Uso de LLMs no Agrupamento e Classificação de Estratégias de Programação em Juízes On-line

Davi Aguiar[1], Rafaela Melo[1,2], Fernanda Pires[1], Marcela Pessoa[1], David Fernandes[2]

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

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

{dam.eng23,fpires,msppessoa}@uea.edu.br

{rmelo,david}@icomp.ufam.edu.br

Abstract. Introduction: *Online judges are widely used in programming courses, but most only consider the correctness of the code, without taking other aspects into account. Understanding the strategies adopted to solve problems is important because it allows the teacher to assess, for example, whether students are assimilating the course content and applying it appropriately.* ***Objective:*** *This work investigates the use of LLMs to identify, group, and describe strategies for solving programming exercises.* ***Methodology:*** *To this end, six different LLMs were used to make the clustering and description.* ***Results:*** *Preliminary results indicate that, with the appropriate formulation of the prompts, LLMs have the potential to perform the strategy classification task, with GPT showing the most consistent performance among the models evaluated.*
Keywords: *Online Judges, Computer Science Education, Large Language Models, Programming Strategies*

Resumo. Introdução: *Juízes on-line são amplamente utilizados em disciplinas de programação, mas em sua maioria, consideram apenas a corretude dos códigos, sem levar em conta outros aspectos. Compreender as estratégias adotadas para resolver problemas é importante, pois permite ao professor avaliar, por exemplo, se os alunos estão assimilando os conteúdos da disciplina e aplicando-os adequadamente.* ***Objetivo:*** *Neste trabalho, investiga-se o uso de LLMs para identificar, agrupar e descrever estratégias para resolver exercícios de programação.* ***Metodologia:*** *Foram utilizados seis LLMs para realizar o agrupamento e descrição.* ***Resultados:*** *Os resultados preliminares indicam que, com a formulação adequada dos prompts, os LLMs têm potencial para realizar a tarefa de classificação de estratégias, com o GPT apresentando o desempenho mais consistente entre os modelos avaliados.*
Palavras-Chave: *Juízes On-line, Educação em Computação, Large Language Models, Estratégias de Programação*

1. Introdução

Juízes on-line são sistemas de correção automática de códigos-fonte comumente empregados como ferramentas de apoio pedagógico em disciplinas de programação

[Carvalho et al. 2016]. Embora esses sistemas sejam úteis para facilitar e agilizar a correção de exercícios [Wasik et al. 2018], seu uso pode acabar por distanciar o docente da análise direta dos códigos produzidos pelos estudantes. Esse fato é normalmente visto como uma desvantagem do uso de juízes on-line, visto que esses sistemas geralmente não avaliam critérios pedagógicos ou qualitativos do código submetido [Barbosa et al. 2023].

Seguindo essa linha, 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 a sequência de Fibonacci até certa posição, os estudantes podem escolher seguir uma abordagem 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 de entendimento dos estudantes, quanto para fazer intervenções caso seja necessário [Barbosa et al. 2023].

Porém, analisar cada código individualmente é uma tarefa que demanda bastante tempo, principalmente porque disciplinas de programação costumam ter muitos estudantes. Além disso, considerando que programar é algo que exige prática, os professores precisam preparar e corrigir diversos exercícios ao longo da disciplina. Nesse sentido, alguns trabalhos na literatura propõem o agrupamento de soluções similares com uso de algoritmos de clusterização [Melo et al. 2024], o que pode auxiliar o trabalho do professor no processo de avaliação. No entanto, a maioria desses algoritmos funciona com base em similaridade sintática, capturando apenas diferenças superficiais, como estilo de escrita ou quantidade de linhas de código, fatores esses que não necessariamente estão vinculados à estratégia de programação abordada pelo estudante [Glassman et al. 2015].

Paralelamente a isso, os *Large Language Models* (LLMs) têm emergido nos últimos anos como uma tecnologia relevante, sendo aplicados em diferentes tarefas de linguagem natural, como geração de resumos, traduções, análises e classificação de texto [Pires et al. 2023]. Por conta disso, esses modelos são explorados em diversas áreas de estudo, incluindo a educação em computação [Mehta et al. 2023]. Uma tarefa em que os LLMs também são frequentemente utilizados é a geração e análise de código [Chen et al. 2021, Li et al. 2022], o que faz com que se apresentem como boas ferramentas de apoio em disciplinas de programação. Dessa forma, como possuem potencial para interpretar código, podem ser explorados, por exemplo, para auxiliar professores na revisão de códigos de estudantes [Piscitelli et al. 2025].

