código, mesmo que as exigências da forma de resolução do problema não fossem cumpridas (uso ou não de troca de valores entre as variáveis), e isso não ocorreu com o Gemini.

Comparando os testes realizados entre os dois LLMs, observa-se que o ChatGPT demonstrou uma maior capacidade de adaptação às exigências formuladas no *prompt*, conseguindo ajustar sua resposta de acordo com as solicitações adicionais feitas ao longo das interações. Enquanto que, o Gemini apresentou um desempenho menos consistente, pois além de gerar uma resposta inicial incorreta para determinados casos de entrada, mostrou dificuldades em corrigir seus próprios erros ao longo das interações.

5.1.3 Estrutura de repetição

Nas linguagens de programação, quando é necessário que uma ou mais instruções sejam repetidas *n* vezes, são utilizadas as estruturas de repetição (também conhecidas como laços de repetição). As estruturas de repetição mais comumente encontradas em praticamente todas as linguagens de programação são o *for* e o *while*, sendo que o segundo possui também uma variação, o *do-while*.

Nessas estruturas é executado um teste lógico e, caso o resultado desse teste seja verdadeiro, uma ou mais instruções são executadas novamente, até que o teste lógico resulte em falso. Em geral, a estrutura *for* é mais usada quando é preciso repetições por um número conhecido de vezes (por exemplo, 10 vezes). Por outro lado, a estrutura *while*, e sua variação *do-while*, são mais usadas quando o número de repetições é imprevisível como, por exemplo, quando as instruções devem ser repetidas até que o usuário entre com um valor válido.

Para o teste relacionado às estruturas de repetição foi proposto aos dois LLMs o seguinte problema: *"faça um programa que simule dois semáforos de trânsito. O primeiro semáforo muda de estado (vermelho para verde ou vice-versa) a cada x segundos. O segundo semáforo muda de estado a cada y segundos, sendo $x < y$. Indique, para um período de 1000 segundos, quantos segundos os dois semáforos estarão no mesmo estado (vermelho ou verde) ao mesmo tempo. Considere os semáforos iniciando iguais."*.

Um fato importante sobre esse problema é que, para resolvê-lo, é preciso usar pensamento abstrato e criatividade, uma vez que não é necessário fazer um código que realmente execute por 1000 segundos, mas sim uma contagem até 1000 simulando a passagem do tempo. Entender e praticar conceitos de abstração como este é importante para estudantes iniciantes. Em (Pereira Filho et al., 2023), o código gerado pelo ChatGPT estava correto, entretanto não havia controle para garantir que $x < y$ e que ambos fossem maiores que 0. Para a lógica do exercício, não há problema em relação aos valores de x e y não obedecerem o que é pedido, porém esses valores devem ser maiores que 0. A restrição proposta tem apenas o intuito de exercitar o estudante em relação ao uso de regras em sua resposta. Após algumas interações, o código gerado estava correto.

No teste refeito, a resposta gerada pelo ChatGPT foi diferente, pois o código gerado possuía detalhes que dificultavam a compreensão como, por exemplo, as variáveis criadas para simular os semáforos foram nomeadas *semaforo1_verde* e *semaforo2_verde*; foram utilizadas duas variáveis diferentes; entre outras. Além disso, o código gerado não apresentava a resposta correta e, ao ser questionado com *"o código acima não dá a resposta correta"*, foi gerada uma nova resposta, ainda mais confusa, pois utilizava duas funções, uma para calcular o máximo divisor comum e outra para calcular o mínimo múltiplo comum entre dois números. O código gerado não possuía nenhuma referência a nenhum dos dois semáforos, estando completamente errado. A partir desse ponto, foram feitas várias interações com o ChatGPT na tentativa de fazê-lo gerar o código correto, mas esse não se comportava mais corretamente, gerando códigos sem sentido em relação ao que era pedido. Somente após a inserção de um código de exemplo que este gerou uma resposta totalmente correta. Do ponto de vista de um estudante iniciante, esse teste comprovou que é possível a geração de respostas equivocadas pelo ChatGPT e que este pode atingir um estado no qual este se torna incapaz de gerar respostas corretas, o que indica um problema que pode gerar confusão em quem a requisita.

Em relação ao Gemini, a resposta gerada estava correta, porém no código gerado era considerado que cada semáforo tinha quantidades de segundos distintas que esses permaneciam em cada cor. Trata-se de uma complicação adicional ao problema que não constava no enunciado inserido no *prompt*. Devido a essa complicação adicional, o restante do código, apesar de correto, também considerava esses dois tempos diferentes para cada semáforo, o que complicaria a análise de um estudante iniciante, caso esse não entendesse essa diferença. Em seguida, foi inserido *"repita, mas quero que os valores de x e y sejam inseridos pelo usuário"*. A nova versão gerada

estava correta, mas ainda considerava dois tempos distintos para os estados dos dois semáforos. Foi então inserido *"em nenhum momento foi colocado no enunciado que cada semáforo tem ciclos diferentes para verde e vermelho, somente que mudam em x e y segundos, respectivamente."* e foi pedida uma nova resposta. A última resposta gerada estava correta e de acordo com o enunciado.

Nesse teste, é possível afirmar que o Gemini teve um desempenho superior ao do ChatGPT, pois conseguiu gerar uma resposta correta logo na primeira tentativa, ainda que tenha introduzido uma complicação adicional ao problema, ao assumir tempos distintos para cada semáforo. Apesar desse detalhe não constar no enunciado original, o código gerado estava funcional e coerente dentro da lógica proposta. Com algumas interações adicionais, foi possível corrigir esse equívoco conceitual, e o modelo se ajustou de forma satisfatória ao que era solicitado. Enquanto que o ChatGPT apresentou, após algumas interações, um estado em que gerava respostas sem sentido em relação ao que era pedido em seu *prompt*.

