VRIJE
UNIVERSITEIT
BRUSSEL

Master thesis submitted in partial fulfilment of the requirements for the degree of
Master of Science in Applied Informatics

CODEELEVATE: A NEW PARADIGM IN PROGRAMMING EDUCATION

Utilising Large Language Models for Personalised Coding Exercises, Feedback, and Chatbot Assistance

Mishkat Haider Chowdhury

2023-2024

Promotor(s): Prof. Dr. Beat Signer
Advisor: Yoshi Malaise

Science and Bio-Engineering Sciences

VRIJE
UNIVERSITEIT
BRUSSEL

Masterproef ingediend ter gedeeltelijke vervulling van de vereisten voor het behalen van de graad van Master of Science in Toegepaste Informatica

CODEELEVATE: EEN NIEUW PARADIGMA IN PROGRAMMEERONDERWIJS

Gebruik van Grote Taalmodellen voor Gepersonaliseerde Programmeeroefeningen, Feedback en Chatbot-ondersteuning

Mishkat Haider Chowdhury

2023-2024

Promotor(s): Prof. Dr. Beat Signer
Advisor: Yoshi Malaise

Science and Bio-Engineering Sciences

Abstract

Large Language Models (LLMs) have become an essential part of various domains, including natural language understanding, content creation, education, and research. Its capability to generate human-like text has resulted in more spontaneous interaction with humans. Before the advent of LLMs, programming education relied on traditional approaches like textbooks, manuals, classroom instructions, online tutorials, documentation, etc. The core questions driving this research are "Can Large Language Models effectively enhance programming education? Can LLMs be used to complement and enhance the pedagogical approaches of traditional education?". To answer these questions, we conducted this research to understand how AI models can be utilised to generate programming education content on which we mainly focused on generating exercises. We have integrated OpenAI's GPT-4 model into a web-based application that generates programming exercises based on the user's difficulty level and topic preferences. It not only generates exercises but also checks the correctness of the exercise answers through the LLM model. Additionally, we have added a chatbot that provides guidance while answering the generated question. While developing this app, we considered the various Usability Goals which include effectiveness, efficiency, utility, learnability, etc. We followed the Design Science Research Methodology (DSRM) [2] to develop the application. After establishing requirements, we went through different design alternatives and finally came to our final product. Additionally, to evaluate the effectiveness of the app we have conducted user studies, collecting both qualitative and quantitative data through interviews and questionnaires which provided us with necessary feedback. Finally, the findings of this research along with the results indicate that Large Language Models can significantly enhance the learning experience in the programming education sector.

Acknowledgements

I would like to express my gratitude to Prof. Dr. Beat Signer for allowing me to conduct my thesis under him and for putting his trust in me to carry out this research. I also want to thank Yoshi Malaise, my advisor, for his constant support and assistance throughout the journey. Finally, I would like to thank those who helped me with the evaluation.

Contents

Chapter 1

Introduction

1.1 Motivation

Programming education is a crucial component in the field of Computer Science and other related fields. Traditional methods of programming education, such as school education, online education, community education, and self-directed learning have been widely utilised. However, these methods at times face challenges such as inadequate teaching resources, and evaluation of teaching outcomes. Large Language Models (LLMs) can address these challenges by providing personalised learning, offering access to a broader range of educational materials and many more, driving research into the evolving landscape of LLMs in education [7].

Integrating LLMs in education can improve the education system through personalised education, reduce long-delayed tasks, provide multilingual support, and many more [1]. The use of LLMs can greatly benefit programming education which can also enable a more flexible and responsive approach to education than the traditional ways.

Research about computing education has been going on for more than 50 years [9]. However, the integration of LLMs in programming education is still in its early stage due to the recent advances in LLM technology [25]. The capability of LLMs to provide a more personalised learning experience is immense. For example, they can enhance programming education by providing immediate coding feedback [17].

Overall, it can be stated that using LLMs for programming education will not only have an impact on the students but also on the teachers. It can enhance the learning activities of students by making it more interactive. Also, it will lead to more retention of conceptual materials among students. For educators, it can help as a very powerful tool by providing an extensive amount of resources, aiding in the generation of diverse examples and instant feedback [19].

In short, the main objective of this study is to improve programming education by providing a more personalised and interactive learning experience that will be beneficial to all.

1.2 Problem Statement

LLMs have notable potential in programming education, yet there is a gap between their capabilities and their actual use in programming education [7]. One of the major problems is the scarcity of empirical research in this field. This study mainly aims to enhance programming education by utilising Large Language Models (LLMs).

There is a significant scope for this problem as it will positively affect both educators and learners, perhaps enhancing the future of the technology industry. The main research questions motivating this research are: How can LLMs effectively enhance programming education? And, how can LLMs be used to complement and enhance the pedagogical approaches of traditional education?

1.3 Contribution

The significance of conducting this research about integrating LLMs in programming education mainly lies in its capability to improve programming education. At the same time, it will open more paths of learning experience for the learners [31].

The impact of this research is beyond the classroom. With the rapid technological advancement, the demand for skilled programmers is rapidly increasing [10]. As a result, the need for effective learning has become important as well.

Moreover, this study is time-relevant in the context of the rapid growth of technological advancement. According to Lyu et al., [19], integrating LLMs into programming education can improve student learning by providing immediate feedback and generating diverse examples. This will help the students in mastering syntax and problem-solving skills. This approach can ensure that the students can be equipped with the relevant skills for the future.

1.4 Methodology

Our methodology follows the Design Science Research Methodology (DSRM) [2], which includes 6 steps that are as below:

Problem Identification and Motivation: The initial phase involved understanding the needs of the programmers. We went through several research papers to understand the common challenges in learning programming. Also, we gathered information from different research papers on integrating LLMs in education.

Define the Objectives for a Solution: Based on the problems we identified from the previous step, we defined the objectives for the solution. The goal includes improving programming education by integrating large language models (LLMs).

Design and Development: We developed multiple prototypes with different functionalities

based on the requirements. We evaluated each prototype to find out the most effective one, leading to the final development phase where we continued with the final selected prototype CodeElevate.

Demonstration: The use of the artifact, CodeElevate was demonstrated to solve instances of the problem which includes, generating personalised exercises and providing immediate feedback to the user-submitted solutions.

Evaluation: The evaluation of CodeElevate was conducted in a mixed method. We have gathered both qualitative and quantitative data to assess their effectiveness. We selected common programming misconceptions from Progmiscon.org [4], generated an exercise from CodeElevate and intentionally gave a wrong answer with a misconception to determine if the application can identify those. Moreover, we did user studies through interviews followed by filling up a questionnaire to gather more feedback.

Communication: The findings of CodeElevate were communicated with relevant stakeholders through detailed documentation and presentation. The goal for this part was to ensure that all aspects of the problems are covered.

1.5 Thesis Structure

The following is the orientation of all the chapters.

- **Chapter 1: Introduction -** This chapter provides an overview of Motivation, Problem Statement and Significance.
- **Chapter 2: Background and Related Work -** This chapter reviews existing research papers mostly related to computer science education with LLMs, their potential and challenges.
- **Chapter 3: Explorative Prototypes -** This chapter discusses the development of all four prototypes, their strengths and limitations. The last section of this chapter discusses the selection of the final prototype.
- **Chapter 4: Final Product: Code Elevate -** This chapter describes the final product, its User Interface, Data Flow and how a typical interaction works.
- **Chapter 5: Evaluation -** This chapter presents the evaluation of CodeElevate, including gathering qualitative and quantitative data collected from user studies to discuss the findings regarding its effectiveness.
- **Chapter 6: Discussion & Future Work -** This chapter discusses the main outcomes of this thesis, challenges and limitations, and future directions.
- **Chapter 7: Conclusion -** This chapter summarises the thesis, including related work, solution, and evaluation.

Chapter 2

Background & Related Work

2.1 Computer Science Education with Large Language Models

Integrating Large Language Models(LLMs) in programming education has opened doors to facilitate the generation of programming education content. On top of that, LLMs such as OpenAI's Codex can analyze the code written by students and provide feedback on where and how the code can be improved [29]. This section discusses the capabilities and implications of generating content with LLMs, highlighting key advancements and challenges.

2.1.1 Generating Educational Content through LLMs

LLMs like GPT-3 and Codex can produce different kinds of learning materials including programming exercises, code explanations and comprehensive learning modules. Their ability to generate natural language allows for the dynamic creation of challenging educational content, which will aid in personal learning experiences [29, 20].

Types of Content Generated

LLMs such as GPT-3 and Codex have significantly enhanced the ability to generate educational content. For example, the work by Sarsa et al. [29] shows us the use of Codex which creates programming exercises (including examples of answers and test cases) and code explanations. This might help students with continuous learning, enabling them to understand and rectify their mistakes. However, the study also observed that the programming exercises were not fully in a state where one can add them to a course.

Moreover, MacNeil et al. [21] in their work demonstrates how LLMs can break down complex

programming problems into understandable and easier steps. The models provide more detailed and step-by-step explanations that help beginners grasp difficult concepts easily. By explaining the logic behind code and making it easier, LLMs make programming education more understandable to learners of different levels.

Benefits of LLM-generated Content

Firstly, educators will be able to create a large amount of educational content through LLMs, which will allow them to engage more with the students rather than on a time-intensive process of content creation. As highlighted by Sarsa et al. [29] LLMs help in the rapid production of educational content of high quality. This efficiency supports a large number of students across a broad range of topics.

Secondly, MacNeil et al. [20] illustrated how LLMs like GPT-3 can generate programming assignments and explanations that align with the student's expertise level.

Thirdly, LLMs can create a variety of educational content starting from programming exercises to comprehensive tutorials and interactive learning to many more [20, 22]. MacNeil et al. [22] demonstrated the potential of LLMs in generating different kinds of programming content such as analysing time complexity, identifying common mistakes by beginner programmers and summarising code.

2.1.2 Evaluating LLM-generated Educational Content

Evaluating the content generated by LLMs is the only way to measure if it meets the user's needs. It further ensures that it enhances the education system and fulfils the education standard.

Methodological Approach

The evaluation of LLM-generated content is typically done in both qualitative and quantitative methods to understand its educational impacts.

Jury et al. [12] employed expert assessments and conducted a large-scale user study involving 400 students as participants to examine the effectiveness of LLMs in an introductory programming course. Their approach included examining the clarity of the LLM-generated explanations and how these explanations can help students have a better learning experience.

MacNeil et al. [21] integrated LLM-generated content on an E-Book related to web software development. They added interactive buttons for the students with different functionalities such as line-by-line code explanations, high-level summaries etc. Then they evaluated by tracking how students interacted with the E-Book, mainly focusing on how often they accessed LLM-generated explanations. Also, they collected data and analyzed which types of explanations the users were accessing more.

Chen et al. [3] examined the code generated by LLMs which are trained on code. They conducted automated testing by generating code and testing if the models generated expected outcomes. They also conducted statistical analysis and evaluated the generated codes by humans.

Findings

The integration of LLMs in educational settings has shown a very positive impact in enhancing the learning experience. Jury et al. [12] found out from their evaluation that LLMs are effective in generating worked examples which reduces the cognitive load and makes the learning of programming education easier. Also, MacNeil et al. [21] reported that they found very positive feedback from the students about the E-Book that contains LLM-generated explanations.

However, these outcomes are not without any challenges. Jury et al. [12] stated that while LLM-generated content facilitates learning, the quality of the generated content can vary and it cannot be stated that the LLM-generated content is always fully accurate. Also, Chen et al. [3] highlighted that the success rate of the LLM-generated code snippets varied and at times it needed further refinement to ensure correctness. They encourage the need for more validation of LLM outputs to prevent incorrect generated content among students.

2.2 LLM-based Programming Assistant Tool

This section discusses the integration of LLMs in programming assistant tools and their outcome. These tools enhance traditional methods by providing real-time programming assistance.

2.2.1 CodeHelp

CodeHelp is an LLM-based programming assistant tool that uses guardrails to prevent students from directly getting the coding solution. Instead, it guides them so that they can independently solve the problem with its assistance without receiving the full solution [18].

Interaction with CodeHelp

The User Interface of CodeHelp looks like this as shown in Figure 2.1. During a session with CodeHelp, the student input the problem they are facing with their code. This input includes the language of the code, the code, the error message of the code and what problem exactly are they facing. CodeHelp processes this information and responds with guidance and hints to solve the problem [18].

