A Generative Artificial Intelligence Tool to Correct Programming Exercises

Filipe Portela
Algoritmi Centre, University of Minho, Guimarães, Portugal

Abstract

This paper presents an innovative strategy for assessing programming exercises in higher education, leveraging generative artificial intelligence (GAI) to support automated grading while ensuring transparency, fairness, and pedagogical relevance. The proposed approach is framed within the TechTeach paradigm and integrates multiple tools – HackerRank for code development, Google Forms and Sheets for submission and prompt generation, and the ChatGPT API for intelligent evaluation. The correction process is personalised using student-specific variables (e.g., student ID, birth date, performance in group work), which are dynamically embedded into the statement and prompt. The GAI algorithm evaluates the code and performs authorship verification using peer-assessed effort data, enabling the detection of potential plagiarism or misuse of AI tools. A case study was conducted in the 2023/2024 edition of the Web Programming course at the University of Minho, which involved 118 students. Results indicate that the method produced consistent and meaningful grades, reflecting a balanced perception of difficulty from students. The system also includes a gamification mechanism (Grade Rescue) for managing contested cases. The achieved findings (>90% of students approved the exercise model) support the viability of GAI-based evaluation as a scalable and effective solution for programming education, while maintaining academic integrity and enhancing the student experience.

2012 ACM Subject Classification Information systems → Information retrieval

Keywords and phrases TechTeach, Information Systems, Higher Education, Generative AI, Code Exercises

Digital Object Identifier 10.4230/OASIcs.ICPEC.2025.7

Funding This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Unit Project Scope UID/00319/Centro ALGORITMI (ALGORITMI/UM).

1 Introduction

The emergence of Generative Artificial Intelligence (GAI) technologies has recently transformed the academic landscape. Tools such as ChatGPT have raised significant concerns among professors, who fear that students might misuse them to complete assignments without truly engaging in the learning process. This is particularly evident in higher education, where these tools proliferate. However, GAI also presents unprecedented opportunities as an educational assistant. It can support instructors by generating exercises, preparing slides, designing interactive content, structuring curricula, and even evaluating assignments.

In this context, the present work arises, framed within the TechTeach paradigm, a new gamified paradigm to engage students in the classroom [11]. This paper presents an innovative algorithm designed to correct programming exercises using GAI. The approach involves the development of a specific enunciation format, a prompt capable of interpreting the exercise statement, and a classification system based on three grading levels:

- **-1**: The student does not demonstrate basic knowledge.
- **0**: The student demonstrates basic but incomplete knowledge.
- **1**: The student demonstrates an expert-level understanding of the subject.

The algorithm was developed and tested over two academic years at the University of Minho, involving more than two hundred students enrolled in the Web Programming course.

This paper is made up of five sections. Section 2 provides the necessary background on Generative Artificial Intelligence and its educational applications. Section 3 describes the materials and methods used to develop the proposed approach. Section 4 presents the GAI-based correction algorithm, detailing its structure and logic. Section 5 reports the case study conducted with students from the University of Minho and the results (section 6) discusses the results. Finally, Section 7 draws the main conclusions and outlines potential future work.

2 Background

This section presents the main topics of the work and some similar works.

2.1 Generative AI in Education

Generative Artificial Intelligence (GAI) has rapidly expanded across multiple domains, with education emerging as a particularly impacted sector. At the core of many GAI tools are Large Language Models (LLMs), a class of deep learning systems trained on extensive textual corpora and capable of producing coherent, context-aware natural language output. These models, such as OpenAI's GPT series, have demonstrated impressive capabilities in a wide range of language understanding and generation tasks, making them increasingly prominent in educational technologies due to their adaptability and scalability [2].

The introduction of powerful LLM-based applications, such as OpenAI's ChatGPT, has created new opportunities and challenges in the teaching and learning landscape. On one hand, GAI tools can support educators by generating instructional materials, summarising content, or responding to complex student queries. On the other hand, their misuse, particularly by students relying on these systems to produce unoriginal or plagiarised work, raises significant ethical, pedagogical, and assessment-related concerns. [7, 14, 1]. Recent research has explored how GAI can become a powerful ally in the classroom, especially when integrated thoughtfully into the curriculum. Its use can foster innovation in assessment, provide personalised feedback, and help teachers scale their efforts to large student groups [8]. Other approaches, such as GAMAI [9] or Socratique [15], help professors create exercises or students learn a topic and receive feedback.

2.2 TechTeach

In 2020, Filipe Portela proposed an innovative teaching paradigm named *TechTeach*, aimed at increasing student engagement through gamification and active learning strategies [11]. This approach integrates digital tools like Kahoot!, project-based learning, and collaborative exercises to create dynamic, student-centred classrooms. Over the years, the model has evolved with new techniques and adaptations being introduced annually, including innovative mechanisms for assessment using real-time quizzes [12].