5.1.4 Vetores

Vetores são estruturas de dados, comuns nas linguagens de programação, que permite o armazenamento de diversos valores de um mesmo tipo, eliminando a necessidade de se criar diversas variáveis independentes.

O acesso a cada um dos valores de um vetor é feito por meio de um índice que indica a posição do valor dentro deste. Por exemplo, na linguagem C, um vetor pode ser declarado da seguinte forma: *int nums[5]*. Essa instrução declara um vetor com 5 posições que podem armazenar valores do tipo *int*. Para acessar uma posição específica, é utilizado o índice, tal como: *nums[2] = 25*, ou seja, essa instrução indica que o valor 25 será armazenado na posição com índice 2 (sendo que a primeira posição dos vetores é a posição 0).

Como teste relacionado com vetores, foi inserido no *prompt* do ChatGPT e do Gemini o seguinte: *"escreva um programa em C que receba dez valores diferentes do tipo float e os apresente na tela, primeiro na ordem em que foram inseridos e depois na ordem contrária de inserção. Utilize vetor"*. O código gerado pelos dois LLMs testados estavam corretos, assim como sua explicação. Ambos apresentaram códigos simples, claros e corretos. O ChatGPT já tinha tido esse comportamento nos testes apresentados em (Pereira Filho et al., 2023).

5.1.5 Matrizes

Nas linguagens de programação, vetores com mais de uma dimensão são chamados de matrizes. Essas se comportam da mesma forma que vetores, permitindo o armazenamento de vários valores de um mesmo tipo, porém, a diferença é encontrada ao se acessar uma posição: é necessário dois índices, um de linha e outro de coluna. Por exemplo, a inserção de um valor em uma posição específica de uma matriz *m* deve ser feita da seguinte forma: *m[0][3] = 25*, no qual a posição na linha 0 e coluna 3 recebe o valor.

Para o teste, foi proposto para o ChatGPT e o Gemini o seguinte: *"faça um programa em C que utilize uma matriz e preencha-a valores aleatórios entre 0 e 500. Em seguida o programa deve apresentar o maior e o menor valor encontrados na matriz"*. O código gerado pelo ChatGPT, tanto para (Pereira Filho et al., 2023) quanto nos testes refeitos, estavam corretos e claros. A única diferença entre eles foi que o primeiro apresentou uma matriz 3x3 e nos testes refeitos foi usada

uma matriz 5x5, detalhe que não altera em nada a resolução do problema. O código gerado pelo Gemini também estava correto e claro, sendo bem parecido com o gerado pelo ChatGPT. No caso do Gemini, foi usada uma matriz 3x4, o que também não influencia na resposta.

5.1.6 Strings

Em programação, uma sequência de caracteres pode representar, por exemplo, um nome completo, um endereço, etc, é chamada de *string*. Na linguagem C, uma *string* é armazenada em um vetor do tipo *char*, que permite o armazenamento de um caractere por posição desta.

Como teste referente a esse assunto, foi proposto para os dois LLMs testados: *"escreva um programa em C que recebe duas palavras e testa se essas palavras são iguais"*. Tanto em (Pereira Filho et al., 2023) quanto nos testes refeitos, o código e a explicação gerados pelo ChatGPT eram corretos, porém, possuíam um problema importante: era feito o uso da função *scanf* utilizando o parâmetro *%s* para o recebimento de caracteres que devem ser digitados pelo usuário do programa. Essa função é conhecidamente insegura com o parâmetro *%s*, pois esta não controla se a quantidade de caracteres digitados ultrapassa o tamanho do vetor que irá recebê-los, o que possibilita um problema de estouro de memória *buffer overflow*. Esse problema, além de fazer com que o programa não funcione como o esperado, pode ser explorado por programadores maliciosos, possibilitando inclusive a invasão do computador que o executa (Matthews et al., 2021). Foi então inserida a seguinte questão: *"o que pode ser melhorado no código anterior?"*. O ChatGPT informou que a função *scanf* poderia ser substituída pela *fgets*, que realmente é uma opção à primeira e possibilita controlar a quantidade de caracteres recebidos. Porém, na explicação, o ChatGPT informou que a sugestão de substituição da função foi feita pelo motivo que a função *scanf* para de receber uma sequência de caracteres ao receber um espaço em branco, comportamento que não ocorre com a *fgets*, não mencionando a questão de segurança da primeira função. Foi então questionado *"mas e a questão de segurança relacionada à função scanf usando "%s"?"*. Essa questão gerou uma explicação correta em relação à falta de segurança da função *scanf* com o parâmetro *%s* e a sugestão segura da *fgets*.

