Explorando o uso de LLMs para rotular estratégias de programação

Rafaela Melo[1,2], Tiago Souza[1], Elaine Oliveira[1], Leandro Galvão[1], Marcela Pessoa[2], David Fernandes[1]

[1]Instituto de Computação – Universidade Federal do Amazonas (ICOMP - UFAM)
Av. Gen. Rodrigo Octávio, 6200, Coroado I – 69080-900 – Manaus – AM – Brasil

[2]Escola Superior de Tecnologia – Universidade do Estado do Amazonas (EST/UEA)
Manaus - AM - Brasil

{rmelo,tiago.souza,elaine,galvao,david}@icomp.ufam.edu.br

msppessoa@uea.edu.br

Resumo. *Em disciplinas de programação, um único exercício pode ser resolvido de diferentes formas. Compreender as estratégias adotadas pelos estudantes para solucionar problemas é pedagogicamente relevante, pois permite avaliar, por exemplo, se eles estão assimilando os conteúdos abordados em sala e aplicando-os corretamente em seus códigos. Para ajudar os professores nesse contexto, o presente artigo investiga o uso de LLMs para rotular códigos de alunos de acordo com a estratégia que eles usaram para solucionar os exercícios de programação. Para tanto, foi criada uma base de dados com códigos rotulados e posteriormente realizados experimentos com diferentes LLMs. Os resultados preliminares indicam que, com a formulação de prompts adequados, os LLMs têm potencial para desempenhar a tarefa de rotulagem automática de códigos.*

Abstract. *In programming courses, a single exercise can be solved in different ways. Understanding the strategies adopted by students to solve problems is pedagogically relevant, as it allows us to assess, for example, whether they are assimilating the content covered in class and applying it correctly in their codes. To help teachers in this context, this paper investigates the use of LLMs to label students' codes according to the strategy they used to solve programming exercises. To this end, a database with labeled codes was created and experiments were subsequently carried out with different LLMs. Preliminary results indicate that, with the formulation of appropriate prompts, LLMs have the potential to perform the task of automatic code labeling.*

1. Introdução

O uso de sistemas de correção automática de códigos, conhecidos como juízes online, tem se tornado comum em disciplinas de programação, facilitando e agilizando a correção de exercícios [Wasik et al. 2018]. Apesar dos benefícios que oferecem para professores e alunos, sua adoção reduz o contato direto dos professores com os códigos dos alunos, visto que esses sistemas funcionam com base em casos de teste [Koivisto and Hellas 2022]. Como consequência, os professores têm menos oportunidade de compreender as estratégias e os recursos utilizados pelos alunos para resolver os exercícios [Barbosa et al. 2023].

Por outro lado, estudantes de disciplinas de programação podem resolver um mesmo exercício usando diferentes estratégias [Joyner et al. 2019]. Por exemplo, em um exercício onde é necessário calcular o fatorial de um número, os estudantes podem escolher seguir uma estratégia recursiva ou iterativa. Do ponto de vista pedagógico, pode ser importante identificar as diferentes estratégias de resolução adotadas pelos alunos, tanto para avaliar o nível entendimento dos estudantes, quanto para fazer intervenções caso seja necessário [Glassman et al. 2015].

Apesar da existência de trabalhos na literatura que exploram o uso de algoritmos de clusterização para agrupar estratégias semelhantes, ainda há lacunas nessa área [Melo et al. 2024]. A maioria desses algoritmos baseia-se na similaridade sintática do código, o que nem sempre reflete a abordagem adotada pelo estudante para resolver um problema. Como consequência, os rótulos gerados por esses estudos podem ser imprecisos.

Nos últimos anos, os *Large Language Models* (LLMs) têm ganhado atenção por conta do ótimo desempenho em tarefas de linguagem natural, principalmente com o surgimento de modelos como GPT (*Generative Pre-trained Transformer*) e LLaMA (*Large Language Model Meta AI*) [Zhao et al. 2023]. A crescente popularidade dos LLMs tem aberto novas possibilidades na análise automática de textos, como por exemplo: resumo de textos, classificação de textos, tradução automática, conversão de fala para texto e análise de textos em contextos específicos [Pires et al. 2023].

Diante disso, este artigo apresenta um estudo sobre o uso de LLMs para rotular códigos desenvolvidos por alunos segundo as estratégias que esses adotaram para solucionar exercícios de programação. Neste trabalho, investigou-se a capacidade de modelos como ChatGPT[1] (versões 4o-mini e 4o) e Gemini[2] na identificação e rotulagem automática de estratégias utilizadas por estudantes para resolver exercícios de programação.

