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

Submission: 10/09/2025; Camera ready: 19/12/2025; 1st round notif.: 04/12/2025; Edition review: 09/02/2026; Available online: 09/02/2026; Published: 09/02/2026;

Exploring the Use of Clustering Algorithms and LLMs to Identify Programming Strategies

Rafaela Melo
Federal University of Amazonas (UFAM)
ORCID: 0000-0001-6198-8962
rmelo@icomp.ufam.edu.br

Tiago Souza
Federal University of Amazonas (UFAM)
ORCID: 0009-0002-9407-992X
tiago.souza@icomp.ufam.edu.br

Fernanda Pires
University of the State of Amazonas (UEA)
ORCID: 0000-0002-8180-9850
fpires@uea.edu.br

Elaine Oliveira
Federal University of Amazonas (UFAM)
ORCID: 0000-0003-2884-9359
elaine@icomp.ufam.edu.br

Leandro Carvalho
Federal University of Amazonas (UFAM)
ORCID: 0000-0003-2970-2084
galvao@icomp.ufam.edu.br

Marcela Pessoa
University of the State of Amazonas (UEA)
ORCID: 0000-0002-3064-5585
msppessoa@uea.edu.br

David Fernandes
Federal University of Amazonas (UFAM)
ORCID: 0000-0002-2887-2324
david@icomp.ufam.edu.br

Abstract

In programming courses, students may use various strategies to solve the same problem. Understanding these strategies can be important for teachers and instructors to assess student progress and provide targeted feedback. Traditional clustering methods have been widely used to group similar programming solutions. However, these techniques often rely on syntactical similarities that do not always reflect the strategies for solving a problem. This research uses clustering algorithms and Large Language Models (LLMs) to identify programming strategies. We conducted the experiments using a dataset of correct student solutions collected from ten Algorithms and Data Structures classes at Federal University of Amazonas. Although the Mean Shift and Affinity Propagation clustering algorithms provided us with visually well-separated clusters, quantitative results showed that the algorithms were not accurate in grouping strategies. In contrast, LLMs demonstrated a better ability to identify strategies aligned with human labels. The results suggest that LLMs can be valuable tools to assist programming instructors in analyzing student solutions.
Keywords: *Programming strategies; LLMs; Clustering; Programming learning*

Cite as: Melo, R., Souza, T., Pires, F., Oliveira, E., Carvalho, L., Pessoa, M. & Fernandes, D. (2025). Exploring the Use of Clustering Algorithms and LLMs to Identify Programming Strategies. Revista Brasileira de Informática na Educação, vol. 34, pp. 59–82. https://doi.org/10.5753/rbie.2026.6550.

1 Introduction

In programming courses, a single exercise may have multiple possible solutions. The differences between students' code can be minor, such as variable naming, or more substantial, such as using different programming strategies to solve the problem (Huang et al., 2013). For example, in a task involving sorting a list of numbers, one student might use bubblesort, while another opts for mergesort (Knuth, 1998). Understanding the strategies students employ throughout a programming course can help instructors determine whether the concepts taught are being applied in assignments and exams (Koivisto and Hellas, 2022).

From a pedagogical perspective, evaluating only whether a solution is correct or not provides a limited view of the student's learning process. Functionally correct code may conceal inefficient strategies, fragile solutions, or partial understandings of the underlying concepts (Koivisto and Hellas, 2022). In contrast, identifying problem-solving strategies allows instructors to understand how students structure their algorithmic reasoning, which concepts they have mastered, and which difficulties persist, even when the final result is correct. This type of analysis enables more precise pedagogical interventions and more meaningful feedback (Glassman et al., 2015).

However, manually evaluating code can be demanding for instructors, especially considering the large number of students per class and the high volume of exercises typically assigned in programming courses. In response to this challenge, several studies in the literature have explored ways to support instructors in this process (Effenberger and Pelánek, 2021; Glassman et al., 2015), for example, by proposing methods to group similar solutions using clustering algorithms (Melo et al., 2024). Despite these advances, most of these methods rely on syntactic similarity, which does not always accurately reflect the programming strategies adopted by students (Melo et al., 2025).

In this regard, some studies indicate that syntactic similarity captures only superficial differences, such as variable naming, instruction order, or writing style, without necessarily reflecting the algorithmic essence of the solution. Huang et al. (2013), for example, observed that in thousands of submissions from a massive online course, the enormous syntactic diversity corresponded to a relatively small number of distinct approaches, highlighting that syntax does not accurately convey the strategy employed. Similarly, Glassman et al. (2015), through the OverCode tool, reinforced the need to go beyond textual comparison, resorting to transformations and dynamic analyses to identify solutions that, although visually distinct, implemented the same underlying logic.

More recently, Paiva et al. (2024) reinforced the limitations of purely syntactic approaches and advanced by proposing the use of semantic representations of code, capable of identifying similar algorithmic strategies even when expressed in different ways. These methods seek to capture deeper aspects of program behavior, approaching the notion of "problem-solving strategy" students adopt.

With the emergence of Large Language Models (LLMs), many research opportunities involving natural language tasks such as text analysis, summarization, and translation have arisen. In computing education, it is no different; some studies have explored using LLMs to support programming courses, acting as assistants for students (Mehta et al., 2023).

In this context, LLMs have proven promising as tools for directly supporting students and as resources capable of assisting teachers in code evaluation (Piscitelli et al., 2025). Leinonen et al. (2023) emphasize that these models can abstract problem-solving patterns and identify programming strategies in a way that is closer to human interpretation, capturing the underlying logic without relying exclusively on syntactic or semantic representations, as occurs in clustering methods. Furthermore, recent studies demonstrate the feasibility of employing LLMs to generate automatic and varied explanations about code fragments, expanding the possibilities for pedagogical analysis and providing teachers with additional support to understand the different approaches adopted by students (MacNeil et al., 2022).

Given this context, this article investigates the use of clustering algorithms to group programming strategies adopted by students when solving exercises, as well as the use of Large Language Models (LLMs) to identify and label such strategies. The main objective is to analyze to what extent these approaches are able to organize and group solutions in a manner consistent with the strategies actually implemented. Within this scope, the following research questions are formulated:

RQ1: To what extent can clustering algorithms group programming solutions according to the strategies used by students?

RQ2: To what extent can Large Language Models (LLMs) identify and describe programming strategies?

To address the research questions, two distinct experiments were conducted, one with clustering algorithms and one with LLMs, based on a dataset of codes submitted by students from ten classes of the Algorithms and Data Structures course at the Federal University of Amazonas, collected through the CodeBench online judge[1] (Galvão et al., 2016). The results point to limitations in traditional clustering methods, However, they show that LLMs hold potential for identifying and describing programming strategies, as they do not rely solely on syntactic aspects of the code in the analysis. Thus, the main contributions of this work are: i) the proposal of a method to cluster student code according to the programming strategies adopted; and ii) a new research avenue in computing education, through the use of LLMs to label code clusters.

We organize this paper as follows: Section 2 presents the theoretical foundation and related work, Section 3 describes the proposed method, Section 4 discusses the experimental results, Section 5 presents the discussion, and Section 6 outlines the conclusions.

2 Background and Related Work

This section presents recent studies on using clustering algorithms to group programming exercise solutions and using LLMs in the context of computing education.

[1]https://cb.icomp.ufam.edu.br

2.1 Clustering of programming solutions

Clustering is an unsupervised learning technique, meaning that training does not require predefined labels or categories. It is an approach focused on grouping similar data into clusters (Xu and Tian, 2015), in which instances share similar characteristics (Jain and Dubes, 1988). Clustering algorithms can be organized into different categories, such as partition-based methods (e.g., k-means (MacQueen et al., 1967)), hierarchical methods, density-based approaches (e.g., DBSCAN (Ester et al., 1996)), and graph- or model-based methods (Xu and Tian, 2015). While partition-based methods require prior specification of the number of groups, hierarchical and density-based approaches allow the identification of grouping structures without the need to predefine the number of clusters. In the context of computing education, this characteristic is particularly relevant, as the number of programming strategies adopted by students is not known in advance.