Em relação ao Gemini, o primeiro código gerado estava correto, mas também usava a função *scanf* com o parâmetro *%s*, também de forma insegura. Foi, então, feita a mesma pergunta em relação ao que poderia ser melhorado no código. A resposta para essa pergunta foi confusa, pois o Gemini citou o problema da função somente receber caracteres até o espaço em branco e sugeriu, para resolver essa questão, o uso de uma outra função (*strcasecmp*), a qual não tem esse objetivo, mas sim auxiliar na comparação entre palavras. O Gemini, inclusive, gerou um novo código apresentando o uso dessa nova função. Ainda na mesma explicação, o Gemini comentou que às vezes não é necessário comparar palavras inteiras e apresentou uma terceira versão do código que comparava somente os 5 primeiros caracteres das palavras, sem que essa versão tenha sido requisitada. É interessante que, na explicação dessa terceira versão, o Gemini explicou a questão relacionada com a segurança computacional do código apresentado e sugeriu a utilização da função *fgets* ou da função *getline*, sendo que essa segunda não é uma função nativa da linguagem C, apesar de existirem versões desta implementada para essa linguagem. Após isso, foi questionado *"apresente uma nova versão do programa, mas que seja seguro"*. Foi então gerada uma versão do programa que usa *fgets* e controla a quantidade de caracteres inserida, evitando o problema de segurança.

É importante notar que em várias das interações com os LLMs testados, para que fosse obtida uma versão adequada do código, era demandado conhecimento prévio e específico em relação à insegurança do uso da função *scanf* no recebimento de caracteres de uma *string*. Um estudante iniciante de programação não teria esse conhecimento e aceitaria a primeira resposta dos LLMs como totalmente adequada.

5.2 Testes quantitativos

Nos testes quantitativos, foram inseridos exercícios de programação nos LLMs e a resposta obtida foi analisada, sem nenhuma interação adicional. Nas tabelas desta seção, na coluna Tópicos são apresentados os conceitos de programação testados; na coluna Total é apresentado o total de exercícios para cada tópico; na coluna Compilou/Executou são apresentados quantos exercícios submetidos não tiveram erro de compilação ou execução; na coluna Correção é apresentada a quantidade de exercícios que apresentaram resposta correta, ou seja, a saída do programa atendeu o que foi pedido no enunciado; e na coluna Explicação é apresentada a quantidade de exercícios que possuíam explicação correta. A única interação que foi feita, além da inserção do exercício, acontecia quando a explicação do código era muito simples. Nesse caso, foi inserido no *prompt* a frase *"Explique o código."* para estimular uma explicação mais elaborada.

Assim como em (Pereira Filho et al., 2023), o ambiente de testes utilizado foi a plataforma Replit (Replit, 2024), na qual é possível fazer a compilação usando o compilador GCC (*GNU Compiler Collection*) na versão 11.3.0, e a execução no sistema operacional Linux Ubuntu 20.04. Os exercícios de programação propostos, obtidos e adaptados de (Mizrahi, 2008), foram os mesmos para ambos LLMs. Para comparação com o este trabalho, os resultados dos testes quantitativos apresentados em (Pereira Filho et al., 2023) podem ser vistos na Tabela 1.

Tabela 1: Testes quantitativos com ChatGPT apresentados em (Pereira Filho et al., 2023).

Tópicos	Total	Compilou/Executou	Correção	Explicação
Variáveis e Operadores	9	9 (100%)	9 (100%)	9 (100%)
Estruturas de Repetição	6	6 (100%)	6 (100%)	6 (100%)
Estruturas de Decisão	10	10 (100%)	10 (100%)	10 (100%)
Strings	15	15 (100%)	8 (∼53,3%)	10 (∼66,6%)
Vetores e Matrizes	6	6 (100%)	5 (∼83,3%)	5 (∼83,3%)
Totais gerais	**46**	**46 (100%)**	**38 (∼82,6%)**	**40 (∼86,9%)**

5.2.1 Testes refeitos com o ChatGPT

Assim como em (Pereira Filho et al., 2023), considerando o ChatGPT, foram testados 9 exercícios sobre variáveis e operadores; 6 exercícios sobre estruturas de repetição; 10 exercícios sobre estruturas de decisão; 15 exercícios sobre *strings* e 6 exercícios sobre vetores e matrizes. Os novos resultados obtidos são apresentados na Tabela 2. Em resumo, de um total de 46 questões apresentadas para o ChatGPT, 100% foram resolvidas com sintaxe correta, compilando e executando sem erros, mesmo resultado do artigo anterior; 36 questões, ou ∼78,2%, atenderam corretamente os requisitos dos enunciados, apresentando saídas corretas para as entradas (resultado menor que os ∼82,6% obtidos anteriormente) e 41 exercícios possuíam explicação adequada, representando ∼89,1% do total (resultado melhor que os ∼86,9% do trabalho anterior).

Tabela 2: Testes quantitativos refeitos com ChatGPT.

Tópicos	Total	Compilou/Executou	Correção	Explicação
Variáveis e Operadores	9	9 (100%)	8 (∼88,8%)	9 (100%)
Estruturas de Repetição	6	6 (100%)	5 (∼83,3%)	6 (100%)
Estruturas de Decisão	10	10 (100%)	10 (100%)	10 (100%)
Strings	15	15 (100%)	7 (∼46,6%)	10 (∼66,6%)
Vetores e Matrizes	6	6 (100%)	6 (100%)	6 (100%)
Totais gerais	**46**	**46 (100%)**	**36 (∼78,2%)**	**41 (∼89,1%)**

Em relação aos resultados por tópicos, na Tabela 2 são apresentados os resultados dos 9 exercícios propostos sobre variáveis e operadores e, pode-se observar que apenas um exercício teve resposta incorreta (coluna Correção). Nesse problema em que houve o erro, o programa solicita ao usuário o número total de cabeças e pés de patos e coelhos em um cercado, e então calcula a quantidade de cada tipo de animal. O ChatGPT cometeu um erro na formulação da equação para determinar o número de coelhos, gerando a Equação (1), ao invés de gerar a Equação (2), que está correta.