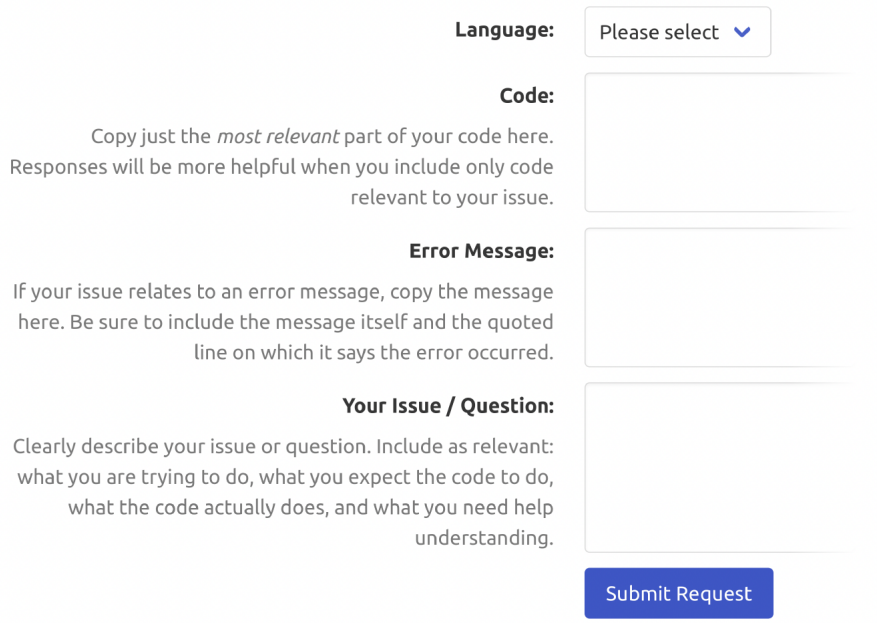


Figure 2.1: User Interface of CodeHelp [18]

Evaluation & Results

Data was collected over 12 weeks from 52 students. The evaluation was done by tracking the student's interaction with the tool. Also, feedbacks were taken from the instructor and the teaching assistant.

The students were noticed to have a sustainable engagement throughout the 12 weeks. Students appreciated the help as it provided additional support to them to solve programming issues. Instructors also found this tool helpful as it was easy to deploy and integrate into their teaching. It served as a valuable addition to their traditional teaching method [18].

2.2.2 CodeAid

CodeAid is a programming assistant tool that provides personalised feedback without providing the actual solution. It helps students with a step-by-step problem-solving process to help them become conceptually strong which facilitates in deeper understanding of the concept. [13].

Interaction With CodeAid

The User Interface of CodeAid looks like this as shown in Figure 2.2. Students interact with CodeAid with a structured interface where they input their programming queries including di-

rect questions about programming concepts or particular code issues. The tool does not give direct code solutions, instead it provides pseudocode of the coding solution or conceptual explanations [13].

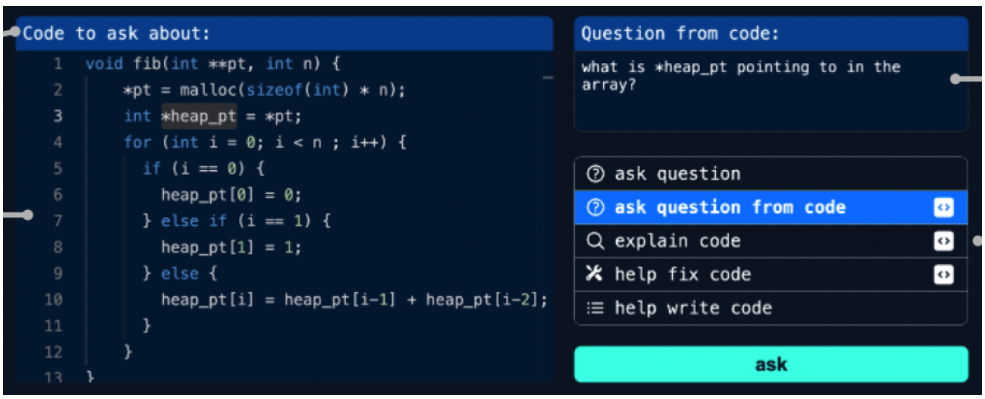


Figure 2.2: User Interface of CodeAid [13]

Evaluation & Results

The evaluation was conducted by deploying this tool in a large introductory programming course where it was used by around 700 students, generating over 8000 interactions. The evaluation method included analysis of student interaction with the tool, weekly surveys, and structured interviews with both students and instructors [13].

The result indicated that CodeAid significantly improved the problem-solving skills of the students along with their conceptual understanding. The students reported high-level satisfaction with the tool for its ability to aid learning. Instructors were also satisfied with its ability to offer personalized help that could be scaled up, making it easier to manage large classes.

2.2.3 Similarities and Dissimilarities

While both tools have similarities in not providing the actual solution, they also have their key differences which are mainly in their approach and focus. CodeHelp focuses on improving problem-solving skills by providing guidance that will help them find solutions on their own without directly providing the solution. This involves providing feedback to students on how to approach and solve the problem independently [18].

Conversely, CodeAid focuses on improving students' conceptual knowledge and code comprehension. Instead of providing solutions or guiding them towards solutions, CodeAid provides detailed explanations to help student understand the underlying concepts of their code. This involves providing pseudo-codes with step-by-step explanations and suggesting corrections without providing the actual code [13].

2.3 LLM-based Programming Feedback

Integrating LLMs in programming education brings a significant enhancement which leads to more detailed and personalised feedback. These tools help students to understand the error well, get tips to solve them and receive guidance on what to do next. This section discusses the ways these technologies can make programming education better.

2.3.1 Comprehensive Enhancement of Feedback Mechanisms

LLMs have greatly improved the programming error feedback. Sometimes the errors that the students encounter are not understandable and hard to read. Instead of confusing errors, LLMs provide easy and understandable explanations of the errors which the students find easy to understand. For instance, Codex was able to explain the error message 57% of the time and these error messages were considered better than the original error message for more than half of the cases [16]. This suggests that the error feedback by LLM is more user-friendly to the students than the traditional error messages.

Kiesler et al. [14] have conducted research on how LLMs like ChatGPT respond to students' programming questions and they found out these models can generate various types of feedback such as explanations, suggestions and also code completion.

Moreover, LLMs provide "next step" hints to students so that they are not stuck with a problem and can think effectively and move forward with their problem. Roest et al. [28] described how these hints can help students to continue their work when they are stuck. The generation of the next-step hint can be achieved without any historical student data or models but with the extensive dataset of LLMs.

2.3.2 Impact

Using LLMs in programming education has shown a significant amount of benefits for students. According to Leinonen et al. [16], LLMs improve the programming error messages and make them more understandable which reduces the frustration of students and improves their learning outcomes. Kiesler et al. [14] in their research observed that LLMs like ChatGPT can generate formative feedback on programming tasks like explanations, and corrections which helps students to understand their mistakes and learn from it. Furthermore, Roest et al. [28] found in their research that students using LLMs for feedback on their code showed improvement in their programming skills and problem-solving abilities. This kind of assistance makes learning less complicated.

2.3.3 Challenges

With the integration of LLMs in programming education, there are many benefits, but there are still many challenges as well that need to be addressed.

In the research of Kiesler et al. [14] they found that 61 out of 99 generated responses of LLMs were incorrect or unhelpful. This could mislead novice programmers. This suggests that while LLMs can provide valuable feedback, their response is not fully reliable and needs more validation. To mitigate this issue, educators should guide the students on how to effectively use and cross-check the feedback generated by LLM.

Leinonen et al. [16] highlighted that although LLMs can provide valuable feedback, they often struggle with complex errors, providing fixes that are correct only half of the time. For example, Codex explanations were considered an improvement over original error messages only 54% of the time. This inconsistency will affect novice programmers badly who will depend on Codex for error explanations.

2.4 Comparative Analysis of ChatGPT's Role in Programming Education

This section does a thorough comparative analysis of chatGPT with both students and teachers. Also, it examines the effectiveness of LLM-generated content against human-generated content.

2.4.1 ChatGPT vs. Human

Quality of Educational Content

Comparing the quality of the code explanations generated by LLMs and student-generated explanations reveals various findings. Leinonen et al. [15] found in their research that in terms of accuracy and understandability, the LLM-generated code explanations are better than those of students.

Accuracy in Source Code Generation

Studies such as those by Denny et al. [6] suggest that in comparison between LLM-generated content and human-generated content, AI can match the accuracy of human-generated content. In their study, Denny et al. [6] found that the median scores for code correctness, explanation correctness and helpfulness were both raised 4 out of 5 for the student-generated content and AI-generated content.

However, the advantage of AI lies in its ability to produce a greater volume of resources quickly and consistently. The study highlighted that the AI-generated content tends to be more concise, with the median length of AI-generated code examples being 224 characters compared to 377 characters for student-generated examples.

2.4.2 The Role of Teachers and ChatGPT

Comparing ChatGPT and Teachers

The evolution of ChatGPT shows that it can outperform some aspects of teaching but not all. It can be used as a helping tool but not as a substitute. GPT-4 at times perform similarly or better than human educators in terms of creating programming content but in most cases, the human educators outperform ChatGPT [27].

Enhancing Teaching Roles

ChatGPT can assist teachers in various ways. A few notable ways are by creating educational content, providing supplementary resources for teaching, etc. This support can help the teachers to be more focused on direct instruction and student interaction which will overall improve the learning system [11].

2.4.3 Challenges

Although AI is revolutionising, there still exist different challenges and limitations. One of the major issues is whether AI can create quality and reliable content like teachers. At times, AI might not understand the depth of the problem properly and give an oversimplified or wrong answer to students which could mislead students [6]. Another concern is about bias - AI might sometimes show biases in their result based on the data they are trained on which can also mislead students [23]. Furthermore, relying too much on AI might make students less capable of solving problems on their own which will not be suitable for the long run [15].

Chapter 3

Explorative Prototypes

In this chapter, we present the development of four exploratory prototypes designed to enhance programming education using Large Language Models (LLMs). Each prototype uses the capabilities of LLMs in different ways to support learning such as generating exercises, explaining code snippets, reviewing code etc. For every prototype, we discussed the user interface, functionalities, API integrations, data flows, evaluation, advantages and disadvantages. Finally, by analyzing the pros, cons and evaluation results, we select the most promising prototype for further development.

3.1 Debugger Lens Prototype

The Debugger Lens web application is designed to help understand JavaScript code by breaking it into different parts and getting LLM-generated explanations for each part. This section discusses the User Interface of the project, its functionalities and its high-level architecture and advantages-disadvantages. The inspiration for this prototype came from the research of Sarsa et al. [29] and MacNeil et al. [20] where they demonstrated the advancements of using LLMs in education. Sarsa et al. demonstrated the use of LLMs for code explanations which we took as an influence and integrated into our application. Also, MacNeil et al. highlighted the usefulness of LLMs in breaking down complex programming problems into understandable steps from which we got influenced to break code into steps and use LLMs for explanation. These studies have helped us with the scientific foundation for the use of LLM in an improved code understanding and debugging, which helped us develop our prototype.

3.1.1 User Interface

The front-end is built using HTML, CSS and JavaScript creating a responsive interface which is seen in 3.1. The User Interface includes:

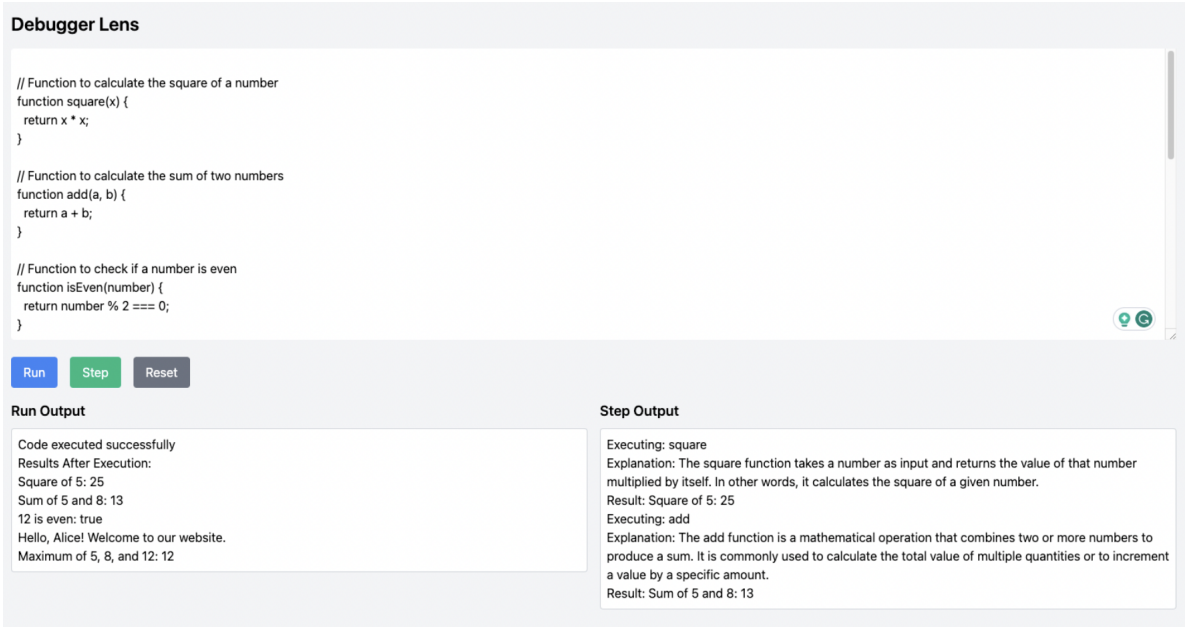


Figure 3.1: User Interface of Debugger Lens

- A text area to input JavaScript code and
- Three Buttons which are **Run, Step** and **Reset** for different functionalities.

