Abordagem com LLM para Geração Automatizada de Feedback no Ensino de Programação de Computadores

Edson P. Pimentel[1], **Lucas D. Lima da Silva**[1], **Rodrigo H. Takeuti**[1], **Juliana C. Braga**[1]

[1]Universidade Federal do ABC - UFABC - Santo André - SP - Brasil

{edson.pimentel, dantas.lucas, rodrigo.h, juliana.braga}@ufabc.edu.br

Abstract. *Learning programming, particularly in introductory courses, is often associated with high failure rates. Tools such as online judges, when integrated into virtual learning environments, can support this process by providing automatically graded exercises. However, the feedback typically offered is limited to error messages or test results, which are not particularly effective in fostering learning. This paper proposes an approach based on Large Language Models (LLMs) to generate more detailed and pedagogically meaningful feedback from students' solutions to programming exercises. We present the results of experiments conducted in an Introduction to Programming course, demonstrating the potential of LLMs to enhance the teaching and learning experience.*

Resumo. *A aprendizagem de programação, especialmente em disciplinas introdutórias, costuma apresentar altos índices de reprovação. Ferramentas como Juízes online, integrados a ambientes virtuais, oferecem correção automática de exercícios, mas o feedback fornecido geralmente se limita a mensagens de erro ou resultados de testes, que são pouco eficazes para promover a aprendizagem. Este artigo propõe uma abordagem com Modelos de Linguagem de Grande Escala (LLMs) para gerar feedback mais detalhado e pedagógico com base nas soluções dos estudantes. Apresentamos resultados de experimentos realizados em um curso de Introdução à Programação, evidenciando o potencial dos LLMs para apoiar o processo de ensino e aprendizagem.*

1. Introdução

Segundo Bosse (2020), muitos estudantes enfrentam dificuldades significativas logo em sua primeira experiência com programação, o que pode se tornar uma barreira substancial em seu desenvolvimento acadêmico e profissional. Essas dificuldades frequentemente resultam em altas taxas de reprovação e se estendem para as disciplinas mais avançadas, perpetuando um ciclo de desafios educacionais.

Para apoiar a aprendizagem em disciplinas introdutórias de programação, os professores têm utilizado ferramentas como juízes online, que realizam correções automáticas, integradas a ambientes virtuais de aprendizagem, para a aplicação de exercícios. No entanto, conforme destacado por Messer et al. (2024), o *feedback* fornecido por essas ferramentas é frequentemente limitado, restringindo-se a informar se o código passou nos testes ou se apresentou erros de compilação.

De acordo com o estudo empírico apresentado por Campos e Ferreira (2020), os *feedbacks* são essenciais para os alunos identificarem problemas em seus códigos e superarem as dificuldades encontradas durante o processo de aprendizado. O artigo destaca

DOI: 10.5753/sbie.2025.12543

que, nas disciplinas introdutórias de programação, os *feedbacks* mais eficazes são aqueles que não apenas informam sobre o erro, mas também oferecem orientações sobre como corrigi-lo.

Com o avanço da inteligência artificial, *Modelos de Linguagem de Grande Escala* (do inglês, *Large Language Models* - LLMs), como o ChatGPT[1], o Gemini[2] do Google e o LLaMA[3] da Meta, têm gerado um impacto significativo especialmente em tarefas que envolvem o *Processamento de Linguagem Natural* (*Natural Language Processing* - NLP) (MOHAN et al., 2024). Esses modelos, treinados com bilhões, ou até trilhões, de parâmetros, demonstram eficácia na compreensão de textos, imagens e até áudios em diversos contextos (TAJIK et al., 2024). Essa evolução tem impactado diversas áreas, incluindo a educação, na qual esses modelos têm o potencial de possibilitar que alunos e professores façam perguntas e recebam respostas contextualmente relevantes. Embora as LLMs apresentem limitações inerentes, como erros gramaticais, falta de autoconsciência e possíveis vieses, eles representam uma possibilidade de mudança significativa nas formas de ensinar e aprender, proporcionando novas oportunidades para melhorar a experiência educacional, como destacado por Tajik et al. (2024).

O uso de LLM também tem sido investigado para apoiar o ensino de programação. Por exemplo, Deus et al. (2024) apresentam um estudo de caso sobre como o ChatGPT aborda e resolve problemas de programação introdutória.

Este artigo tem por objetivo apresentar uma abordagem baseada em Large Language Models (LLMs) para gerar *feedbacks* mais detalhados a partir das soluções dos estudantes em exercícios de programação. Apresentam-se os resultados de experimentações com LLMs em uma disciplina de Introdução à Programação, evidenciando seu potencial no apoio ao processo de ensino-aprendizagem.