Inspired by the success of this paradigm, the GAI-based correction strategy presented in this paper incorporates several principles originally introduced in *TechTeach*, particularly those related to engagement and formative feedback through gamified interactions.

2.3 Assistive Coding Correction Tools

In this context, several tools have been developed in recent years, like Gradescope, to support the automatic evaluation of programming exercises. Traditional automatic graders rely heavily on predefined unit tests and static analysis techniques, often failing to understand

creative or alternative correct solutions. With the emergence of GAI, new possibilities arise for interpreting code semantically and evaluating student submissions based on reasoning and problem-solving rather than rigid test cases. These systems can complement or even surpass conventional tools in specific contexts, especially when coupled with pedagogical strategies that promote iterative learning and constructive feedback [8, 7]. The strategy proposed in this paper builds on these recent advances by combining generative AI capabilities with a structured evaluation approach. Thus, it addresses some limitations of traditional systems and leverages a more flexible and nuanced assessment process.

3 Material and Methods

This section presents the Case study methodology that drives the work and the tools used.

3.1 Case Study

This study follows a practice-oriented methodology structured into four main phases: *Design*, *Implementation*, *Analysis*, and *Interpretation*. These phases guided the development and evaluation of a Generative Artificial Intelligence (GAI) algorithm that automatically assesses student solutions to programming exercises. The method was applied over two academic years in the Web Programming course at the University of Minho, involving over 200 students.

- **Design Phase:**
  - Identify the pedagogical needs and challenges in evaluating programming exercises in large classes.
  - Define the classification rubric, consisting of three levels: **-1** (no understanding), **0** (partial understanding), and **1** (complete understanding).
  - Design a structured statement format for programming problems to facilitate prompt comprehension by the GAI model.
  - Develop and test different prompting strategies to effectively and consistently evaluate code.
- **Implementation Phase:**
  - Develop the GAI-based correction algorithm, integrating prompt templates with programmatic evaluation workflows.
  - Prepare datasets with anonymised student submissions from the Web Programming course.
  - Implement automation scripts to process code submissions, interact with the GAI model (e.g., ChatGPT), and collect responses.
  - Generate automatic classifications and feedback based on the GAI's output.
- **Analysis Phase:**
  - Compare the GAI-generated classifications with those manually assigned by experienced instructors.
  - Assess the generated feedback's consistency, accuracy, and usefulness.
  - Analyse how well the rubric and prompt strategy align with the intended learning outcomes.
- **Interpretation Phase:**
  - Reflect on the impact of the proposed strategy on assessment processes in programming education.
  - Identify the strengths and limitations of the GAI approach compared to traditional manual correction.
  - Discuss the implications of adopting this type of algorithm for formative and summative assessment in higher education.

All procedures adhered to ethical standards, ensuring that student data remained anonymous and protected throughout the process.

Adopting a case study methodology is appropriate for this research, as it enables the exploration of a GAI approach without compromising the course's core learning objectives. This method is particularly well-suited for studies conducted in academic environments because it allows professors to observe student behaviours in context and relate these behaviours to their outcomes. [3].

3.2 Tools

Regarding the technological infrastructure supporting this study, several digital tools were integrated into the workflow to support each phase of the process:

- **HackerRank** [6] – Used as the coding environment where students developed and tested their programming solutions.
- **Google Forms** [4] – Employed as the primary submission platform for students to submit their final code.
- **Google Sheets** [5] – Served as the central hub for aggregating responses, generating dynamic prompts, and connecting to the GAI evaluation system.
- **ChatGPT API** [10] – Integrated with the spreadsheet to automatically analyse and grade student submissions based on customised prompts.
- **ioEduc** [13] – Includes the rescue system and allows students to individually and anonymously complete project assessments, both individual and group. These grades are then used by the algorithm.

4 GAI Algorithm

This section presents the Generative Artificial Intelligence (GAI) algorithm developed to support the automated assessment of programming exercises. The algorithm features a modular and scalable architecture, enabling adaptation to various educational contexts. Its integration into the Web Programming course at the University of Minho illustrates its applicability to real teaching scenarios. The algorithm includes a well-defined workflow, error mitigation strategies, and a customised prompt structure that guides the evaluation process.

4.1 Assessment Guidelines

The design and implementation of the GAI-based correction system were guided by a set of pedagogical and operational principles to ensure both educational value and fairness in the assessment process. These guidelines aim to align the use of generative models with authentic student evaluation practices:

- **Assessment Objective:** The programming exercise assesses students' understanding of the covered topics. Students are allocated 50 minutes to respond to the proposed challenge and demonstrate their knowledge under time constraints.
- **Realistic Learning Context:** The assessment is framed within a broader narrative. Students are encouraged to approach the task as a practical and contextualised project, including peer-assessment and reflection components, thereby mimicking real-world development environments.
- **Personalised Submission:** At the beginning, students must fill out a form, including their personal data. This helps prevent plagiarism and also provides valuable context for the LLM to interpret.