Sendo assim, a proposta desse trabalho é avaliar o comportamento de modelos de linguagem natural na tarefa de identificar, agrupar e descrever diferentes estratégias adotadas por estudantes ao resolverem exercícios de programação em juízes on-line. Para tanto, foram explorados seis diferentes modelos: *GPT-5-mini*, *Gemini 3.0 Pro Preview*, *DeepSeek-Chat*, *DeepSeek-Reasoner*, *Claude Sonnet 4.6* e *Claude Opus 4.6*. Os resultados demonstraram que os LLMs têm potencial para interpretar e descrever estratégias, com destaque para o *GPT-5-mini*, que apresentou melhor desempenho.

O restante do trabalho está organizado como segue: a Seção 2 apresenta trabalhos relacionados com a pesquisa, a Seção 3 discorre sobre a metodologia aplicada para avaliar o uso de LLMs na tarefa de identificação de estratégias de programação, a Seção 4 apresenta os resultados encontrados, na Seção 5 está a discussão dos resultados, e, por fim, na Seção 6, as considerações finais do trabalho.

2. Trabalhos Relacionados

Juízes on-line são muito utilizados em disciplinas de programação, por fornecer benefícios como variedade de exercícios e, principalmente, a correção automática de código [Wasik et al. 2018], sendo ferramentas de apoio úteis para os professores. Esses sistemas funcionam com base em casos de teste previamente definidos, ou seja, o código do estudante é executado para um determinado conjunto de entradas e a saída resultante é comparada com a saída esperada pelo sistema. Dessa forma, não são considerados outros aspectos relevantes, como qualidade do código ou até mesmo a forma como o estudante desenvolveu a solução [Barbosa et al. 2023].

Apesar dos benefícios apresentados pelos juízes on-line, o professor pode acabar ficando dependente da correção fornecida por essas ferramentas, limitando a capacidade do docente de identificar dificuldades ou as estratégias de programação adotadas na resolução dos exercícios. Além disso, em um estudo recente, Figueras et al. [2025] apontaram que, em juízes on-line, muitas vezes os estudantes se preocupam apenas em produzir a saída correta esperada para um exercício, o que pode causar impacto negativo no desenvolvimento adequado das habilidades de programação dos alunos.

Entretanto, analisar manualmente os códigos dos estudantes também não é uma tarefa trivial, pois demanda tempo e esforço consideráveis da parte do professor. Diante desse contexto, alguns trabalhos na literatura buscaram formas de auxiliar os professores nessa tarefa. Koivisto e Hellas [2022] investigaram o uso de algoritmos de clusterização para gerar agrupamentos de códigos de alunos, por meio de um sistema intitulado CodeClusters. A ideia dos autores era utilizar a ferramenta em conjunto com um juíz on-line e facilitar o processo de revisão em larga escala, ou seja, fornecer *feedback* mais direcionado a grupos de estudantes com códigos sintática e estruturalmente semelhantes.

Em Paiva et al. [2025], os autores apontam que a maioria das abordagens de clusterização existentes se baseiam em características sintáticas do código. Então, é proposta uma ferramenta chamada AsanasCluster, baseada em aspectos semânticos das soluções, onde, quando uma nova submissão chega, um algoritmo de clusterização determina qual o agrupamento mais adequado. Os autores realizaram uma comparação com outras ferramentas de clusterização de códigos existentes na literatura e, em alguns casos, o AsanasCluster demonstrou desempenho melhor, além de ser mais eficaz em identificar estratégias distintas.

Com relação ao uso de LLMs para análise de código no ensino de programação, Chen et al. [2021] demonstraram a capacidade dos LLMs de gerar código a partir de descrições em linguagem natural, demonstrando que essas ferramentas podem ser utilizadas como suporte em disciplinas de programação. Já Poldrack et al. [2023] avaliaram o ChatGPT-4 para geração e refatoração de código, concluindo que, embora o modelo tenha grande potencial, a interação humana ainda é necessária para garantir a qualidade dos resultados obtidos.

Mehta et al. [2023] avaliaram o ChatGPT como assistente de *feedback* em uma disciplina introdutória de programação, observando que o modelo não é totalmente confiável para avaliar a corretude das soluções, mas fornece boas sugestões de melhoria. Em MacNeil et al. [2023], os LLMs são explorados para geração automática de explicações de código em plataformas educacionais, evidenciando o potencial dessas ferramentas para

auxiliar os estudantes a entenderem suas próprias soluções e como podem melhorá-la. Esses resultados também indicam que os LLMs são eficientes na tarefa de interpretar código e consideram aspectos além da sintaxe do código.

