New York Institute of Technology
College of Engineering and Computing Sciences
Department of Computer Science

# Group Report:
# Code Grader

AI Powered Assignment Grading System

CSCI 870 Master Project I
Prof. Wenjia Li

Himani Parikh [1322085, hparik07@nyit.edu],
Kashyap Shukla [1317566, kshukl03@nyit.edu]

# Code Grader

AI Powered Assignment Grading System

1th Wenjia Li
Project Supervisor
New York Institute of Technology

2rd Himani Parikh
Team Member
New York Institute of Technology

3th Kashyap Shukla
Team Member
New York Institute of Technology

***Abstract*—The Code Grader project aims to revolutionize the assignment grading process in educational settings by leveraging advanced language models and automation techniques. Unlike traditional grading methods that are often time-consuming and prone to errors, Code Grader offers an innovative solution that enhances efficiency and accuracy. By integrating Language Model models like LLM with the Ollama API, Code Grader automates the grading process, significantly reducing grading time by up to 60%. This automation not only saves valuable time for educators but also ensures consistent and objective grading standards across assignments.**

***Keywords— Machine Learning, MongoDB, Ollama, Express.js, ReactJS, Node.js, CodeLlama***

## I. INTRODUCTION

In today's fast-paced, technology-driven world, education is undergoing a significant transformation. One of the most intriguing advancements on this journey is the integration of artificial intelligence into the classroom, particularly in the realm of grading and assessment. This blog post explores the realm of "AI-Graded Assessments" and delves into the pros and cons of automated grading systems.

The impact of technology is pervasive in the quickly changing landscape of education. The incorporation of artificial intelligence (AI) into the grading process is one of the many revolutionary innovations that stands out as noteworthy. This combination is rethinking conventional approaches to evaluating student performance and provides a wealth of benefits that enhance learning for both teachers and students types of systems that can suggest music could really help user to improve the mental health.

With its innovative approach, Code Grader aims to revolutionize the assignment grading experience, making it more efficient, accurate, and user-friendly. By combining advanced technology with practical use cases, Code Grader seeks to enhance the learning experience for educators and students alike.

## II. LITERATURE REVIEW

The existing manual grading processes in computer science courses, exacerbated by growing class sizes, present significant challenges. Professors and teaching assistants struggle with increased workloads, causing delays in providing timely feedback to students.[6] Recognizing the inefficiencies in traditional grading methods, there is a need for a solution to streamline assessment workflows, expedite grading, and enhance the educational experience. [7] This project addresses these issues by proposing the implementation of an Automated Grading AI. Leveraging the opportunities presented by large language models like ChatGPT, particularly the CodeLlama model, the study explores the potential for automatic grading of student assignments [8]. By utilizing the advanced capabilities of Meta's CodeLlama, the aim is to provide valuable insights and contribute to the development of tools that can optimize educational resources and improve the grading process.[9]

### *A. Exiting Techniques for Automated Grading*

1) *Unit testing:*

In this test, the targeted software should be clean of any syntax errors, by passing the targeted software to the compiler then apply the test. The test outcome indicates whether the software produces the correct or incorrect results based on predefined inputs outlined in the specification document or assignment requirements. However, keeping unit tests updated with system changes can be time-consuming and may lead to misleading results if not properly maintained. Running extensive unit tests on a large grading system can strain resources and impact performance.[13]

2) *Sketching Synthesis and Error Statistical Modeling :*

The authors presented a novel tool utilizing sketching synthesis and Error Statistical Modeling (ESM) for immediate feedback on introductory programming assignments. The tool was applied to MIT's "Introduction to Computer Science and Python Programming Language." It works by providing the system with a reference implementation for simple computational problems. However, limitations include a lack of checks for structural requirements, limited support for large constant values, and no coverage of Object-Oriented Programming (OOP) concepts.[13]

3) *Peer-To-Peer Feedback:*

In this approach [13], the instructor makes the students randomly grade each other's answers. It can help students spot and get used to common mistakes, but there are problems with this method. For instance, there might not be quick feedback and the feedback could be wrong or incomplete because students might not know everything, especially beginners.