A sequência do artigo está organizada como segue: a Seção 2 apresenta a revisão de literatura, incluindo trabalhos relacionados; a Seção 3 apresenta a metodologia utilizada no experimento para gerar *feedbacks* com o uso de LLM; a Seção 4 apresenta e discute os resultados do experimento; e por fim, a Seção 5 estabelece as considerações finais.

2. Revisão de Literatura

O processo de revisão de literatura buscou identificar o estado da arte a partir da problematização e dos objetivos estabelecidos seguindo um protocolo de mapeamento sistemático de literatura (DERMEVAL; COELHO; BITTENCOURT, 2020).

A seguinte *string* de busca foi utilizada: { ("Evaluation"OR “feedback”) AND ("LLM"OR "Large Language Models") AND ("programming exercises"OR"online judges")} a partir da questão principal: *"Como as LLMs têm sido utilizadas no contexto da geração de feedback em atividades de programação?"*.

A busca foi realizada a partir do *Google Scholar* com a a *string* apenas no idioma inglês. A escolha do período mais recente (2023 a 2024) justifica-se pela ascensão de modelos de Inteligência Artificial Generativa, como os Large Language Models (LLMs),

[1]<https://openai.com/chatgpt>
[2]<https://ai.google/>
[3]<https://www.llama.com/>

que têm avançado de forma significativa nos últimos anos. A busca inicial resultou em 634 artigos.

Os seguintes critérios de inclusão foram definidos: (I1) artigos em inglês; (I2) artigos publicados entre 2023 e 2024; (I3) artigos completos. Os critérios de exclusão foram: (E1) artigos duplicados; (E2) artigos curtos.

Após a aplicação dos critérios de inclusão e exclusão, o processo resultou na seleção de 34 artigos, que foram considerados relevantes para o mapeamento sistemático, com destaque para 3 artigos que serão discutidos adiante.

Posteriormente, artigos escritos em português e publicados em eventos e/ou periódicos de informática na educação foram considerados na complementação da revisão de literatura.

Essa seção tem por objetivo apresentar os fundamentos conceituais necessários para a compreensão do trabalho e alguns trabalhos relacionados.

2.1. Fundamentos Conceituais

Os Modelos de Linguagem de Grande Escala (LLMs, *Large Language Models*) representam um avanço significativo no campo do Processamento de Linguagem Natural (PLN). Os LLMs têm sido aplicados em uma ampla gama de áreas, como na educação, para a geração automatizada de *feedbacks* em exercícios de programação, e na saúde, auxiliando na redação de relatórios médicos (NAVEED et al., 2023). Na educação, por exemplo, ferramentas baseadas em LLMs podem fornecer *feedbacks* instantâneos e direcionados, auxiliando no aprendizado de alunos em cursos de programação (MESSER et al., 2024).

Para processar o texto, os LLMs utilizam técnicas de *tokenização*, que dividem o texto em unidades menores, chamadas *tokens* (palavras ou subpalavras). Modelos como GPT-3 e LLaMA empregam métodos de codificação, como *Byte Pair Encoding* (BPE), que permitem comprimir o vocabulário para lidar com grandes quantidades de dados textuais (OPENAI et al., 2024). Após a tokenização, os modelos passam por uma fase de *pré-treinamento*, onde são expostos a grandes volumes de dados não rotulados, aprendendo padrões linguísticos de forma não supervisionada.

A arquitetura de *transformers*, amplamente adotada em LLMs como GPT-3, GPT-4 e LLaMA, introduz mecanismos como a *auto-atenção*, que permite ao modelo identificar quais partes do texto são mais relevantes para a tarefa que está sendo executada (NAVEED et al., 2023). A *atenção* funciona diretamente sobre os tokens gerados pela fase de *tokenização* que servem como entrada para que os transformadores possam estabelecer conexões contextuais e semânticas entre diferentes partes do texto (VASWANI et al., 2017).

Segundo Hanke et al. (2024), a execução de Modelos de Linguagem de Grande Escala (LLMs) localmente, tem se tornado cada vez mais relevante devido às demandas de eficiência e privacidade em diversas indústrias. Modelos como LLaMA, desenvolvidos pela Meta, são exemplos de LLMs que alcançam grandes avanços na compreensão e geração de texto (PETRUZZELLIS; TESTOLIN; SPERDUTI, 2024). Tradicionalmente, a execução de tais modelos exigia infraestruturas robustas baseadas em nuvem, devido ao grande número de parâmetros e à quantidade significativa de recursos de hardware neces-

sária para processá-los. O Ollama[4] surge como uma solução para esse desafio, permitindo que usuários executem modelos de IA localmente, sem depender de serviços em nuvem, promovendo uma nova era de acessibilidade e eficiência na IA (LILJEDAHL, 2024).