- **Generative Model for Evaluation:** A large language model (LLM), such as ChatGPT, is used to analyse the code and the statement and to produce an assessment aligned with a simplified three-level scale instead of traditional numeric scores (e.g., 0–20). This strategy encourages qualitative feedback and conceptual mastery rather than point-based evaluation.
- **Learning from Human Input:** The evaluation process includes data from peer-assessment and other manual assessments (e.g., project contributions), which serve as training context and calibration reference for the LLM, helping it to interpret performance with greater nuance.
- **Human Validation:** To mitigate errors and ensure the quality of the automated evaluation, the system includes human-in-the-loop verification. Specifically, all extreme results (i.e., grades of **-1** or **1**) are manually reviewed by the professor. Additionally, a representative sample of other responses is also validated to improve the reliability of the overall process.
- **Feedback Transparency:** Final evaluations, including those produced by the LLM and any manual corrections, are shared with students. This ensures that learners receive meaningful feedback and can understand the rationale behind their results.
- **Rescue possibility:** Students should have a mechanism to contest their grades. This mechanism should include rules and may be approved automatically or manually.

4.2 Statement Design Guidelines

The success of a Generative AI (GAI) algorithm in evaluating student programming exercises is directly influenced by the clarity, structure, and technical precision of the exercise statement. Since LLMs interpret natural language through prompt engineering, the statements must be designed to avoid ambiguity, favour structure, and anticipate edge cases. The following guidelines have been defined to ensure that exercise statements are effectively interpreted by GAI systems while also maintaining pedagogical consistency and technical correctness according to the rules defined by professor:

- **Use a modular structure:** Divide the statement into logical parts (e.g., Part A, Part B) when multiple technologies or concepts are involved. This approach helps both students and AI models grasp the scope and sequence of tasks. Each part can have distinct evaluation rules/prompts.
- **Present requirements using ordered lists:** Use bullet points (e.g., a., b., c.) or numbered lists to clearly separate tasks. Each instruction should focus on a single objective or functionality.
- **Define all custom attributes and variables:** When referring to identifiers like `eid`, `SID`, or contextual values (e.g., date of birth, student ID), always clarify their origin, format, and usage to prevent students' plagiarism or copying. For example: "margin set to the last two digits of the student id (SID)."
- **Maintain consistent naming conventions:** Select a uniform style for IDs, variable names, and file names (e.g., camelCase, lowercase, or snake_case) and apply it consistently throughout the documentation.
- **Specify expected interactions clearly:** Clearly define which user actions trigger responses (e.g., clicking a button triggers an alert). If JavaScript is involved, indicate the event handlers and the expected messages or outcomes.
- **Clarify data processing steps:** If the task involves data transformation (e.g., calculating a value, subtracting an offset from an ID), provide an explicit formula or explanation to ensure consistent interpretation.

- **Include dynamic elements and explain them:** When utilising runtime-generated content (e.g., DOM manipulation, dynamic tables, loops over API data), describe the expected output structure (e.g., a table with three columns: X, Y, Z).
- **Describe external resources and formats:** If the task involves fetching data from external APIs or files, include the endpoint, HTTP method, expected format (e.g., JSON), and the structure of the response object. For instance: "Access the property `data.URL` in the JSON response."
- **Avoid vague or overly open-ended language:** Replace expressions like "do something similar" or "create a generic solution" with specific technical instructions. LLMs require clear guidance to produce accurate feedback.
- **In case of need, indicate what libraries or frameworks students can use:** If students must use external libraries (e.g., Bootstrap, jQuery), specify the version and provide a link or state how they should be included (CDN, local file, etc.).
- **Be explicit about expected outputs:** Whether it is a table, form, alert, or LocalStorage entry, describe the expected structure, data, and user experience. This is essential for automated evaluation to ensure presence, correctness, and completeness.

4.3 Personalisation and Anti-Plagiarism Strategy

Beyond supporting the automated correction process, the exercise statement also plays a crucial role in promoting academic integrity and preventing plagiarism. In educational settings where assignments are automatically corrected using generative models, the risk of students sharing answers or reusing solutions significantly increases if all statements are identical.

To mitigate this risk, the statement should be structured to ensure consistency while remaining adaptable for each student. A recommended strategy involves integrating personalised variables, such as the Student Identification Number (SID), name, or date of birth, into the exercise's body. These variables can be collected at the beginning of the assessment and programmatically embedded within the task description.

For example, the student ID (SID) can define the margin of an HTML element, serve as input to a function, or determine the value passed to an API endpoint. This not only ensures that each student receives a unique version of the same pedagogical challenge, but also preserves fairness by standardising difficulty and learning outcomes across the cohort.