Mais próximo da proposta deste trabalho, em um estudo anterior conduzido pelos presentes autores [Melo et al. 2026], foi investigado o uso de algoritmos de clusterização para agrupar códigos de programação, onde os resultados demonstraram que os *clusters* resultantes não correspondiam realmente a estratégias semelhantes. Diante disso, também foram realizados experimentos com LLMs, *ChatGPT-4.1 Mini* e *Gemini 2.5 Flash*, com o objetivo de rotular automaticamente *clusters* de código de estudantes de Algoritmos e Estruturas de Dados. Os experimentos incluíram abordagens *Zero-Shot*, *One-Shot* e *Few-Shot*, utilizando como entrada os enunciados dos exercícios e grupos de códigos já pré-agrupados. Os resultados demonstraram que os modelos desempenharam bem a tarefa de rotular estratégias, principalmente com *prompts* bem estruturados. Nesse sentido, os LLMs demonstraram maior potencial para interpretar estratégias de programação, considerando características semânticas, em comparação com algoritmos tradicionais de clusterização.

O presente trabalho se diferencia dos demais por: (i) utilizar LLMs tanto para agrupar quanto para descrever as estratégias presentes em cada agrupamento, em vez de usar algoritmos de clusterização tradicionais, como os mencionados CodeClusters [Koivisto and Hellas 2022] e AsanasCluster [Paiva et al. 2025]; ii) solicitar aos LLMs que realizem o próprio agrupamento a partir de sequências de submissão individuais, sem receber previamente exemplos de *clusters* já definidos (semelhante à abordagem de Zero-Shot em Melo et al. [2026]); e, (iii) explorar seis diferentes LLMs para a tarefa de identificar estratégias (*GPT-5-mini*, *Gemini 3.0 Pro Preview*, *DeepSeek-Chat*, *DeepSeek-Reasoner*, *Claude Sonnet 4.6* e *Claude Opus 4.6*).

3. Metodologia

Para avaliar o uso de LLMs na tarefa de agrupar, identificar e descrever estratégias adotadas para solucionar exercícios de programação, o procedimento metodológico adotado neste trabalho é composto por cinco etapas: i) preparação dos *prompts*, usando como entradas os códigos de estudantes de programação, ii) execução de *prompts* manual, iii) preparação de *workflow* automatizado, iv) execução dos *prompts* nos LLMs e v) análise dos resultados. As etapas estão ilustradas na Figura 1 e descritas a seguir.

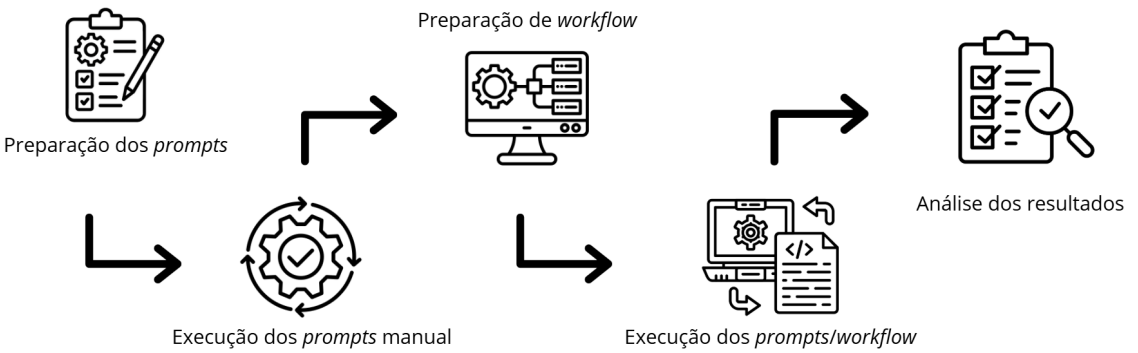

Figura 1. Procedimento metodológico.

Preparação dos *prompts*

Como etapa inicial, é realizada a preparação dos *prompts* para obter as respostas fornecidas pelos modelos de linguagem. Para isso, são utilizados códigos reais de estudantes de programação. Assim, o *prompt* padrão é composto por três partes: o enunciado da questão, os códigos dos alunos e a solicitação de resposta ao modelo. Na solicitação, é pedido que o LLM realize uma descrição curta de cada código, não podendo passar de 100 caracteres. O objetivo dessa regra é evitar a alucinação por parte desses modelos. Após essas descrições feitas, é então solicitado que seja feito o agrupamento dos códigos com base nas estratégias aplicadas. Por fim, é instruído que seja feita uma descrição para cada agrupamento.