Some studies in the literature have applied clustering algorithms to group students' programming solutions. For example, Gao et al. (2019) used this type of approach across different sets of exercises. In one of the exercises analyzed, the goal was to calculate the strength of a password based on its length and character diversity. The algorithm identified six distinct groups, including: (i) the use of a single `for` statement combined with conditionals to check different characters; (ii) the use of helper functions; (iii) the excessive use of logical operators `&&`; and (iv) the redundant use of `for` statements for the same task. The two remaining groups were not detailed. The authors concluded that, in some exercises, the clusters can provide relevant insights into the different types of solutions adopted by students; however, they highlighted the need to improve cluster interpretability as a limitation. The main goal of the study was to identify similar behaviors among students when solving programming problems, in line with other research in the area, such as Glassman et al. (2015), Luo and Zeng (2016), Fu et al. (2021), and Effenberger and Pelánek (2021).

Silva et al. (2023) applied a hierarchical clustering algorithm to group students with similar programming styles. The authors identified four distinct clusters of students, based on the characteristics of the code they developed: (i) students who wrote concise functions with few lines of code; (ii) students who produced lengthy functions with a large number of lines; (iii) students whose code contained fewer functions than expected for the task; and (iv) students who achieved results far below expectations.

The use of clustering algorithms in the context of computing education has also been employed to identify common errors among students (Combéfis and Schils, 2016; Emerson et al., 2020; Head et al., 2017; Kawabayashi et al., 2021), generate personalized feedback (Beh et al., 2016; Joyner et al., 2019; Koivisto and Hellas, 2022; Rahman et al., 2021), support pedagogical decision-making (Rahman et al., 2022; Silva and Silla, 2020; Silva et al., 2023), enhance instructors' review capacity (Barbosa et al., 2018; Rosales-Castro et al., 2016), and identify disengaged students (Lokkila et al., 2022).

Although other studies in the literature have addressed the use of clustering to identify similar programming styles, as can be seen in the works of Gao et al. (2019) and Silva et al. (2023), it is clear that the approaches presented focus mainly on syntactic aspects of the codes, such as the use of conditional structures, functions, and even the number of lines used. The focus of the method presented in this journal is identifying programming strategies adopted by students, investigating whether conventional clustering approaches can group solutions based on aspects beyond syntactic similarity. To this end, clustering algorithms that do not require the number of clusters to be defined in advance were selected, since the number of different strategies that may exist to solve a given programming problem is unknown.

2.2 LLMs in computing education

Large Language Models (LLMs) are large-scale generative models, typically based on the Transformer architecture, trained on vast amounts of textual and code data (Brown et al., 2020; Vaswani et al., 2017). These models are capable of performing tasks such as text and program generation, analysis, summarization, and explanation.

With the dissemination of LLMs, several studies in the literature have been exploring the potential of these language models in educational contexts, including the field of computing education (Raihan et al., 2025). For example, Neumann et al. (2024) developed and evaluated an educational chatbot integrated with an LLM designed for undergraduate students in Database and Information Systems courses. As a result, an experiment conducted with 47 students found that the chatbot helped support exam preparation and review key concepts. However, in some cases, the responses provided by the chatbot were vague or incorrect. Similarly, in the study by Lyu et al. (2024), an experiment was conducted with students enrolled in introductory programming courses using an LLM-based assistant called CodeTutor. The authors reported positive results, including higher scores among the experimental group (which used the tool), even in assessments where using CodeTutor was not allowed, indicating that the tool contributed to long-term learning. However, over time, students' trust in the assistant decreased, and they began to prefer help from human tutors. Both studies, Neumann et al. (2024) and Lyu et al. (2024), highlighted the importance of formulating good questions for the LLM to obtain better answers. In other words, the potential of these language models is directly tied to the ability to use them effectively.

Still, within the context of programming education, Miguel et al. (2025) presented a programming expert designed to guide students during problem-solving tasks without providing ready-made code. The expert was built using ChatGPT-4, with the addition of clear instructions to ensure that the LLM would not return complete code solutions. An initial experiment conducted with three participants showed that, when answering a question, the expert focused on explaining concepts and logical flows, demonstrating the potential to support the programming learning process, especially in the current context, where students have become increasingly dependent on LLMs (Lima, 2023).

The work of MacNeil et al. (2022) explored the ability of GPT-3 to automatically generate multiple and varied code explanations, considering that high-quality feedback and explanations are important for learning introductory programming. The authors conducted a systematic analysis of the types of explanations GPT-3 can generate, which include: time complexity analysis, identification of common beginner programming errors, code summarization at different levels of abstraction (execution details, code purpose, related concepts, etc.), bug correction, analogies with real-life situations, and listing of important programming concepts. In addition to serving as feedback for students to understand their solutions and how to improve them, the work of MacNeil et al. (2022) demonstrates that LLMs have the potential to explain code based on aspects beyond syntax. Similarly, Leinonen et al. (2023) compared code explanations created by students with those generated by LLMs. The results showed that LLMs can support students in learning programming by helping them understand code more clearly. Furthermore, the authors noted that LLMs: (i) provide more accurate explanations than those of students; (ii) can aid in code comprehension given the current context, where many automatic code generation tools exist; and (iii) represent a viable alternative for providing explanation examples in large classes.

Considering the ability of LLMs to generate high-quality explanations of programming code, as noted by MacNeil et al. (2022) and Leinonen et al. (2023), and the potential of these models in the context of computing education, the method proposed in our paper is not limited to providing feedback to students. Its main objective is to support teachers in analyzing and understanding the strategies employed by students in solving programming exercises.

3 Proposed method

The method proposed in this paper encompasses two distinct steps to identify and cluster programming strategies students adopt. The first step is based on applying clustering algorithms to student code, aiming to identify clusters of solutions that share similar strategies for a single programming exercise.

The second step explores the use of large-scale language models (LLMs), which have excelled in various natural language processing tasks and are increasingly being applied in the educational context (MacNeil et al., 2022; Mehta et al., 2023; Miguel et al., 2025). In this scenario, LLMs are evaluated for their ability to identify and describe programming strategies, considering different example configurations (zero-shot, one-shot, and few-shot). In Figure 1 we present an overview of the proposed method, including the two steps we detail in the following subsections.

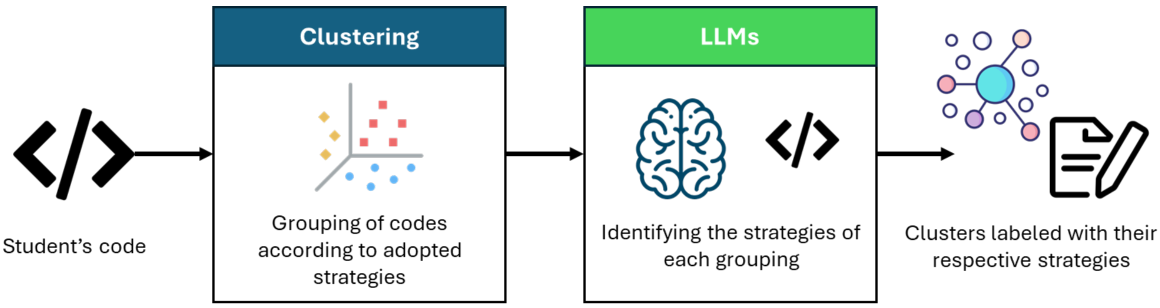


Figure 1: Overview of the proposed method.

3.1 Clustering of similar solutions

Clustering is an unsupervised machine learning technique that allows identifying groups based on existing patterns in data (Xu and Tian, 2015). Some studies in the literature have used clustering algorithms to group programming solutions (Effenberger and Pelánek, 2021; Koivisto and Hellas, 2022; Lokkila et al., 2022; Silva et al., 2023), using students' codes as data.

Therefore, this paper proposes using clustering algorithms to group codes with similar strategies. To this end, we used the correct codes submitted by students as answers to different programming exercises in an online judge (Figure 2, step 1). To convert the codes into a highdimensional numerical representation (embeddings), we used the CodeBERT model [2] (Feng et al., 2020), a transformer network pre-trained on large volumes of source code (Figure 2, step 2). Each solution is transformed into a d-dimensional embedding vector, where d represents the dimension of the latent space generated by the model (Figure 2, step 3). We chose this approach because it considers the student's source code as a whole, without removing comments, for example, and allows us to capture syntactic and semantic characteristics of the code.