3.1.2 Functionalities

- The "Run" button is used to execute the code which displays the output of the code in the "Run Output" section.
- The "Step" button is used to traverse through the code step by step which displays the output of that step along with the LLM-generated explanation in the "Step Output" section.
- The "Reset" button is used to clear all the execution results which then resets the whole application.

3.1.3 API Integration

OpenAI API: The application queries information from the OpenAI API during the step execution to fetch explanations about JavaScript functions. The prompt to the API query has been well crafted so that the descriptions provided are accurate and helpful. Normally, the prompt will contain a short description of the task and a specific segment of the code for which the explanation is requested.

3.1.4 Data Flow

The data flow of the Debugger Lens is shown in a sequence diagram in 3.3

Run: When the user inputs their JavaScript code and clicks the "Run" button, it immediately triggers the `runCode()` function. In the `runCode()` function, the input code is executed using the `eval()` function. If the code is executed then the execution results are displayed in the "Run Output section".

Step: When the user triggers the "Step" button, it triggers the `step()` function. It parses the input code into an Abstract Syntax Tree (AST) using the Acorn JavaScript parser. It goes through each part of the code, executes it, requests function explanation to the OpenAI API and returns the output with the LLM-generated explanation produced by the OpenAI API. The prompt used for requesting specific JavaScript functions is shown in 3.1.4.

```
messages: [
 {  role: 'system', content: "You are a helpful assistant." },
 {
    role: 'user',
    content: `Explain the following JavaScript function step-by-step
    including its purposes, how it works: ${functionName}.`
 }
]
```

Figure 3.2: Prompt for Function Explanation

Reset: When the "Reset" button is clicked, it triggers the `reset()` function which clears all the execution results.

3.1.5 Evaluation of the First Prototype

During the development of this prototype, there were several limiting factors. One major limiting factor was its debugging capability is quite basic compared to the advanced debugging tools, meaning it will not be suitable for complex debugging tasks. Additionally, this app was unsuitable for large codebases as you can only copy-paste a code snippet.

3.1.6 Advantages

- This tool helps in breaking code into different parts through debugging which helps to understand the programming construct.

- It also helps with a detailed explanation of each function.

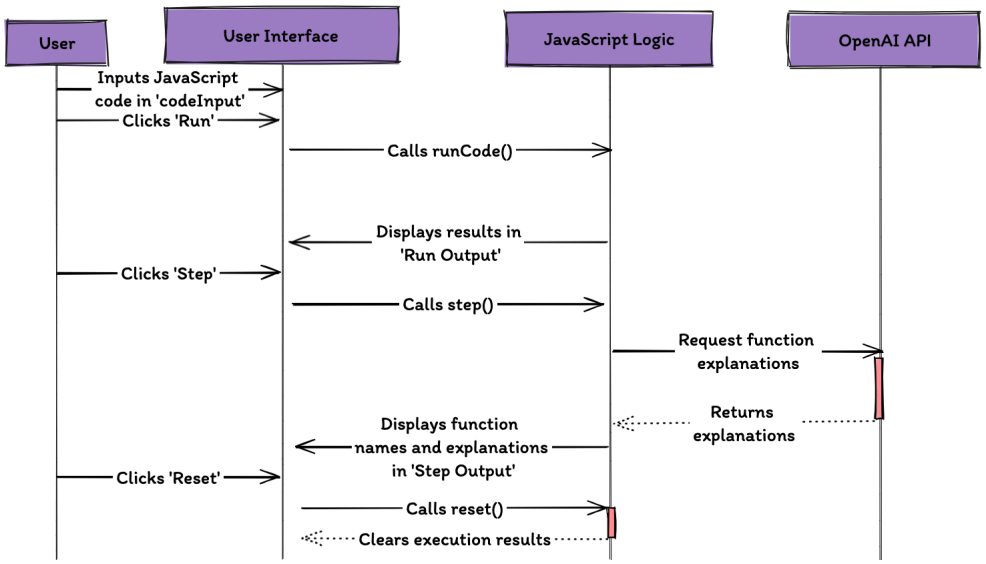


Figure 3.3: Sequence Diagram of Debugger Lens

3.1.7 Disadvantages

- It has limited language support. It only supports JavaScript.
- Its debugging capability is very basic whereas there are already better debugging tools now.
- For larger codebases, it is quite impossible to go through the code part by part.

3.2 Code Analyzer Prototype

Code Analyzer is a web application which is designed to generate exercises from code input. It supports any programming language. It uses OpenAI API to generate code-relevant programming exercises. The inspiration for building this tool came from the work of Jury et al. [12] where they generated worked examples using LLMs from question input by the user. Instead of generating worked examples, Code Analyzer focuses on generating exercises related to user-input code.

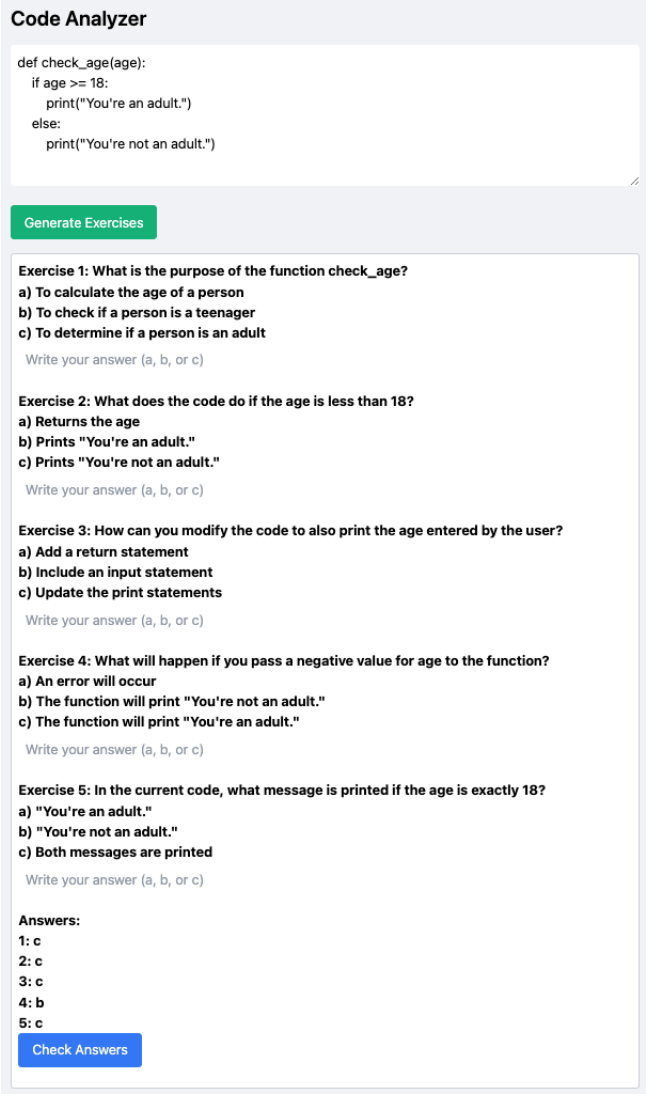


Figure 3.4: User Interface of Code Analyzer

3.2.1 User Interface

The front-end of Code Analyzer is built using HTML, CSS and JavaScript, creating a responsive interface seen in 3.4. The User Interface contains:

- An input `<textarea>` for the users to input their code.
- A "Generate Exercise" button.

3.2.2 Functionalities

- When the "Generate Exercise" button is clicked, it displays exercises based on the user's input code.
- Along with the exercises, it also displays input fields for answers where the user can write their answer.
- Finally, when the "Check Answer" button is clicked, it displays the answers in a pop-up.

3.2.3 API Integration

OpenAI API: Code Analyzer uses the OpenAI API to generate exercises. The application sends a POST request with the user's input code and retrieves a set of questions related to the input code. The prompt to the API query has been well crafted so that the exercises provided are accurate and helpful.

3.2.4 Data Flow

The data flow of the application Code Analyzer is shown in a sequence diagram in 3.6.

Generate Exercises: When the "Generate Exercise" button is clicked, the `generateExercises()` function is triggered. This function retrieves the user-input code and sends it to the `fetchExerciseFromAPI()` function which then sends a POST request to the OpenAI API with the prompt in 3.5. The API then returns with a set of exercises related to the code.

Display Explanation: The displayExplanation() function splits the LLM-generated response into 5 exercises and also includes an input field for the user to enter their answer.

Check Answers: When the user writes their answer and clicks the "Check Answers" button, it triggers the checkAnswers() function which retrieves the user's answers and compares them to the answers given by the LLM. Finally, an alert is displayed with the result of the answer being right or wrong.

```
messages: [
 {
    role: 'user',
    content: `You are a personal instructor. You will receive a
    sample code snippet: "${code}" and generate 5 exercises for the
    student to learn in depth about that code.
    Just ask the questions only. Don't add anything extra. Provide 3
    options and provide 1 answer. Format the questions like the
    following example. First you will provide 5 questions and then
    provide the answers.
    Exercise 1: Question?
    a) option 1
    b) option 2
    c) option 3

    answers:
    1: c`
 },
]
```

Figure 3.5: Prompt for Exercise Generation

3.2.5 Evaluation of the Second Prototype

Testing Procedure

To validate the reliability and accuracy of Code Analyzer we have conducted two tests which are:

- Relevance Test: In this test, a set of 30 Python code snippets was entered to check if it generates relevant questions. These snippets were chosen from a diverse range of Python programming features including class definitions, error handling, web framework usage, file handling and many more different types. Each snippet was selected to represent basic operations and common challenges typical of Python programming for a comprehensive assessment of the system's capabilities. Relevance, in this context, means that if we ask the app to generate a specific type of question, such as an if-else question, the app should indeed generate an exercise that involves if-else statements.
- Answer Checking Accuracy Test: In this test, a set of 30 answers was manually entered to check if the application can correctly recognise correct or incorrect answers.

Results

- Relevance Test: Out of 30 code snippets, it was seen that the LLM has generated 30 relevant exercises which shows an accuracy of 100%.

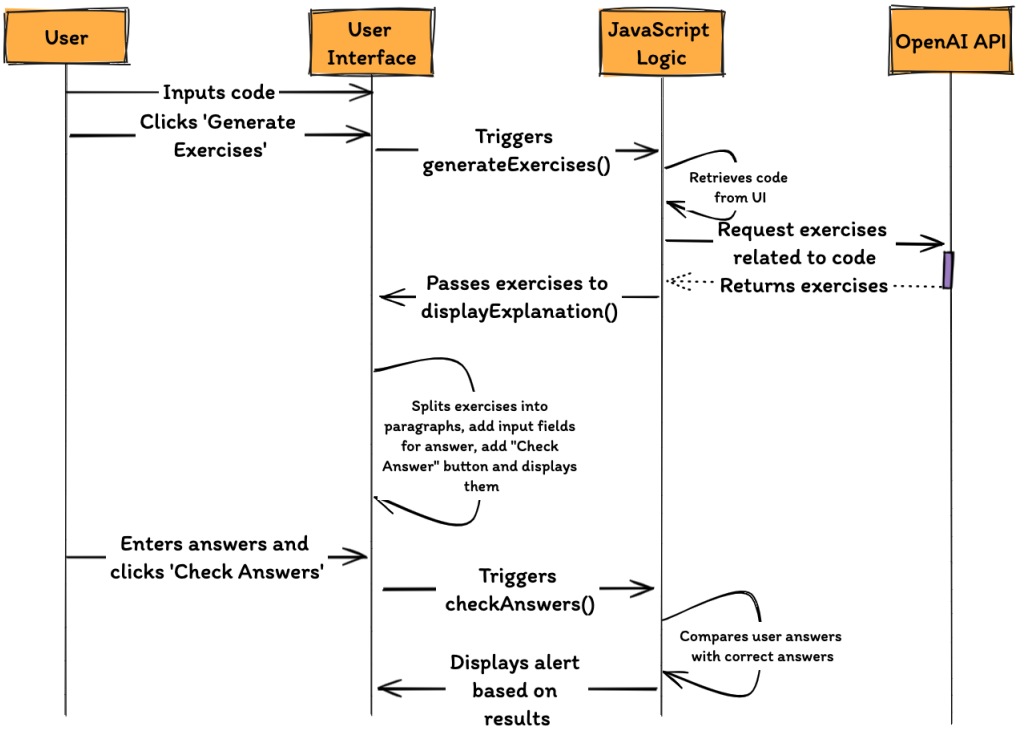


Figure 3.6: Sequence Diagram of Code Analyzer

- Answer Checking Test: Out of 30 answers, our application was able to correctly match only 14 answers which shows an accuracy of only 46.7%.