$$\text{Número de coelhos} = \frac{2 \times \text{Total de cabeças} - \text{Total de pés}}{2} \tag{1}$$

$$\text{Número de coelhos} = \frac{\text{Total de pés} - 2 \times \text{Total de cabeças}}{2} \tag{2}$$

Em relação ao tópico estruturas de repetição, de um total de 6 exercícios, o ChatGPT respondeu apenas um incorretamente. Nesse problema, o programa deveria calcular um termo *n* da sequência de Fibonacci a partir da entrada do usuário. O ChatGPT apresentou resposta incorreta na identificação de termos da sequência de Fibonacci, devido a subtração de uma unidade do termo solicitado pelo usuário, ou seja, ao solicitar o termo *n*, o programa retornava o termo *n - 1*. A Figura 3 mostra um trecho do código gerado pelo ChatGPT, com a subtração incorreta destacada na linha 22. A correção do código é relativamente simples, mesmo no contexto de conteúdo introdutório de programação.

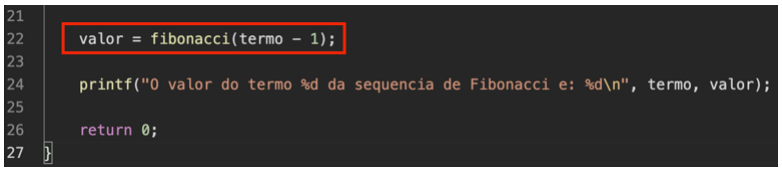

Figura 3: Trecho do código gerado pelo ChatGPT para o cálculo de um termo da sequência de Fibonacci.

Em relação às estruturas de decisão, de 10 exercícios, nenhum teve resposta errada em nenhum dos itens considerados. O mesmo comportamento foi obtido em relação a vetores e matrizes: de um total de 6 exercícios, todos compilaram e executaram, estavam corretos e sua explicação também era correta.

Em relação ao tópico de *strings*, foram propostos 15 exercícios, dos quais, todos aqueles gerados pelo ChatGPT compilaram e executaram, semelhante ao observado em (Pereira Filho et al., 2023). Entretanto, apenas 7 respostas tiveram soluções totalmente corretas (a respeito do

código e da explicação gerados). Oito respostas apresentaram saídas incorretas, sendo 6 devido ao uso inseguro de *scanf* para leitura de *strings* e 2 com outros erros de saída. É importante notar que, nos testes refeitos, o ChatGPT forneceu explicações menos detalhadas, necessitando de solicitações adicionais para obter explicações mais completas, ao contrário do ocorrido em (Pereira Filho et al., 2023).

No primeiro código sobre *strings* com a saída errada, o programa deveria inicializar uma *string s* com *n* repetições do caractere armazenado na variável *ch*, sendo todos esses valores obtidos como entrada do usuário. Entretanto, a entrada da *string s* foi omitida pelo programa. No segundo, o programa deveria receber uma *string* de entrada, um tamanho desejado *n* e um modo (com valores possíveis iguais a 0, 1 ou 2) para adicionar espaços em branco, atingindo o tamanho *n*. No modo 0 os espaços são adicionados no final da *string*, no modo 1 os espaços devem ser centralizados na *string*, e no modo 2 os espaços são adicionados no início desta. No código gerado pelo LLM, os modos 1 e 2 apresentaram comportamento divergente do especificado, adicionando espaços entre *strings* concatenadas ao invés de preencher a *string* de entrada conforme solicitado.

Em relação ao tópico de estruturas de repetição, de um total de 6 exercícios, o ChatGPT utilizou o laço *for* em 2 respostas, utilizou o laço *while* em 2 respostas e em 2 questões fez o uso de recursão, conceito considerado avançado. Nas soluções dos exercícios de estruturas de decisão, foram usadas funções em 3 respostas, outro conceito também considerado avançado. O tópico de matrizes não apresentou erros de compilação e execução, correção ou explicação, superando os resultados obtidos no artigo anterior (Pereira Filho et al., 2023). Porém, em 3 soluções, o ChatGPT fez uso de funções para encapsular as lógicas das soluções, conceito considerado avançado para estudantes iniciantes. O uso desses recursos podem dificultar o entendimento por programadores iniciantes, uma vez que eles podem ainda não ter tido contato com tais estruturas até esse estágio do aprendizado.

5.2.2 Testes com o Gemini

Na Tabela 3 são mostrados os resultados para os testes feitos no Gemini, nos quais foram utilizados os mesmos exercícios de testes do ChatGPT. Em resumo, de um total de 46 exercícios propostos, o Gemini apresentou códigos que compilaram e executaram sem erro 44 respostas (∼95,6%), estavam corretas 32 respostas (∼69,6%) e a explicação era adequada em 42 respostas (∼91,3%).

Tabela 3: Testes quantitativos Gemini.

Tópicos	Quantidades	Compilou/Executou	Correção	Explicação
Variáveis e Operadores	9	8 (∼88,8%)	7 (∼77,7%)	9 (100%)
Estruturas de Repetição	6	6 (100%)	5 (∼83,3%)	6 (100%)
Estruturas de Decisão	10	10 (100%)	9 (90%)	10 (100%)
Strings	15	15 (100%)	7 (∼46,6%)	12 (80%)
Vetores e Matrizes	6	5 (∼83,3%)	4 (∼66,6%)	5 (∼83,3%)
Totais gerais	**46**	**44 (∼95,6%)**	**32 (∼69,6%)**	**42 (∼91,3%)**

Dos 9 exercícios propostos sobre variáveis e operadores, 8 compilaram e executaram sem erros, 7 apresentaram soluções corretas e 9 incluíam explicações adequadas. O Gemini também errou o exercício que solicitava ao usuário o número de cabeças e pés de patos e coelhos em um cercado para calcular a quantidade de cada animal, apresentando equação errada (conforme mostrado em (3)). É importante notar que o ChatGPT também elaborou esse exercício incorretamente.