A abordagem adotada nos experimentos é a *Zero-Shot* [Reiss 2023], ou seja, nenhum exemplo de rótulo ou agrupamento é fornecido ao modelo, sendo disponibilizados apenas o enunciado e os códigos. A ideia, além de explorar a capacidade dos LLMs em agrupar e descrever estratégias, é também simular um ambiente real, onde os códigos não chegam pré-agrupados. Incluir exemplos de agrupamentos rotulados seria menos natural neste cenário.

Execução dos *prompts* manual

Inicialmente, são usados dois LLMs para avaliar a capacidade de rotulação de estratégias de códigos. Para isso, ambos os modelos receberam *prompts* contendo o enunciado da questão e os códigos desenvolvidos pelos estudantes, identificados anonimamente. Essa etapa foi realizada de forma manual, sendo necessário executar cada *prompt* individualmente nos dois LLMs.

Organização do *workflow*

Para facilitar o processo de execução dos *prompts*, é criado um *pipeline* de envio e coleta de respostas com a ferramenta n8n[1], uma plataforma de automação de fluxos de trabalho que permite integrar APIs de diferentes serviços. O fluxo processa cada exercício da seguinte forma:

1. O enunciado do exercício é concatenado com as sequências de código de todos os alunos do respectivo exercício.
2. Um *prompt* estruturado é enviado simultaneamente para diferentes LLMs.
3. O *prompt* solicita, em três passos: **(a)** uma descrição curta (até 100 caracteres) da estratégia utilizada em cada código; **(b)** o agrupamento dos códigos baseado na descrição dessas estratégias; e **(c)** uma descrição da estratégia daquele grupo.
4. As respostas são coletadas e armazenadas em uma planilha para análise comparativa.

Execução dos *prompts*/*workflow*

Após a preparação do *workflow* automatizado, os *prompts* são executados nos LLMs selecionados e as respostas são armazenadas em uma planilha para análise posterior. Esta etapa exige a criação de chaves API para executar os modelos e as conexões com Google Drive e Google Planilhas.

Análise dos resultados

A análise dos dados segue uma abordagem qualitativa, comparando os agrupamentos gerados por cada LLM com os agrupamentos e rótulos esperados. A

[1] `https://n8n.io`

avaliação considera: (a) correspondência entre os grupos gerados e os grupos esperados; (b) a granularidade das distinções realizadas pelo modelo (se o LLM identificou subgrupos relevantes ou colapsou grupos distintos em um único); e (c) a qualidade pedagógica das descrições geradas.

4. Resultados

Esta seção apresenta os resultados obtidos através dos experimentos realizados neste trabalho, além das discussões acerca desses resultados.

4.1. Construção do *prompt*

Para a construção do *prompt*, utilizou-se uma base de códigos construída em um estudo anterior[2] [Melo et al. 2026]. Esses códigos foram utilizados como entrada para solicitar aos LLMs a identificação das estratégias presentes neles. A estrutura do *prompt* pode ser visualizada na Tabela 1.

Tabela 1. Estrutura do *prompt* utilizado nos experimentos.

Prompt
Dado o enunciado: {enunciado do exercício}
Estou lhe fornecendo soluções de código abaixo:
[{{id}: {código 1}}, {{id}: {código 2}}, ..., {{id}: {código n}}]
Você deve descrever a estratégia de cada código. A descrição deve ser curta e não deve passar de 100 caracteres. Em seguida, faça o agrupamento desses códigos por estratégia. Realize a descrição de cada agrupamento.

A base de dados é composta por seis exercícios de programação de disciplinas de Algoritmos e Estruturas de Dados, ofertadas nos cursos de Ciência da Computação, Engenharia da Computação e Engenharia de Software, da Universidade Federal do Amazonas (UFAM). As submissões foram capturadas entre os anos de 2021 e 2023 por meio do juíz on-line CodeBench[3]. Os exercícios cobrem os seguintes tópicos: listas encadeadas (Exercício 1), pilhas (Exercício 2), listas duplamente encadeadas (Exercício 3), árvores genéricas (Exercício 4), ordenação com *Selection Sort* (Exercício 5) e rotação em árvores AVL (Exercício 6).