Para alcançar esse objetivo, foram realizados experimentos variados, incluindo abordagens Zero-Shot, One-Shot e Few-Shot [Brown 2020]. Esses experimentos também exploraram diferentes formas de apresentação dos códigos aos LLMs, com variações como a inclusão ou omissão dos enunciados dos exercícios e o fornecimento de exemplos rotulados. Como contribuições, o trabalho apresenta uma nova possibilidade de estudo no contexto da educação em computação e experimentos iniciais para a rotulagem automática de *clusters* de código a partir de LLMs. O restante do trabalho está organizado como segue: a Seção 2 aborda conceitos sobre LLMs e trabalhos relacionados, a Seção 3 apresenta a metodologia, a Seção 4 discorre sobre os resultados alcançados, a Seção 5 contempla uma consolidação dos resultados dos experimentos e na Seção 6 estão as considerações finais do trabalho.

2. LLMs em diferentes contextos

Os LLMs são modelos de inteligência artificial baseados em aprendizado profundo, treinados com grandes quantidades de dados textuais para prever e gerar texto de maneira contextualizada, permitindo aplicações em tarefas como tradução, resumo e assistência em processamento de linguagem natural [Zhao et al. 2023, Chang et al. 2024]. Muitos trabalhos na literatura buscam aplicar esses modelos para tarefas em diferentes contextos.

[1] `chatgpt.com`
[2] `gemini.google.com`

Reiss [2023] investigou a capacidade do ChatGPT-3.5 em classificar textos de notícias e não notícias, utilizando uma abordagem Zero-Shot, que se refere a quando um modelo de aprendizagem de máquina é treinado para realizar uma tarefa sem possuir exemplos específicos da tarefa. Para tanto, foram consideradas diferentes formulações de *prompts* e repetições de entrada, visando investigar a consistência das classificações. Como resultado, os experimentos mostraram que utilizar ChatGPT com a abordagem Zero-Shot para a classificação de notícias pode estar abaixo dos padrões científicos comuns. Além disso, repetir os mesmos *prompts* levou a saídas diferentes, o que questiona a confiabilidade do ChatGPT-3.5.

Uma das áreas em que os LLMs têm sido muito explorados é a educação em computação. Poldrack et al. [2023] estudaram o uso do ChatGPT-4 para a geração de códigos de computador e também para a refatoração de códigos fornecidos pelo usuário. Primeiramente, os autores utilizaram um *prompt* autoral e um humano avaliou o código resultante. Em seguida, enviaram *prompts* adicionais para tentar corrigir os problemas. Caso o humano não encontrasse uma solução válida em 5 minutos, o problema era dito como não resolvido. Os resultados mostraram que, apesar do ChatGPT ter grande potencial para a geração de código, a interação do humano é importante para garantir a qualidade. Com relação à refatoração, o ChatGPT conseguiu melhorar o código com base em métricas de qualidade, porém não conseguiu eliminar todos os problemas.

Mehta et al. [2023] avaliaram o ChatGPT-3.5 como um assistente para a disciplina de Introdução à Programação, que recebe os códigos dos estudantes e fornece *feedback* sobre a solução apresentada. Para verificar se o ChatGPT analisa a qualidade do código da maneira certa, os autores compararam os resultados com métricas de qualidade de código, além de avaliarem a relevância do *feedback* fornecido. Os resultados apontaram que o ChatGPT não é completamente confiável para avaliar a corretude do código, mas com relação ao *feedback*, fornece boas sugestões para a melhoria do código.

Além da abordagem *Zero-Shot*, usada em Reiss [2023], existem a *One-Shot*, em que o modelo é treinado com um único exemplo, e a *Few-Shot*, em que um pequeno conjunto de exemplos é fornecido para treinamento [Brown 2020]. No presente artigo, investigou-se a capacidade do ChatGPT e do Gemini usando essas três abordagens, no contexto da educação em computação, para a tarefa de rotular estratégias de programação adotadas por estudantes para solucionar exercícios.

3. Metodologia

Para avaliar o uso de LLMs na tarefa de rotulagem de códigos desenvolvidos por alunos, segundo as estratégias adotadas para solucionar exercícios de programação, foi utilizada uma metodologia composta por três etapas: criação da base de dados, experimentos com LLMs e análise dos resultados, que são apresentadas na Figura 1 e descritas a seguir.

Criação da base de dados

Para a construção do *dataset* foram selecionados exercícios de programação, cada um acompanhado por soluções (códigos-fonte) desenvolvidas por alunos da disciplina Algoritmos e Estruturas de Dados. A construção detalhada da base de dados está sendo apresentada na Seção 4.1.

Após a coleta, os códigos foram rotulados manualmente com base nas estratégias

Figura 1. Metodologia experimental.