3.2.6 Strengths

- This tool is designed to generate exercises from code input in any programming language.
- This tool also promotes interactive learning by not only generating questions from the code but also providing input fields for the user to test their understanding by writing the answer and checking them against the correct solution.

3.2.7 Limitations

- The biggest disadvantage was the limitation of the answer-checking mechanism.
- Lack of detailed explanation of the code is another drawback, which limits the tool's effectiveness.

3.3 Code Review Assistant Prototype

Code Review Assistant is a web application designed to review code snippets provided by the user. It uses the OpenAI GPT-3.5-turbo model. The inspiration for developing this web application came from the work of Chen et al. [3], where they highlighted the strengths and limitations of LLMs in generating correct code and detecting errors. This influenced us to develop a web application that can detect errors in code and improve code.

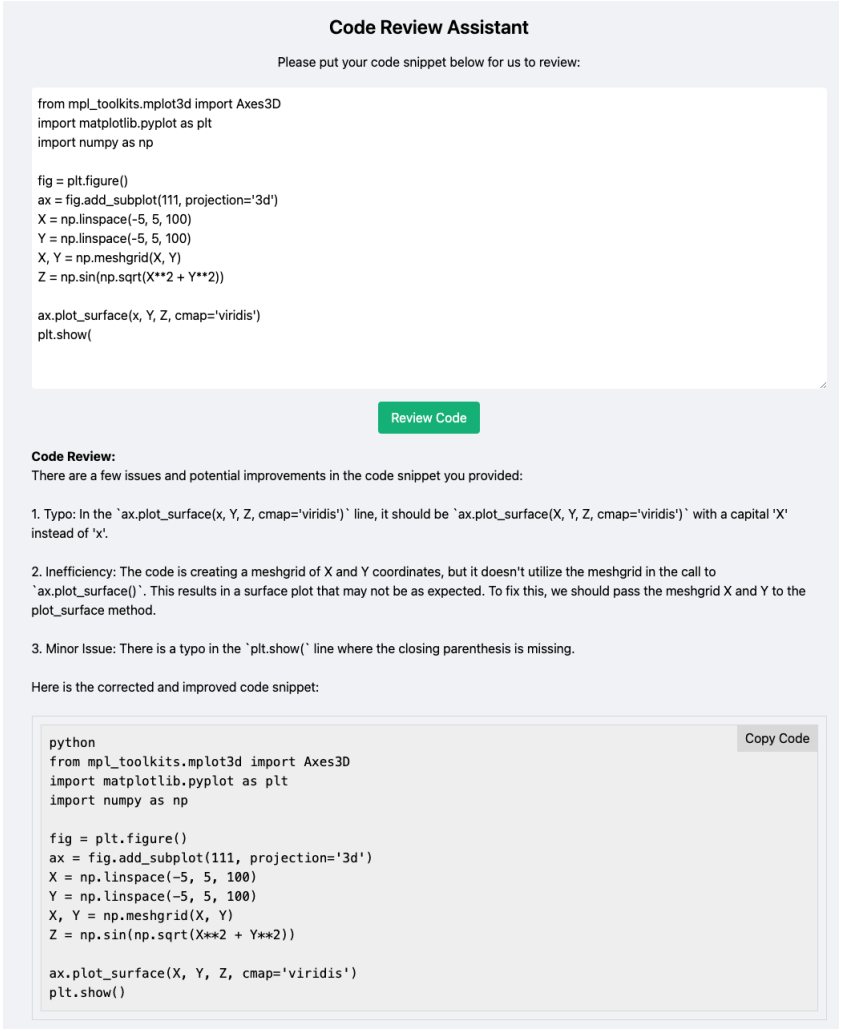


Figure 3.7: User Interface of Code Review Assistant

3.3.1 User Interface

The front-end of the application uses HTML, CSS and JavaScript, creating a responsive User Interface which is shown in 3.7. The User Interface includes:

- An input `<textarea>` for the user to input their code.
- A "Review Code" button to submit the code for review.

3.3.2 Functionalities

- When the "Review Code" button is clicked, the code review process starts where the application analyzes the code.
- Upon completion of the review process, the application displays the review.

3.3.3 API Integration

OpenAI API: Code Review Assistant uses the OpenAI API to generate code reviews. It sends a POST request with the user's entered code and receives a detailed review of the code. The prompt to the API query has been well crafted so that the responses provided are accurate and helpful.

3.3.4 Data Flow

The data flow for Code Review Assistant is shown in 3.9.

Generate Review: Upon clicking the "Review Code" button, the `generateReview()` function is triggered. This function retrieves the user's entered code and calls the `fetchReviewFromAPI()` function which then sends a POST request to OpenAI's API with the prompt in 3.8 asking to review the code.

Display Review: If the API is successful, the response is passed to the `displayReview()` function. Finally, the `displayReview()` function formats the review and displays it on the webpage.

3.3.5 Evaluation of Third Prototype

Testing Procedure

To validate the usefulness and effectiveness of Code Review Assistant, we have conducted three structured tests which are:

```
messages: [
 {
    role: 'user',
    content: `Please review the following code snippet: "${code}".
    Analyze the code for any errors or inefficiencies. If any are found,
    please explain the issue and suggest improvements. Also, provide a
    comparison between the original code and your suggested code,
    highlighting why your version is more efficient or better in terms
    of time complexity. If the code snippet is too long for a detailed
    review, please return an appropriate error message.`
 },
]
```

Figure 3.8: Prompt for Code Review

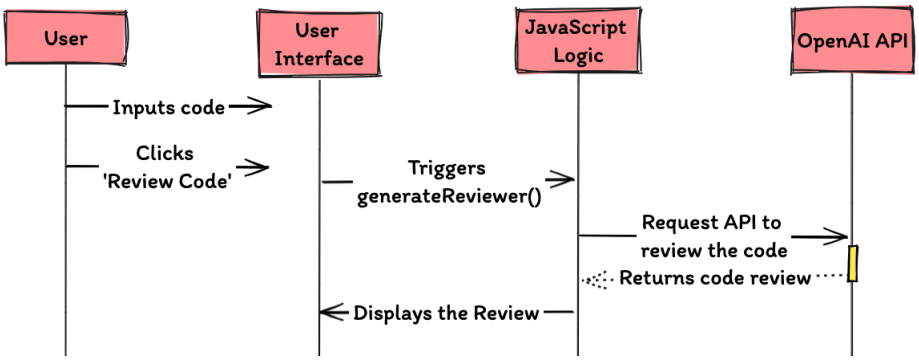


Figure 3.9: Sequence Diagram of Code Review Assitant

- Code Improvement Suggestion Test: A set of 15 different code snippets was input into the application to check if the LLM-generated explanation is relevant and beneficial.
- Error Detection Test with small code snippet: A set of 10 code snippets with intentional errors was input into the application to check if the application can successfully detect the errors.
- Error Detection test with large code snippet: A set of 5 code snippets with more than 100 lines of code was input with a lot of intentional errors to check if the application can successfully detect.

Results

- Code Improvement Suggestion Test: Out of 15 cases, it passed all 15 achieving a 100% success rate.
- Error Detection Test with small code snippet: Out of 10 cases, it also passed all 10 achieving a 100% success rate.

- Error Detection Test with large code snippet: Out of 5 cases, it failed 4 of them resulting in only a 20% success rate.

3.3.6 Strengths

- The app is very simple and easy to use with its user-friendly interface.
- It provides immediate feedback.
- It uses OpenAI's GPT-3.5-turbo model which is a powerful AI model capable of understanding code.

3.3.7 Limitations

- One main limitation here is scalability. Although it performs well in small code snippets, we saw it struggles when the codebase is large. In our test with large codebases, it showed only a 20% success rate.
- Another drawback is the single file limitation. The app is designed to only one code snippet where as in real-world scenarios, projects contain multiple interconnected files.

3.4 CodeElevate Prototype

CodeElevate is a web application designed to generate programming exercises and validate user-input answers. It uses OpenAI's GPT 4 model to create exercises based on the user's selected skill-level and topic and also to check the correctness of the answers. This prototype aims to enhance the learning experience by providing immediate feedback on the programming exercises. This prototype was influenced by the work of Sarsa et al. [29] where they highlighted the potential of LLMs in creating challenging programming exercises. Also, integrating the feedback part was influenced by the research of Kiesler et al. [14] where they explored how LLMs can be used to generate formative feedback.

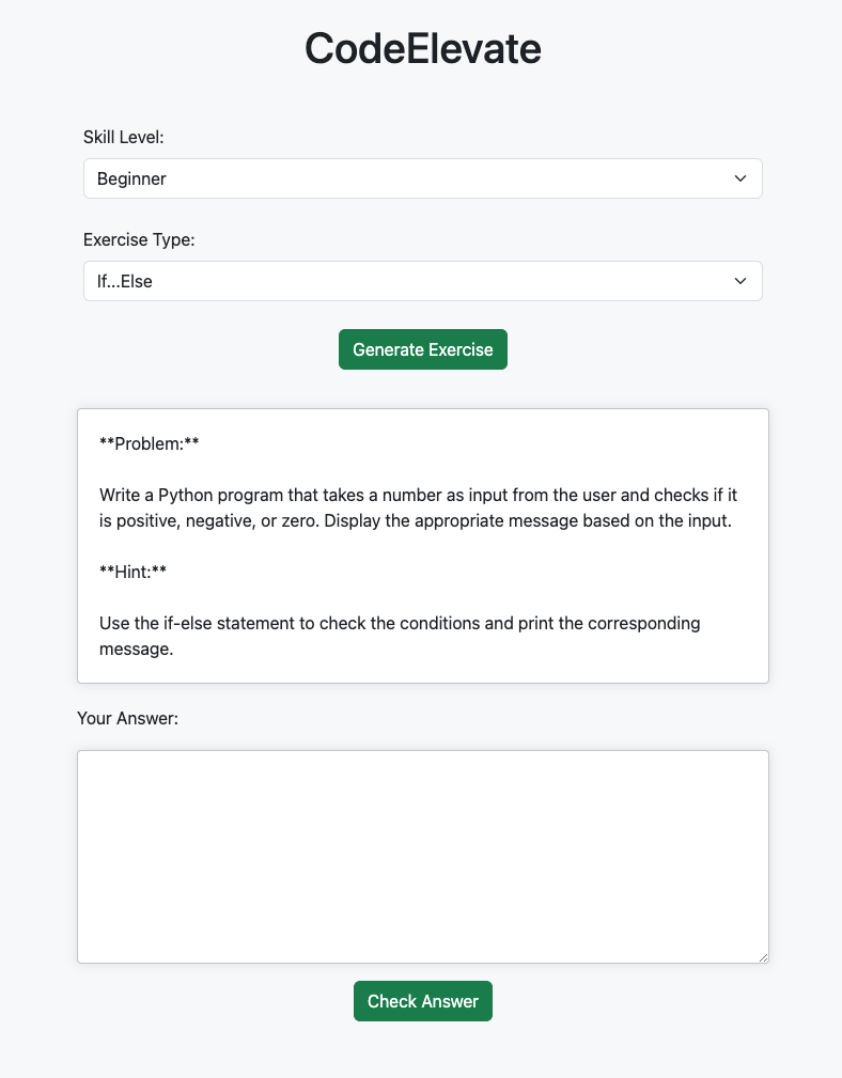


Figure 3.10: User Interface of CodeElevate

3.4.1 User Interface

The front-end of CodeElevate is a responsive interface built using HTML, CSS and JavaScript as shown in 3.10. The User Interface contains:

- A dropdown menu for selecting the difficulty level (Beginner, Intermediate, Advanced).
- A dropdown menu to select the exercise of choice (e.g., Lists, if...Else, Tuples, Functions, Arrays, Classes).
- A "Generate Exercise" button to generate exercise based on the user's preference.
- An output area with the generated exercise.
- An input `<textarea>` for the user to input their answer.
- A "Check Answer" button for the user to check the correctness of the answer.