Tabela 2. Resumo dos exercícios, agrupamentos e estratégias existentes na base de dados.

Exercício	**Tópico**	**Grupos**	**Resumo de estratégias**
1	Intercalação de listas	4	Iterativo; recursivo com nova lista; recursivo sem nova lista; dupla reversão
2	Validação com pilhas	2	Comparação simbólica direta vs. aritmética ASCII
3	Listas duplamente encadeadas	4	Tipo de lista; direção de inserção/remoção; lista circular
4	Árvore genérica	3	Representação filho-irmão; ordem de busca; lista de filhos
5	*Selection Sort*	5	Vetor de structs; vetor de ponteiros; lista simplesmente encadeada; lista duplamente encadeada; qsort
6	Rotação AVL	2	Representação por classe vs. dicionário

Para cada exercício existem seis códigos produzidos por diferentes estudantes. Sendo assim, a base contém ao todo 36 arquivos de códigos. Cada exercício é acompanhado por seu enunciado e por arquivos de texto contendo os rótulos, ou seja, descrições de

[2]Disponível em: `https://github.com/rafaelamwlo/exercisesdata.git`
[3]`https://cb.icomp.ufam.edu.br`

estratégias, que correspondem a agrupamentos de códigos com estratégias semelhantes. Esses agrupamentos e rótulos foram feitos e validados por professores com experiência em programação. A Tabela 2 apresenta, para cada exercício, o número de agrupamentos e um resumo das estratégias utilizadas para resolver os problemas. Esses rótulos e agrupamentos serviram como base para avaliar o comportamento dos LLMs nos experimentos, como descrito a seguir.

4.2. Experimentos manuais com LLMs

Como uma fase prévia dos experimentos, decidiu-se testar o desempenho de dois modelos (ChatGPT[4] e Gemini[5]) na tarefa de rotulação das estratégias. O objetivo desta etapa foi avaliar como os modelos de linguagem se comportavam ao receber a tarefa de rotulação das estratégias. A Tabela 3 mostra o *prompt* utilizado com os dois LLMs, seguindo o padrão definido anteriormente. A Tabela 4 demonstra as respostas exatas que ambos os modelos forneceram, seguindo as instruções solicitadas.

Tabela 3. Estrutura do *prompt* utilizado nos experimentos.

Dado o enunciado:
[Enunciado do exercício]
Estou lhe fornecendo soluções de código em anexo. Você deve descrever a estratégia de cada código. A descrição deve ser curta e não deve passar de 100 caracteres. Em seguida, faça o agrupamento desses códigos por estratégia. Realize a descrição de cada agrupamento
Códigos em anexo: 3594_5095_5710.log, 3616_5095_5710.log, 5704_5095_5710.log, 3609_5095_5710.log, 1790_5095_5710.log, 1924_5095_5710.log

Tabela 4. Respostas dos modelos.

Resposta do Gemini	**Resposta do ChatGPT**
Estratégia: Classe com nós **Códigos:** 3594, 3609, 3616, 5704 **Descrição:** Implementam a árvore com classes de nós (No, Node, Arvore), usam inserção recursiva e aplicam rotação RR alterando ponteiros de objetos. Percursos pre e pós-ordem são feitos com recursão direta	**Agrupamento 1: Estrutura baseada em Classes** • **Códigos:** 3594_5095_5710.log, 3616_5095_5710.log, 5704_5095_5710.log, 3609_5095_5710.log • **Descrição:** Este agrupamento utiliza a abordagem de programação orientada a objetos. Uma classe é definida para representar cada nó da árvore, contendo o valor e as referências para os filhos esquerdo e direito. As operações de inserção, rotação e percurso são implementadas como funções que manipulam os objetos dessa classe.
Estratégia: Estrutura com dicionários **Códigos:** 1790, 1924 **Descrição:** Modelam os nós como dicionários com chaves 'valor', 'esq' e 'dir'. Inserem elementos e aplicam rotação RR manipulando os dicionários recursivamente. Simulam estrutura de árvore sem classe.	**Agrupamento 2: Estrutura baseada em Dicionários** • **Códigos:** 1790_5095_5710.log, 1924_5095_5710.log • **Descrição:** As soluções neste grupo representam os nós da árvore utilizando dicionários, conforme sugerido pelo enunciado. Cada dicionário contém chaves para o valor do nó ('valor') e para as sub-árvores esquerda ('esq') e direita ('dir'). Funções independentes são utilizadas para manipular essas estruturas de dados para construir a árvore, aplicar a rotação e realizar os percursos.