Fonte: Própria.

adotadas pelos alunos para resolver cada exercício. Em seguida, códigos que utilizaram a mesma estratégia foram agrupados em *clusters*. Desta forma, o *dataset* passou a ser composto por um conjunto de exercícios, cada um contendo um conjunto de *clusters*, os quais agrupam códigos que empregaram estratégias semelhantes para resolver o mesmo exercício.

Experimentos com LLMs

Foram realizados três experimentos com diferentes configurações, utilizando os modelos Gemini e ChatGPT-4o-mini em seus planos gratuitos, e ChatGPT-4o (versão paga), como descrito a seguir:

1. **Experimentos iniciais (Zero-Shot):** foi avaliada a capacidade dos modelos de rotular estratégias de programação sem exemplos de rótulos predefinidos (Zero-Shot), apenas com os *clusters* de códigos. Os *prompts* foram elaborados contendo os enunciados dos problemas e exemplos de códigos agrupados. A partir das respostas, foi avaliada a capacidade da LLM de rotular as estratégias adotadas em cada *cluster* de código;
2. **Experimentos com a abordagem One-Shot:** nesta etapa, foram aplicadas as abordagens Zero-Shot e One-Shot. Para Zero-Shot o prompt não continha exemplos de rótulos, apenas dos clusters. Na abordagem One-Shot, exemplos de *clusters* de códigos previamente rotulados foram incluídos no *prompt*. As respostas dos modelos foram analisadas para avaliar o impacto da ausência e da presença de exemplos nos resultados;
3. **Experimentos com a abordagem Few-Shot:** por fim, foi introduzida a abordagem Few-Shot, que incluiu múltiplos exemplos de *clusters* de códigos rotulados nos *prompts*. Essa etapa também utilizou o modelo ChatGPT-4o (versão paga), permitindo comparações com os modelos anteriores.

Análise dos resultados

Após a realização dos experimentos, os resultados foram analisados qualitativamente, considerando a coerência, precisão e clareza das respostas fornecidas pelos modelos. Essa análise forneceu *insights* sobre o comportamento das abordagens utilizadas, possibilitando ajustes nos *prompts* e melhorias nas futuras aplicações dos modelos. A consolidação desses resultados está sendo apresentada na Seção 5.

4. Resultados e Discussões

Esta seção apresenta a descrição das atividades desenvolvidas em cada etapa do trabalho, descrevendo os resultados dos experimentos.

4.1. Criação da base de dados

Conforme descrito na Seção 3, a primeira etapa da metodologia foi a construção de um *dataset* para os experimentos. Para isso, foram selecionados aleatoriamente 20 exercícios resolvidos por estudantes matriculados em turmas de Algoritmos e Estrutura de Dados, que é uma disciplina do segundo semestre dos cursos de Ciência da Computação e Engenharia de Software, da Universidade Federal do Amazonas (UFAM). Os códigos foram desenvolvidos nas linguagens Python e C e foram submetidos pelos alunos por meio do juiz on-line CodeBench [Galvão et al. 2016]. Para exemplificar, abaixo é descrito um dos exercícios selecionados, junto com as cinco estratégias observadas:

Enunciado do exercício: elabore um programa que leia duas listas de números inteiros e, como saída, imprima as duas listas concatenadas separadas por um espaço. A quantidade de elementos de cada lista (n_1 e n_2) deve ser lida antes das respectivas leituras das listas. Assuma que n_1 e n_2 são maiores que 0 (zero) e que cada lista possa ter até 15 elementos. Implemente as duas listas como vetores.

Rótulos:

- Estratégia 0: estrutura de dados dinâmicas (listas ligadas);
- Estratégia 1: vetores estáticos;
- Estratégia 2: vetores dinâmicos (alocação dinâmica de memória);
- Estratégia 3: manipulação simples (direta) de vetores;
- Estratégia 4: estruturas com funções auxiliares.

A partir dos dados, espera-se que as LLMs forneçam respostas com estratégias semelhantes às que foram mapeadas no *dataset* e com o mesmo formato, ou seja, uma resposta simples e direta sobre a estratégia dos códigos. Por exemplo, em um exercício onde é solicitada a implementação da estrutura de dados pilha, uma das estratégias aplicadas pode ser "Implementação de pilha estática com vetores".

4.2. Experimentos iniciais com Gemini e ChatGPT-4o-mini (Zero-Shot)

Nos experimentos iniciais, foram usados os modelos Gemini e ChatGPT-4o-mini, ambos em seus planos gratuitos, com o objetivo de avaliar suas capacidades de rotular estratégias de códigos sem rótulos predefinidos. Para isso, os modelos receberam *prompts* contendo o enunciado do problema e os *clusters* de códigos sem indicação de rótulo, ou seja, utilizando uma abordagem Zero-Shot (sem exemplos de rótulos para os *clusters*), conforme exemplo da Tabela 1.