4) *Random Inputs Test Cases:*

This approach is proposed in [13], where instructor prepares a collection of distinct inputs that used to check if the student assignment output is false positive or false

negative. However, utilizing this method for grading assignments has limitations, as it does not provide feedback on specific errors made by students, rendering it less comprehensive for effective assessment.

A whole new method of marking exams has emerged as a result of the development of artificial intelligence. Automated grading systems powered by AI are positioned to overcome the drawbacks of traditional grading techniques, providing a range of advantages that have the potential to transform education [2] .

As technology evolves at a rapid pace, the field of education is experiencing transformative changes. An area that has particularly advanced is exam evaluation. Historically, grading exams has been labor-intensive and subjective, often susceptible to human error and bias. However, the advent of automated grading systems utilizing Artificial Intelligence (AI) is revolutionizing the process of exam evaluation.

the literature reviewed underscores the pressing need for innovative solutions to streamline the assignment grading process in educational settings. While traditional grading methods have been prone to inefficiencies and inconsistencies, the emergence of advanced technologies like language models and automation techniques presents an opportunity for significant improvement. By leveraging these technologies, projects like Code Grader aim to enhance the efficiency, accuracy, and user experience of the grading process, ultimately contributing to a more effective and equitable educational environment.

## III. Methodology

### A. Data Collection and Preparation

To train the language model effectively, a diverse dataset of coding assignments and their corresponding solutions is required. This dataset will be collected from various sources, including educational institutions, online coding platforms, and open-source repositories. The assignments will cover a wide range of programming languages, difficulty levels, and coding styles to ensure the model's generalizability.

Additionally, an existing large language model (CodeLlama) pre-trained on code-related tasks will be utilized to generate synthetic coding assignments and their corresponding solutions. This approach will augment the dataset and provide additional training examples, particularly for underrepresented coding styles or problem types.

The collected data will undergo a rigorous pre-processing step, including cleaning, formatting, and tokenization. Code snippets will be tokenized while preserving their syntax, structure, and contextual information, ensuring that the model can effectively learn from the provided examples.

### B. LLM Model Selection

A suitable large language model (LLM) architecture will be selected, considering its pre-training on programming-related tasks and adaptability to the educational context. Potential candidates include CodeLlama transformer-based models with demonstrated performance in code generation and understanding.

The CodeLlama will be utilized directly, without fine-tuning, to leverage its pre-existing capabilities in understanding, reviewing, and generating code. Instead of fine-tuning, the focus will be on developing effective prompting strategies and integrating the LLM into the grading algorithm and feedback generation system.

### C. Grading Algorithm Development

A grading algorithm will be designed and implemented, integrating the pre-trained LLM as a core component. This algorithm will consider multiple criteria for evaluating student code, including correctness, efficiency, code readability, and adherence to coding standards and best practices.

CodeLlama is a powerful language model developed by Meta (formerly Facebook) specifically for code-related tasks. While CodeLlama itself is not directly accessible to the public, we can integrate it using a separate tool called Ollama, which allows users to interact with various language models, including CodeLlama.

Ollama is an open-source command-line interface (CLI) written in Python, hosted on the GitHub repository (https://github.com/ollama/ollama). It serves as a bridge between users and language models like CodeLlama. This interface abstracts away the underlying complexities of accessing and querying different language models, allowing developers to focus on their core application logic. To integrate CodeLlama with Ollama, the following steps can be taken:

*a) Install Ollama*:

Ollama can be installed by cloning the GitHub repository (https://github.com/ollamh/ollama) or using pip to install the package directly from the repository.

*b) Obtain Access to CodeLlama*:

Since CodeLlama is developed by Meta, you will need to obtain access to the language model through appropriate channels, which may involve requesting access from Meta or obtaining a license.

*c) Configure Ollama*:

After installation, Ollama needs to be configured with the necessary credentials or access tokens to connect to CodeLlama. The specific configuration steps may vary depending on the access method provided by Meta.