Os modelos apresentaram algumas diferenças nas estruturas das respostas. Por exemplo, o Gemini exibiu apenas a primeira parte do ID de cada código (o que não é um problema, visto que é a única parte diferente entre os nomes dos arquivos). Além

[4]`https://chatgpt.com/`
[5]`https://gemini.google.com/`

disso, nota-se o uso da nomenclatura de "estratégia" pelo Gemini e "agrupamento" pelo ChatGPT, evidenciando o comportamento não-determinístico dos modelos de linguagem em geral.

Apesar dessas diferenças, foi possível constatar que ambos os LLMs foram capazes de identificar as estruturas de dados e as estratégias utilizadas pelos estudantes. Com base nesses resultados, decidiu-se realizar experimentos com outros exercícios. Além disso, os modelos do DeepSeek e da Anthropic também foram adicionados para realizar a comparação do desempenho da rotulação com seis LLMs, com o fim de verificar como diferentes modelos de linguagem realizam essa tarefa.

4.3. Comparação entre os LLMs

Como mencionado na Seção 3, utilizou-se o n8n para envio do *prompt* e obtenção das respostas de cada modelo via API, o que permitiu facilitar a execução dos *prompts* e aumentar o número de LLMs testados. Para isso, foi necessário construir um *workflow*[6] que implementa a busca dos códigos e do enunciado por exercício, monta o *prompt* concatenando os códigos e enunciado, envia-o via requisição para os modelos e salva as respostas em uma planilha. A Figura 2 representa o fluxo de automação construído com a ferramenta.

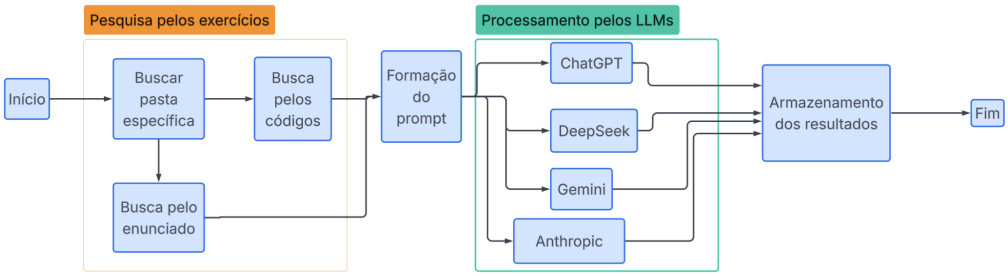

Figura 2. *Workflow* para salvar respostas dos LLMs.

Nessa etapa foram realizados testes com seis LLMs distintos: *GPT-5-mini*, *Gemini 3.0 Pro Preview*, *DeepSeek-Chat*, *DeepSeek-Reasoner*, *Claude Sonnet 4.6* e *Claude Opus 4.6*. A Tabela 5 apresenta um resumo comparativo dos agrupamentos produzidos pelos LLMs em relação ao gabarito para os seis exercícios analisados, além do tipo de acesso (gratuito ou pago) à API de cada LLM.

Tabela 5. Resultados dos experimentos realizados.

Modelo	Acesso	Ex. 1	Ex. 2	Ex. 3	Ex. 4	Ex. 5	Ex. 6
GPT-5-mini	Gratuito	✓	✓	✓	✗	✗	✓
Gemini 3 Pro	Gratuito	✗	✓	✗	✗	✗	✓
DeepSeek-chat	Pago	✗	✗	✗	✗	✗	✗
DeepSeek-reasoner	Pago	✗	✗	✗	✗	✗	✓
Claude Sonnet 4.6	Pago	✓	✗	✗	✗	✗	✗
Claude Opus 4.6	Pago	✗	✗	✗	✗	✗	✓

Legenda: ✓ Rotulagem correta ✗ Rotulagem incorreta.

[6]Disponível em: `https://github.com/daviagrm/n8nworkflow.git`

Para o critério de rotulagem, foi avaliado se o modelo foi capaz de realizar o mesmo número de agrupamentos e separar as mesmas estratégias previstas no gabarito (Tabela 2). É possível visualizar que, entre os LLMs testados, o GPT apresentou o melhor desempenho, conseguindo rotular corretamente quatro dos seis exercícios. Além disso, percebe-se que os modelos que não tiveram custo obtiveram maior assertividade em relação aos que tiveram, o que é o oposto do esperado, já que os modelos pagos são mais recentes em relação aos gratuitos. Na seção a seguir são discutidos os resultados encontrados neste trabalho.