Além do exemplo demonstrado, os modelos foram testados com outros três enunciados (dos 20 disponíveis no *dataset*), mantendo a estrutura do *prompt* sem rótulos para os *clusters* de códigos. Como resultado, ambos os LLMs demonstraram uma boa capacidade de análise dos códigos, conforme apresentado na Tabela 2.

Com relação ao comportamento dos LLMs, observou-se que o Gemini avaliou a estratégia adotada em cada código de maneira individual em vez de avaliar o *cluster* de maneira conjunta. Já o ChatGPT fez a análise como foi solicitado no *prompt*, avaliando

os *clusters* de forma geral. Percebe-se que no caso do *cluster* 0, ambos conseguiram identificar a estratégia baseada em classes para a manipulação de árvores AVL.

O ChatGPT foi ainda mais preciso ao mencionar a utilização de Programação Orientada a Objetos (POO), que consiste em uma abordagem de programação que utiliza classes e objetos como formas de abstração. Quanto ao *cluster* 1, o Gemini destacou o uso de dicionários para a representação dos nós da árvore AVL, demonstrando que operações poderiam ser realizadas diretamente com essa estrutura de dados. O GPT também identificou a utilização de dicionários para resolver a questão.

Tabela 1. Estrutura do prompt para o primeiro experimento.

Dado o enunciado:
• Neste exercício você deverá elaborar todas as funções necessárias para inserir elementos na árvore AVL e fazer rotações (simples e duplas) para que a árvore resultante seja simplesmente balanceada. Escreva um programa principal que cria várias árvores, efetuando rotações para mantê-las simplesmente balanceadas. Para cada árvore, imprima os resultados dos percursos pré-ordem e pós-ordem. • A entrada deste exercício é uma lista contendo uma sequência de chaves que devem ser inseridas em uma árvore inicialmente vazia. A entrada termina quando aparecer uma lista vazia. • Como saída, imprima primeiro a frase "Arvore {}", sendo que o especificador "{}" deve ser substituído pelo número da árvore que está sendo processada. Em seguida, imprima os percursos pré-ordem e pós-ordem da árvore obtida após a operação de rotação. Siga a formatação contida nos exemplos. Imprima uma linha em branco após cada caso de teste.
E os *clusters* abaixo (para este exemplo, foram identificadas apenas 2 estratégias de solução, cujos códigos foram agrupados em *clusters* e passados em seguida):
• Códigos do *cluster* 0, sem rótulo (total de 2 códigos) • Códigos do *cluster* 1, sem rótulo (total de 3 códigos)
Analise a estratégia de resolução abordada em cada conjunto de códigos.

Fonte: Própria.

Tabela 2. Respostas do Gemini e GPT-4o-mini no primeiro experimento.

Resposta do Gemini	**Resposta do GPT-4o-mini**
***Cluster* 0** • **Código 1:** Utiliza uma classe ThreeNode para representar os nós. Implementa as funções de altura, fator de balanceamento, atualização de altura e rotações de forma clara e concisa. A função de inserção atualiza a altura e verifica o fator de balanceamento para realizar as rotações necessárias. • **Código 2:** A função `set_altura` é responsável por atualizar a altura do nó. As rotações são implementadas de forma correta, mas poderiam ser mais detalhadas nos comentários.	***Cluster* 0** • Este *cluster* utiliza uma abordagem baseada em classes para representar os nós da árvore. A lógica de balanceamento e as rotações estão implementadas de forma clara. Ambos os códigos compartilham semelhanças significativas, mas têm algumas diferenças na organização das funções e terminologia.
***Cluster* 1** • **Código 1:** Utiliza dicionários para representar os nós, o que pode ser menos intuitivo para alguns programadores. • **Código 2:** Utiliza uma classe noArvore para representar os nós. A implementação é clara e concisa, com comentários explicando as etapas das rotações. • **Código 3:** Estrutura similar ao código 1, utilizando dicionários. A função `get_fb` calcula o fator de balanceamento.	***Cluster* 1** • Este *cluster* segue uma abordagem funcional, sem utilizar classes para representar os nós da árvore. Cada nó é representado como um dicionário contendo chaves para os atributos relevantes.

Fonte: Própria.