No exercício com problema de compilação, o código gerado pelo Gemini continha a função *getch* encontrada na biblioteca *conio.h*, que não é padrão da linguagem C e, portanto, não é suportada em ambiente Linux.

$$\text{Número de patos} = \frac{\text{Total de pés} - 2 \times \text{Total de cabeças}}{2} \tag{3}$$

Dos 6 exercícios de estruturas de repetição propostos, todos foram compilados, executados e possuíam explicação correta, sendo que, apenas um apresentou solução incorreta. O exercício incorreto solicitava ao usuário um número inteiro entre 3 e 18, representando a soma desejada ao lançar três dados simultaneamente. A probabilidade de obter essa soma deveria ser calculada pela fórmula *P = (n1 / n2) * 100*, na qual *n1* é o número de resultados cuja soma dos três dados é igual ao número fornecido pelo usuário, e *n2* é o número total de resultados possíveis. O erro cometido pelo modelo Gemini ocorreu na declaração da variável probabilidade como um tipo inteiro (*int*), conforme ilustrado na Figura 4. Como probabilidade é calculada com um valor que não é inteiro, a variável deveria ter sido declarada como um tipo de ponto flutuante, por exemplo *float* ou *double*. A correção, em um contexto introdutório de programação, é simples e requer apenas o conhecimento dos tipos de dados básicos da linguagem C.

Figura 4: Trecho do código gerado pelo Gemini para calcular a probabilidade de um valor dado o lançamento de 3 dados.

Em relação às estruturas de decisão, dos 10 exercícios propostos, todos compilaram e executaram com sucesso. Nove foram resolvidos corretamente, com explicações adequadas, sendo que o único exercício incorreto envolvia a determinação se um número fornecido pelo usuário era par ou ímpar, imprimindo 1 para ímpar ou 0 para par. Embora o Gemini tenha apresentado uma lógica correta para identificar a paridade, a saída divergiu do enunciado: em vez de imprimir 0 ou 1, o programa exibiu a frase *"O número N é ímpar/par"*. Essa discrepância, facilmente corrigida em um contexto introdutório de programação, requer apenas a substituição da mensagem de saída pelos valores numéricos especificados no enunciado.

Sobre *strings*, de um total de 15 exercícios, todos compilaram e executaram, 7 estavam corretos e 12 possuíam explicação precisa. O uso de funções inseguras (`scanf` e `gets`) foi observado em 6 exercícios. Entre os 2 exercícios incorretos, em um deles, o Gemini cometeu o mesmo erro que o ChatGPT para o exercício que solicitava um programa que deveria inicializar uma *string* com repetições de um mesmo caractere, e tais parâmetros deveriam ser recebidos como entrada do usuário, o que não ocorreu. Em um segundo exercício incorreto, que consistia em inverter uma *string* fornecida pelo usuário, o programa gerado pelo modelo Gemini retornava sempre uma *string* vazia. Isso ocorria devido ao acesso do programa a um índice inválido, ou seja, fora dos limites da *string* de entrada. A resolução desse problema demandaria uma depuração mais aprofundada, possivelmente além do contexto introdutório de programação. É importante citar que o assunto de *strings* é considerado mais complexo na linguagem C, se comparado com outras linguagens, devido à própria natureza desta. A linguagem C oferece maior liberdade ao pro-

gramador, pois não dispõe de muitas soluções prontas, transferindo assim toda a responsabilidade e complexidade do desenvolvimento para aquele que escreve o código fonte.

Dos 6 exercícios propostos envolvendo vetores e matrizes, 5 foram compilados e executados com sucesso. Quatro apresentaram soluções corretas, enquanto 5 continham explicações corretas. Em um dos exercícios respondidos incorretamente pelo Gemini, era solicitado a geração de 50 apostas de loteria, cada uma com 6 números distintos. Em vez disso, o modelo gerou 50 sequências contendo os 60 números possíveis em ordem aleatória, conforme ilustrado na Figura 5. O outro erro foi devido a não compilação do código, por conta do uso de variáveis não declaradas.

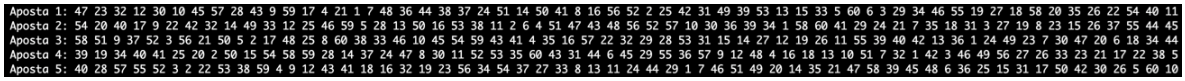

Figura 5: Trecho da saída gerada pelo código escrito pelo Gemini para realizar 50 apostas da loteria..

Em relação às estruturas da linguagem C utilizadas nas soluções apresentadas para o tópico de estruturas de repetição, de um total de 6 exercícios, o Gemini utilizou o laço *for* em 3 respostas, utilizou o laço *while* em 2 respostas e em 1 questão fez o uso de recursão, conceito considerado avançado, que pode dificultar o entendimento por programadores iniciantes.