A arquitetura do Ollama segue uma abordagem modular, cujos componentes podem ser acessados de forma independente e gerenciados com comandos simples. Cada comando executa uma função específica que facilita o uso local dos modelos de linguagem. O Ollama facilita a execução de uma ampla gama de modelos de linguagem, incluindo o LLaMA 2, LLaMA 3 e variações de modelos ajustados para tarefas específicas. Para garantir eficiência na execução local, o Ollama utiliza técnicas de quantização e outras formas de otimização de modelos, que reduzem a quantidade de memória e processamento necessários para rodar esses modelos sem uma perda significativa de desempenho.

Conforme evidenciado pelo trabalho de Liu, Kang e Han (2024), o Ollama também se integra bem com outras plataformas de desenvolvimento de IA, como o LangChain, permitindo a construção de pipelines de processamento de linguagem natural mais complexos. Ao conectar Ollama a outras ferramentas, os usuários podem criar aplicações que combinam modelos de linguagem avançados com fluxos de trabalho personalizados, melhorando ainda mais a capacidade de automatizar tarefas baseadas em linguagem, como análise de texto e geração de conteúdo.

Segundo Zhang et al. (2023), a quantização é uma técnica crucial para viabilizar a execução de Modelos de Linguagem de Grande Escala (LLMs) em dispositivos com recursos limitados, reduzindo a necessidade de memória e poder computacional. Técnicas de ajuste são essenciais para maximizar a eficiência dos LLMs quantizados, permitindo que plataformas como o Ollama ofereçam modelos de grande escala em dispositivos locais, com um bom equilíbrio entre economia de recursos e desempenho preciso (FRANTAR et al., 2023).

2.2. Trabalhos relacionados

Os trabalhos apresentados a seguir, foram identificados como relacionados, considerando aspectos como: (a) abordam o uso de *feedback* em disciplinas introdutórias de programação; (b) fazem uso de LLM para fornecer *feedback* em atividades de programação.

Silva (2020) apresentam uma ferramenta destinada a apoiar o ensino de disciplinas de programação introdutória, oferecendo um ambiente no qual os alunos podem receber *feedback* automatizado em relação às suas submissões de código. A ferramenta utiliza técnicas de análise automática de código para identificar erros comuns e fornecer orientações específicas para corrigir esses problemas.

O trabalho de McBroom, Koprinska e Yacef (2021) explora diversas técnicas de geração automatizada de dicas para exercícios de programação, apresentando o *framework* HINTS (*Hint Iteration by Narrow-down and Transformation Steps*) como uma maneira de organizar essas técnicas em uma estrutura coesa. O estudo contribui para a compreensão das metodologias que podem ser empregadas na geração de *feedbacks* automáticos.

O artigo de Estévez-Ayres et al. (2024) apresenta uma análise detalhada do uso de LLMs para a geração de *feedback* em um contexto de programação concorrente. Os

[4]<https://ollama.com>

autores realizaram uma avaliação abrangente da capacidade dos LLMs, como ChatGPT e Bard, de identificar e fornecer *feedback* sobre erros típicos de concorrência, como *deadlocks*, condições de corrida e *starvation*. Os estudos revelam tanto o potencial quanto as limitações do uso de LLMs na detecção de erros complexos em programação, destacando que, embora essas ferramentas ofereçam *insights* valiosos, elas ainda não são suficientemente confiáveis para substituir a avaliação humana em contextos específicos, como a programação concorrente.

Gabbay e Cohen (2024) exploram a integração de *feedback* gerado por Modelos de Linguagem de Grande Escala (LLMs) com sistemas tradicionais de *feedback* baseado em testes automatizados (ATF) em um curso massivo *online* de programação (MOOC). Os autores desenvolveram e testaram a aplicação Gipy, que permite aos alunos receber *feedback* automatizado gerado por LLMs sobre suas soluções de código, complementando o *feedback* baseado em testes. A pesquisa revelou que, embora os LLMs, como o GPT-3.5 e o GPT-4, sejam eficazes na detecção de erros de código, o *feedback* gerado frequentemente apresenta inconsistências e erros, o que pode limitar sua confiabilidade como única fonte de *feedback*.

O artigo de Pankiewicz e Baker (2023) investiga a aplicação de LLMs, especificamente o GPT-3.5, para automatizar a geração de *feedback* em tarefas de programação. Em um estudo experimental, os autores integraram o GPT-3.5 a uma plataforma de avaliação automatizada, permitindo a criação de dicas automatizadas para estudantes de um curso de programação orientada a objetos. Os resultados mostraram que os alunos que receberam *feedback* gerado pelo GPT-3.5 apresentaram um desempenho superior em comparação aos que receberam apenas o *feedback* regular da plataforma, sugerindo que o uso de LLMs pode aprimorar a aprendizagem ao fornecer orientações mais detalhadas e específicas. No entanto, o estudo também identificou uma possível dependência excessiva dos alunos em relação ao *feedback* gerado pelo GPT, o que levanta preocupações sobre o equilíbrio entre a utilização de LLMs e a promoção de uma aprendizagem mais independente.