*d) Provide Input*:

Once configured, Ollama can be used to interact with CodeLlama through the command line. Users can provide code snippets, natural language prompts, or entire files as input.

*e) Leverage CodeLlama's Capabilities*:

Depending on the input provided, CodeLlama can be leveraged for various code-related tasks, such as code generation, explanation, refactoring, or error debugging. Ollama acts as an interface, translating the

user's input into prompts that CodeLlama can understand and process.

*f) Retrieve Output*:

After processing the input, CodeLlama's output will be returned through Ollama, which can be in the form of generated code, natural language explanations, or suggestions for improvement.

The grading algorithm will leverage the LLM's capabilities to analyze the student's code, compare it against the expected solution, and identify potential issues or areas for improvement. Additionally, the algorithm will incorporate performance evaluation techniques, such as test case execution or static code analysis, to assess the code's functionality and efficiency.

### D. User Interface and Feedback Generation

A user-friendly web interface will be developed for instructors to upload coding assignments, configure grading parameters, and view automated assessments. The interface will provide a seamless experience for managing and grading student submissions.

Leveraging the CodeLlama, the system will generate constructive and detailed feedback on student code. The feedback will address specific coding issues, such as logic errors, inefficient algorithms, or style inconsistencies, and offer actionable suggestions for improvement.

The feedback generation process will utilize the LLM's natural language generation capabilities to produce human-readable and understandable feedback, ensuring that students receive clear and meaningful guidance for enhancing their coding skills.

### E. Testing, Validation, and Iteration

Rigorous testing and validation will be conducted using a diverse set of coding assignments and solutions not included in the training dataset. The system's accuracy and effectiveness will be evaluated by comparing its assessments and feedback with human-graded assignments.

The validation process will involve collaboration with educators and subject matter experts to ensure the system's alignment with educational standards and grading practices. Feedback from instructors and students will be collected and analyzed to identify areas for improvement and potential refinements.

Based on the evaluation results and user feedback, iterative improvements will be made to the language model's fine-tuning process, the grading algorithm, and the feedback generation system. This iterative approach will ensure the continuous enhancement of the system's performance and its ability to adapt to evolving educational needs.

## IV. PROJECT REQUIREMENTS

### A. Software Requirement- Development

- IDE: Visual Studio, Co-lab
- Design: Figma, Miro Board, Idea Board
- Database: MongoDB
- Development Technology: React, Tailwind CSS, Express.js, Node.js CodeLlama, Ollama, MongoDB and Vite
- Version Control: GitHub
- Project Management: JIRA
- Documentation: Microsoft Word, Excel, Google Docs

### B. Hardware Requirement

- Minimum of 6GB RAM
- Basic graphics GPU

### C. Functional Requirement

- Ability for students to upload their assignments securely through a user-friendly interface.
- Integration with the LLM (Language Model) API to generate feedback and grades automatically upon submission.
- Display of feedback and grades to students in a clear and accessible format.
- Option for students to view detailed feedback provided by the AI model.
- Separate panel accessible to Teaching Assistants for assignment review and feedback modification.
- TA interface should display a list of student assignments along with their automatically generated feedback and grades.
- Functionality for TAs to review and modify the feedback provided by the AI model.
- Ability for TAs to add personalized comments and annotations to the feedback.
- Option for TAs to adjust grades based on their evaluation of the assignment.
- Seamless integration of the AI LLM model with the assignment submission platform to ensure automatic feedback generation.
- Implementation of robust security measures to protect student data and ensure confidentiality of assignments and feedback.
- Secure authentication and authorization mechanisms for both students and TAs to access the system and their respective panels.