6 Discussão dos resultados e comparação entre o ChatGPT e o Gemini

A respeito dos testes qualitativos, ambos LLMs geraram respostas erradas em alguns momentos, exigindo interações adicionais para conseguirem gerar respostas corretas, assim como relatado em (Finnie-Ansley et al., 2022), no qual mais de 50% das questões apresentadas para o Codex (LLM antecessor do ChatGPT) precisaram de interações adicionais para fornecer uma resposta correta. Entretanto, houve situações que, mesmo após várias interações, o LLM não foi apto a corrigir o que era pedido, como descrito em alguns casos apresentados na Seção 5.

Em alguns casos, os LLMs geraram respostas que utilizavam estruturas que não são familiares a iniciantes em programação, como o uso de funções. Tanto as interações adicionais exigidas nas respostas incorretas, quanto as estruturas mais avançadas usadas pelos LLMs nos códigos gerados podem ser complicadores para o entendimento de estudantes iniciantes em programação, assim como observado no trabalho de (Finnie-Ansley et al., 2023), que relata o uso, por parte do Codex, de estruturas inconsistentes com as instruções dos cursos introdutórios de programação. Em (Sarsa et al., 2022) é observado que algumas respostas geradas pelo Codex apresentavam erros comuns a iniciantes em programação, os quais, embora necessitassem de poucos ajustes para correção, ainda exigiam conhecimento prévio do aluno. Em outras situações, ainda nos testes qualitativos do presente trabalho, era preciso conhecimento específico e avançado, por exemplo, em relação ao uso de funções inseguras como a *scanf* com o parâmetro *%s*.

O desempenho geral dos LLMs nos testes quantitativos, considerando somente os códigos corretos, foi de ∼73,9%. O ChatGPT alcançou ∼78,2% (valor é inferior aos ∼82,6% do artigo anterior) e o Gemini teve aproveitamento de ∼69,6%, ficando abaixo do desempenho geral e mostrando uma diferença significativa para o primeiro. Em contraste com esses resultados, Sarsa et al.

(2022) observou que o Codex, embora gerasse código executável em 90% dos casos, produzia soluções corretas em apenas 30% das vezes.

Ainda em relação aos testes quantitativos, considerando os tópicos mais fundamentais, como variáveis e operadores e estruturas de decisão e repetição, o aproveitamento geral dos LLMs foi de ∼87%. Em relação à esses tópicos, o desempenho do ChatGPT variou de 100% no primeiro artigo para ∼92% nos testes refeitos para o atual. O Gemini teve um aproveitamento de ∼84%, abaixo do desempenho geral e consideravelmente abaixo do ChatGPT.

Nos exercícios que envolvem manipulação de *strings*, tema complexo na linguagem C, o aproveitamento dos LLMs foi igual, figurando ∼46,7% de aproveitamento. No artigo anterior, o ChatGPT resolveu corretamente ∼53,3% dos problemas. *Strings* foi o tópico com os percentuais mais baixos, mostrando que, possivelmente, é o conteúdo mais complexo dentre os utilizados nos testes. Ambos os LLMs produziram códigos com o uso inadequado das funções *scanf* e *gets*, totalizando 8 respostas com o uso dessas funções inseguras, representando ∼27% dos problemas apresentados sobre este tópico.

Em relação ao tópico de vetores e matrizes, o desempenho dos LLMs foi de ∼83%. O aproveitamento do ChatGPT variou de ∼80% no primeiro artigo para 100% de acerto neste trabalho. O Gemini, por sua vez, resolveu ∼67% dos problemas corretamente, ficando significativamente abaixo das porcentagens do ChatGPT.

Em 17 respostas, o ChatGPT usou funções para agrupar e encapsular partes do código. As funções foram nomeadas tanto em inglês quanto em português, e seus nomes foram coesos em relação ao propósito de cada uma. No artigo anterior, foram 16 respostas usando funções com todas nomeadas em inglês. O Gemini fez uso de funções próprias apenas 5 vezes, sendo todas nomeadas em português e, assim como as nomeadas pelo ChatGPT, como nomes significativos e coesos. Como funções não é um dos assuntos introdutórios, é necessário que o aluno busque esse conhecimento para compreender as respostas fornecidas, alterando os códigos ou solicitando as alterações para adequar as soluções aos enunciados dos exercícios. É importante destacar que os LLMs são treinados com bases de códigos disponíveis em repositórios da Internet, com todos os níveis de conhecimento, portanto, esse comportamento de criar funções pode ser considerado comum, dependendo do contexto e da complexidade do exercício proposto.

No geral, portanto, para os testes realizados para este artigo, o ChatGPT teve um aproveitamento melhor que o Gemini em praticamente todos os aspectos analisados. Um ponto importante é que, alguns erros cometidos pelos dois LLMs podem ser facilmente corrigidos, inclusive pelos próprios LLMs, porém demanda conhecimento prévio de quem recebe a resposta. E para os erros apontados que, por exemplo, vão contra o enunciado do problema proposto, é preciso analisar o código gerado, o que demandaria ainda mais conhecimento prévio do estudante. Outro ponto importante a ser considerado é que em relação o conteúdo de *strings* na linguagem C, ambos LLMs não tiveram um bom aproveitamento, o que indica que esse deve ser um assunto que demande maior supervisão em relação às respostas geradas.

Dessa forma, a partir da análise feita, é possível responder as duas questões de pesquisa definidas no início deste trabalho:

- **QP1**: Os LLMs ChatGPT e Gemini são adequados para serem utilizados por estudantes iniciantes de programação?

Sim, porém é importante supervisionar esse uso, uma vez que os LLMs cometem erros, muitas vezes facilmente corrigíveis, porém esse fato que pode ser um obstáculo para alunos iniciantes em programação.