A partir das respostas, foi possível constatar que ambos os LLMs foram capazes de identificar as estruturas de dados e as estratégias utilizadas nos códigos dos *clusters*. Com base nesses resultados, decidiu-se testar os LLMs com outras abordagens, visando

identificar se a adição de exemplos nos *prompts* influenciaria nas respostas fornecidas pelos modelos. Então, no segundo experimento foi aplicada a abordagem One-Shot, adicionando um exemplo de rótulo no *prompt*, os resultados estão descritos a seguir.

4.3. Experimentos com a abordagem One-Shot

Na segunda etapa dos experimentos, foi empregada a abordagem One-Shot, utilizando os modelos ChatGPT-4o-mini e Gemini. O objetivo dessa fase foi avaliar como o uso de um único *cluster* rotulado (em contraste com o uso de *clusters* não rotulados do experimento anterior) nos *prompts*, impacta no desempenho dos LLMs.

Para isso, os *prompts* foram estruturados para refletir o cenário de uso real, permitindo uma análise comparativa entre as abordagens Zero-Shot e One-Shot. A estrutura do *prompt* utilizado, para a abordagem One-Shot, encontra-se na Tabela 3.

Tabela 3. Estrutura do *prompt* para o segundo experimento (One-Shot).

Olá. Eu, como professor da disciplina de Algoritmos e Estrutura de Dados, preciso identificar a estratégia utilizada pelos alunos para atingir um determinado objetivo. Irei te passar um conjunto de códigos para você analisar e me dizer qual foi a estratégia usada pelos alunos que implementaram esses códigos. Me dê somente a estratégia como resposta, nada mais. Abaixo te darei exemplos de códigos e a estratégia utilizada por eles:
Cluster de códigos e seu rótulo.
Agora, seguem os códigos dos alunos para você destacar a estratégia:
Cluster de códigos a ser rotulado.

Fonte: Própria.

Tabela 4. Resultados do segundo experimento com os LLMs.

N° Exp.	LLM	Abordagem	Formato	Rotulou corretamente
1	GPT-4o-mini	Zero-shot	Não agradável	✓
2	GPT-4o-mini	Zero-shot	Agradável	✓
3	GPT-4o-mini	Zero-shot	Agradável	✓
4	GPT-4o-mini	**One-shot**	Agradável	✓
5	GPT-4o-mini	**One-shot**	Agradável	✓
6	GPT-4o-mini	**One-shot**	Agradável	✓
7	GPT-4o-mini	**One-shot**	Agradável	✗
8	GPT-4o-mini	**One-shot**	Agradável	✗
9	GPT-4o-mini	**One-shot**	Agradável	✗
10	Gemini	Zero-shot	Não agradável	✓
11	Gemini	Zero-shot	Não agradável	✗
12	Gemini	Zero-shot	Não agradável	✓
13	Gemini	**One-shot**	Não agradável	✓
14	Gemini	**One-shot**	Não agradável	✓
15	Gemini	**One-shot**	Não agradável	✓
16	Gemini	**One-shot**	Não agradável	✗
17	Gemini	**One-shot**	Não agradável	✗
18	Gemini	**One-shot**	Não agradável	✓

Fonte: Própria.

A Tabela 4 apresenta o comportamento dos LLMs para diferentes testes realizados com o *prompt*. Ao total, seis testes utilizando One-Shot foram reproduzidos em cada

LLM e as suas respostas foram analisadas para verificar consistência na rotulação dos *clusters*. Ao adicionar um exemplo de *cluster* com códigos e rótulo correto, notou-se que o GPT-4o-mini se manteve fiel ao formato de resposta, mas não rotulou corretamente três dos seis experimentos realizados com One-Shot. Na abordagem Zero-Shot, o GPT-4o-mini acertou o rótulo em todos os três experimentos realizados. No entanto, em um dos casos, a resposta não foi apresentada em um formato agradável. É importante destacar que, ao adicionar um exemplo de *cluster* com seu respectivo rótulo, o GPT-4o-mini manteve a consistência na estrutura das respostas, sem que um formato não agradável fosse retornado.

O principal problema identificado no Gemini foi o formato prolixo de suas respostas, já observado nos experimentos anteriores (Tabela 2). Embora a estratégia fosse apresentada em uma frase, o restante do texto foi acompanhado por seções de análise dos códigos, propostas de melhorias e comentários adicionais que não foram solicitadas no *prompt*, mesmo com a inclusão de um exemplo de rótulo (One-Shot).

Tendo em vista os resultados alcançados no segundo experimento, onde a abordagem One-Shot, apesar de fornecer as respostas em um formato agradável, não rotulou de acordo com o que era esperado, resolveu-se aplicar a abordagem Few-Shot. Além de adicionar mais de um *cluster* rotulado ao *prompt*, também foi incluída a versão paga do ChatGPT-4o, visto que os resultados do GPT foram melhores do que os do Gemini.