## V. SYSTEM DIAGARMS

### A. Class Diagram

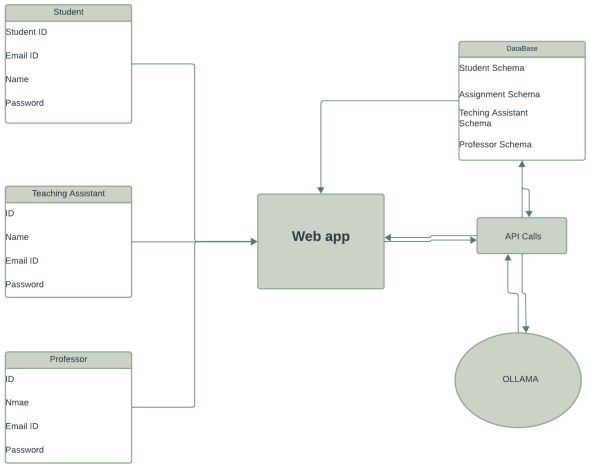


*Figure 1 Class Diagram*

The provided class diagram illustrates the structural design of an academic web application. It showcases the four core entities—Students, Teaching Assistants, Professors, and the Web Application itself, each with essential attributes including ID, Name, Email ID, and Password, which are fundamental for user identification and security protocols. Central to the diagram is the Web Application, depicted as the nexus that facilitates interaction between the user entities and the back-end database. The database is logically partitioned into distinct schemas, each responsible for handling data pertaining to students, assignments, teaching assistants, and professors, ensuring an organized and efficient management system. The diagram also integrates an 'API Calls' mechanism that indicates the application's capability to communicate with external APIs, thereby extending its functionality. One such external service is represented by 'OLLAA,' implying a potential third-party integration or an external module that interacts with the main system to enhance its operational scope.

### B. Conceptual Architectural Diagram

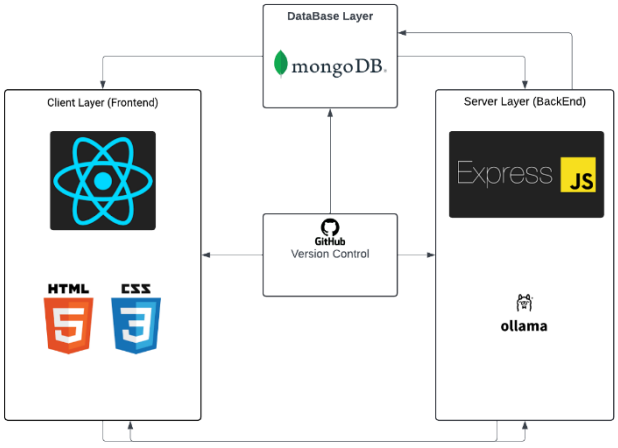


*Figure 2 Conceptual Architectural Diagram*

The conceptual architectural diagram represents the multi-layered structure of a web application. In the front-end client layer, we see the use of HTML5 and CSS3 for markup and styling, along with React, a popular JavaScript library for building user interfaces. This combination suggests a focus on a responsive, interactive user experience.

The server layer is based on Express.js, a web application framework for Node.js that allows for the building of server-side logic and APIs in a scalable and efficient manner. It's depicted alongside 'Ollama', which might refer to a custom-built service or an integration within the server's framework, although its exact purpose is not specified in the diagram.

The database layer showcases MongoDB, indicating the application's reliance on a document-oriented NoSQL database, which is often chosen for its scalability and flexibility with unstructured data. Connecting these layers, GitHub is illustrated as the version control system, essential for collaborative development and maintaining the application's codebase.

### C. Sequence Diagram:

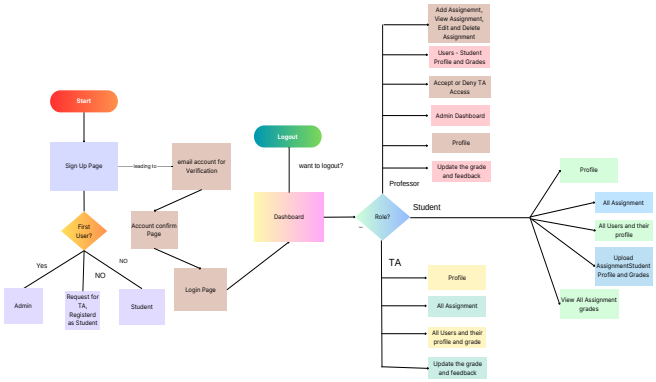


*Figure 3 Sequence Diagram*

The sequence diagram illustrates the workflow of an assignment grading system within an educational web application. It features an integration with the 'Ollama's' LLM model, which evaluates student submissions and generates feedback automatically. The process begins with students either signing up for a new account or logging into an existing one. Once logged in, they can submit assignments, which are then processed by the 'Ollama' within the system's backend.