Such dynamic personalisation enhances the complexity of sharing solutions while ensuring consistency in assessment criteria. It also promotes deeper engagement from students, who must apply core concepts to contexts with their own data.

These guidelines ensure that statements are not only pedagogically sound but also technically compatible with automated correction systems based on generative AI. They support both human and AI comprehension, enhancing fairness, accuracy, and the quality of feedback.

4.4 Prompt

The following pseudocode outlines a generic engine for generating a structured prompt to be interpreted by a large language model (LLM) like ChatGPT. This prompt aims to assess students' programming exercises by integrating contextual data provided by students and peer assessment information.

Algorithm 1 GeneratePromptForGAIEvaluation (example).

```
Input: statement, rules, variablesText, studentCode, peerScore, groupScore: text
Output: promptText: text
// 1. Build the exercise description
statement ← "Develop a web page considering the variables: " + variablesText;
rules ← "The rules to follow are: " + rules;
// 2. Format the student's submitted code
studentCode ← "<!– (start of code) –>" + studentCode + "<!– (end of code) –>";
// 3. Build technical evaluation section
promptText ← "For the statement: " + statement + "\n";
promptText ← promptText + rules + "\n";
promptText ← promptText + "The following code was submitted:\n" + studentCode
 + "\n";
promptText ← promptText + "GPT, considering all specified rules, evaluate the
 code using a scale from 0.00% to 100.00%.\n";
promptText ← promptText + "Return only: final grade: (value) | missing: list of
 what was not done.";
// 4. Add behavioural evaluation based on peer scores
promptText ← promptText + "1) Determine if the code was authored by the
 student:\n";
promptText ← promptText + "- Peer score: " + peerScore + "\n";
promptText ← promptText + "- Group score: " + groupScore + "\n";
promptText ← promptText + "Low score and perfect code suggests external
 help.\n";
promptText ← promptText + "High score with normal errors suggests authentic
 student work.";
// 5. Add plagiarism and AI detection analysis
promptText ← promptText + "2) Evaluate likelihood of plagiarism or AI-generated
 code using a scale from 0 (none) to 5 (highly suspicious).\n";
promptText ← promptText + "Respond only with: plagiarism: 0–5 | done: yes/no |
 justification: [reasoning]";
// 6. Output the final prompt
return promptText;
```

The algorithm receives a set of textual parameters as input that define the context and content of a programming exercise:

- **statement** – It is the programming exercise statement;
- **rules** – The specific technical/structural requirements to be followed when developing the code
- **variablesText** – a list of variables to be used in the implementation and has students' characteristics (e.g. BirthDay, Favourite Colour, others);
- **studentCode** – the code developed and submitted by the student;
- **peerScore** – the individual participation score attributed by peers' assessment in the group;
- **groupScore** – the evaluation score attributed to the student's performance in a group project.

The output is a fully structured prompt (**promptText**) that is interpreted by a Generative AI model (e.g., ChatGPT), combining all contextual elements and evaluation criteria in

natural language.

4.5 Mitigation Strategies

To mitigate potential misclassifications and ensure fairness in the grading process, the system integrates a gamified appeal mechanism derived from the TechTeach framework, referred to as *Grade Rescue*. This mechanism enables students to request a grade revision if they believe their assessment was inaccurate or if they encountered exceptional circumstances that may have affected their performance.

Requests for grade revision must be formally submitted and are subject to the professor's approval. If a request is accepted, the corresponding grade is temporarily suspended, and the student is closely monitored throughout the remainder of the subject. At the end of the term, the case is reassessed, taking into account the student's overall engagement, consistency, and contributions to determine the final grade.

By incorporating this human-in-the-loop approach into the GAI-driven evaluation process, the system enhances its commitment to transparency, pedagogical integrity, and continuous improvement. It also empowers students to participate actively to ensure the fairness and accuracy of the assessments, with the professor as the jury.

Although the proposed approach is scalable and independent of any specific tools or platforms, it is crucial to demonstrate its application in a real-world context. The following section presents a practical example of applying the algorithm using a selected set of tools.

5 Case Study

The case study section presents the results obtained during the 2023/2024 edition of the Web Programming course at the University of Minho. In this edition, 118 students from the third year of the Engineering and Management of Information Systems Bachelor's program participated in the proposed programming exercises.

To provide a comprehensive understanding of how the generative evaluation process was implemented, the following subsections describe the structure of the exercise statement, the mechanism for generating prompts, and the methodology used to calculate final grades.

5.1 Workflow

The correction process incorporates various digital tools into a unified pipeline that guarantees efficiency, scalability, and pedagogical relevance. The following steps outline the complete operational workflow of the algorithm:

1. The professor designs the programming problem and publishes it on the **HackerRank** platform.
2. Students are instructed to develop and test their solutions using HackerRank's interface.
3. Upon finalising their code, students submit it via a **Google Form**.
4. Submissions are automatically collected and organised in a **Google Sheet**.
5. The spreadsheet is connected to the **ChatGPT API**, which analyses each submission using a tailored prompt that incorporates the problem statement and a predefined evaluation rubric.
6. The algorithm considers three primary inputs: the student's submitted code, the problem statement, and the student's performance in the group project (collected through the ioEduc platform). Based on these inputs, the system generates a grade in the range: **-1**, **0**, or **1**.