4.4. Experimentos com as abordagens Zero-Shot, One-Shot e Few-Shot

Na etapa final, foi incluído o ChatGPT-4o, em sua versão paga, que foi testado com os *prompts* do segundo experimento, adicionando dois exemplos de One-Shot, como pode ser visto na Tabela 5. Esses exemplos consistiam em exercícios adicionais usados como *shots* ou códigos a serem rotulados pelo LLM. Visto que nos experimentos anteriores o Gemini não seguiu o que era solicitado no *prompt*, ou seja, fornecia mais informações do que era solicitado, optou-se por fazer esse experimento somente com o ChatGPT-4o (versão paga) e o ChatGPT-4o-mini (versão gratuita).

Tabela 5. Estrutura do *prompt* para o terceiro experimento (Few-Shot).

Olá. Eu, como professor de um curso de Algoritmos e Estrutura de Dados, preciso identificar a estratégia utilizada pelos alunos para atingir um determinado objetivo. Irei te passar um conjunto de códigos para você analisar e me dizer qual foi a estratégia usada pelos alunos que implementaram esses códigos. Me dê somente a estratégia como resposta, nada mais. Abaixo te darei dois exemplos de códigos e a estratégia utilizada por eles:
Cluster de códigos e seu rótulo.
Cluster de códigos e seu rótulo.
Agora, seguem os códigos dos alunos para você destacar a estratégia:
Cluster de códigos a ser rotulado.

Fonte: Própria.

Adicionalmente, foi introduzida a abordagem Few-Shot, que incluiu um exemplo extra de *cluster* rotulado. Para essa abordagem, foram utilizados cinco exemplos de exercícios (dos 20 disponíveis no dataset), cada um com três estratégias (*clusters*), para construir os *shots*.

O formato utilizado neste experimento, agora com o ChatGPT-4o (versão paga), resultou em uma melhora consistente na qualidade das respostas. Essa versão acertou todos os rótulos nos casos em que os *prompts* foram bem estruturados e incluíam pelo

menos um exemplo do formato de saída esperado, ou seja, os rótulos estavam mais claros para cada estratégia.

Com a inclusão de mais um exemplo de cluster rotulado no *prompt* (Few-Shot), o ChatGPT-4o e o ChatGPT-4o-mini mantiveram um desempenho semelhante aos exemplos que usaram a abordagem One-Shot. Entretanto, só foi possível avaliar a resposta do ChatGPT-4o-mini com dois *prompts*, como mostra a Tabela 6. Ao testar com os demais *prompts*, o ChatGPT-4o-mini apresentou limitações por conta da quantidade de caracteres do *prompt*.

Tabela 6. Resultados dos experimentos com ChatGPT-4o.

N° Exp.	LLM	Abordagem	Formato	Rotulou corretamente
1	GPT-4o	Zero-shot	Não agradável	✗
2	GPT-4o	Zero-shot	Agradável	✓
3	GPT-4o	Zero-shot	Agradável	✓
4	GPT-4o	Zero-shot	Agradável	✓
5	GPT-4o	Zero-shot	Agradável	✓
6	GPT-4o	One-shot	Agradável	✓
7	GPT-4o	One-shot	Agradável	✓
8	GPT-4o	One-shot	Agradável	✓
9	GPT-4o	One-shot	Agradável	✗
10	GPT-4o	Few-shot	Agradável	✓
11	GPT-4o	Few-shot	Agradável	✓
12	GPT-4o	Few-shot	Agradável	✓
13	GPT-4o-mini	Few-shot	Agradável	✓
14	GPT-4o-mini	Few-shot	Agradável	✓

Fonte: Própria.

Em resumo, o ChatGPT-4o (versão paga) demonstrou maior consistência na identificação das estratégias, enquanto o ChatGPT-4o-mini, no cenário Few-Shot, embora não tenha sido testado com vários casos por conta das limitações, acertou os dois testes realizados com ele, destacando a importância dos rótulos para o refinamento das respostas. Os resultados dos experimentos com ChatGPT-4o, reproduzindo os experimentos feitos anteriormente (Zero-Shot e One-Shot), e ChatGPT-4o-mini, com ambos utilizando Few-Shot, podem ser observados na Tabela 6.

5. Consolidação dos resultados

Os resultados tabulados e descritos indicaram que direcionar as respostas com exemplos de saída e códigos rotulados contribuiu significativamente para a precisão das respostas nos modelos ChatGPT-4o e ChatGPT-4o-mini. Esse direcionamento permitiu que ambos os modelos formulassem respostas concisas e próximas da expectativa, o que demonstra que um contexto bem estruturado e exemplos claros podem ajudar esses modelos a evitar ambiguidades e redundâncias. Esse padrão, entretanto, não foi observado no modelo Gemini que, apesar de atribuir rótulos corretos, gerou respostas prolixas e detalhadas além do necessário.