Professors have the capability to oversee all activities within the platform, including assignment submissions, the grading process, and the dissemination of feedback to students. They log in through a different sequence and have access to a dashboard where all student activities are visible.

Teaching Assistants (TAs), on the other hand, can see students' submissions and the grades and feedback provided by the 'Ollama' LLM model. Their interaction with the system appears to be geared towards assisting with the academic evaluation process, although the specific details of their permissions and capabilities are not fully outlined in the diagram.

This system streamlines the grading process by employing an advanced LLM model to provide timely and consistent evaluations of student work, with the professors

maintaining oversight of the entire process and TAs assisting in managing the workflow. The architecture ensures data is systematically stored in a database, maintaining integrity and traceability of student submissions and feedback.

### D. ER Diagram


*Figure 4 ER Diagram*

The ER diagram for our application showcases the intricate relationships and structure of our database, centered around four main entities: User, Token, Assignment, and StudentSubmission. The User entity captures comprehensive details such as name, email, password, and various role-related attributes, highlighting the system's ability to handle complex user profiles and permissions. The Token entity, linked to the User entity via an ObjectId, manages authentication elements like refresh tokens and IP addresses, ensuring secure user sessions. The Assignment entity stores information about educational tasks, including title, requirements, and due dates, which are created and tracked within the system. Finally, the StudentSubmission entity relates to both User and Assignment entities, documenting student responses, grades, and feedback, thereby facilitating a seamless tracking of academic progress and assessment. This diagram not only delineates the data architecture but also reinforces our application's capacity to handle multifaceted user interactions and educational data management efficiently.