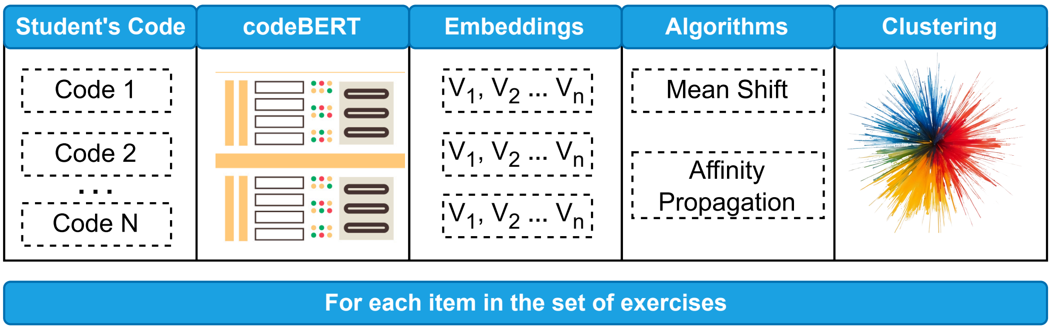


Figure 2: Steps for exercise clustering.

After transforming the codes into embedding vectors, we applied the Mean Shift and Affinity Propagation algorithms (Figure 2, step 4) to generate clusters for each exercise (Figure 2, step 5). Mean Shift is a density-based algorithm that identifies dense regions of points in vector space by iteratively shifting each point toward the local maximum of the estimated density. This shift allows us to detect clusters of arbitrary shapes without the need to previously define the number of groups (Comaniciu and Meer, 2002), which is particularly relevant in a context where the number of distinct strategies in a programming exercise is unknown and may vary depending on the type of exercise or the number of solutions available for analysis.

Affinity Propagation clusters the data through a process of "messages" exchanged between pairs of points, automatically determining which examples serve as representatives of each cluster (Frey and Dueck, 2007). This algorithm suits the problem because it can capture similarity relationships between embedding vectors without relying on an initialization with a fixed number of clusters.

Thus, these algorithms were chosen because they are both capable of handling situations in which the number of strategies (i.e., the number of clusters) is unknown in advance, and they also allow for the exploration of different perspectives: while Mean Shift emphasizes the notion of density, Affinity Propagation relies on similarity measures.

To evaluate the resulting clusters, we performed qualitative analyses, including visual analysis with dimensionality reduction techniques such as UMAP (Uniform Manifold Approximation and Projection) and PCA (Principal Component Analysis) to visualize the clusters in two dimensions, allowing us to see whether there is clear separation between clusters and whether there is significant overlap.

Furthermore, we conducted a quantitative evaluation of the clusters using the Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI) metrics, which compare the generated

[2]Available at: https://github.com/microsoft/CodeBERT

clusters with previously defined labels (ground truth), thus providing an objective indicator of the quality of the clusters in terms of identifying similar programming strategies.

3.2 Using LLMs to identify programming strategies

In recent years, LLMs have been applied to various natural language processing tasks, including translation, text summarization and classification, semantic analysis in different contexts, and text-to-speech. More recently, these models have also gained prominence in computer science education, being used, for example, to provide feedback on programming assignments (Mehta et al., 2023), generate practical examples in introductory courses (Jury et al., 2024), and support teachers in analyzing student solutions.

In this paper, we explore LLMs such as ChatGPT and Gemini to verify their ability to identify and describe programming strategies adopted by students. We performed this analysis considering three distinct usage configurations: zero-shot, one-shot, and few-shot (Brown et al., 2020). These configurations vary depending on the number of examples presented to the model before the assignment:

- Zero-shot: the model receives only the instruction, without additional examples;
- One-shot: the model receives a single clustering example with its corresponding strategy;
- Few-shot: the model receives two or more examples, allowing the model to recognize more consistent patterns.

The Zero, One, and Few-shot configurations are relevant because, in the context of strategy identification, the number of examples provided can directly influence the responses' accuracy, consistency, and generalizability. To operationalize this step, we used the n8n platform[3], a workflow automation tool that allows us to execute and record the prompts sent to the models systematically. Furthermore, we used pre-labeled code sets to construct the prompts (necessary in the One-shot and Few-shot scenarios) and evaluate the responses.

The evaluation follows two main criteria: i) correctness: it verifies whether the description provided by the LLM aligns with the expected strategies and previously defined labels; and ii) description quality: assesses whether the responses are simple, direct, and focused on characterizing the strategies, facilitating interpretation by teachers. Thus, using LLMs aims to automate strategy identification, offering a more accessible and interpretable view of how students solve programming exercises. Figure 3 presents the steps for identifying programming strategies using LLMs, where there is a set of code clusters that we use to prepare the prompts; after that, the prompts are processed by the LLMs, which return descriptions of the strategies contained in the clusters.

[3]Available at: https://n8n.io/

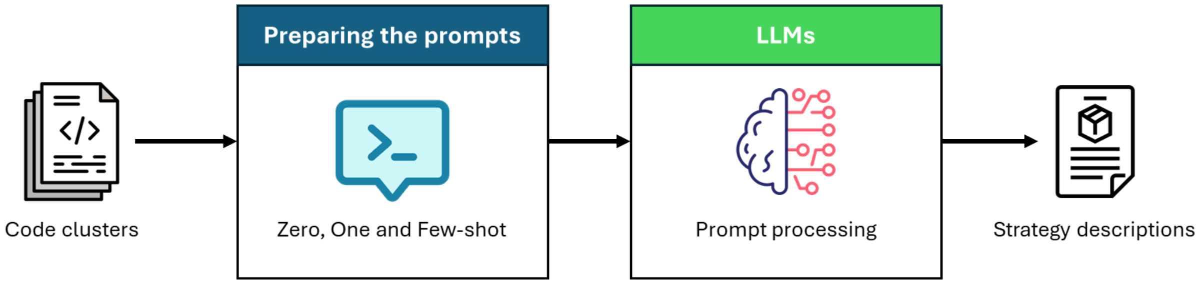


Figure 3: Steps for identifying strategies with LLMs.

4 Experimental results

In this section, we present the results of the experiments to evaluate the proposed method. We describe the dataset used, the student code clustering experiment, and the experiment with LLMs to identify programming strategies.

4.1 Dataset

The data we used to evaluate the proposed method were collected through the CodeBench online judge, from Algorithms and Data Structures (AED) courses at the Federal University of Amazonas, offered in the Computer Science, Computer Engineering, and Software Engineering programs between 2021 and 2023. The AED course is part of the second semester of the course curriculum. It covers content such as elementary data structures, operators, functions, conditional and repetition structures, local and global parameters, recursion, vectors, matrices, records, lists, queues, and stacks.

The online judge can be used for any programming course using C, C++, Java, Python, Haskell, Lua, Bash (with AWK and SED), Assembly, and SQL. For each exercise provided to students, the instructor must develop the statement and a set of test cases. The test cases will be used to evaluate the correctness of the solutions submitted by the students.

The database contains code collected from ten different classes. The classes had an average of 30 students, and each student was required to solve approximately ten sets of exercises, each with approximately twelve questions. Typically, a set of exercises is associated with a curriculum, with varying difficulty levels.

The codes for the ten selected classes were in the Python and C programming languages, typically used in the Algorithms and Data Structures course at UFAM.

To conduct the experiments, we selected only the students' correct solutions. That is, we removed from the dataset all code submitted by students to the online judge that did not meet the requirements of the exercises. In total, the set of correct solutions contains over 12,000 code files.

We present the database structure in Figure 4. Each class consists of different exercise lists, each containing the codes of different students.