A natureza do problema abordado, ou seja, analisar estratégias de código e gerar rótulos adequados, exigiu que os modelos fossem capazes de identificar rapidamente

informações relevantes. Nessa tarefa, tanto o GPT-4o e o ChatGPT-4o-mini mostraram-se eficazes ao utilizar estratégias Zero-Shot, One-Shot e Few-Shot. A versão completa do ChatGPT-4o ofereceu respostas ligeiramente mais específicas do que o ChatGPT-4o-mini em alguns casos. Além disso, como foi abordado no último experimento, é importante considerar a limitação de caracteres do ChatGPT-4o-mini ao construir *prompts* mais complexos, o que restringe o uso dessa LLM e a coloca em desvantagem em comparação com a versão completa.

Em relação ao Zero-Shot, ficou claro que a construção de um contexto e a inclusão de um exemplo de saída são fatores importantes para que o modelo produza resultados melhores. Ao adicionar mais exemplos de código e seus rótulos (Few-Shot), não se observou uma mudança importante nos resultados em comparação ao One-Shot. As respostas mantiveram um formato semelhante, mas as respostas do ChatGPT-4o e do ChatGPT-4o-mini ficaram mais alinhadas, apresentando frases mais consistentes. Em testes onde o Zero-Shot foi aplicado sem um contexto bem estruturado, os resultados mostraram-se menos precisos e menos alinhados com as expectativas, o que reforça a importância da engenharia de *prompt* para atingir melhores resultados.

Os resultados encontrados neste estudo sugerem que, com ajustes adequados nos *prompts* e direcionamentos claros, modelos como o ChatGPT-4o e ChatGPT-4o-mini têm potencial para serem utilizados em tarefas de rotulagem automática, onde a precisão pode ser necessária.

Em trabalhos sobre clusterização de soluções de programação semelhantes, os algoritmos são geralmente aplicados para agrupar os códigos com base em critérios sintáticos ou de qualidade de código. Por exemplo, Beh et al. [2016] classificaram os *clusters* considerando a média de linhas dos códigos, assumindo que soluções mais longas seriam mais complexas—uma abordagem que nem sempre reflete a estratégia adotada pelos estudantes. Além disso, a análise desses *clusters* é frequentemente realizada manualmente [Yin et al. 2015], o que pode torná-la trabalhosa e suscetível a erros, especialmente quando há um grande número de *clusters* ou códigos por grupo. Nesse contexto, o uso de LLMs surge como uma alternativa promissora para avaliar as abordagens empregadas pelos estudantes na resolução de problemas.

Dentre as limitações do trabalho podem ser citadas: i) rotulagem manual do *dataset* feita por somente um humano; ii) a avaliação dos resultados ter sido feita de maneira qualitativa em sua maioria; iii) o formato dos *prompts* que melhor trabalhados poderiam resultar em respostas diferentes e a adição de mais exemplos nos *prompts* também poderia influenciar nos resultados; iv) alucinações por parte dos LLMs, ou seja, os modelos podem gerar respostas incorretas; e, v) versões gratuitas dessas ferramentas apresentam limitações quanto ao tamanho do *prompt*, o que tornou necessário o uso da versão paga.

6. Considerações finais

Este estudo analisou o desempenho de três modelos de linguagem (ChatGPT-4o, ChatGPT-4o-mini e Gemini) na identificação de estratégias de programação. Ao direcionar os modelos com exemplos de saída e rótulos corretos, tanto o ChatGPT-4o quanto o ChatGPT-4o-mini alcançaram resultados satisfatórios, gerando respostas concisas e alinhadas com as expectativas. Por outro lado, o modelo Gemini apresentou uma tendência a fornecer respostas prolixas, o que, embora tenha resultado em rótulos corretos, adicionou

um volume desnecessário de informações.

A análise destaca que o sucesso das abordagens Zero-Shot e Few-Shot depende de um contexto bem estruturado e de exemplos claros de saída. Sem essa orientação, os *prompts* geraram respostas menos direcionadas, o que pode limitar a aplicabilidade dos modelos em cenários onde a precisão é necessária, como neste estudo. Nesse sentido, o GPT-4o, nas duas versões, demonstrou ser mais robusto e adequado para a tarefa de rotulagem automática de estratégias de programação, sendo que a versão completa forneceu detalhes adicionais que podem ser úteis em contextos que exigem maior especificidade.