5. Discussão

De forma geral, percebe-se que a maioria dos modelos não obteve desempenho satisfatório na tarefa de rotulagem das estratégias dos códigos dos estudantes. O *GPT-5-mini* foi o modelo com melhor desempenho geral, acertando quatro dos seis exercícios. As descrições produzidas foram concisas e objetivas.

Um bom exemplo do desempenho positivo do modelo ocorreu no Exercício 2 (validação de expressões com pilhas), onde ele agrupou corretamente os alunos que utilizam comparação simbólica direta daqueles que empregaram aritmética baseada em valores ASCII. As discrepâncias ocorreram com o Exercício 4 (árvore genérica), onde o modelo fundiu soluções que o gabarito distingue pela ordem de busca (ainda que sua descrição tenha indicado percepção da diferença) e o Exercício 5 (*SelectionSort*), onde o modelo uniu soluções que utilizam vetores de *structs* e ponteiros no mesmo agrupamento, mas também mencionando a diferença na descrição.

O *Gemini 3.0 Pro* produziu descrições mais ricas e detalhadas entre os seis modelos, identificando nuances como diferenças de eficiência assintótica entre implementações iterativas (O(N) vs. O(1) por inserção no Exercício 1) e distinções entre as formas de validação da entrada do código, como interromper no meio do *loop* quando a entrada está errada ou realizar o processamento da entrada completa, e retornar o erro depois (Exercício 2). Nos Exercícios 1, 3 e 5, o modelo cometeu erros semelhantes ao *GPT-5-mini*, ao fundir duas estratégias em um só *cluster*.

Os dois modelos da família *DeepSeek* apresentaram desempenho insatisfatório, ainda que com perfis de erro distintos. O *DeepSeek-Chat* não acertou nenhum dos seis exercícios, cometendo erros tanto ao agrupar códigos com estratégias diferentes quanto ao separar códigos com estratégias semelhantes. O caso mais evidente ocorreu no Exercício 2, no qual o modelo reuniu todos os seis alunos em um único grupo, descrevendo apenas que "todos utilizam pilha para empilhar aberturas e desempilhar ao encontrar fechamento correspondente", ignorando completamente a distinção entre comparação simbólica e aritmética ASCII.

O *DeepSeek-Reasoner*, por sua vez, apresentou desempenho ligeiramente superior, acertando apenas o Exercício 6, que envolve a distinção entre o uso de classes e dicionários. Ainda assim, nos demais exercícios, o modelo também apresentou dificuldades em capturar distinções mais sutis entre as estratégias, como diferenças na ordem de busca em árvores ou no tipo de estrutura de lista utilizada.

Os modelos da *Anthropic*, *Claude Sonnet 4.6* e *Claude Opus 4.6*, também apresentaram desempenho limitado, acertando apenas um exercício cada. O *Claude Sonnet*

4.6 acertou o Exercício 1 (intercalação de listas), distinguindo corretamente as estratégias iterativas e recursivas, incluindo a variante que utiliza a estratégia recursiva duas vezes. O *Claude Opus 4.6*, por sua vez, acertou o Exercício 6 (rotação AVL), conseguindo separar adequadamente as soluções baseadas em classes daquelas representadas por dicionários. Nos demais exercícios, ambos os modelos cometeram erros ao reunir em um único grupo estratégias que o gabarito distinguia em dois ou até mais grupos. Vale destacar que, nos exercícios em que erraram, os modelos produziram descrições individuais coerentes para cada código, mas falharam na etapa de agrupamento, o que indica que a dificuldade está menos na interpretação semântica e mais na decisão de quando uma diferença justifica a criação de um grupo distinto.

Em síntese, os LLMs têm potencial real para identificar automaticamente estratégias de programação. A contribuição adicional deste trabalho é mostrar que esse potencial se mantém mesmo quando o modelo precisa construir os agrupamentos a partir das submissões individuais, sem receber os *clusters* pré-formados. A diferença de desempenho entre os modelos reforça que a escolha do LLM é um fator crítico: o GPT-5-mini mostrou-se o mais equilibrado para essa tarefa na configuração *Zero-Shot*, o Gemini pode ser mais adequado quando se deseja riqueza descritiva, e os modelos DeepSeek e Claude revelaram-se menos confiáveis em cenários de distinções sutis.