3.4.2 Functionalities

- Upon clicking the "Generate Exercise" button CodeElevate generates an exercise based on the selected skill and exercise type.
- Upon clicking the "Check Answer" button after completing the answer, CodeElevate responds with feedback on the answer to the question. If the answer is correct, the application confirms the correctness and if the answer is wrong, the application replies with the correct answer and explanation.

3.4.3 API Integration

- **Exercise Generation:** CodeElevate uses OpenAI's GPT-4 model to generate exercises. It sends a POST request to the API with the user's chosen skill level and exercise type. The API responses with a generated exercise.
- **Answer Validation:** CodeElevate uses OpenAI's GPT-4 model to validate the answer. It sends a POST request to the API with the exercise and the user's answer. The API checks the answer and responds with feedback.

3.4.4 Data Flow

The data flow for CodeElevate is shown in 3.13

Generate Exercise: When the user clicks "Generate Exercise" the `generateExercise()` function is called. The function retrieves the user's selected skill level and exercise type and then the function `fetchExerciseFromAPI()` is called with the skill level and exercise type as arguments.

```
messages:
[
 {
    role: 'user',
    content: `Generate a ${skillLevel} level problem solving python
    programming exercise about ${exerciseType}. Do not provide any
    example or solution code. Just provide the problem statement and hint.`
 },
]
```

Figure 3.11: Prompt for Generating Exercise

Then this function sends a POST request to OpenAI's API with the prompt in 3.11 asking to generate a coding question.

Check Answer: When the user clicks the "Check Answer" button, the `fetchCorrectnessFromAPI()` function is called with the exercise and the user's answer as arguments. This function then sends a POST request to OpenAI's API with the prompt in 3.12 requesting to provide feedback on the answer to the question.

```
messages:
[
 {
    role: 'user',
    content: `The exercise presented is: "${exercise}".
    The answer submitted is: "${answer}".
    Please evaluate the correctness of this response.
    If the answer is incorrect, kindly provide the correct answer,
    detail the mistake, and supply the corrected code.
    If the answer is correct, no further action is required.`
 },
]
```

Figure 3.12: Prompt for Answer Checking

3.4.5 Strengths

- User-friendly Interface.
- Provides Exercises based on difficulty level and exercise type.
- Provides feedback immediately.

3.4.6 Limitations

- Supports only Python Programming.

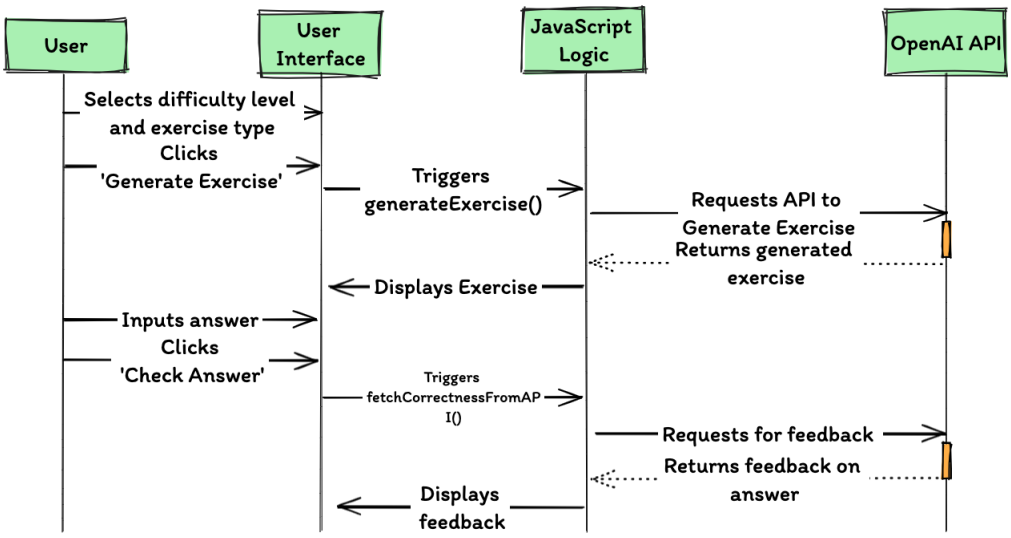


Figure 3.13: Sequence Diagram of CodeElevate

- Runs on GPT-4 model. There are newer models now

3.5 Selection of Final Prototype

In this section, we discussed the pros and cons of all four prototypes developed during the research. By understanding the strengths and limitations of each prototype we justify why we ultimately chose to focus and move forward with the fourth one which is CodeElevate.

The first prototype, Debugger Lens is a debugging web application which provides a step-by-step explanation of JavaScript code which aims to help in understanding the programming construct. However, its limitations were significant, including being restricted to simple debugging capabilities whereas there are better debugging tools now such as VSCode's debugger [24] and Chrome DevTools [8]. These tools have more functionalities than Debugger Lens. Moreover, it is only restricted to JavaScript.

The second Prototype, Code Analyzer shows great accuracy in correctly generating relevant exercises. However, its major drawback is the answer-checking mechanism. It only accepts exact matches(e.g., 'a' matches with only 'a', but if the user enters 'A', it cannot match properly). LLMs also generate the answers in different patterns at different times which causes the application to return correct answers as incorrect. It only showed 46.7% accuracy in correctly matching.

The third Prototype, Code Review Assitant's user-friendly interface and immediate feedback were notable strengths. However, it showed inconsistency in scalability, mainly with large codebases where it showed a success rate of only 20%. Also, the single code file limitation is another major

drawback. Furthermore, there are better tools that use AI for extensive code improvement and error correction like Amazon CodeGuru Reviewer [30], CodeRabbit [5], etc. These tools make the limitation of Code Review Assistant more apparent.

In contrast, the fourth Prototype, CodeElevate, shows considerable strengths, making it the most prominent candidate for further development. CodeElevate offers a tailored learning experience by offering exercises based on the user's selected difficulty and topic. It also offers immediate correctness for the user to get instant feedback from where they can learn from their mistakes.

Ultimately we chose to continue our further development with CodeElevate for the following reasons:

- **Adaptibility:** CodeElevate's ability to generate personalised exercises of various difficulty levels and topics has made it more versatile and useful compared to the other three prototypes.
- **Comprehensive Feedback:** CodeElevate's feedback mechanism contains a detailed explanation for incorrect answers that offers a more in-depth learning experience compared to the other three prototypes.
- **Integration of New Models:** Its framework is extensible to newer models like GPT-4 Turbo and GPT-4o which can generate more sophisticated exercises, and a more sophisticated answer-checking mechanism which can enhance the overall learning system.

Chapter 4

Final Product: CodeElevate

In this chapter, we present the final product, CodeElevate which is built on the strengths of the prototypes discussed in Chapter 3 and the related work as presented in Chapter 2. CodeElevate is designed to generate programming exercises utilizing OpenAI's GPT-4 model. It generates exercises based on the user's preferred difficulty level and topic. Also, it aims to enhance the learning experience by providing immediate feedback on the programming exercises. Additionally, we have integrated a chatbot feature to assist the user with programming without providing the actual solution. Our inspiration for this chatbot feature came from the work of Liffiton et al. [18] which we discussed in chapter 2 on CodeHelp, where a similar approach was used like providing guidance instead of the solution.

4.1 User Interface

The front-end of CodeElevate is a responsive interface built using HTML, CSS and JavaScript as shown in 4.1. The User Interface contains:

- A dropdown menu for selecting the difficulty level (Beginner, Intermediate, Advanced).
- A dropdown menu to select the exercise of choice (e.g., Lists, if...Else, Tuples, Functions, Arrays, Classes).
- A "Generate Exercise" button to generate exercise based on the user's preference.
- An output area with the generated exercise.
- An input textarea for the user to input their answer.
- A "Check Answer" button for the user to check the correctness of the answer.
- A "Send" button in the chatbot interface to send messages to the chatbot.
- A "Minimize Chatbot" button to minimize the chatbot.

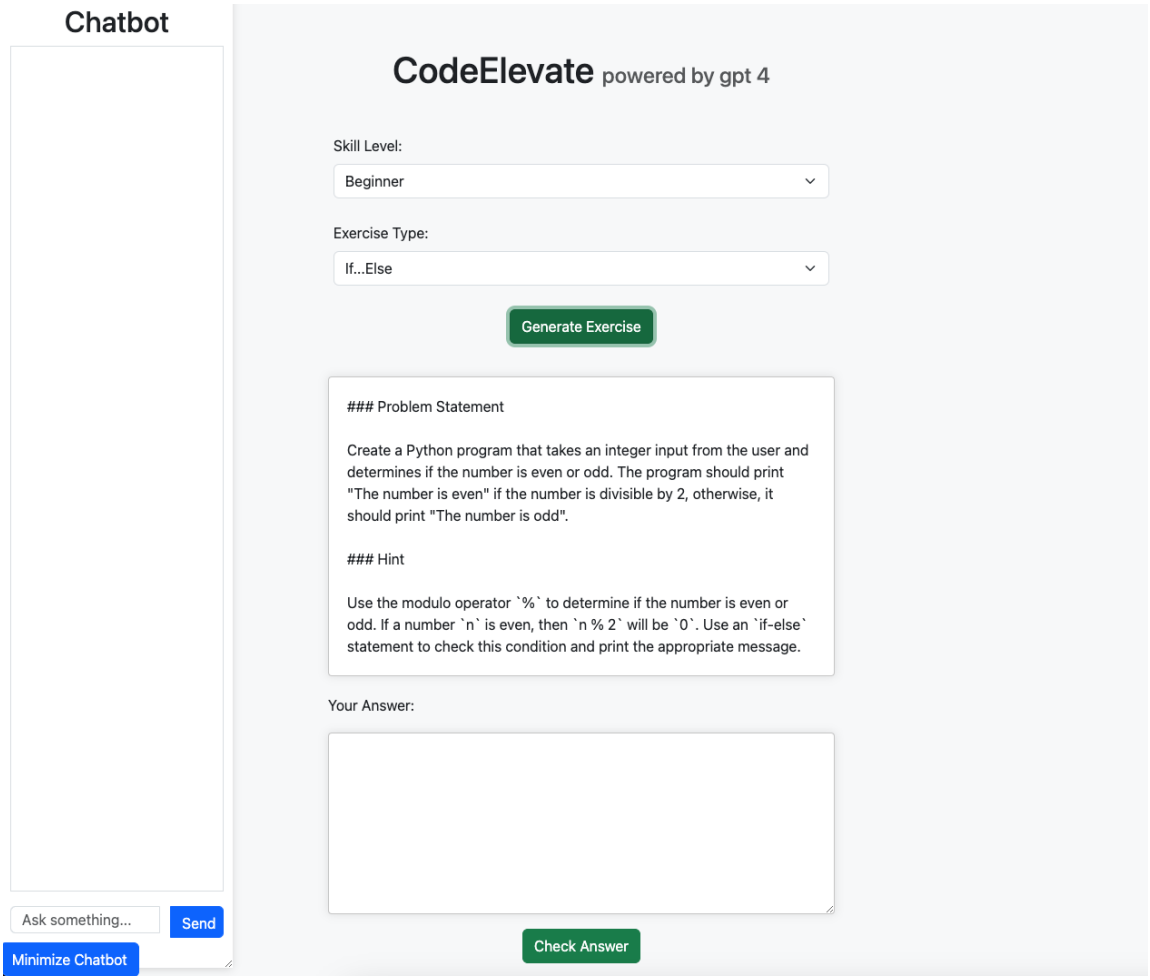


Figure 4.1: User Interface of CodeElevate with Chatbot

4.2 Chatbot Integration

The chatbot assists users by providing hints and syntax help without providing an actual answer to the problem. This feature enhances the learning experience by guiding users to find the answer on their own.

4.2.1 API Integration

The chatbot feature uses OpenAI's GPT-4 model to respond to the user's programming questions. It sends a POST request to the API with the question asked by the user and the programming question so that the chatbot is aware of the question. The API responds with an answer to the user's question.