- **QP2**: É possível utilizar os LLMs ChatGPT e Gemini para o aprendizado em programação?

  Sim, porém é preciso critério e algum conhecimento prévio, ainda que mínimo, para aproveitar da melhor maneira as respostas que estes geram.

7 Ameaças à validade

É possível citar alguns pontos que podem influenciar nos resultados apresentados neste trabalho: i) a quantidade de exercícios propostos para os LLMs testados não foi extenso, sendo menor que outras referências citadas na Seção 2; ii) os LLMs estão em constante evolução, podendo haver atualizações que alterem seu comportamento e qualidade de suas respostas constantemente, melhorando os resultados apresentados aqui; iii) neste trabalho foram analisados somente códigos gerados na linguagem C, que pode ter diferença para outras linguagens; iv) foram avaliadas apenas quatro características da geração de código, podendo-se expandir a análise para legibilidade e clareza do código, qualidade dos comentários, tratamento de erros e exceções, outras práticas seguras de programação e uso padrões; v) a percepção em relação à visão de estudantes iniciantes em programação foi dada pelos autores do artigo, os quais já possuem experiência com esse conteúdo.

8 Conclusão

Atualmente, os LLMs, tais como o ChatGPT e o Gemini estão cada vez mais sendo utilizados no cotidiano das pessoas, devido a sua facilidade de uso e respostas que, no geral, são corretas. Porém, sabe-se que estes cometem equívocos e as respostas fornecidas devem sempre serem checadas.

Dada essa característica, o uso dos LLMs para aprendizado é especialmente delicado. Por um lado, trata-se de uma ferramenta interessante e que pode trazer benefícios para estudantes, por outro, para os iniciantes em algum assunto, a falta de conhecimento em relação ao conteúdo estudado somado ao fato de que os LLMs cometem equívocos, pode ser desastroso.

Este trabalho apresentou uma análise das respostas dadas por dois LLMs, o ChatGPT e o Gemini, em relação a conteúdos introdutórios de programação. Nas análises das respostas geradas pelos LLMs, foi considerado o ponto de vista de estudantes iniciantes em programação.

A análise feita demonstrou que os LLMs possuem potencial para auxiliar o aprendizado em programação. Contudo, é necessário destacar que as respostas geradas pelos LLMs não podem ser consideradas totalmente precisas e exigem um certo conhecimento prévio por parte do usuário para aproveitá-las corretamente. A principal contribuição deste trabalho é uma análise crítica acerca das respostas dadas pelos LLMs, ChatGPT e Gemini a exercícios de programação, no contexto de estudantes iniciantes e, para esse tipo de usuário é indicada a supervisão de um docente

ou tutor. A partir dos resultados obtidos e relatados, espera-se que a comunidade acadêmica da área de Informática na Educação possa compreender as limitações dessas tecnologias, bem como as oportunidades de novas análises técnicas e pedagógicas.

Como trabalhos futuros vislumbra-se a realização de novos testes para aferir outras características em relação a criação de código, não somente em relação ao conteúdo introdutório da programação de computadores. Além disso, espera-se ampliar os testes para outros LLMs, comparando os resultados com os obtidos e apresentados neste artigo. Por fim, almeja-se planejar estudos exploratórios com estudantes iniciantes em programação, para analisar suas percepções em relação ao uso de LLMs durante o processo de aprendizagem de programação.

Artigo Premiado Estendido

Esta publicação é uma versão estendida de artigo premiado no XXXIV Simpósio Brasileiro de Informática na Educação (SBIE 2023), intitulado "Análise das Respostas do ChatGPT em Relação ao Conteúdo de Programação para Iniciantes", DOI: https://doi.org/10.5753/sbie.2023.234870.

Referências

Aljanabi, M., Ghazi, M., Ali, A. H., Abed, S. A., & ChatGpt. (2023). ChatGpt: Open Possibilities. *Iraqi Journal For Computer Science and Mathematics*, *4*(1), 62–64. https://doi.org/10.52866/20ijcsm.2023.01.01.0018 [GS Search].

Cámara, J., Troya, J., Burgueño, L., & Vallecillo, A. (2023). On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML. *Software and Systems Modeling*, *22*(3), 781–793. https://doi.org/10.1007/s10270-023-01105-5 [GS Search].

Dengel, A., Gehrlein, R., Fernes, D., Görlich, S., Maurer, J., Pham, H. H., Großmann, G., & Eisermann, N. D. g. (2023). Qualitative Research Methods for Large Language Models: Conducting Semi-Structured Interviews with ChatGPT and BARD on Computer Science Education. *Informatics*, *10*(4). https://doi.org/10.3390/informatics10040078 [GS Search].

du Boulay, J. B. H. (1986). Some Difficulties of Learning to Program. *Journal of Educational Computing Research*, *2*(1), 57–73. https://www.taylorfrancis.com/chapters/edit/10.4324/9781315808321-18/difficulties-learning-program-benedict-du-boulay [GS Search].

Dunder, N., Lundborg, S., Wong, J., & Viberg, O. (2024). Kattis vs ChatGPT: Assessment and Evaluation of Programming Tasks in the Age of Artificial Intelligence. *Proceedings of the 14th Learning Analytics and Knowledge Conference*, 821–827. https://doi.org/10.1145/3636555.3636882 [GS Search].

Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., & Prather, J. (2022). The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming, 10–19. https://doi.org/10.1145/3511861.3511863 [GS Search].