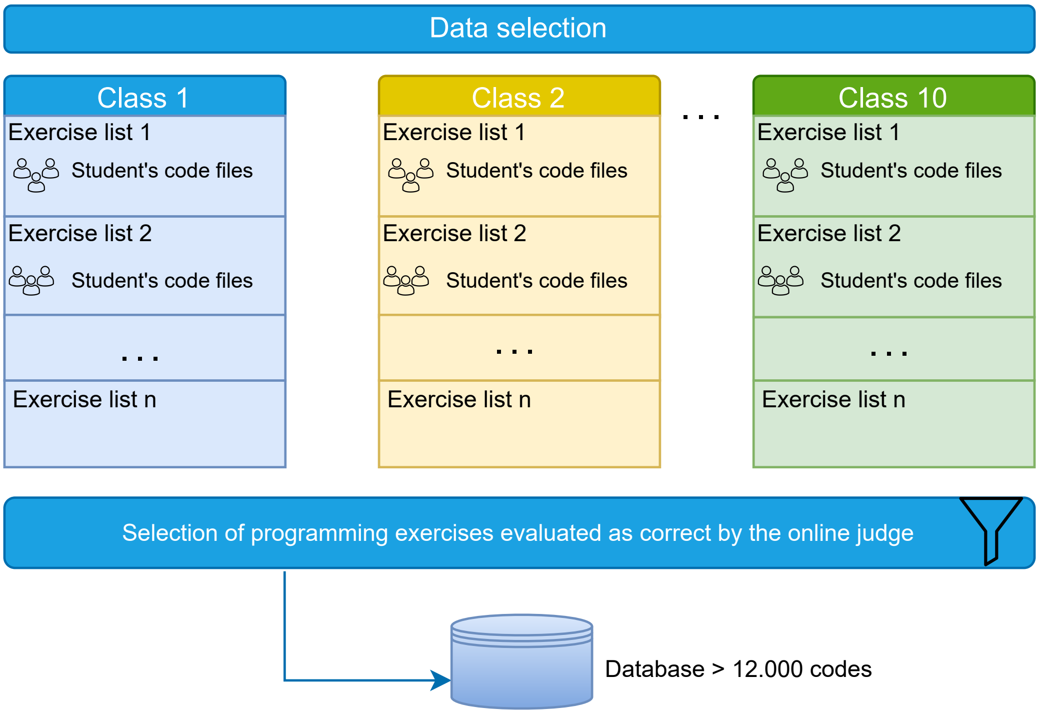


Figure 4: Database description.

4.1.1 *Set of exercises*

Considering the diversity of exercises available in the database, we randomly selected only six programming exercises to evaluate the proposed method. Since the focus of the experiments is: i) the identification of groups of similar strategies and ii) the description of these strategies, an important step in the evaluation consisted of the manual analysis of a subset of solutions. For this purpose, six code submissions were selected for each exercise, totaling 36 manually analyzed codes. These codes were grouped according to similar strategies and used as the basis for defining the reference labels and strategy descriptions[4].

We, the authors, performed this labeling work manually. One author, a professor with experience teaching programming in higher education, was responsible for reviewing the defined strategies. We present a summary of the exercise statements and the identified strategies in Table 1. Therefore, we used this set of exercises to evaluate the proposed method, as described below.

4.2 Student code clusters

Several strategies for solving a problem may exist within a programming class, and it is impossible to determine the exact number in advance. Therefore, to group similar solution strategies, we explore clustering algorithms such as Affinity Propagation and Mean Shift, as these algorithms do not require pre-determining the number of clusters. To use these algorithms, we first apply a technique to transform the codes into embedding vectors. We use the CodeBERT model and tokenizer, which Microsoft provides, to do this.

[4]Available at: https://github.com/rafaelamwlo/exercisesdata.git

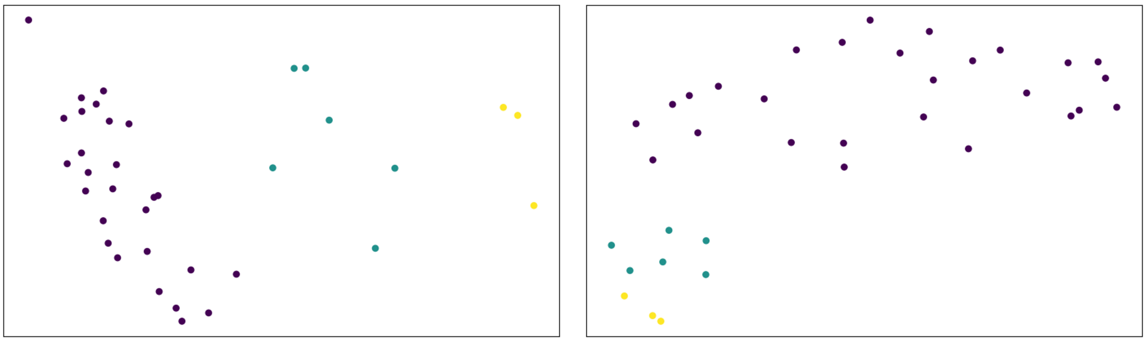
Table 1: Summary of exercises and strategies.

Example	Exercise summary	Strategies
Exercise 1	Merge two linked lists by alternating their elements, empty and free the original lists, and preserve the order of any remaining elements in the new list.	**Cluster 1:** Interleaving is done iteratively with the explicit construction of a new list. **Cluster 2:** Recursive interleaving without creating a new list; only prints the data, which does not fully meet the requirements. **Cluster 3:** Recursive interleaving with the creation and population of a new linked list. **Cluster 4:** Interleaves data into an auxiliary list in reverse order, then reverses it again to maintain the correct sequence.
Exercise 2	Check if sequences of parentheses, brackets, and braces are well-formed using dynamic stacks, printing "YES" or "NO" for each input until "###" is encountered.	**Cluster 1:** Stack implementations that push and pop brackets using ASCII value difference calculations. **Cluster 2:** Stack implementations that push and pop brackets through direct symbol comparisons.
Exercise 3	Simulate a card game using a doubly linked list, repeatedly discarding the top card and moving the next one to the bottom until only one card remains, printing the discarded cards and the last remaining card for each input value N.	**Cluster 1:** Uses a circular doubly linked list to simulate the continuous rotation of the cards. **Cluster 2:** Uses a doubly linked list as a queue, removing the top card and inserting the next at the end. **Cluster 3:** Reverses the list direction: inserts at the beginning and removes from the end, treating the top card as the last element. **Cluster 4:** Uses a singly linked list with insertions at the end and removals from the beginning, but does not meet the requirement of using a doubly linked list.
Exercise 4	Build a generic n-ary tree from (child, parent) pairs and print it recursively in the following order: node, then its children from left to right.	**Cluster 1:** N-ary tree with a search that prioritizes siblings before children. **Cluster 2:** N-ary tree with children stored in linked lists. **Cluster 3:** N-ary tree with a search that prioritizes children before siblings.
Exercise 5	Sort the Olympic medal table using Selection Sort by prioritizing countries with more gold medals, then silver, then bronze, and finally alphabetically in case of ties, and print the sorted list with each country's medal counts.	**Cluster 1:** Uses a doubly linked list with manual node swapping based on multiple criteria (gold, silver, bronze, name). **Cluster 2:** Uses a singly linked list and swaps node contents using Selection Sort according to the tie-breaking criteria. **Cluster 3:** Uses the standard library's qsort function instead of implementing Selection Sort, which does not meet the requirements. **Cluster 4:** Uses an array of pointers to structs and an auxiliary function to handle tie-breaking after comparing gold medals. **Cluster 5:** Uses an array of structs and directly implements the tie-breaking logic within the Selection Sort.
Exercise 6	Build binary search trees from sequences of keys, apply a single right rotation (RR) at the root of each tree, and print the resulting pre-order and post-order traversals.	**Cluster 1:** Represents tree nodes using dictionaries with the keys 'value', 'left', and 'right'. **Cluster 2:** Uses classes with attributes to represent the nodes, which does not fully meet the requirement, as the prompt specifies the use of dictionaries.

Next, we applied the algorithms using the scikit-learn library, which has ready-made implementations of clustering algorithms, where we only need to define the parameters and similarity metrics. We present in Figure 5 the visual results of the clustering with the Mean Shift algorithm and one of the sets of codes referring to one of the exercises available in the database. In Figure 5(a), we used the PCA dimensionality reduction technique, and in Figure 5(b), we used the UMAP technique. It is possible to see that there was good separation of the groups since there was no apparent overlap; however, the visual results do not mean that the codes identified in each cluster represent similar strategies.