4.3 Data Flow

The data flow for the updated CodeElevate along with the chatbot assistance is shown in 4.3

Generate Exercise: When the user clicks "Generate Exercise" the generateExercise() function is called. The function retrieves the user's selected skill level and exercise type and then the function fetchExerciseFromAPI() is called with the skill level and exercise type as arguments. Then this function sends a POST request to OpenAI's API with the prompt in 3.11 asking to generate a coding question.

Check Answer: When the user clicks the "Check Answer" button, the fetchCorrectness-FromAPI() function is called with the exercise and the user's answer as arguments. This function then sends a POST request to OpenAI's API with the prompt in 3.12 requesting to provide feedback on the answer to the question.

Chatbot Interaction: When the user types a message and clicks the "Send" button, the user's message along with the system message is sent to the OpenAI API with the prompt in 4.2 through a POST request. The API responds to the question which is then shown in the chatbot interface.

```
messages: [
 {
    role: 'system',
    content: `The current exercise is: "${currentExercise}".
    You are a helpful coding assistant. You provide hints and syntax help
    the user when asked, but you do not provide direct answers to coding
    problems. You aim to guide the user to find the solution themselves
    through a conversational and interactive approach.`
 },
 {
    role: 'user',
    content: input_value
 }
]
```

Figure 4.2: Prompt for Chatbot

4.4 Interaction with CodeElevate

4.4.1 First Look

When a user navigates to the CodeElevate web application, the first look they can see is shown in 4.4. The user can choose their desired skill level from the "Skill Level" dropdown and choose between Beginner, Intermediate or Advanced. Also, from the dropdown of "Exercise Type" the

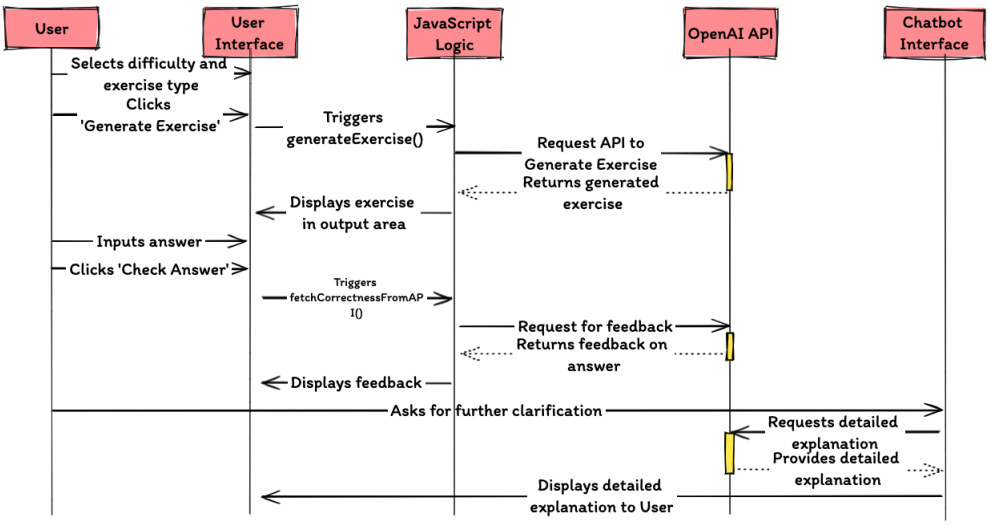


Figure 4.3: Sequence diagram of CodeElevate with Chatbot

user can select the type of exercise they want such as Lists, tuples, Sets, Arrays, etc. Afterwards, they can click on "Generate Exercise" to generate an exercise based on the desired skill level and exercise type.

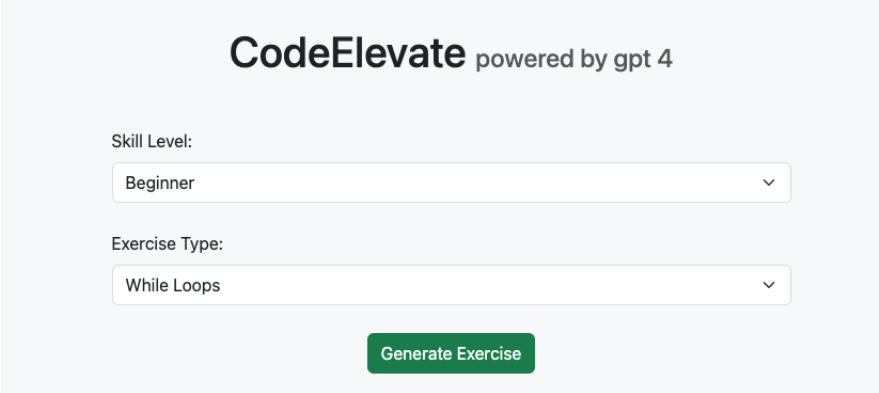


Figure 4.4: First Look of CodeElevate

4.4.2 Exercise Generated

After clicking the "Generate Exercise" button, an interface like 4.5 will be displayed. A chatbot will also appear in the left side of the interface where the user can interact with it.

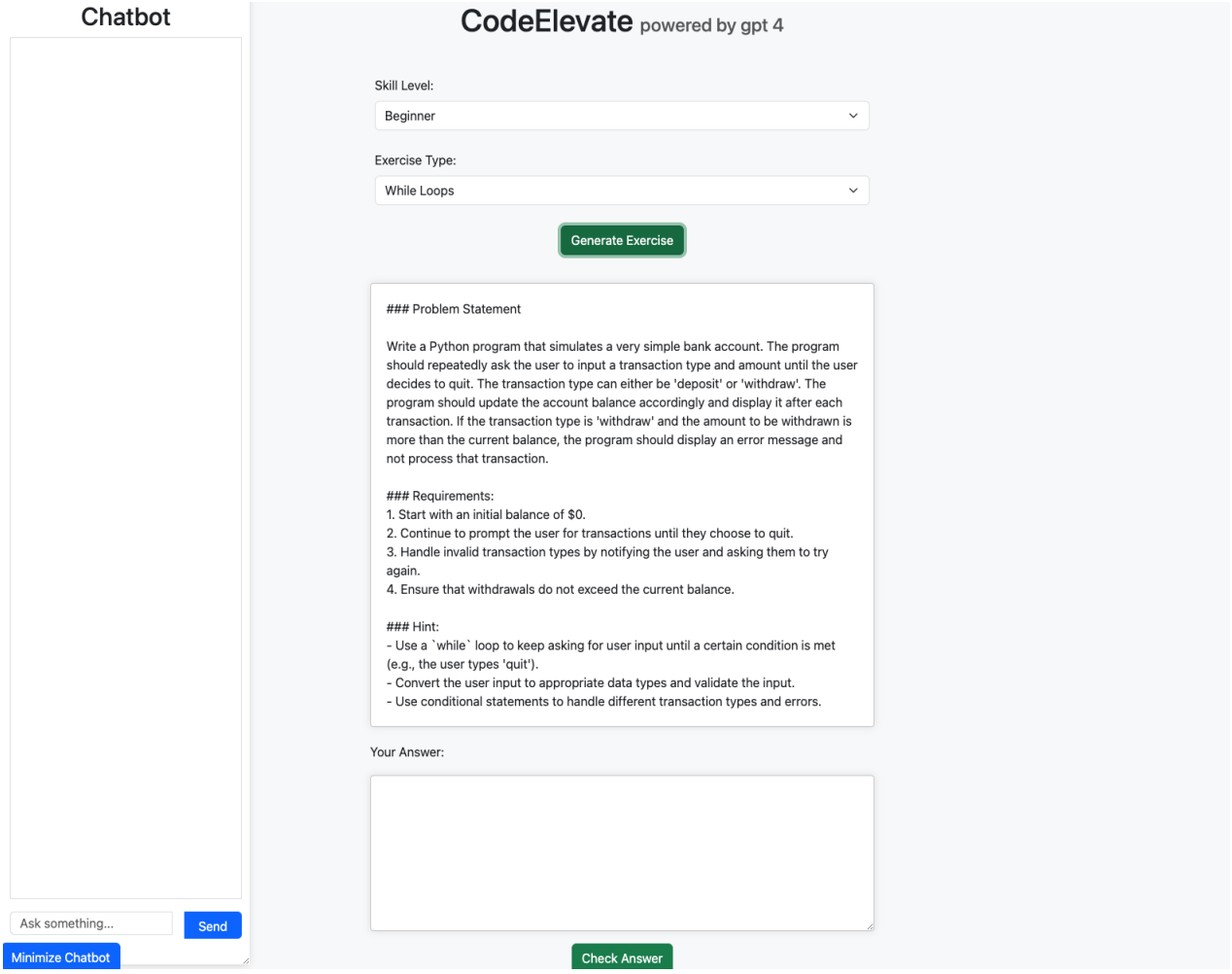


Figure 4.5: Interface after Exercise Generation

4.4.3 Chatbot Interaction

The user can ask questions about the problem to the chatbot. The chatbot is instructed to provide syntax help and guidance regarding the questions. However, the chatbot will not provide a direct solution. A typical interaction with the chatbot looks like 4.6.

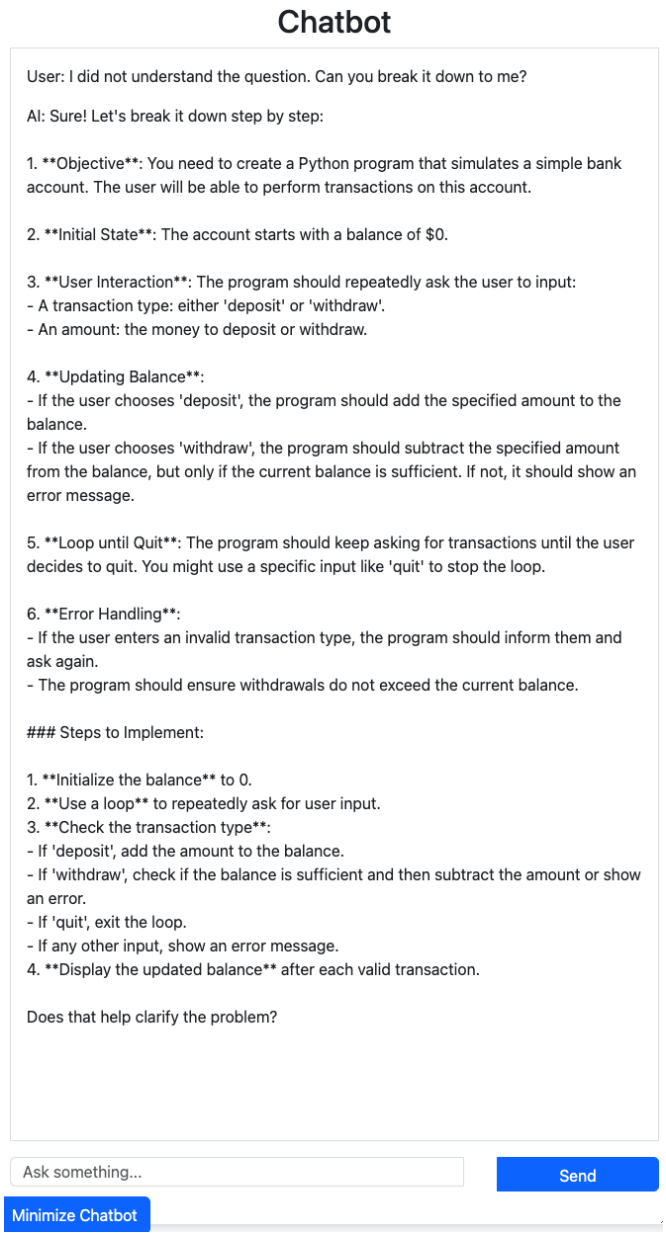
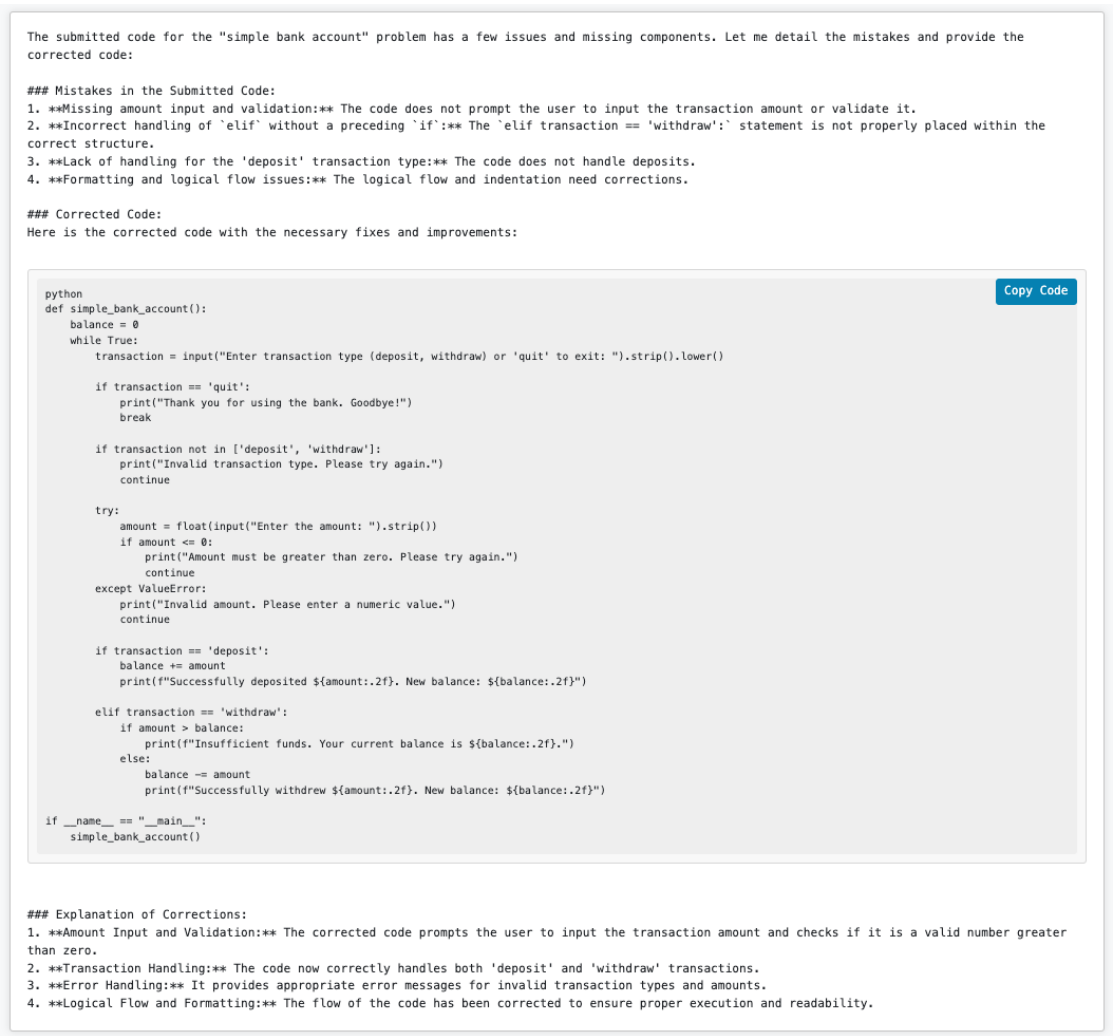