Em linhas gerais, os trabalhos que podem ser considerados mais correlatos são:

- O artigo de Estévez-Ayres et al. (2024) apresenta uma análise do uso de LLMs para a geração de *feedback* em um contexto de programação concorrente;
- Gabbay e Cohen (2024) exploram a integração de *feedback* gerado por Modelos de Linguagem de Grande Escala (LLMs) com sistemas tradicionais de *feedback* baseado em testes automatizados;
- O artigo de Pankiewicz e Baker (2023) investiga a aplicação de LLMs, especificamente o GPT-3.5, para automatizar a geração de *feedback* em tarefas de programação.

Este artigo buscar avançar em relação aos correlatos ao avaliar o desempenho de modelos de LLM, na geração de *feedbacks* a partir de soluções de atividades de programação realizadas com o suporte do Virtual Programming Lab (VPL) [5]. O VPL é um *plugin* para o ambiente virtual Moodle que atua como um juiz online, possibilitando a avaliação automática de atividades de programação.

[5] <https://moodle.org/plugins/mod_vpl>

3. Metodologia

Esta seção tem por finalidade apresentar a metodologia utilizada para realizar os experimentos de geração de feedback, fazendo uso de Modelos de LLM, a partir de soluções de exercícios de programação elaboradas por estudantes.

Um *prompt* foi elaborado para realizar as avaliações. Um *prompt* é uma entrada para um modelo de IA generativa, que é usado para orientar sua saída (MESKÓ, 2023).

As seguintes premissas foram estabelecidas para a construção do *prompt*: (i) uso do enunciado para fins de contextualização; (ii) uso da solução do estudante; (iii) uso dos resultados da execução do estudante; (iv) orientação para informar se a solução está correta ou incorreta; (v) orientação para informar os erros e explicá-los; (vi) não fornecer a solução.

A Listagem 1 apresenta o *prompt* final utilizado nas avaliações das soluções dos alunos e geração de *feedback*.

```
Considere que você seja um monitor em um curso introdutório de programação.
A tarefa proposta para um aluno é:
[descrição do problema]
A solução fornecida pelo aluno foi:
[código do aluno]
A execução do código para os casos de teste foi:
[resultados da execução dos testes]

Qual o feedback para esta solução?

Se a solução estiver correta, somente escreva "A solução está correta!".

Senão, escreva "A solução está incorreta!", seguido dos erros e divergências das
    instruções.

Não mostre a solução completamente nem partes dela.
```

Listagem 1. Prompt para avaliação de solução e geração de feedback

O *prompt* exibido na Listagem 1 é composto por três parâmetros principais:

- **Descrição do Problema**: corresponde ao enunciado da tarefa proposta ao aluno e fornece o contexto necessário para que o modelo compreenda o objetivo do código;
- **Código do Aluno**: elemento central da avaliação, a partir do qual o modelo identifica erros e gera *feedbacks*.
- **Execução do Código**: incluem mensagens de erro de compilação, saídas do programa ou resultados de testes específicos, ajudando o modelo a avaliar o comportamento do código em relação às expectativas.

Para os experimentos foram utilizados códigos submetidos por estudantes de uma disciplina introdutória de programação, de um curso de graduação de uma universidade pública. Os códigos foram submetidos por meio do ambiente virtual Moodle configurado com o *plugin* Virtual Programming Lab (VPL).

Foram disponibilizados, para os experimentos, enunciados de 30 exercícios e soluções de 150 estudantes para serem selecionados.

3.1. Seleção dos Códigos

Dado que a avaliação da qualidade dos *feedbacks* gerados seria realizada manualmente, por dois avaliadores, optou-se por selecionar apenas 30 códigos, distribuídos em três atividades distintas (10 soluções de cada atividade), cada uma abordando aspectos essenciais das estruturas de repetição, como a validação de dados e a análise de sequências numéricas, a saber:

- Atividade 1 (AT1): Estruturas de Repetição com N Fixo.
- Atividade 2 (AT2): Estruturas de Repetição com Validação (Idades).
- Atividade 3 (AT3) : Estruturas de Repetição com Validação (Salário e Filhos).