Esses achados sugerem que os modelos da série GPT-4o têm um potencial significativo para aplicações que envolvem análise e classificação de estratégias de programação. Para melhorar ainda mais o desempenho em futuras aplicações, recomenda-se explorar diferentes técnicas de engenharia de *prompt* que possam maximizar a precisão e concisão das respostas, ajustando o uso dos modelos conforme a complexidade e especificidade dos cenários de rotulagem.

Além disso, no contexto da educação em computação, o uso de LLMs para rotular estratégias de programação pode auxiliar os professores a compreender as abordagens adotadas pelos estudantes na resolução de questões propostas em sala de aula e verificar sua adequação ao conteúdo da disciplina. Isso também possibilita a identificação de estratégias excessivamente básicas ou avançadas, fornecendo indícios sobre o nível de compreensão dos alunos. Com essa análise, os professores podem intervir de maneira mais direcionada para oferecer suporte aos estudantes, quando necessário.

Como trabalhos futuros, pretende-se realizar novos experimentos para rotular estratégias, aprimorando os rótulos por meio de *prompts* mais estruturados, de modo que sejam mais adequados, ou seja, que forneçam informações úteis e de fácil compreensão para os professores. Além disso, planeja-se integrar os rótulos gerados pelas LLMs a outras técnicas, com o objetivo de desenvolver ferramentas mais robustas para a análise de código em disciplinas de programação.

Agradecimentos

Esta pesquisa, realizada no âmbito do Projeto Samsung-UFAM de Ensino e Pesquisa (SUPER), de acordo com o Artigo 39 do Decreto n°10.521/2020, foi financiada pela Samsung Eletrônica da Amazônia Ltda, nos termos da Lei Federal n°8.387/1991, através do convênio 001/2020 firmado com a UFAM e FAEPI, Brasil.

O presente trabalho também foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES-PROEX) - Código de Financiamento 001. Este trabalho foi parcialmente financiado pela Fundação de Amparo à Pesquisa do Estado do Amazonas - FAPEAM - por meio do projeto POSGRAD 2024/2025 e também recebeu apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (Processo 303443/2023-5).

Referências

Barbosa, A., Barros Costa, E., and Brito, P. H. (2023). Juízes online são suficientes ou precisamos de um VAR? In *Anais do III Simpósio Brasileiro de Educação em Computação*, pages 386–394. SBC.

Beh, M. Y., Gottipatti, S., LO, D., and Shankararaman, V. (2016). Semi-automated tool for providing effective feedback on programming assignments.

Brown, T. B. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., et al. (2024). A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 15(3):1–45.

Galvão, L., Fernandes, D., and Gadelha, B. (2016). Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 27, page 140.

Glassman, E. L., Scott, J., Singh, R., Guo, P. J., and Miller, R. C. (2015). Overcode: Visualizing variation in student solutions to programming problems at scale. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 22(2):1–35.

Joyner, D., Arrison, R., Ruksana, M., Salguero, E., Wang, Z., Wellington, B., and Yin, K. (2019). From clusters to content: Using code clustering for course improvement. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 780–786.

Koivisto, T. and Hellas, A. (2022). Evaluating codeclusters for effectively providing feedback on code submissions. In *2022 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. IEEE.

Mehta, A., Gupta, N., Balachandran, A., Kumar, D., Jalote, P., et al. (2023). Can ChatGPT play the role of a teaching assistant in an introductory programming course? *arXiv preprint arXiv:2312.07343*.

Melo, R., Pessoa, M., and Fernandes, D. (2024). Clusterização de soluções de exercícios de programação: um mapeamento sistemático da literatura. In *Anais do 35º Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 1715–1729. Sociedade Brasileira de Computação.

Pires, R., Abonizio, H., Almeida, T. S., and Nogueira, R. (2023). Sabiá: Portuguese large language models. In *Brazilian Conference on Intelligent Systems*, pages 226–240. Springer.

Poldrack, R. A., Lu, T., and Beguš, G. (2023). AI-assisted coding: Experiments with GPT-4. *arXiv preprint arXiv:2304.13187*.

Reiss, M. V. (2023). Testing the reliability of ChatGPT for text annotation and classification: A cautionary remark. *arXiv preprint arXiv:2304.11085*.

Wasik, S., Antczak, M., Badura, J., Laskowski, A., and Sternal, T. (2018). A survey on online judge systems and their applications. *ACM Computing Surveys (CSUR)*, 51(1):1–34.

Yin, H., Moghadam, J., and Fox, A. (2015). Clustering student programming assignments to multiply instructor leverage. In *Proceedings of the second (2015) ACM conference on learning@ scale*, pages 367–372.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al. (2023). A survey of large language models. *arXiv preprint arXiv:2303.18223*.