Finnie-Ansley, J., Denny, P., Luxton-Reilly, A., Santos, E. A., Prather, J., & Becker, B. A. (2023). My AI Wants to Know If This Will Be on the Exam: Testing OpenAI's Codex on CS2 Programming Exercises. *Proceedings of the 25th Australasian Computing Education Conference*, 97–104. https://doi.org/10.1145/3576123.3576134 [GS Search].

Gil, A. C. (2002). *Como elaborar projetos de pesquisa* (4ª ed.). Editora Atlas S.A.

Google. (2021). *LaMDA: our breakthrough conversation technology* [Acessado em: 25/04/2024]. https://blog.google/technology/ai/lamda

Google. (2023a). *Bard now helps you code* [Acessado em: 25/04/2024]. https://blog.google/technology/ai/code-with-bard/

Google. (2023b). *A Message From Our CEO: An important next step on our AI journey* [Acessado em: 25/04/2024]. https://blog.google/technology/ai/bard-google-ai-search-updates/

Google. (2024a). *Bard becomes Gemini: Try Ultra 1.0 and a new mobile app today* [Acessado em: 25/04/2024]. https://blog.google/products/gemini/bard-gemini-advanced-app/

Google. (2024b). *How Gemini for Google Cloud works* [Acessado em: 25/04/2024]. https://cloud.google.com/gemini/docs/discover/works

Kiesler, N., & Schiffner, D. (2023). Large Language Models in Introductory Programming Education: ChatGPT's Performance and Implications for Assessments. *ArXiv*, *abs/2308.08572*. https://api.semanticscholar.org/CorpusID:261030787 [GS Search].

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Lago, A. D., Hubert, T., Choy, P., de Masson d'Autume, C., Babuschkin, I., Chen, X., Huang, P.-S., Welbl, J., Gowal, S., Cherepanov, A., ... Vinyals, O. (2022). Competition-level code generation with AlphaCode. *Science*, *378*(6624), 1092–1097. https://doi.org/10.1126/science.abq1158 [GS Search].

Lo, C. K. (2023). What Is the Impact of ChatGPT on Education? A Rapid Review of the Literature. *Education Sciences*, *13*(4). https://doi.org/10.3390/educsci13040410 [GS Search].

MacNeil, S., Tran, A., Mogil, D., Bernstein, S., Ross, E., & Huang, Z. (2022). Generating Diverse Code Explanations Using the GPT-3 Large Language Model. *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 2*, 37–39. https://doi.org/10.1145/3501709.3544280 [GS Search].

Matthews, S. J., Newhall, T., & Webb, K. C. (2021). Dive into Systems: A Free, Online Textbook for Introducing Computer Systems. *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 1110–1116. https://doi.org/10.1145/3408877.3432514 [GS Search].

Mizrahi, V. V. (2008). *Treinamento em Linguagem C* (Vol. 1). Person Prentice Hall.

OpenAI. (2024a). *ChatGPT* [Acessado em: 24/04/2024]. https://openai.com/chatgpt

OpenAI. (2024b). *Model index for researchers* [Acessado em: 24/04/2024]. https://platform.openai.com/

Ouh, E. L., Gan, B. K. S., Jin Shim, K., & Wlodkowski, S. (2023). ChatGPT, Can You Generate Solutions for my Coding Exercises? An Evaluation on its Effectiveness in an undergraduate Java Programming Course. *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, 54–60. https://doi.org/10.1145/3587102.3588794 [GS Search].

Pereira Filho, L. C., Souza, T. P. C., & Paula, L. B. (2023). Análise das Respostas do ChatGPT em Relação ao Conteúdo de Programação para Iniciantes. *Anais do XXXIV Simpósio Brasileiro de Informática na Educação*, 1738–1748. https://doi.org/10.5753/sbie.2023.234870 [GS Search].

Piccolo, S. R., Denny, P., Luxton-Reilly, A., Payne, S. H., & Ridge, P. G. (2023). Evaluating a large language model's ability to solve programming exercises from an introductory

bioinformatics course. *PLOS Computational Biology*, *19*(9), 1–16. https://doi.org/10.1371/journal.pcbi.1011511 [GS Search].

Rasul, T., Nair, S., Kalendra, D., Robin, M., Santini, F., Ladeira, W., Sun, M., Day, I., Rather, A., & Heathcote, L. (2023). The Role of ChatGPT in Higher Education: Benefits, Challenges, and Future Research Directions. *Journal of Applied Learning Teaching*, *6*, 41–56. https://doi.org/10.37074/jalt.2023.6.1.29 [GS Search].

Replit. (2024). *Replit* [Acessado em: 25/04/2024]. https://replit.com/

Sarsa, S., Denny, P., Hellas, A., & Leinonen, J. (2022). Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. *Proc. of the 2022 ACM Conf. on International Computing Education Research V.1*. https://doi.org/10.1145/3501385.3543957 [GS Search].

Sok, S., & Heng, K. (2023). ChatGPT for Education and Research: A Review of Benefits and Risks. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.4378735 [GS Search].

Tsai, M.-L., Ong, C. W., & Chen, C.-L. (2023). Exploring the use of large language models (LLMs) in chemical engineering education: Building core course problem models with Chat-GPT. *Education for Chemical Engineers*, *44*, 71–95. https://doi.org/https://doi.org/10.1016/j.ece.2023.05.001 [GS Search].

Wermelinger, M. (2023). Using GitHub Copilot to Solve Simple Programming Problems. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 172–178. https://doi.org/10.1145/3545945.3569830 [GS Search].