Esse tipo de avaliação, focada na qualidade e precisão dos *feedbacks* gerados, demanda um tempo significativo para a identificação, categorização e interpretação dos resultados fornecidos, o que justifica a escolha por uma amostra que fosse representativa, mas administrável dentro das limitações de tempo e recursos do projeto.

Os três enunciados (atividades) foram escolhidos para garantir uma cobertura abrangente das habilidades fundamentais necessárias para a compreensão e aplicação de estruturas de repetição na resolução de problemas computacionais. A fim de garantir a diversidade dos dados analisados, foi selecionada apenas uma submissão por aluno para cada atividade, priorizando a variedade entre os tipos de erros e acertos. A amostra incluiu códigos que continham:

- Erros de lógica - a solução não atingiu o resultado correto devido a falhas na implementação do algoritmo.
- Erros de sintaxe - que impossibilitaram a compilação ou execução do código.
- Soluções corretas - código atendia a todos os critérios do enunciado.

Essa seleção visou garantir uma avaliação equilibrada, testando a capacidade da LLM de identificar e fornecer *feedback* em diferentes contextos de erros e acertos. Os códigos utilizados para os testes, estão disponíveis no repositório GitHub do projeto, na pasta `ExercisesAnalysis`[6].

3.2. Critérios de Avaliação

A eficácia das LLMs na avaliação de soluções e geração de *feedback* foi avaliada com base em um conjunto de cinco critérios definidos, a saber:

- **C1 - A solução foi adequadamente identificada como correta ou incorreta?** - Este critério avalia a capacidade da LLM em distinguir se o código submetido está correto ou se contém erros. Se o código apresenta falhas — sejam de lógica, sintaxe ou execução —, a ferramenta deve classificá-lo como incorreto. Por outro lado, se a solução foi implementada corretamente, ela deve ser reconhecida como correta. Este critério é essencial, pois um insucesso nesse critério inviabiliza a análise dos próximos.
- **C2 - A solução identificou ao menos um erro presente no código corretamente?** - Este critério verifica se a LLM conseguiu reconhecer pelo menos um erro adequadamente, quando a solução é incorreta.

[6]Link para o repositório: <https://github.com/rodrigohisashi/codeinsight/tree/main/ExercisesAnalysis>

- **C3 - A solução identificou todos os erros presentes no código corretamente?** - Este critério avalia se a LLM conseguiu fazer uma análise abrangente e detectar todos os problemas de lógica, sintaxe ou execução presentes no código submetido.
- **C4 - A solução adicionou erros que não existem?** - Um dos grandes desafios em ferramentas automáticas de geração de *feedback* é evitar falsos positivos — isto é, identificar erros que não existem.
- **C5 - O *feedback* gerado forneceu o código da solução ao aluno?** - No prompt fornecido, a orientação era a de não fornecer a solução, na perspectiva que o *feedback* deve ajudar o aluno a progredir no aprendizado sem fornecer diretamente a solução do problema.

Cada um desses critérios foi rigorosamente aplicado na avaliação do retorno fornecido pelas 5 (cinco) LLMs, para cada uma das 30 (trinta) soluções selecionadas, comparando-se o retorno com a solução do aluno. Dois avaliadores realizaram uma avaliação independente, atribuindo "Sim"ou "Não"a cada um dos critérios. Em seguida, os avaliadores discutiram e unificaram sua avaliação, em caso de discrepância. Os resultados dessas avaliações foram utilizados para calcular a taxa de sucesso dos modelos em cada um dos critérios, a serem apresentados na seção de resultados e discussão.

3.3. Tecnologias e Etapas

A análise dos códigos de programação submetidos pelos alunos foi realizada de forma local, utilizando a ferramenta Ollama para rodar modelos de linguagem de código aberto em um ambiente controlado. Essa abordagem permitiu a execução de modelos LLMs em máquinas com capacidade limitada, viabilizando a experimentação com diferentes arquiteturas.

Os modelos escolhidos para essa análise foram: (i) Llama 3.1 (8b); (ii) Mistral-Nemo (12b); (iii) Zephyr (7b); (iv) Deepseek-Coder v2 (16b) e (v) Codellama (7b). Adicionalmente, o Chat GPT-4o foi utilizado como modelo de controle para comparação, sendo executado remotamente em um ambiente otimizado. Essa abordagem permitiu avaliar as diferenças de desempenho entre modelos locais e uma solução online avançada.