Therefore, we next performed a quantitative evaluation using ground truth; that is, we used a small manually labeled and clustered dataset to verify whether the algorithm returned something similar to what was expected. To do so, we randomly selected six exercises from the database, and from each exercise, we selected a random set of six codes from different students. We present the summary of the exercises in Table 1, along with the strategies identified for each. It is worth mentioning that a programming teacher validated the descriptions of the strategies.

Subsequently, we use the Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI) metrics to compare the results obtained by the algorithms with the labeled set. In Table 2, we present the results of applying the Mean Shift and Affinity Propagation algorithms to six



(a) Plot results with PCA. (b) Plot results with UMAP.

Figure 5: Visual results of clusters with Mean Shift.

different exercises involving linked lists, stacks, AVL trees, and sorting algorithms. The ARI metric ranges from -1 (completely random) to 1 (clustering identical to the expected one). NMI, in turn, ranges from 0 (no relationship) to 1 (perfect correspondence).

Table 2: Ground truth results with Mean Shift.

Example	**Mean Shift**		**Affinity Propagation**	
	ARI	**NMI**	**ARI**	**NMI**
Exercise 1	0.143	0.506	0.062	0.579
Exercise 2	**0.348**	**0.404**	-0.132	0.369
Exercise 3	-0.071	0.247	-0.098	0.760
Exercise 4	-0.250	0.115	-0.176	0.339
Exercise 5	-0.129	**0.524**	-0.111	0.722
Exercise 6	-0.174	0.139	0.151	0.579

Among the results obtained in Exercise 2, the ARI metric of 0.348 indicates a moderate agreement between the groupings generated by the method and those generated manually. For the other exercises, we noticed negative values, which indicate that the grouping is worse than random; the algorithm divided the groups very differently when compared to the labeled base.

As for the MNI metric, the 0.404 obtained in Exercise 2 indicates recognizable patterns between the clusters and the labels provided. However, the division is imperfect; there is still a combination of different strategies in the same cluster. The clustering is not entirely random, but it is not perfect either. The same can be said for the other exercises since the maximum MNI value achieved with Mean Shift was 0.524..

Regarding the results obtained with Affinity Propagation, the behavior was similar to that of Mean Shift, especially with the ARI metric, which showed low and negative values. As for the NMI, some values were higher, but the groupings were still far from perfect.

The results show that although the algorithms can capture specific patterns (as evidenced in Exercise 2), most clusters exhibit low agreement or performance below chance, as indicated by the negative ARI values. The NMI values, while higher in some cases, suggest only partial alignment, in which clusters combine distinct strategies. These findings indicate that when applied in isolation, clustering is not sufficient to identify programming accurately.

4.3 Using LLMs to label strategies

This section presents the experiment results with LLMs for identifying programming strategies using Zero-shot, One-shot, and Few-shot. We selected two distinct models for this experiment: ChatGPT-4.1 Mini and Gemini 2.5 Flash.

Before presenting the results of the LLM experiment, it is important to clarify that this experiment is independent of the clustering experiment. Although both experiments rely on the same dataset, the clustering experiment evaluates unsupervised grouping of solutions, whereas the LLM experiment evaluates the ability of language models to identify and describe programming strategies based on a manually labeled reference set. The clusters obtained in the first experiment are not used in the second.

For the execution of the prompts, a workflow was created in n8n, as illustrated in Figure 6. This workflow aimed to facilitate interaction with the LLMs and automate the storage of responses. As shown in Figure 6, the prompt is sent simultaneously to both language models; the responses are then recorded individually, merged, and finally stored in a Google Sheets spreadsheet.

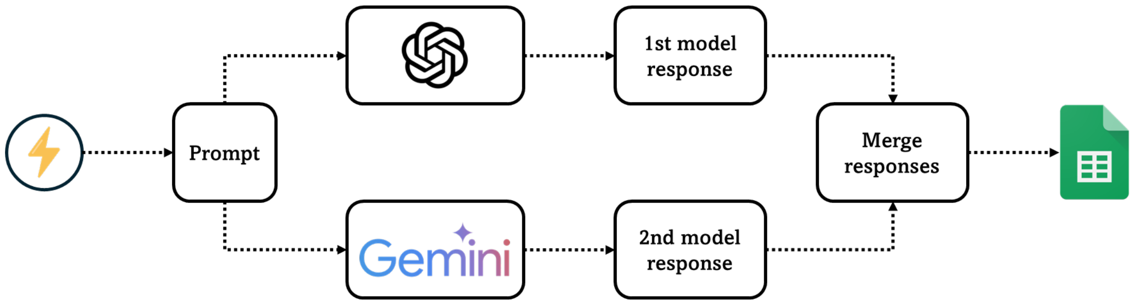


Figure 6: Workflow created in n8n.

The dataset used for this experiment is the same as in the clustering experiment: the set of examples presented in Table 1.

4.3.1 Zero-shot approach

First, we applied the zero-shot approach. No label examples (i.e., example descriptions of programming strategies) are sent to the LLM, only the code clusters that should be labeled, as shown in Table 3.

It is worth noting that this prompt underwent a refinement process until it reached a more appropriate response format. The first attempts, where we did not specify the number of characters per strategy, resulted in long responses, with code-by-code analysis and suggestions for improvements, which was not the experiment's intention.

We tested the LLMs with the same prompt model in the other five exercises, but we needed minor adjustments, such as specifying the number of clusters submitted. These adjustments were necessary because the models now describe the strategy of each code individually, rather than the set of codes. Thus, we added the following to the prompt for the remaining exercises: "Note that I

Table 3: Prompt structure with Zero-shot.

Given the statement:
Implement a program that reads two linked lists, generates, and prints the interleaving of the two lists. At the end of the function, the original lists must be empty and freed. If one list has more elements than the other, the remaining elements are transferred in the same order to the new list.
And the clusters below (for this example, only 4 solution strategies were identified, whose codes were grouped into *clusters* and then provided):
Codes from cluster 0, unlabeled (total of 2 codes) Codes from cluster 1, unlabeled (total of 1 code) Codes from cluster 2, unlabeled (total of 2 codes) Codes from cluster 3, unlabeled (total of 1 code)
Provide a description of at most 120 characters for the problem-solving strategy addressed in each code cluster.

am not asking for a description of each code, but rather of the clusters. In this case, there are only N clusters." For each exercise, we explicitly indicated the number of clusters.

Table 4 presents the results for the exercise cited in the prompt in Table 3 (and corresponds to the first exercise in Table 1), where you can see that both LLMs demonstrated a good ability to return the cluster strategies concisely.

Table 4: Gemini and GPT-4o-mini results with Zero-shot..

Gemini Response	GPT-4.1-mini Response
Here are the descriptions of the strategies for each cluster: **Cluster 0:** Creates a new interleaved list by copying data from the originals; originals freed after use. **Cluster 1:** Interleaves by printing elements and tries to empty the original lists recursively (fails due to pass by value). **Cluster 2:** Recursive interleaving creates a new list by copying elements, with the originals freed. **Cluster 3:** Creates an auxiliary list to interleave in reverse order, then reverses to the final list. Frees originals but leaks new ones.	**Cluster 0:** Interleaves by creating a new list, inserts data by copying; original lists must be freed manually. **Cluster 1:** Interleaves by printing element by element and removing nodes; original lists remain empty after use. **Cluster 2:** Interleaves using recursion and insertion at the end, with dynamic lists and helper functions. **Cluster 3:** Interleaves by reversing with insertions at the beginning, creates an auxiliary list, and then reverses to the correct order.

Both LLMs were able to identify the strategies adopted in the code to solve the linked list problem presented, such as the use of recursion to solve the problem or the creation of auxiliary lists. The strategies are similar to those presented in Table 1, aligning with our expectations. In cluster 1, the models identified that the values were printed, instead of creating a new list, as requested in the question statement.