7. As a quality control measure, all responses graded with **-1** (indicating a lack of understanding) or **1** (indicating mastery) are manually reviewed by the professor to confirm the accuracy of the AI-generated evaluation.

This workflow enables the correction of large volumes of student submissions with speed and consistency, while still accommodating more complex or non-standard solutions.

5.2 Exercise Statement

The exercise statement includes some initial rules. It follows a controlled and semi-personalised structure to ensure fairness, consistency, and academic integrity. The key rules are:

- Duration: **50 minutes + 5-minute buffer**.
- The exercise consists of a single statement divided into two distinct parts (Part A and Part B).
- Students must develop and test their code **online using the HackerRank platform**.
- To solve the project, students must consider their answers (e.g., birthdate, student ID, and others) and use the value for the underlined variables.
- Students can consult the class slides and previously developed project code.
- The use of **Generative AI tools is strictly prohibited**.
- Final code solutions must be submitted via a dedicated **Google Sheets form**.

Statement – Part A

Develop a complete HTML page using **Bootstrap** and **LocalStorage**, incorporating the following features:

a. A `div` element with the following characteristics:
- `id = eid`;
- Margin set to the last two digits of the student's SID;
- An `<h3>` element with the text `"DSI Courses"`, centred, and with a background colour (hex) corresponding to the student's birthdate (format: `yyyy-mm-dd`);
- An image containing the DSI logo must be a hyperlink to the DSI website and open in a new tab.

b. A Bootstrap grid layout (8 + 4) with the following elements:
- A table with header and body, containing data from three DSI courses: *study cycle*, *acronym*, and *name*;
- A form for collecting student opinions about the courses, including placeholders for: `fname`, `course`, and `opinion`.

c. A JavaScript file that:
- Submit event on the form bottom with name(id);
- Displays an alert message: `"Thank you name for your opinion"`.

d. Stores the form data in LocalStorage as an object named after the eid.

Statement – Part B

Part B has ten distinct JSON versions, with the version each student receives determined by their Student ID. The goal is to develop a solution that interacts with external data using JavaScript and the fetch API. The student must know how to make an API call to access the exercise. Otherwise, he cannot do Part B due to a lack of knowledge. To solve this part, student should:

1. Create an HTML element and ensure it is correctly referenced and manipulated in the corresponding JavaScript code.
2. Implement a **FETCH API request** using the `GET` method and the `JSON.parse()` function to process the response.
3. Access the following endpoint: `URL/mt2/:id` where `:id` must be replaced with the result of `SID - 13500`.
4. Iterate over the JSON object received in the `fetch` response using any of the following: `map`, `for`, or `while`.
5. Extract the value of the `data.URL` property from the API response.
6. Iterate over the content of `data.URL`, printing all sub-objects and arrays present in this property.
7. Dynamically create and display an HTML table within the `fetch` block, representing the extracted content.
8. Create additional HTML elements of type `img` (image) and `ul/li` (lists) dynamically within the `fetch` response processing using JavaScript.

Student Variables

Students were asked to complete an initial questionnaire to personalise the solution and dynamically generate specific requirements. The collected data was then used to parameterise the exercise statement. The following variables were gathered:

- `birth` – Student's birth date (format: `yyyy-mm-dd`)
- `eid` – Student identification number
- `fname` – Student's full name
- `SID` – Project Group identifier
- `name` – Student's name
- `course` – Course
- `email` – Student's institutional email address

The collected data was essential for personalising the task, ensuring each student received a unique and context-aware version of the exercise.

5.3 Evaluation Dimensions and Scoring

The Generative AI (GAI) algorithm was designed to assess student performance by considering three complementary evaluation dimensions:

- **Student's Response to the exercise:** Evaluates whether the submitted code adequately addresses the exercise statement by following the specified variables and technical requirements. The LLM directly analyses it based on the structured prompt, which already includes student variables correctly placed within the exercise statement. The result is calculated in a (0–100%) grade.
- **Student's Participation in the Project (0–100%):** Based on peer assessment, each group member can indicate whether another member contributed to the project. The participation score is calculated as:

$$\text{Participation } (\%) = 100 - \left(\frac{\text{notWork}}{\text{TotalAssessments}} \times 100 \right)$$

where `notWork` is the number of times the student was marked as a non-contributor.