Diante disso, dentro do contexto da educação em computação, o uso de LLMs para agrupar, identificar e descrever estratégias de programação pode auxiliar os docentes a compreender a forma como os estudantes resolvem problemas. Isso também permite a identificação de abordagens extremamente simples ou complexas por parte dos estudantes, fornecendo *feedbacks* sobre o nível de entendimento dos alunos. Com a análise desses *feedbacks*, o professor pode usar isso a seu favor para oferecer suporte pedagógico aos alunos, caso necessário.

6. Considerações Finais

Este trabalho investigou o uso de LLMs para classificar automaticamente as estratégias de resolução de problemas de programação adotadas por estudantes em juízes on-line. O *pipeline* proposto, implementado com n8n, submete sequências de código a seis modelos usando uma abordagem *Zero-Shot*.

Os experimentos com seis exercícios de Algoritmos e Estruturas de Dados indicam que o *GPT-5-mini* apresentou o melhor desempenho geral, com concordância plena em quatro dos seis exercícios. O *Gemini 3.0 Pro* produziu descrições mais ricas, mas com um desempenho menos satisfatório. Os modelos *DeepSeek* e *Claude* demonstraram desempenho inferior, colapsando grupos distintos e ignorando estratégias relevantes.

Como limitações do trabalho, destacam-se: (i) a avaliação qualitativa, sem métricas formais de concordância; (ii) a base de dados pequena (36 códigos de 6 exercícios); (iii) o gabarito construído por um único especialista; e (iv) o uso exclusivo de abordagem *Zero-Shot*, sem explorar os efeitos de *prompts* com exemplos (*Few-Shot*), que mostraram resultados promissores em Melo et al. [2026].

Como trabalhos futuros, pretende-se ampliar a base de dados com mais exercícios e alunos; avaliar abordagens *One-Shot* e *Few-Shot*; realizar experimentos com mais modelos de linguagem (e.g., Qwen e Grok); investigar o impacto de diferentes *designs* de *prompt*; e integrar o *pipeline* ao juiz on-line para oferecimento de *feedback* em tempo real.

7. Declaração sobre uso de Inteligência Artificial

Declara-se que foram utilizados recursos de Inteligência Artificial para a realização dos experimentos desta pesquisa, bem como auxílio na escrita, para melhorar alguns trechos do artigo.

Agradecimentos

O presente trabalho foi realizado com o apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (AUXPE-CAPES-PROEX) - Código de Financiamento 001. Adicionalmente, este trabalho foi parcialmente financiado pela Fundação de Amparo à Pesquisa do Estado do Amazonas - FAPEAM - por meio do projeto PDPG-CAPES.

Os autores expressam sua gratidão à Universidade do Estado do Amazonas (UEA) pelo apoio institucional. Agradecem também aos seus colegas do ThinkTEd Lab pelas contribuições e discussões que enriqueceram este trabalho. Os autores reconhecem ainda o apoio do centro de pesquisa Nexus, que forneceu recursos materiais essenciais, incluindo espaço físico, computadores e infraestrutura relacionada.

Referências

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

Carvalho, L. S., Oliveira, D. B., and Gadelha, B. F. (2016). Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. In *Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 140–149. SBC.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Figueras, C., Farazouli, A., Cerratto Pargman, T., McGrath, C., and Rossitto, C. (2025). Promises and breakages of automated grading systems: a qualitative study in computer science education. *Education Inquiry*, pages 1–22.

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.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. (2022). Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

MacNeil, S., Tran, A., Hellas, A., Kim, J., Sarsa, S., Denny, P., Bernstein, S., and Leinonen, J. (2023). Experiences from using code explanations generated by large language models in a web software development e-book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 931–937.

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. *Simpósio Brasileiro de Informática na Educação (SBIE)*, pages 1715–1729.

Melo, R., Souza, T., Pires, F., Oliveira, E., Carvalho, L., Pessoa, M., and Fernandes, D. (2026). Exploring the use of clustering algorithms and llms to identify programming strategies. *Revista Brasileira de Informática na Educação*, 34:59–82.

Paiva, J. C., Leal, J. P., and Figueira, Á. (2025). Clustering source code from automated assessment of programming assignments. *International Journal of Data Science and Analytics*, 20(2):1581–1592.

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.

Piscitelli, A., De Rosa, M., Fuccella, V., Costagliola, G., et al. (2025). Large language models for student code evaluation: Insights and accuracy. In *CSEDU (2)*, pages 534–544.

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.