Based on these results, we decided that the next step in the LLM experiment would be to test whether adding manually labeled examples influences the responses. We then applied the one-shot approach, adding a labeled example to the prompt.

4.3.2 *One-shot approach*

For the One-shot approach, we structured the prompt the same way as the Zero-shot approach. However, we added a labeled code cluster to the prompt before asking the LLM to identify the strategy of the submitted code sets to be labeled, as shown in Table 5.

Table 5: Prompt structure with One-shot.

Given the statement:
Implement a program that reads two linked lists, generates, and prints the interleaving of the two lists. At the end of the function, the original lists must be empty and freed. If one list has more elements than the other, the remaining elements are transferred in the same order to the new list.
And the cluster below with its respective description:
Cluster 0 – Description: The codes iteratively traverse the lists and explicitly insert the elements alternately into a new list. Codes from cluster 0 (total of 2 codes)
Provide a description of at most 120 characters for the code clusters below.
Codes from cluster 1, unlabeled (total of 1 code) Codes from cluster 2, unlabeled (total of 2 codes) Codes from cluster 3, unlabeled (total of 1 code)

Table 6: Results with One-shot approach.

Example	LLM	Format	Labeled Correctly
Exercise 1	Gemini Flash 2.5	Agreeable	Y
	GPT 4.1 Mini	Agreeable	Y
Exercise 2	Gemini Flash 2.5	Agreeable	Y
	GPT 4.1 Mini	Agreeable	N
Exercise 3	Gemini Flash 2.5	Agreeable	Y
	GPT 4.1 Mini	Agreeable	N
Exercise 4	Gemini Flash 2.5	Agreeable	Y
	GPT 4.1 Mini	Agreeable	Y
Exercise 5	Gemini Flash 2.5	Agreeable	N
	GPT 4.1 Mini	Agreeable	Y
Exercise 6	Gemini Flash 2.5	Agreeable	Y
	GPT 4.1 Mini	Agreeable	Y

During the execution of the experiment, as in the One-shot approach, it was also necessary to make adjustments to the prompt, since in some cases the models recognized more clusters than requested. As shown in Table 6, the LLMs provided a well-structured response format in all cases, that is, a concise description of the code cluster strategies, limited to 120 characters. In Exercise 3, despite the specification of the number of clusters, GPT added one more cluster to the response, which we did not expect, so we consider that the model did not label correctly.

Regarding the other exercises, in Example 3, GPT focused on the data structure being used rather than on the strategy itself, and for this reason, we considered the response incorrect. In Example 5, where it was necessary to implement the Selection Sort algorithm, Gemini did not indicate in its response that one of the clusters used a built-in language function to solve the problem, unlike GPT. For the remaining exercises, the models provided similar strategies.

Based on the results of the One-shot approach, where in some cases the LLMs did not label as expected or even close to what we expected, we decided to add more examples to the prompt, that is, to apply the Few-shot approach in order to assess the impact of additional examples on the results.

4.4 Few-shot approach

For the Few-shot approach, we used the same prompt as in the One-shot approach, but with the addition of two examples instead of one, as shown in Table 7.

Table 7: Prompt structure with Few-shot.

Given the statement:
Implement a program that reads two linked lists, generates, and prints the interleaving of the two lists. At the end of the function, the original lists must be empty and freed. If one list has more elements than the other, the remaining elements are transferred in the same order to the new list.
And the cluster below with its respective description:
Cluster 0 – Description: The codes iteratively traverse the lists and explicitly insert the elements alternately into a new list. Codes from cluster 0 (total of 2 codes)
Cluster 1 – Description: Performs the interleaving recursively, without creating a new explicit list: directly prints the data during the recursive call. Codes from cluster 1 (total of 1 code)
Provide a description of at most 120 characters for the code clusters below. Codes from cluster 2, unlabeled (total of 2 codes) Codes from cluster 3, unlabeled (total of 1 code)

During the experiment, we needed to adjust the prompt, indicating the number of clusters being sent for strategy identification, as in the zero-shot and one-shot approaches. We ran the tests with four exercises, as shown in Table 8. We did not use exercises 2 and 6 because they only had two labeled clusters.

Table 8: Results with Few-shot approach.

Example	LLM	Format	Labeled Correctly
Exercício 1	Gemini Flash 2.5	Agreeable	Y
	GPT 4.1 Mini	Agreeable	Y
Exercício 3	Gemini Flash 2.5	Agreeable	Y
	GPT 4.1 Mini	Agreeable	Y
Exercício 4	Gemini Flash 2.5	Agradável	Y
	GPT 4.1 Mini	Agreeable	Y
Exercício 5	Gemini Flash 2.5	Agreeable	Y
	GPT 4.1 Mini	Agreeable	Y

As shown in Table 8, when using the Few-shot approach, the LLMs returned a well-structured format in all cases, as in the other approaches tested. This outcome is related to the constructed prompt, which already specified the desired character limit for the output of each strategy. Although the strategies provided by the models were not identical to what we expected (Table 1), the responses presented similar strategies, sometimes highlighting another aspect of the clusters, such as the use of recursion or the use of specific data structures.

Table 9 presents the percentage of correct identifications achieved by the LLMs across the different approaches. It can be observed that, for both tools, providing examples in the prompt directly impacts performance. While the Zero-shot and One-shot approaches show variations between models, the Few-shot configuration achieves 100% accuracy for both GPT and Gemini. These quantitative results corroborate the qualitative analysis, indicating that the use of explicit examples enables LLMs to identify programming strategies more consistently with human-defined labels.

Table 9: Percentage of correct answers for LLMs by approach.

Approach	GPT (%)	Gemini (%)
Zero-shot	83.3	66.7
One-shot	66.7	83.3
Few-shot	100.0	100.0

Overall, the experiments show that prompt construction and providing examples directly influence the quality of LLM responses. The Zero-shot approach presented limitations, particularly in the models' tendency to describe code by code rather than synthesizing the strategies of the clusters. Adding a single example (One-shot) brought gains in consistency, but misinterpretations still occurred in some exercises. In the Few-shot configuration, the responses were closer to what was expected, with concise descriptions aligned with the strategies, although not identical to the previously labeled ones. These results reinforce that LLMs, when guided with clear examples, can play an important role as a supporting tool for labeling programming strategies.

5 Discussion

The results we obtained in the two experiments offer different perspectives on identifying programming strategies students adopt. We guide the discussion by the two research questions proposed in this work: (i) to what extent clustering algorithms can group solutions according to the strategies used (RQ1), and (ii) to what extent LLMs can identify and describe these strategies (RQ2). Below, we discuss the main findings concerning each question, highlighting limitations, potentialities, and pedagogical implications.

5.1 QP1 - Clustering of programming solutions

The results showed that, despite the good visual separation of clusters using techniques such as PCA and UMAP, the low or negative values of the ARI and NMI metrics suggest that the groupings obtained with the Mean Shift and Affinity Propagation algorithms do not coincide with the actual strategies used by students. These findings may indicate that clustering algorithms based solely on syntactic embeddings (even semantic ones, such as CodeBERT) do not adequately capture the strategy adopted in problem-solving. Thus, it can be concluded that clustering with code embeddings alone is not sufficient to identify programming strategies, making it necessary to apply these techniques in combination with others.

Furthermore, we observed that even when the NMI values were relatively high (above 0.5 in some cases), this did not imply a semantically correct separation of the strategies. This result suggests that the similarity between codes may be more related to syntactic patterns than strategic reasoning.

Another important point observed during the execution of the clustering experiment is that, when analyzing the labeled exercise examples to evaluate the clusters (Table 1), it becomes evident that within a small set of codes in certain exercises, the strategies vary considerably, as in Examples 3, 4, and 5. The variation in student responses may lead us to understand that, within a classroom of fifty students, several different strategies may exist, which may not be fully identified by code clustering.

Furthermore, among the strategies, we found that in some cases the code was considered correct by the online judge (it passed the online judge's test cases), but the code did not fully follow the instructions, as is the case in exercise four, where the problem was supposed to be solved using doubly linked lists, but one of the labeled clusters presented answers with linked lists. Within programming learning, the fact that an answer does not fulfill the instructions is also interesting to explore, as many institutions use online judges for automatic code correction (Barbosa et al., 2023). Test cases alone do not guarantee that students followed the instructions, as there are other ways to solve the problem besides the one requested, as seen in the labeled codes in this research.