- **Student's Performance in the Project (0–100%):** Group members evaluate each other's contribution on a scale from -4 to 4. The performance score is calculated as that normalises peer-assessed performance into a 0–100% scale:

$$\text{Performance } (\%) = \frac{\sum \text{grades} + 4}{8} \times 100$$

The overall grade produced by the LLM is then interpreted according to the following thresholds:

- **-1:** The student does not demonstrate basic knowledge ($< 25\%$).
- **0:** The student demonstrates partial understanding (25%–74%).
- **1:** The student shows mastery of the content ($>= 75\%$).

5.4 Sheet Template

The spreadsheet template includes 30 columns and serves as the primary data collection and processing environment. Each row corresponds to an individual student's response. The sheet's structure is organised as follows:

- **Column A** – Student participation score based on peer evaluation;
- **Column B** – Student performance within the group (group-assessed contribution);
- **Column I** – Exercise statement and student answer - Part A ;
- **Column J** – Exercise statement and student answer - Part B;
- **Columns K to Q** – Student-specific variables collected via the pre-assessment questionnaire (e.g., `birth`, `eid`, `SID`, etc.);
- **Column R** – First prompt generated for code evaluation based on the statement (Column C) and student inputs (Column K to Q);
- **Column U** – Second prompt generated for code evaluation based on the statement (Column D) and student input calculated (Column L);
- **Column Z** – Authorship prompt generated for authorship validation and potential plagiarism detection based on Column A:Q;
- **Column AB** – Final result, calculated based on the outputs from both prompts (Columns Z and AA) and validated by the professor when necessary.

The missing columns are support columns used to parse the results (e.g., split values by |). This spreadsheet-driven architecture enables the automation of prompt generation, integration with the ChatGPT API, and storage of all relevant metadata needed for evaluation.

5.5 Prompt

A dynamic prompt was automatically generated at Google Sheets by combining the student's code submission with their personal data and the exercise rules. The prompt follows a two-step structure:

- **Step 1 – Code Evaluation:** Reads the exercise statement and the student's code, then assesses the solution based on predefined grading rules.
- **Step 2 – Authorship Verification:** Cross-references the code with the student's individual performance to identify potential copy or unauthorised use of generative AI tools.

A simplified version of the evaluation prompt used in Google Sheets is shown below:

Listing 1 Snapshot of the primary prompt generation formula in Google Sheets.

```
=IF(AND(I3<>"",F4=$F$3),
  CONCATENATE(
    "For the statement: 'Develop a web page considering the variables
       : ",
    $K$1,$K4,$L$1,$L4,$M$1,$M4,$N$1,$N4,$O$1,$O4,$Q$1,$Q4,
    "' and the rules present in the following items:\n",
    $I$2,"\nThe following response was obtained:\n",
    "<!-- (start of code) -->\n", I4, "\n<!-- (end of code) -->\n\n",
    "GPT, considering everything requested in each of the rules, ",
    "evaluate the code using a scale of 0.00% to 100.00% and ",
    "send only the result in the format final grade: (value) |
       missing: list of what was not done"
  ),
"")
```

This evaluation prompt was generated and stored in column `S` of the spreadsheet. In this setup, cells `$K$1` to `$Q$1` contain the names of student-specific variables, and the corresponding values are located in `K4` to `Q4`. The complete exercise statement is placed in `$I$2`, while the student's code submission appears in `I4`.

This formula dynamically integrates all necessary contextual and individualised information into a structured prompt, which the GAI model interprets to assign a preliminary grade.

A second prompt was explicitly designed to support the detection of potential academic misconduct. It cross-references the quality of the submitted code with the student's performance within their group. By leveraging peer assessment data, this prompt helps identify inconsistencies between student effort and the code quality, factors which may suggest the use of external assistance or generative AI. The following listing illustrates the structure of this second prompt:

Listing 2 Snapshot of the authorship and plagiarism detection prompt.

```
"GPT, considering the presented code,\n",
"1) Verify if the code was (yes or no) created by a student based on
    peer evaluations, considering ", CONCATENATE($A$3, A4, $B$3, B4),
     ". A score above 0 means the student worked adequately.
    Therefore, if the student put little effort, i.e., had a score
    below 0.5, received a negative score (<-0.50), and the presented
    code is partially or entirely correct, they did not do it. If the
     code is acceptable and the student put in the effort (score >
    0.65) and received a score greater than -0.25, then the code is
    theirs.\n",
"Based on the result of 1) and to identify possible plagiarism or use
     of AI-generated code, evaluate the code on a scale of 0 (nothing
    ) to 5 (everything). Consider if the code is simple, does not
    perfectly meet the requirements, and has normal errors typical of
     human work. If the code is too correct or uses incorrect
    variables, it might have been done by a machine. If the student
    is diligent with a good evaluation and the code is good, then it
    is also done by a human.\n",
"As a response to the request, send only the result in the format:
    plagiarism: 0 to 5 | done: yes or no | justification: plagiarism
    scale and the appropriate: yes or no"
```

In this case, cells `A3` and `B3` store the names of the variables representing peer- and group-assessed performance, while `A4` and `B4` contain the student's actual evaluation results. These values are used to construct a second, personalised prompt that guides the GAI model in making inferences about authorship and originality.