## VI. APPLICATION SCREENSHOT

### A. Home Page

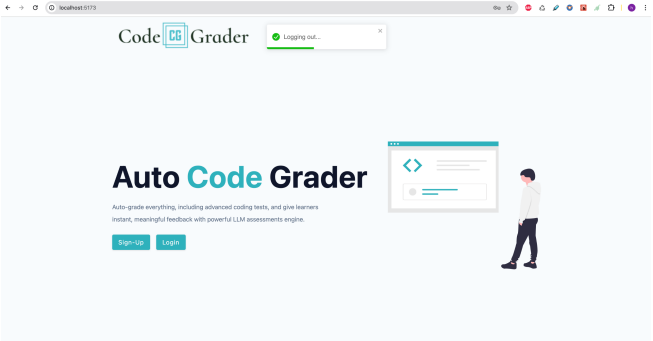


*Figure 5 Home Page*

### B. Sign Up Page

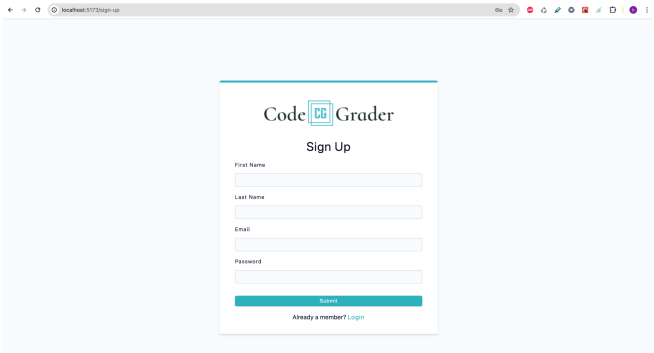


*Figure 6 Sign Up Page*

### C. Student Dashboard

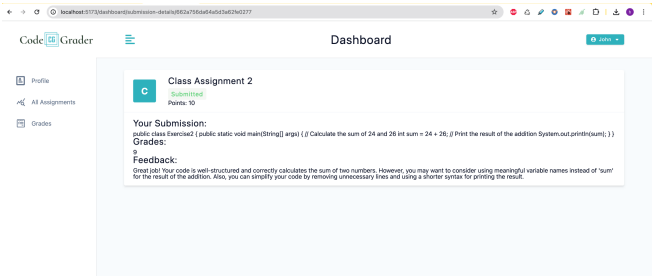


*Figure 7 Student Dashboard*

### D. Professor Dashboard

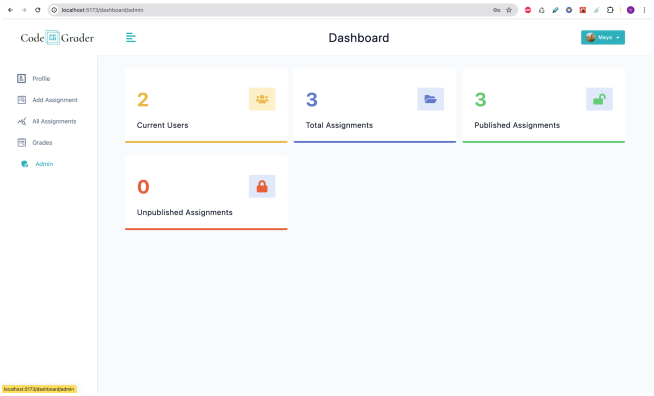


*Figure 8 Professor Dashboard*

### E. Profile Page

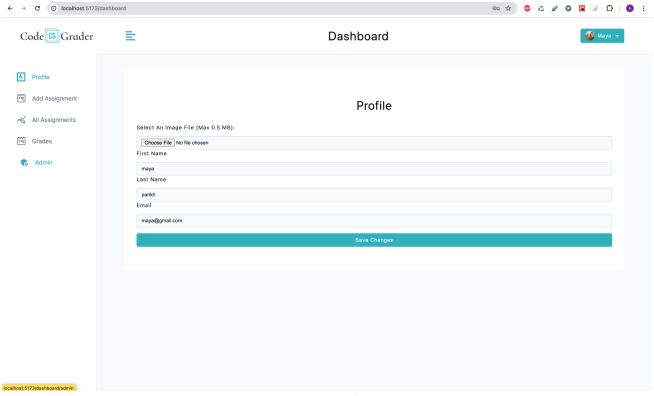


*Figure 9 Profile Page*

## VII. KEY CHALLENGES

During the development of our project, we encountered several technical challenges that required innovative solutions and adaptations. These challenges primarily revolved around the integration of custom machine learning models into our Node.js and Express.js backend, as well as issues connecting our frontend and backend environments.

### *A. Integration of Custom Machine Learning Models:*

Our initial efforts to incorporate custom machine learning models into our backend system encountered multiple obstacles. We explored several approaches, including:

- Utilizing the Hugging Face interface API, which proved incompatible with our specific requirements.
- Attempting to convert a transformer model to a TensorFlow model for integration with TensorFlow.js, but the performance did not meet our expectations.
- Employing tokenizer.json and model files with @xenova/transformers, which failed to achieve our desired performance benchmarks.
- Building an API using the llm-api node module, which also did not adequately address our needs.
- Converting the model to a PyTorch model using the ONNX runtime platform, which faced operational challenges.

The solution to these integration challenges emerged with the adoption of OLLAMA, a lightweight and faster tool designed for running custom language models locally. This tool streamlined the integration process and met our performance and operational criteria, significantly simplifying the incorporation of custom machine learning models into our backend infrastructure.

### *B. Proxy Error Between Frontend and Backend:*

We had a big technical problem where the frontend (user interface part) and backend (server part) of our application could not connect properly due to a proxy error. This connection issue was critical because it prevented the two main parts of the application from talking to each other smoothly.

To fix this problem, we set up the Vite.js server proxy to automatically forward requests from the frontend to the backend server. Configuring the proxy in this way solved the proxy error, allowing the frontend and backend to establish a stable and efficient connection. With this solution in place, the two parts of the application could communicate seamlessly, ensuring the application worked correctly overall.