O processo de análise automatizada seguiu uma sequência de passos que permitiu processar as soluções dos alunos usando os modelos LLMs mencionados. Esses passos são descritos a seguir:

1. **Preparação dos Dados:** Os códigos submetidos pelos alunos foram extraídos diretamente dos arquivos enviados. Cada código foi lido e carregado em formato de texto, preservando sua estrutura original.
2. **Configuração do Prompt:** Para cada código, foi gerado um *prompt* que incluía a descrição do problema, o código submetido e os resultados da execução dos testes. O *prompt* foi cuidadosamente elaborado para garantir que o *feedback* gerado fosse claro, útil e consistente com os objetivos pedagógicos do projeto.
3. **Execução dos Modelos:** Os dados foram enviados aos modelos de linguagem selecionados para análise, através do ambiente local configurado com o Ollama. Os modelos processaram as informações fornecidas e geraram *feedbacks* detalhados para cada código analisado.
4. **Armazenamento do *Feedback*:** Os *feedbacks* gerados foram armazenados em arquivos de texto para posterior análise. Cada arquivo incluía o *feedback* completo gerado pelo modelo, permitindo uma avaliação qualitativa e quantitativa.

4. Resultados e Discussão

Esta seção tem por finalidade apresentar e discutir os resultados obtidos, considerando-se que os prompts foram executados localmente, fazendo uso dos seguintes modelos: Llama 3.1 (8b), Mistral-neMo (12b), Zephyr (7b), DeepSeek-Coder-v2 (16b), CodeLlama (7b), com o Chat GPT-4o sendo usado como controle.

Durante as execuções, foram enfrentados problemas de hardware ao rodar alguns dos modelos com maior número de parâmetros, o que gerou limitações durante a avaliação do desempenho.

4.1. Desempenho Geral das LLMs na Avaliação de Códigos

Os resultados mostraram que os modelos de linguagem diferiram em seu desempenho ao fornecer *feedbacks* precisos. A Tabela 1 apresenta a taxa de sucesso dos modelos em cada um dos critérios:

Modelo	C1	C2	C3	C4	C5
Llama 3.1 8b	83,33%	60,00%	6,67%	23,33%	100,00%
Mistral-Nemo 12b	93,33%	60,00%	13,33%	40,00%	93,33%
Zephyr 7b	46,67%	26,67%	16,67%	23,33%	100,00%
Deepseek-Coder-v2 16b	83,33%	66,67%	38,89%	50,00%	44,44%
Codellama - 7b	70,00%	33,33%	6,67%	10,00%	93,33%
Chat GPT-4o	100,00%	96,67%	76,67%	80,00%	100,00%

Tabela 1. Taxa de sucesso dos modelos nos critérios de avaliação

Analisando os resultados dos modelos para cada critério, observamos diferenças substanciais no desempenho entre os modelos locais e o Chat GPT-4o, que serviu como referência, por rodar remotamente e presumivelmente em um ambiente com maior capacidade computacional.

A análise detalhada de cada critério permite entender melhor como cada modelo local se comporta em tarefas de detecção de erros e aderência às instruções, revelando forças e limitações específicas:

- C1 - o desempenho geral foi satisfatório entre os modelos locais, com exceção do Zephyr, que obteve apenas 46,67% de precisão, uma taxa muito inferior aos demais. Esse resultado sugere que o Zephyr tem dificuldades em realizar até mesmo uma avaliação inicial de erros no código. Tanto o Llama 3.1 quanto o Mistral-Nemo tiveram desempenhos consistentes, superando 80%, mostrando-se razoavelmente competentes nesse critério. O modelo Deepseek-Coder-v2, que teve alguns casos descartados devido a timeouts, também obteve um desempenho sólido neste critério, mas com a ressalva de que sua amostra foi menor, o que pode ter influenciado levemente os resultados. O Chat GPT-4o obteve 100% de precisão.
- C2 - Llama 3.1, Mistral-Nemo e Deepseek-Coder-v2 apresentaram desempenho mediano, conseguindo detectar um erro em aproximadamente metade dos testes. Zephyr e Codellama tiveram um desempenho fraco, encontrando um erro em menos de um terço dos testes. O Chat GPT-4o, teve desempenho preciso em 96,67% dos testes.

- C3 - o desempenho foi geralmente insatisfatório entre os modelos locais, refletindo uma limitação importante na análise mais detalhada. Deepseek-Coder-v2 apresentou uma taxa ligeiramente superior aos demais modelos locais. O Chat GPT-4o, teve desempenho inferior em relação ao critério C2, com uma precisão 76,67%.
- C4 - neste critério, o desempenho foi abaixo da média para praticamente todos os modelos locais, sendo que o Codellama foi o mais problemático, deixando de apresentar alucinações em apenas 10% dos testes. O Chat GPT-4o, embora também tenha apresentado alucinações teve uma precisão 80%.
- C5 - nesse critério os resultados foram bons para quase todos os modelos, com exceção do Deepseek-Coder-v2. Esse modelo, ao contrário dos demais, forneceu frequentemente a solução, contrariando a instrução presente no prompt. O Chat GPT-4o obteve 100% de precisão.