Therefore, regarding RQ1, we concluded that the clustering algorithms explored were helpful in an initial exploratory analysis but showed significant limitations when the goal is to identify programming strategies accurately.

5.2 QP2 - LLMs to identify programming strategies

Regarding the second experiment, the use of LLMs demonstrated a greater ability to recognize student strategies compared to traditional clustering methods. Even with the zero-shot approach, the models were able to identify relevant code characteristics, such as the use of object orientation or dictionary structures.

As One-shot and Few-shot approaches were introduced, with prior examples in the prompts, the LLMs showed improved performance in correctly identifying the strategies and in returning the responses in the expected format. The Few-shot approach stood out by providing the best results, suggesting that supplying well-constructed examples in the prompt is essential for the accuracy and consistency of the models' responses.

In our previous study (Melo et al., 2025), the experiments with LLMs had a more exploratory nature, without formal validation of the labels by experts, which limited the reliability of the analyses. Moreover, Gemini presented significant limitations, such as verbose responses and poor alignment with the requested format, which reduced its applicability. In the present study, however, Gemini 2.5 Flash demonstrated superior performance, sometimes achieving more accurate results than GPT-4.1 Mini. This difference may be related to the evolution of the model itself and to the refinement of the experimental design, using better-structured prompts, specification of format, and definition of the number of clusters. Another aspect that contributed to the greater robustness of the results was the use of exercises with labels previously validated by a programming instructor, which reduced biases and provided a more reliable basis for evaluating the LLM responses.

The findings of this study have important implications for programming education. First, they show that traditional clustering methods, although useful for exploratory analyses, are limited when the goal is to understand cognitive and strategic aspects of students' solutions. On the other hand, LLMs are promising tools to support qualitative code analysis, enabling teachers to identify patterns and strategies even in classes with large volumes of data. This possibility opens pathways for more informed pedagogical practices, such as providing feedback targeted to the type of strategy adopted and identifying unconventional approaches, or even identifying approaches that did not follow the teacher's request in the question statement.

The results of the two research questions point to a scenario in which the approaches analyzed can be complementary. While clustering based on code embeddings proved helpful as an initial exploratory tool for organizing large volumes of solutions (RQ1), LLMs demonstrated a greater ability to produce interpretable descriptions of strategies, predominantly when guided with well-constructed examples in the prompts (RQ2). Thus, integrating these approaches may represent a promising path to broaden the understanding of programming strategies used by students, supporting evidence-based pedagogical practices and enabling more targeted feedback in programming education contexts. While clustering can help instructors understand the overall profile of a class, LLMs can support individualized interventions focused on the student.

6 Conclusion

This research investigated the use of clustering algorithms and LLMs to identify programming strategies in students' solutions to different exercises. The results showed that although the clustering algorithms produced satisfactory visual separations, the quantitative metrics revealed that these groupings did not accurately reflect the actual strategies adopted by the students. This finding reinforces the limitation of methods based solely on code embeddings, which capture syntactic and semantic similarities but not necessarily strategic reasoning.

On the other hand, the experiments with LLMs showed promising results. Even with Zero-shot approaches, the models were able to recognize relevant characteristics in the solutions. This performance was even more satisfactory in the One-shot and Few-shot approaches, in which the provision of examples in the prompt helped the LLM identify the strategies applied by the students with greater accuracy and consistency.

Thus, the use of LLMs emerges as a viable alternative to support teachers in the analysis of large volumes of code, enabling a deeper understanding of the strategies adopted in the solutions. In addition, this approach can assist in identifying correct answers that did not follow the problem statement specifications, a relevant aspect in the educational context where online judges are used.

Among the contributions of this work, the following stand out: (i) a method to cluster student code based on programming strategies; (ii) empirical evidence of the potential of LLMs to identify and describe programming strategies; and (iii) opportunities for research in the field of computing education.

The limitations of this study include: (i) the limited set of exercises and manually labeled solutions used for model evaluation; and (ii) the prompts used with the LLMs, which could be further refined through advanced prompt engineering techniques and the use of adaptive contexts. To mitigate these limitations, as future work we intend to expand the number of analyzed exercises and solutions, explore different language models, and investigate the use of LLMs to perform clustering jointly with strategy identification. This may enhance the automatic analysis of programming strategies and contribute to the development of more robust educational tools for teaching and learning programming.

7 Ethical aspects

The information we use in this paper was obtained from our own dataset, composed of files extracted from the CodeBench online judge, from the Federal University of Amazonas, referring to source codes produced by students in programming courses between August 11, 2021, and November 9, 2023. We emphasize that all data was anonymized correctly and did not contain personally identifiable information.

Therefore, there was no need to submit the research to research ethics committees, since no sensitive data were collected and there was no direct contact with the participants. Even so, we ensure that we applied all ethical and confidentiality precautions at all stages of this research, following good academic practices.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Brazil, Finance Code 001. This work was partially supported by Amazonas State Research Support Foundation (FAPEAM) through the POSGRAD project.

References

Barbosa, A. d. A., Costa, E. d. B., & Brito, P. H. (2018). Adaptive clustering of codes for assessment in introductory programming courses. *Intelligent Tutoring Systems: 14th International Conference, ITS 2018, Montreal, QC, Canada, June 11–15, 2018, Proceedings 14*, 13–22. https://doi.org/10.1007/978-3-319-91464-0_2 [GS Search].

Barbosa, A. d. A., de Barros Costa, E., & Brito, P. H. (2023). Juízes online são suficientes ou precisamos de um var? *Simpósio Brasileiro de Educação em Computação (EDUCOMP)*, 386–394. https://doi.org/10.5753/educomp.2023.228224 [GS Search].

Beh, M. Y., Gottipatti, S., LO, D., & Shankararaman, V. (2016). Semi-automated tool for providing effective feedback on programming assignments. https://ink.library.smu.edu.sg/sis_research/3748 [GS Search].

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901. [GS Search].

Comaniciu, D., & Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, *24*(5), 603–619. https://doi.org/10.1109/34.1000236 [GS Search].

Combéfis, S., & Schils, A. (2016). Automatic programming error class identification with code plagiarism-based clustering. *Proceedings of the 2nd International Code Hunt Workshop on Educational Software Engineering*, 1–6. https://doi.org/10.1145/2993270.2993271 [GS Search].

Effenberger, T., & Pelánek, R. (2021). Interpretable clustering of students' solutions in introductory programming. *International Conference on Artificial Intelligence in Education*, 101–112. https://doi.org/10.1007/978-3-030-78292-4_9 [GS Search].

Emerson, A., Smith, A., Rodriguez, F. J., Wiebe, E. N., Mott, B. W., Boyer, K. E., & Lester, J. C. (2020). Cluster-based analysis of novice coding misconceptions in block-based programming. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 825–831. https://doi.org/10.1145/3328778.3366924 [GS Search].

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. *kdd*, *96*(34), 226–231. [GS Search].

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., et al. (2020). Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*. https://doi.org/10.48550/arXiv.2002.08155 [GS Search].

Frey, B. J., & Dueck, D. (2007). Clustering by passing messages between data points. *science*, *315*(5814), 972–976. https://doi.org/10.1126/science.1136800 [GS Search].

Fu, Y., Osei-Owusu, J., Astorga, A., Zhao, Z. N., Zhang, W., & Xie, T. (2021). Pacon: A symbolic analysis approach for tactic-oriented clustering of programming submissions. *Proceedings of the 2021 ACM SIGPLAN International Symposium on SPLASH-E*, 32–42. https://doi.org/10.1145/3484272.3484963 [GS Search].

Galvão, L., Fernandes, D., & Gadelha, B. (2016). Juiz online como ferramenta de apoio a uma metodologia de ensino híbrido em programação. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, *27*(1), 140. https://doi.org/10.5753/cbie.sbie.2016.140 [GS Search].