### *C. Effective Prompt Engineering:*

To tailor our language model's responses for the specific use case of automated grading, we employed effective prompt engineering techniques. This optimization was crucial in ensuring that the model's responses were both accurate and contextually appropriate for the grading tasks.

These challenges, while significant, provided our team with valuable learning experiences and led to a deeper understanding of advanced machine learning model integration and system architecture optimization. Through innovative solutions and persistent effort, we were able to overcome these obstacles and successfully implement a robust application.

## VIII. RESULT

We have attached the grading results for the student code submissions, generated by our large language model (LLM) based automated code grading system. These results showcase the system's capabilities in accurately evaluating assignments across various programming languages and complexity levels, facilitated by the LLM's advanced natural language processing and context understanding abilities. The attached files provide insightful feedback and scoring, enabling students to identify areas for improvement while streamlining the grading process for educators.

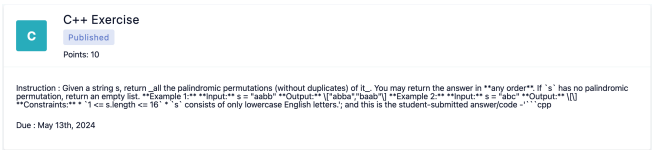


*Figure 11 Assignment Requirement*

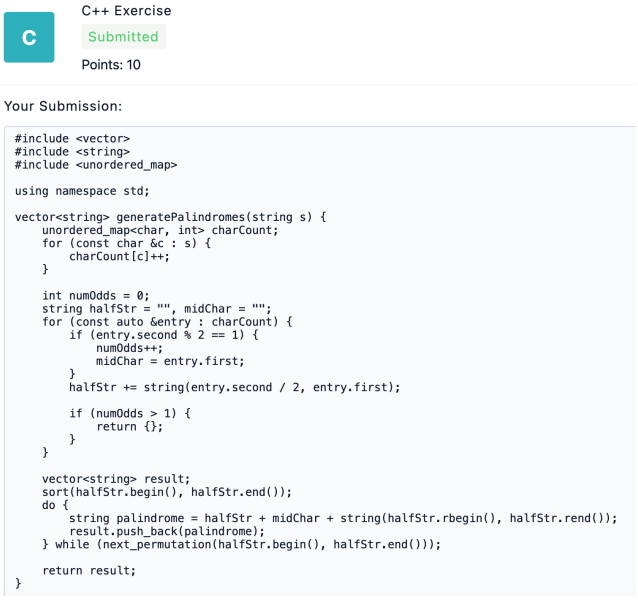


*Figure 12 Student Submission*

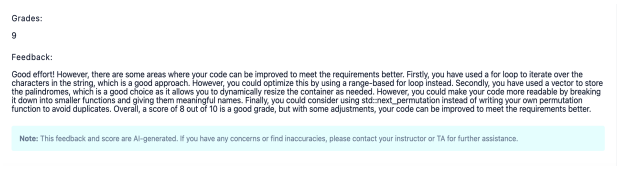


*Figure 13 Feedback and score generated by CodeGrader*

## IX. FUTURE WORK

*A.* Dynamic Prompt Engineering based on Grading Rubrics from Instructors.

*B.* File Submission Interface for Student Assignments.

*C.* OAuth Integration for Secure Authentication.

*D.* Plagiarism Detection and Similarity Checking.

*E.* Conversational User Interface for Student-Instructor Interaction.

*F.* Configurable Assignment Attempt Limits by Professors.

*G.* User Experience (UX) and User Interface (UI) Enhancements.

*H.* The system can grade assignments across multiple programming languages and their different versions, allowing instructors to specify the language version for grading to ensure assignments are evaluated against the correct language syntax, libraries, and features.

*I.* The system facilitates open communication by automatically sending email notifications to teaching assistants and professors when a student raises concerns or requires clarification regarding the grading or feedback provided for their assignment.