Em resumo, o Chat GPT-4o, rodando remotamente, destacou-se em todos os critérios, sugerindo que a infraestrutura e a arquitetura mais avançadas do Chat GPT-4o possibilitam uma análise mais precisa e confiável. Llama 3.1 e Mistral-Nemo se posicionaram como modelos razoáveis para tarefas básicas de verificação, mas com limitações notáveis em análises mais complexas, enquanto Codellama e Zephyr demonstraram dificuldades em vários critérios, indicando que são menos adequados para tarefas de avaliação crítica de código.

Ao detalhar a análise por tipo de atividade (AT1, AT2, AT3) a constatação foi que não houve grande diferença de desempenho dos modelos nos 3 tipos de atividades, mesmo havendo uma complexidade maior na solução de AT2 e AT3.

5. Considerações Finais

Este trabalho teve como ponto de partida o reconhecimento da necessidade de ferramentas para apoiar a aprendizagem de programação de computadores, principalmente para iniciantes, que forneçam *feedback* mais elaborado a partir dos erros em soluções dos aprendizes.

Os modelos de LLM têm gerado um impacto significativo especialmente em tarefas que envolvem o *Processamento de Linguagem Natural* e, portanto, podem ser utilizados na análise de soluções de atividades de programação para geração de *feedback* adequado.

Como contribuição principal, este trabalho apresentou um desenho de experimento que avaliou soluções de 30 estudantes, em 3 tipos de enunciados, com 5 modelos de LLM locais e usando o ChatGPT (remoto) como controle. Foram definidos e aplicados 5 critérios da avaliação e análise.

Os resultados indicaram que, apesar das limitações, os LLMs são capazes de identificar erros de lógica e sintaxe e fornecer *feedbacks* que vão além do escopo dos juízes online convencionais, como o plugin Virtual Programming Lab (VPL). Esses resultados indicam que os LLMs podem complementar e enriquecer as práticas pedagógicas, ajudando os alunos a compreenderem melhor seus erros e a melhorarem suas soluções. No entanto, os problemas enfrentados, como tempos elevados de resposta (considerando a experiência do usuário) e inconsistências nos *feedbacks* gerados por modelos menores,

indicam que melhorias substanciais são necessárias para que essa solução seja implementada de forma eficaz em larga escala.

Como ameaças à validade, para generalizações dos resultados, destacam-se: (a) o caráter limitado da amostra (30 soluções de 3 categorias de problemas, de 10 alunos e (b) o possível viés humano da avaliação dos 5 critérios realizados por dois pesquisadores.

Na sequência da pesquisa, um protótipo denominado CodeInsight foi implementado, dotando o *plugin* VPL do recurso de *feedback* com o apoio de LLM. A figura 1 mostra a arquitetura geral o Ambiente Integrado de Geração de *Feedbacks*.

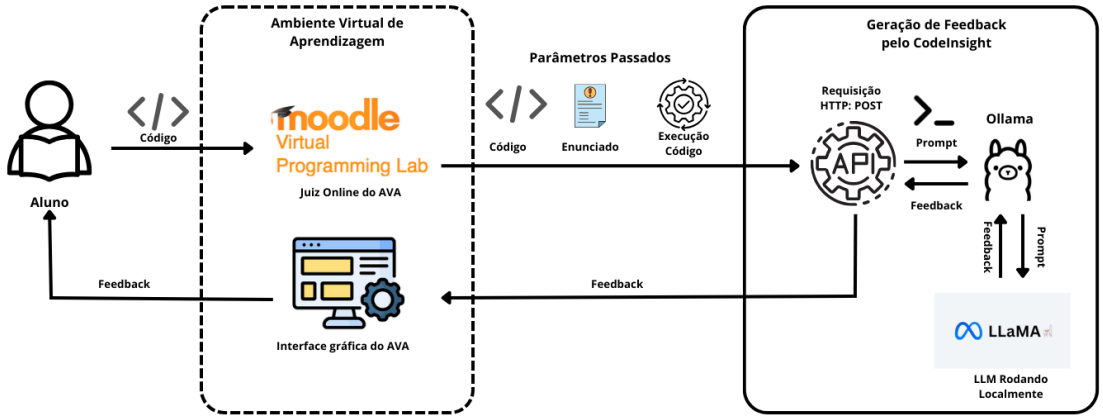

Figura 1. Arquitetura geral do Ambiente Integrado de Geração de *Feedbacks*

Como trabalhos futuros pretende-se: (a) avaliar o protótipo CodeInsight com alunos, durante a realização das atividades; (b) avaliar os ganhos de aprendizagem com o uso da ferramenta.