Figure 4.6: A typical Interaction with the Chatbot

4.4.4 CodeElevate's Feedback

Finally, after answering the question when the user clicks the "Check Answer" button, CodeElevate will evaluate the solution and inform the user whether the answer is correct or not. If the answer is correct, the application confirms the correctness and if the answer is wrong, the application replies with the correct answer and a detailed explanation. CodeElevate's feedback looks like the figure in 4.7.



The submitted code for the "simple bank account" problem has a few issues and missing components. Let me detail the mistakes and provide the corrected code:

Mistakes in the Submitted Code:
1. **Missing amount input and validation:** The code does not prompt the user to input the transaction amount or validate it.
2. **Incorrect handling of `elif` without a preceding `if`:** The `elif transaction == 'withdraw':` statement is not properly placed within the correct structure.
3. **Lack of handling for the 'deposit' transaction type:** The code does not handle deposits.
4. **Formatting and logical flow issues:** The logical flow and indentation need corrections.

Corrected Code:
Here is the corrected code with the necessary fixes and improvements:

Copy Code

```python
def simple_bank_account():
    balance = 0
    while True:
        transaction = input("Enter transaction type (deposit, withdraw) or 'quit' to exit: ").strip().lower()

        if transaction == 'quit':
            print("Thank you for using the bank. Goodbye!")
            break

        if transaction not in ['deposit', 'withdraw']:
            print("Invalid transaction type. Please try again.")
            continue

        try:
            amount = float(input("Enter the amount: ").strip())
            if amount <= 0:
                print("Amount must be greater than zero. Please try again.")
                continue
        except ValueError:
            print("Invalid amount. Please enter a numeric value.")
            continue

        if transaction == 'deposit':
            balance += amount
            print(f"Successfully deposited ${amount:.2f}. New balance: ${balance:.2f}")

        elif transaction == 'withdraw':
            if amount > balance:
                print(f"Insufficient funds. Your current balance is ${balance:.2f}.")
            else:
                balance -= amount
                print(f"Successfully withdrew ${amount:.2f}. New balance: ${balance:.2f}")

if __name__ == "__main__":
    simple_bank_account()
```

Explanation of Corrections:
1. **Amount Input and Validation:** The corrected code prompts the user to input the transaction amount and checks if it is a valid number greater than zero.
2. **Transaction Handling:** The code now correctly handles both 'deposit' and 'withdraw' transactions.
3. **Error Handling:** It provides appropriate error messages for invalid transaction types and amounts.
4. **Logical Flow and Formatting:** The flow of the code has been corrected to ensure proper execution and readability.

Figure 4.7: Feedback from CodeElevate

Chapter 5

Evaluation

CodeElevate was evaluated to test its accuracy, reliability and user experience, consisting of various testing methods, gathering both qualitative and quantitative data. The first test conducted was a quantitative analysis, a structured misconception test, based on common programming misconceptions identified by Chiodini et al. [4]. These tests were conducted to test the accuracy and reliability of CodeElevate in identifying these errors. In addition to the quantitative analysis, qualitative data was collected through interviews and filling out questionnaires. By integrating these evaluation methods, the study aimed to find both the strengths and potential improvements of CodeElevate.

5.1 Programming Misconception Tests

Progmiscon.org is a curated inventory contributed by Chiodini et al. [4], dedicated to documenting common programming misconceptions and providing a solution for them. These misconceptions are often identified by classroom observations and research. The site aims to identify these common programming misconceptions and provide a solution for them.

5.1.1 Methodology

For this test, we have selected 13 misconceptions from progmiscon.org. Our goal was to identify if CodeElevate can identify the common programming misconceptions in the code. So, we generated 13 random exercises from CodeElevate and intentionally answered them with a misconception, to evaluate if CodeElevate's feedback mechanism can identify those.

5.1.2 First Misconception Test: AssignCompares

Description

The misconception with AssignCompares is that the = operator compares two values, for example, 1=3, here it will return false as one is not equal to three.

Problem Statement generated by CodeElevate

Write a Python program that prompts the user to enter the name of a historical figure. The program should then print a message stating whether the figure is famous for their contributions to history or not.

Intentional Wrong Answer with a Misconception

We have answered the question with an intentional mistake on lines 2 and 3 where we used == to assign the value and = to compare two operators.

```
1 famous_figures = ["Abraham Lincoln", "Napoleon Bonaparte", "Albert
      Einstein", "Leonardo da Vinci", "Winston Churchill"]
2 name == input("Enter the name of a historical figure: ")
3 if name = famous_figures:
4     print(name + " is famous for his/her contributions to history.")
5 else:
6     print(name + " is not famous for his/her contributions to history.")
```

Listing 5.1: Intentional Wrong Answer with AssignCompares Misconception

CodeElevate's Feedback

The provided answer is incorrect. In Python, the equality comparison operator is == not =. Also, to check if a value is in a list, you should use the in operator. Here is the corrected code:

```
1 famous_figures = ["Abraham Lincoln", "Napoleon Bonaparte", "Albert
      Einstein", "Leonardo da Vinci", "Winston Churchill"]
2
3 name = input("Enter the name of a historical figure: ")
4
5 if name in famous_figures:
6     print(name + " is famous for his/her contributions to history.")
7 else:
8     print(name + " is not famous for his/her contributions to history.")
```

Listing 5.2: CodeElevate's Response

Result Analysis

We can clearly see here that CodeElevate was successful in identifying the error. Thus, the 1st Misconception test was successful.

5.1.3 Second Misconception Test: ComparisonWith

Description

The misconception with ComparisonWith is that to find out if an expression is True or False, we have to compare relational operators `==` or `!=` with booleans which are True or False.

Problem Statement generated by CodeElevate

Write a function is_positive that takes an integer x as input and returns True if x is positive, otherwise False.

Intentional Wrong Answer with a Misconception

We have answered the question with an intentional mistake on line 2 by comparing an expression with a boolean.

```
1 def is_positive(x):
2     if (x > 0) == True:
3         return True
4     else:
5         return False
```

Listing 5.3: Intentional Wrong Answer with ComparisonWith Misconception

CodeElevate's Feedback

Your function is_positive works, but it includes an unnecessary comparison `(x > 0)` == True. The comparison can be simplified because the expression `x > 0` itself evaluates to a boolean value (True or False).

Here's a simplified version of your function:

```
1 def is_positive(x):
2     return x > 0
```

Listing 5.4: CodeElevate's Response

Result Analysis

We can clearly see here that CodeElevate was successful in identifying the error. Thus, the second misconception test was successful.

5.1.4 Third Misconception Test: MapToBooleanWithIf

Description

The misconception here is that to convert an expression such `c>d` into a boolean, we need an if statement

Problem Statement generated by CodeElevate

Write a Python function that takes two integers as input and returns True if the first integer is greater than the second integer, and False otherwise.

Intentional Wrong Answer with a Misconception

We have answered the question with an intentional mistake by using an if statement to convert to a boolean.

```
def is_greater(a, b):
    if a > b:
        return True
    else:
        return False
```

Listing 5.5: Intentional Wrong Answer with MapToBooleanWithIf Misconception

CodeElevate's Feedback

Your answer is correct in the sense that it achieves the objective of returning True if the first integer is greater than the second integer and False otherwise. However, the implementation can be simplified. In Python, comparison expressions like a ¿ b directly evaluate to boolean values (True or False), so you can return the result of the comparison directly.

Here is the corrected code:

```
def is_greater(a, b):
    return a > b
```

Listing 5.6: CodeElevate's Response

Result Analysis

We can clearly see here that CodeElevate was successful in identifying the error. Thus, the third Misconception test was successful.

5.1.5 Fourth Misconception Test: ParenthesisOnly

Description

The misconception here is we do not need a parenthesis on a function that has no arguments.

Problem Statement generated by CodeElevate

Write a Python function called ‘greet‘ that prints “Hello, World!” when called without any arguments.

Intentional Wrong Answer with a Misconception

We answered the question with an intentional mistake by not putting any parenthesis after the function name.

```
def greet:
    print("Hello, World!")

greet
```

Listing 5.7: Intentional Wrong Answer with ParenthesisOnly Misconception

CodeElevate’s Feedback

The provided answer is almost correct, but there is a syntax error in the function definition. The correct answer should include parentheses after the function name. Here is the corrected code:

```
def greet():
    print("Hello, World!")

greet()
```

Listing 5.8: CodeElevate’s Response

Result Analysis

We can clearly see here that CodeElevate was successful in identifying the error. Thus, the fourth Misconception test was successful.

5.1.6 Other Misconception Tests

Similar to the three tests above we have conducted 9 more tests based on different misconceptions selected from Progmiscon.org [4]. Every test involved generating an exercise from CodeElevate, intentionally answering it with a misconception and checking CodeElevate's response. All of the tests came out successful. All the test results can be seen in Figures: A.1,A.2,A.3,A.4,A.5 and A.6 in Appendix A.

5.2 User Interviews

5.2.1 Methodology

As a part of the evaluation, we conducted interviews with 10 participants where we asked them to generate a question using our application and answer it with or without the help of the chatbot. The participants included both students and professionals. After the interview, the participants were asked to fill out a questionnaire to tell us about their experience using the app.

5.2.2 Participants' Information

Experience Levels: Participants had Python programming experience ranging from no experience to at max 4 years of experience.

Status: Out of 10 participants, 3 of them were professionals and 7 of them were students as shown in 5.1. Among the students, there were students from different levels (Undergraduate, Graduate, Advanced Master).