## X. CONCLUSION

Our AI-powered grading system revolutionizes educational assessment by leveraging cutting-edge artificial intelligence technologies. It provides fast, fair, and insightful evaluations of student assignments, offering detailed constructive feedback to facilitate learning. The system streamlines workflows for educators, automating the grading process and allowing them to focus on other critical aspects of teaching.

Through seamless integration of large language models and meticulously designed algorithms, our system offers an unparalleled level of accuracy and consistency in assessing student assignments across a wide range of subjects and complexity levels. The ability to provide detailed, constructive feedback empowers students to identify areas for improvement and fosters a more engaging and personalized learning experience.

The successful implementation demonstrates AI's profound impact on educational practices, paving the way for continuous innovation in assessment methodologies, adaptive learning, and personalized educational experiences. As we refine and expand the system's capabilities, it empowers students, educators, and institutions to embrace AI's transformative potential in shaping the future of learning.

## XI. REFERENCES


[1] Review of recent systems for automatic assessment of programming assignments. [ [https://www.researchgate.net/publication/216714976_Review_of_recent_systems_for_automatic_assessment_of_programming_assignments]

[2] Better Context Makes Better Code Language Models: A Case Study on Function Call Argument Completion [https://arxiv.org/abs/2306.00381]

[3] Introducing Code Llama, a state-of-the-art large language model for codeing [https://ai.meta.com/blog/code-llama-large-language-model-coding/]

[4] AI Grader Benefits of AI in automated assessment [https://www.kangaroos.ai/AI-GRADER/]

[5] Get up and running with large language models locally [https://github.com/ollama/ollama]

[6] **https://kth.diva-portal.org/smash/get/diva2:1779792/FULLTEXT01.pdf**

[7] Stribling, D., Xia, Y., Amer, M. K., Graim, K. S., Mulligan, C. J., & Renne, R. (2024). The model student: GPT-4 performance on graduate biomedical science exams. *Scientific reports*, *14*(1), 5670. [https://doi.org/10.1038/s41598-024-55568-7]

[8] Enkelejda Kasneci, Kathrin Sessler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, Stephan Krusche, Gitta Kutyniok, Tilman Michaeli, Claudia Nerdel, Jürgen Pfeffer, Oleksandra Poquet, Michael Sailer, Albrecht Schmidt, Tina Seidel, Matthias Stadler, Jochen Weller, Jochen Kuhn, Gjergji Kasneci,ChatGPT for good? On opportunities and challenges of large language models for education, Learning and Individual Differences,Volume 103, 2023, 102274, ISSN 1041-6080, https://doi.org/10.1016/j.lindif.2023.102274. [https://www.sciencedirect.com/science/article/pii/S1041608023000195]

[9] Hengzhi Pei et al. "Better context makes better code language models: A case study on function call argument completion". In: AAAI 2023. 2023. [https : / / www . amazon . science / publications / better-context-makes-better-code-language-modelsa-case-study-on-function-call-argument-completion.]

[10] https://billtcheng2013.medium.com/codellama-fine-tuning-7d40e6ad33b4

[11] R. Gao, N. Thomas and A. Srinivasa, "Work in Progress: Large Language Model Based Automatic Grading Study," *2023 IEEE Frontiers in Education Conference (FIE)*, College Station, TX, USA, 2023, pp. 1-4,doi:10.1109/FIE58773.2023.10343006. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10343006&isnumber=10342889

[12] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models. In Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1 (ICER '22), Vol. 1. Association for Computing Machinery, New York, NY, USA, 27–43. [https://doi.org/10.1145/3501385.3543957]

[13] Aldriye, Hussam & Alkhalaf, Asma & Alkhalaf, Muath. (2019). Automated Grading Systems for Programming Assignments: A Literature Review. International Journal of Advanced Computer Science and Applications.10.10.14569/IJACSA.2019.0100328. [https://www.researchgate.net/publication/332137224_Automated_Grading_Systems_for_Programming_Assignments_A_Literature_Review]

[14] https://ai.meta.com/blog/code-llama-large-language-model-coding/