Gao, L., Wan, B., Fang, C., Li, Y., & Chen, C. (2019). Automatic clustering of different solutions to programming assignments in computing education. *Proceedings of the ACM Conference on Global Computing Education*, 164–170. https://doi.org/10.1145/3300115.3309515 [GS Search].

Glassman, E. L., Scott, J., Singh, R., Guo, P. J., & 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. https://doi.org/10.1145/2699751 [GS Search].

Head, A., Glassman, E., Soares, G., Suzuki, R., Figueredo, L., D'Antoni, L., & Hartmann, B. (2017). Writing reusable code feedback at scale with mixed-initiative program synthesis. *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale*, 89–98. https://doi.org/10.1145/3051457.3051467 [GS Search].

Huang, J., Piech, C., Nguyen, A., & Guibas, L. (2013). Syntactic and functional variability of a million code submissions in a machine learning mooc. *AIED 2013 Workshops Proceedings Volume*, *25*. [GS Search].

Jain, A. K., & Dubes, R. C. (1988). *Algorithms for clustering data*. Prentice-Hall, Inc. [GS Search].

Joyner, D., Arrison, R., Ruksana, M., Salguero, E., Wang, Z., Wellington, B., & Yin, K. (2019). From clusters to content: Using code clustering for course improvement. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 780–786. https://doi.org/10.1145/3287324.3287459 [GS Search].

Jury, B., Lorusso, A., Leinonen, J., Denny, P., & Luxton-Reilly, A. (2024). Evaluating llm-generated worked examples in an introductory programming course. *Proceedings of the 26th Australasian computing education conference*, 77–86. https://doi.org/10.1145/3636243.3636252 [GS Search].

Kawabayashi, S., Rahman, M. M., & Watanobe, Y. (2021). A model for identifying frequent errors in incorrect solutions. *2021 10th International Conference on Educational and Information Technology (ICEIT)*, 258–263. https://doi.org/10.1109/ICEIT51700.2021.9375615 [GS Search].

Knuth, D. E. (1998). *The art of computer programming: Sorting and searching, volume 3*. Addison-Wesley Professional. [GS Search].

Koivisto, T., & Hellas, A. (2022). Evaluating codeclusters for effectively providing feedback on code submissions, 1–9. https://doi.org/10.1109/FIE56618.2022.9962751 [GS Search].

Leinonen, J., Denny, P., MacNeil, S., Sarsa, S., Bernstein, S., Kim, J., Tran, A., & Hellas, A. (2023). Comparing code explanations created by students and large language models. *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, 124–130. https://doi.org/10.1145/3587102.3588785 [GS Search].

Lima, J. (2023). Como o chatgpt afeta a educação e o desenvolvimento universitário. *The Trends Hub*, (3). https://doi.org/10.34630/tth.vi3.5020 [GS Search].

Lokkila, E., Christopoulos, A., & Laakso, M.-J. (2022). A clustering method to detect disengaged students from their code submission history. *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 1*, 228–234. https://doi.org/10.1145/3502718.3524754 [GS Search].

Luo, L., & Zeng, Q. (2016). Solminer: Mining distinct solutions in programs. *Proceedings of the 38th International Conference on Software Engineering Companion*, 481–490. https://doi.org/10.1145/2889160.2889202 [GS Search].

Lyu, W., Wang, Y., Chung, T., Sun, Y., & Zhang, Y. (2024). Evaluating the effectiveness of llms in introductory computer science education: A semester-long field study. *Proceedings of the Eleventh ACM Conference on Learning@ Scale*, 63–74. https://doi.org/10.1145/3657604.3662036 [GS Search].

MacNeil, S., Tran, A., Mogil, D., Bernstein, S., Ross, E., & Huang, Z. (2022). Generating diverse code explanations using the gpt-3 large language model. *Proceedings of the 2022 ACM conference on international computing education research-volume 2*, 37–39. https://doi.org/10.1145/3501709.3544280 [GS Search].

MacQueen, J., et al. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, *1*(14), 281–297. https://cir.nii.ac.jp/crid/1571135649659368064 [GS Search].

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*. https://doi.org/10.48550/arXiv.2312.07343 [GS Search].

Melo, R., Pessoa, M., & Fernandes, D. (2024). Clusterização de soluções de exercícios de programação: Um mapeamento sistemático da literatura. *Simpósio Brasileiro de Informática na Educação (SBIE)*, 1715–1729. https://doi.org/10.5753/sbie.2024.242403 [GS Search].

Melo, R., Souza, T., Oliveira, E., Galvao, L., Pessoa, M., & Fernandes, D. (2025). Explorando o uso de llms para rotular estratégias de programação. *Simpósio Brasileiro de Educação em Computação (EDUCOMP)*, 178–190. https://doi.org/10.5753/educomp.2025.5335 [GS Search].

Miguel, J., Martins, W., Benarrós, Í., & Duarte, J. C. (2025). Especialista em algoritmos para apoio interativo na aprendizagem de programação utilizando chatgpt. *Simpósio Brasileiro de Educação em Computação (EDUCOMP)*, 204–215. https://doi.org/10.5753/educomp.2025.5378 [GS Search].

Neumann, A. T., Yin, Y., Sowe, S., Decker, S., & Jarke, M. (2024). An llm-driven chatbot in higher education for databases and information systems. *IEEE Transactions on Education*. https://doi.org/10.1109/TE.2024.3467912 [GS Search].

Paiva, J. C., Leal, J. P., & Figueira, Á. (2024). Clustering source code from automated assessment of programming assignments. *International Journal of Data Science and Analytics*, 1–12. https://doi.org/10.1007/s41060-024-00554-5 [GS Search].

Piscitelli, A., De Rosa, M., Fuccella, V., Costagliola, G., et al. (2025). Large language models for student code evaluation: Insights and accuracy. *Proceedings of the 17th International Conference on Computer Supported Education-(Volume 2)*, 534–544. https://doi.org/10.5220/0013287500003932 [GS Search].

Rahman, M. M., Watanobe, Y., Matsumoto, T., Kiran, R. U., & Nakamura, K. (2022). Educational data mining to support programming learning using problem-solving data. *IEEE Access*, *10*, 26186–26202. https://doi.org/10.1109/ACCESS.2022.3157288 [GS Search].

Rahman, M. M., Watanobe, Y., Rage, U. K., & Nakamura, K. (2021). A novel rule-based online judge recommender system to promote computer programming education. *Advances and Trends in Artificial Intelligence. From Theory to Practice: 34th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2021, Kuala Lumpur, Malaysia, July 26–29, 2021, Proceedings, Part II 34*, 15–27. https://doi.org/10.1007/978-3-030-79463-7_2 [GS Search].

Raihan, N., Siddiq, M. L., Santos, J. C., & Zampieri, M. (2025). Large language models in computer science education: A systematic literature review. *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, 938–944. https://doi.org/10.1145/3641554.3701863 [GS Search].

Rosales-Castro, L. F., Chaparro-Gutiérrez, L. A., Cruz-Salinas, A. F., Restrepo-Calle, F., Camargo, J., & González, F. A. (2016). An interactive tool to support student assessment in programming assignments. *Advances in Artificial Intelligence-IBERAMIA 2016: 15th Ibero-American Conference on AI, San José, Costa Rica, November 23-25, 2016, Proceedings 15*, 404–414. https://doi.org/10.1007/978-3-319-47955-2_33 [GS Search].

Silva, D. B., Carvalho, D. R., & Silla, C. N. (2023). A clustering-based computational model to group students with similar programming skills from automatic source code analysis using novel features. *IEEE Transactions on Learning Technologies*. https://doi.org/10.1109/TLT.2023.3273926 [GS Search].

Silva, D. B., & Silla, C. N. (2020). Evaluation of students programming skills on a computer programming course with a hierarchical clustering algorithm. *2020 IEEE Frontiers in Ed-*

ucation Conference (FIE), 1–9. https://doi.org/10.1109/FIE44824.2020.9274130 [GS Search].

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*. [GS Search].

Xu, D., & Tian, Y. (2015). A comprehensive survey of clustering algorithms. *Annals of data science*, *2*(2), 165–193. https://doi.org/10.1007/s40745-015-0040-1 [GS Search].