Referências

BOSSE, Y. *Padrões de dificuldades relacionadas com o aprendizado de programação*. Tese (Doutorado) — Universidade de São Paulo, 2020.

CAMPOS, D. S. de; FERREIRA, D. J. Feedback no ensino de lógica de programação com o auxílio de ferramentas para apoiar o ensino e a aprendizagem: Uma abordagem empírica. In: SBC. *Anais do V Congresso sobre Tecnologias na Educação*. [S.l.], 2020. p. 346–355.

DERMEVAL, D.; COELHO, J. A. d. M.; BITTENCOURT, I. I. Mapeamento sistemático e revisão sistemática da literatura em informática na educação. *JAQUES, Patrícia Augustin; SIQUEIRA; Sean; BITTENCOURT, Ig; PIMENTEL, Mariano.(Org.) Metodologia de Pesquisa Científica em Informática na Educação: Abordagem Quantitativa. Porto Alegre: SBC*, 2020.

DEUS, W. S. de et al. Avaliando resoluções de exercícios introdutórios de programação na era das ias generativas: Um estudo de caso com o chatgpt. In: SBC. *Simpósio Brasileiro de Informática na Educação (SBIE)*. [S.l.], 2024. p. 1645–1657.

ESTÉVEZ-AYRES, I. et al. Evaluation of llm tools for feedback generation in a course on concurrent programming. *International Journal of Artificial Intelligence in Education*, Springer, p. 1–17, 2024.

FRANTAR, E. et al. *GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers*. 2023. Disponível em: <https://arxiv.org/abs/2210.17323>.

GABBAY, H.; COHEN, A. Combining llm-generated and test-based feedback in a mooc for programming. In: *Proceedings of the Eleventh ACM Conference on Learning@ Scale*. [S.l.: s.n.], 2024. p. 177–187.

HANKE, V. et al. Open LLMs are necessary for private adaptations and outperform their closed alternatives. In: *ICML 2024 Next Generation of AI Safety Workshop*. [s.n.], 2024. Disponível em: <https://openreview.net/forum?id=uGml3wUL8s>.

LILJEDAHL, J. *Development and Evaluation of an AI-Powered Virtual Coach for an E-Health Tool for Fall Prevention : Open-Source vs Closed-Source LLMs in E-Health Applications*. 2024. 58 p. (UPTEC IT, 24045).

LIU, F.; KANG, Z.; HAN, X. *Optimizing RAG Techniques for Automotive Industry PDF Chatbots: A Case Study with Locally Deployed Ollama Models*. 2024. Disponível em: <https://arxiv.org/abs/2408.05933>.

MCBROOM, J.; KOPRINSKA, I.; YACEF, K. A survey of automated programming hint generation: The hints framework. *ACM Computing Surveys (CSUR)*, ACM New York, NY, v. 54, n. 8, p. 1–27, 2021.

MESKÓ, B. Prompt engineering as an important emerging skill for medical professionals: tutorial. *Journal of medical Internet research*, JMIR Publications Toronto, Canada, v. 25, p. e50638, 2023.

MESSER, M. et al. Automated grading and feedback tools for programming education: A systematic review. *ACM Transactions on Computing Education*, ACM New York, NY, v. 24, n. 1, p. 1–43, 2024.

MOHAN, G. B. et al. An analysis of large language models: their impact and potential applications. *Knowledge and Information Systems*, Springer, p. 1–24, 2024.

NAVEED, H. et al. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*, 2023.

OPENAI et al. *GPT-4 Technical Report*. 2024. Disponível em: <https://arxiv.org/abs/2303.08774>.

PANKIEWICZ, M.; BAKER, R. S. Large language models (gpt) for automating feedback on programming assignments. *arXiv preprint arXiv:2307.00150*, 2023.

PETRUZZELLIS, F.; TESTOLIN, A.; SPERDUTI, A. *Assessing the Emergent Symbolic Reasoning Abilities of Llama Large Language Models*. 2024. Disponível em: <https://arxiv.org/abs/2406.06588>.

SILVA, Â. G. L. d. Uma ferramenta para correção de erros sintáticos usando aprendizado de máquina para programação introdutória [trabalho de conclusão de curso]. *Universidade Federal Rural do Semi-Árido*, 2020.

TAJIK, E. et al. A comprehensive examination of the potential application of chat gpt in higher education institutions. 2024. Disponível em: <https://ssrn.com/abstract=4699304>.

VASWANI, A. et al. Attention is all you need. *Advances in neural information processing systems*, v. 30, 2017.

ZHANG, L. et al. *Dual Grained Quantization: Efficient Fine-Grained Quantization for LLM*. 2023. Disponível em: <https://arxiv.org/abs/2310.04836>.