To calculate the final classification, this second prompt complements the initial code evaluation by incorporating behavioural and performance indicators, as described in Subsection 5.2 and stores in result in column AA.

6 Results

The evaluation results demonstrated a diverse range of student performance levels and highlighted the effectiveness of the grading strategy.

6.1 Examples of GAI Outputs

Below are selected excerpts of feedback generated by the GAI algorithm, which combines code analysis with behavioural performance metrics. Each output includes a plagiarism risk assessment, an authorship judgement, a detailed justification, the corresponding final grade, and an indication of missing elements.

Plagiarism: 2 | **Human:** Yes
Justification: The code presents some common errors that a student might make, such as incorrect CSS formatting and the absence of placeholders in the form. Additionally, the overall structure of the code is simple and does not perfectly meet all specifications, which is typical of work done by a student. The effort evaluation and the completed work grade indicate that the student can produce acceptable code. Therefore, the probability of plagiarism is low.

Final Grade: 70.00% | **Missing:**

- The margin of the `div` with `id=eid` should be defined by the last two digits of the SID (30px), but it was set as `30em`.
- The background color of the `<h3>` should be based on the birth date (22/06/2003), but it was set as `#22062003`, which is not a valid hexadecimal value.
- The courses table is not using the proper HTML table structure (`<table>`, `<thead>`, `<tbody>`, `<tr>`, `<th>`, `<td>`).
- The form is missing placeholders for `fname`, `course`, and `opinion`.
- The JavaScript code contains syntax and logic errors.
- The object saved in `LocalStorage` should be named after the `eid`, but the current implementation does not guarantee this.

Plagiarism: 4 | **Human:** No
Justification: The code presents various errors and inconsistencies typical of code generated by AI or copied from a source without proper understanding. Additionally, the student's effort evaluation is 0, and the completed work grade is 0, indicating that the student did not put in any effort. Therefore, it is highly likely that the code was not done by the student.

6.2 Student Results

The calculated scores ranged from **20% to 78%**, with plagiarism suspicion scores varying between **2 and 4**, and AI usage likelihood scores also ranging from **2 to 4**.

The final grade distribution was as follows:

- **-1 (Insufficient understanding)**: 12 students (10.17%)
- **0 (Basic knowledge)**: 104 students (88.14%)
- **1 (Advanced mastery)**: 2 students (1.69%)

Among the twelve students who received a grade of **-1**, three submitted a successful request for grade redemption through the *Grade Rescue* mechanism. The other students did not meet the criteria for a positive reassessment.

These results suggest that the algorithm, combined with human validation, reliably classified student performance while also offering a pathway for reconsideration in exceptional cases.

6.3 Student's Feedback

A feedback form was included in Google Forms to assess students' perceptions regarding the test and the evaluation strategy based on Generative AI. Students were asked to rate the difficulty of the test and their level of agreement with the proposed model using a 5-point Likert scale. Table 1 summarises the results.

Table 1 Student feedback on test difficulty and model approval (1 = very low, 5 = very high).

Scale (1–5)	Test Difficulty	Model Approval
1 – Very Easy / Totally Disagree	1 (0.83%)	2 (1.67%)
2 – Easy / Disagree	6 (5.00%)	7 (5.83%)
3 – Neutral	63 (52.50%)	34 (28.33%)
4 – Difficult / Agree	44 (36.67%)	48 (40.00%)
5 – Very Difficult / Strongly Agree	6 (5.00%)	29 (24.17%)

Analysing the results presented in Table 1, it is noteworthy that over 90% of students approved of the proposed evaluation model. Regarding the difficulty level, the majority of students (52.50%) considered the exercise to be accessible, selecting the neutral option on the scale.

7 Conclusion

This study presents a novel evaluation methodology for higher education that integrates Generative Artificial Intelligence (GAI) into the correction process of programming assignments, aligned with the TechTeach pedagogical paradigm. Addressing the growing need for adaptable and scalable grading systems, the proposed solution blends personalised assessment, peer feedback, and AI-based prompt analysis to ensure accurate evaluation.

The key to the process is that the exercise statement must have a template that is "prompt-ready". The assessment prompt should consider a global model that operates independently of the exercise, allowing the prompt to work effectively. Another relevant point is that a dynamic exercise statement can yield distinct solutions for each student. This is achieved by incorporating personal variables into the exercise. Then, the system should not be too rigid and should be able to accept partially correct submissions or submissions with syntactic errors (which, in real life, code editors can help to resolve).

The strategy relies on integrating diverse digital tools: HackerRank for code development and validation, Google Sheets and Forms for orchestrating data and prompt automation, and the ChatGPT API for intelligent code evaluation. By incorporating student-specific data, such as identifiers, birthdates, and group assessments, the system not only tailors the assessment to each individual but also detects potential academic dishonesty, including copying or unauthorised AI usage. Additionally, including the *Grade Rescue* mechanism ensures that students can contest results, reinforcing human oversight while demanding more from them. If a student asks for redemption, they will remain under the professor's scrutiny until the end of the course unit. Therefore, if the student believes he does not know more than the result achieved, he avoids requesting redemption. Otherwise, he must enhance their efforts and demonstrate that the exercise result was an error.

This approach combines human and machine strategies to automate the evaluation process.

A case study conducted during the 2023/2024 academic year, involving 118 students from the Web Programming course at the University of Minho, demonstrated the practical viability of this method. Most students (88.14%) received intermediate scores (0), while cases requiring attention (10.17%) were effectively identified and flagged. The combination of behavioural metrics and code quality proved to be a valuable approach for reinforcing academic integrity. The results concerning student perception are encouraging, as fewer than 10% of respondents expressed disagreement with the evaluation model. Furthermore, 52.50% of students rated the difficulty as neutral, suggesting that the assessment was suitable for the course's expected level of challenge. Students with a plagiarism score of 4 were manually assessed to determine the code origin and validate the grade. To prevent misinterpretations by the algorithm, both scores are kept separate.

Future work focuses on optimising the model and exploring other prompts and variables. The long-term goal is to deliver a reliable, student-centred, and pedagogically meaningful system for the automated assessment of coding exercises in higher education.

References

1. David Baidoo-Anu and Lawrence Owusu Ansah. Education in the era of generative artificial intelligence (ai): Understanding the potential benefits of chatgpt in promoting teaching and learning. *Journal of Higher Education Theory and Practice*, 23(2):1–10, 2023. URL: `https://articlegateway.com/index.php/JHETP/article/view/7004`.
2. Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.
3. Kathleen M Eisenhardt. Building theories from case study research. *The Academy of Management Review*, 14(4):532–550, 1989.
4. Google. Google forms — online form builder. `https://forms.google.com`, 2024. Accessed: 2025-04-25.
5. Google. Google sheets — online spreadsheet tool. `https://sheets.google.com`, 2024. Accessed: 2025-04-25.
6. HackerRank. Hackerrank – technical hiring platform. `https://www.hackerrank.com/`, 2024. Accessed: 2025-04-25.
7. Enkelejda Kasneci, Kevin Sessler, Niklas M. Kuhl, Maria Bannert, Daria Dementieva, Florian Fischer, Urs Gasser, Fabian Haag, Philipp Molt, and Manfred Spitzer. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103:102274, 2023. `doi:10.1016/j.lindif.2023.102274`.

8 Nimród Mike, Krisztina Karsai, Gábor Orbán, Alexandra Bubelényi, Csaba Norbert Nagy, and Gábor Polyák. The impact of chatgpt on student performance in higher education. In *CEUR Workshop Proceedings*, volume 3737, pages 1–10, 2024. URL: https://ceur-ws.org/Vol-3737/paper28.pdf.

9 Raffaele Montella, Ciro Giuseppe De Vita, Gennaro Mellone, Tullio Ciricillo, Dario Caramiello, Diana Di Luccio, Sokol Kosta, Robertas Damaševičius, Rytis Maskeliūnas, Ricardo Queirós, and Jakub Swacha. Leveraging large language models to support authoring gamified programming exercises. *Applied Sciences*, 14(18), 2024. doi:10.3390/app14188344.

10 OpenAI. Chatgpt api documentation. https://platform.openai.com/docs, 2024. Accessed: 2025-04-25.

11 Filipe Portela. Techteach—an innovative method to increase the students' engagement at classrooms. *Information*, 11(10), 2020. doi:10.3390/info11100483.

12 Filipe Portela. A new approach to perform individual assessments at higher education using gamification systems. In *4th International Computer Programming Education Conference (ICPEC 2023)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

13 Miguel Silva, Diogo Ferreira, and Filipe Portela. IoEduc - Bring Your Own Device to the Classroom. In Ricardo Queirós, Filipe Portela, Mário Pinto, and Alberto Simões, editors, *First International Computer Programming Education Conference (ICPEC 2020)*, volume 81 of *Open Access Series in Informatics (OASIcs)*, pages 23:1–23:9, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/OASIcs.ICPEC.2020.23.

14 Artur Strzelecki. Chatgpt in higher education: Investigating bachelor and master students' expectations towards ai tool. *Education and Information Technologies*, 2024. doi:10.1007/s10639-024-13222-9.

15 Rafal Wlodarski, Leonardo Sousa, and Allison Pensky. Level up peer review in education: Investigating genai-driven gamification system and its influence on peer feedback effectiveness, April 2025. doi:10.48550/arXiv.2504.02962.