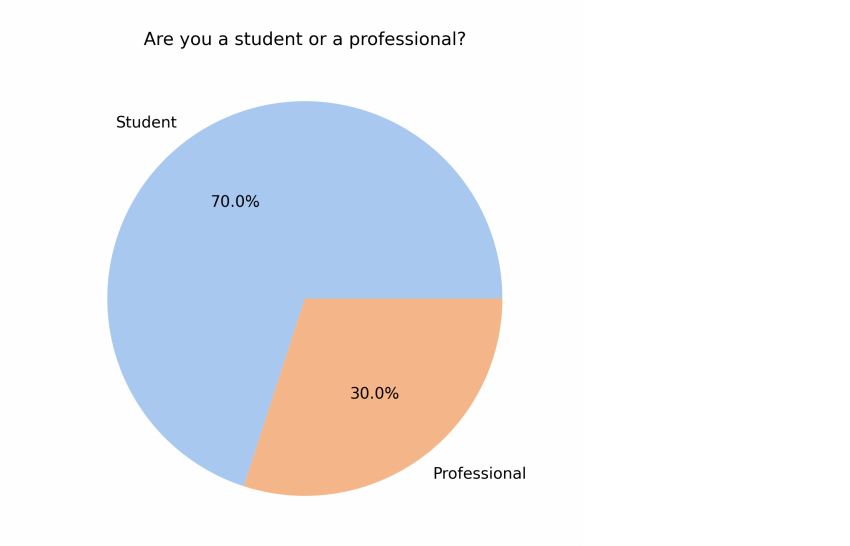


Figure 5.1: Percentage of Students and Professionals

Usage of AI Tools: Participants were asked if they use any AI tools while they do programming, and among them, 80% of them responded positively and only 20% of them responded negatively as shown in 5.2.

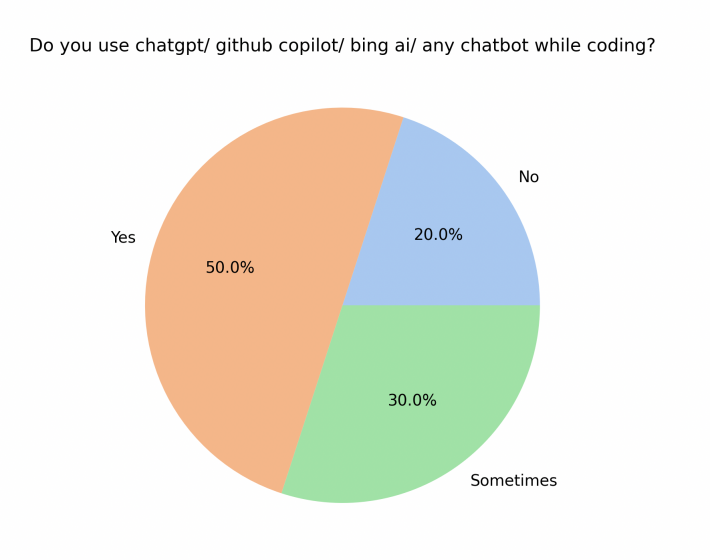


Figure 5.2: Percentage of participants using AI while coding

5.2.3 Key Findings

Ease of Use: One of the questions we asked the participants was a straightforward question, was the app easy to use? We got 100% success there as the participants found the app easy to navigate.

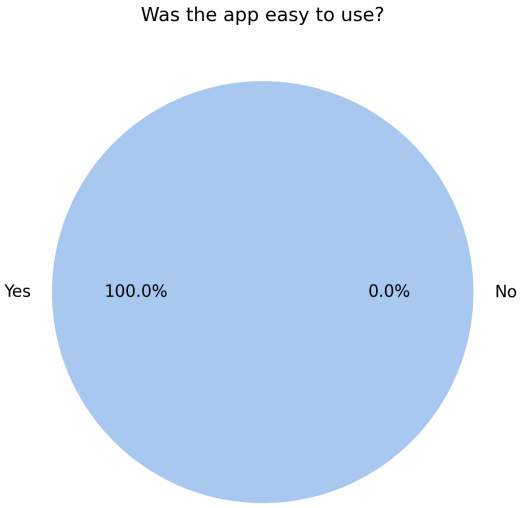


Figure 5.3: Ease of Use

Analysis of Task Completion Assistance Among Participants: The participants were asked if they were able to complete the task without any help. 9 of them among 10 said that they could not, including 7 of them directly saying they took help from the chatbot. Only 1 person completed it without any help. The analysis is shown in 5.4.

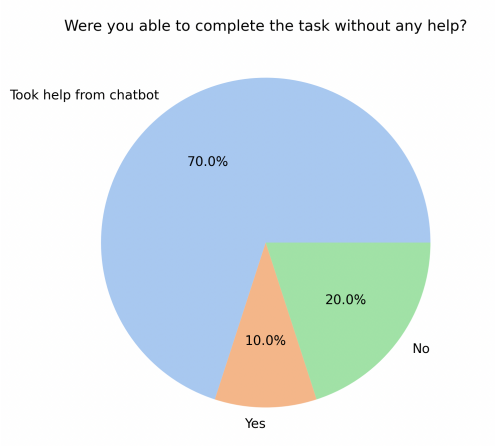


Figure 5.4: Analysis of Task Completion Assistance Among Participants

Relevance of AI-generated Problems: The participants were asked if they found the generated exercise appropriate to their skill level. Specifically, they were queried whether beginner-level exercises were appropriately challenging for beginners and whether intermediate-level exercises matched the expected difficulty for intermediate users. This question was asked to test if CodeElevate can generate exercises that align accurately with the chosen skill. The result came out with a 100% accuracy as all the participants thought that the exercises were relevant to their skill level.

However, one limitation in this evaluation is that the participants might not be able to accurately judge other levels as they are only familiar with their own level.

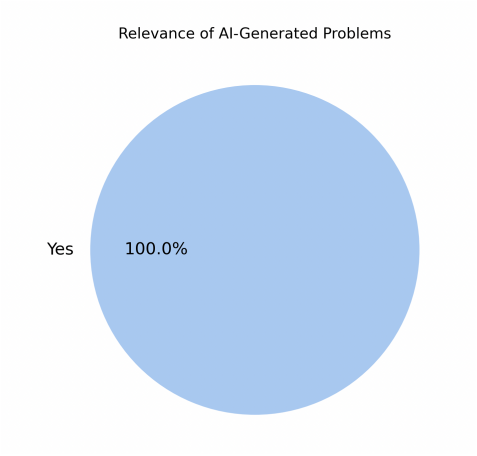


Figure 5.5: Analysis of Relevance of AI-Generated Problems

Helpfulness of the Chatbot in Guiding Towards the Solution: The participants were asked if the chatbot helped them to find the solution. Out of 10, 9 of them responded in the affirmative. An analysis is shown in 5.6.

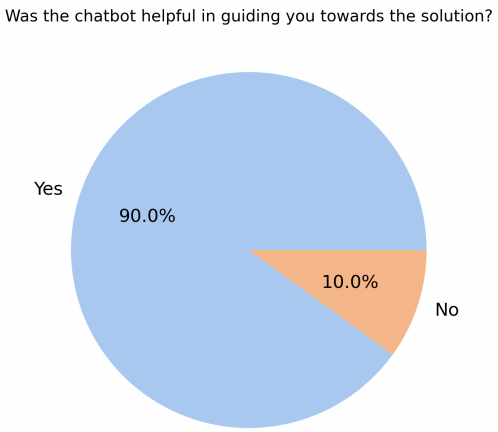


Figure 5.6: Analysis of Chatbot Guiding to Solution

Distribution of Prompts Used in the Chatbot: We collected the data for how many prompts were used by the participants while using CodeElevate. The distribution is shown in a bar chart in 5.7.

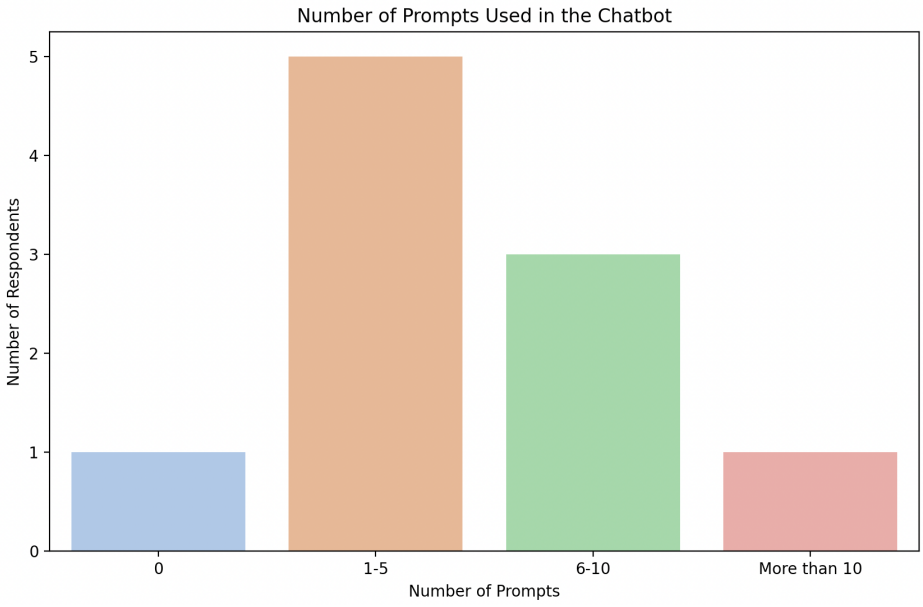


Figure 5.7: Distribution of Prompt

Clarity and Accuracy of Chatbot Hints: The participants were asked if the chatbot helped them with proper hints. 9 of them responded and 8 of them answered in the affirmative that they got clear and accurate hints. An analysis of them is shown in 5.8.

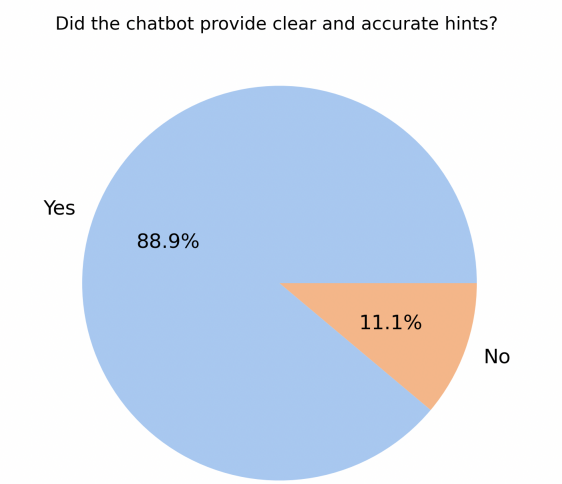


Figure 5.8: Analysis of Clarity and Accuracy of Chatbot Hints

Chatbot Usage Purposes: From the gathered data from the interview, we tried to find for what purposes the participants used the chatbot. Some of them used it for hints, again some for syntax checking, some also for other purposes. The distribution of the chatbot usage purposes is shown in 5.9.

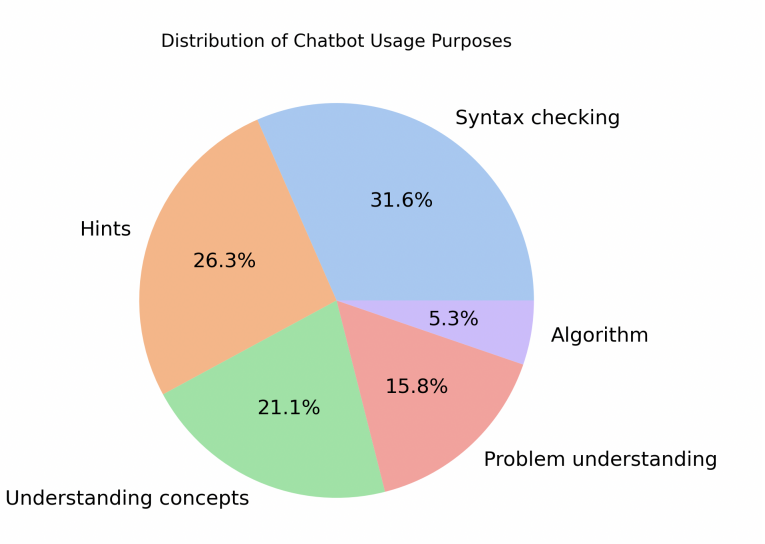


Figure 5.9: Distribution of Chatbot Usage Purposes

5.2.4 Suggested Improvements

We have asked the participants to provide us insights on how to improve the app. We broke down this into three main parts, including User Interface improvement, Chatbot improvements and overall application improvement.

User Interface Suggestions

We have asked the participants if they recommend anything to improve the User Interface. Some of them were satisfied with what there is, again some of them proposed some improvements such as integrating an FAQ sidebar, adding a loading sign, making a more visible differentiation between user input and chatbot as for small screens it was overlapping, etc and many more. A distribution of the User Interface suggestions is shown in 5.10.

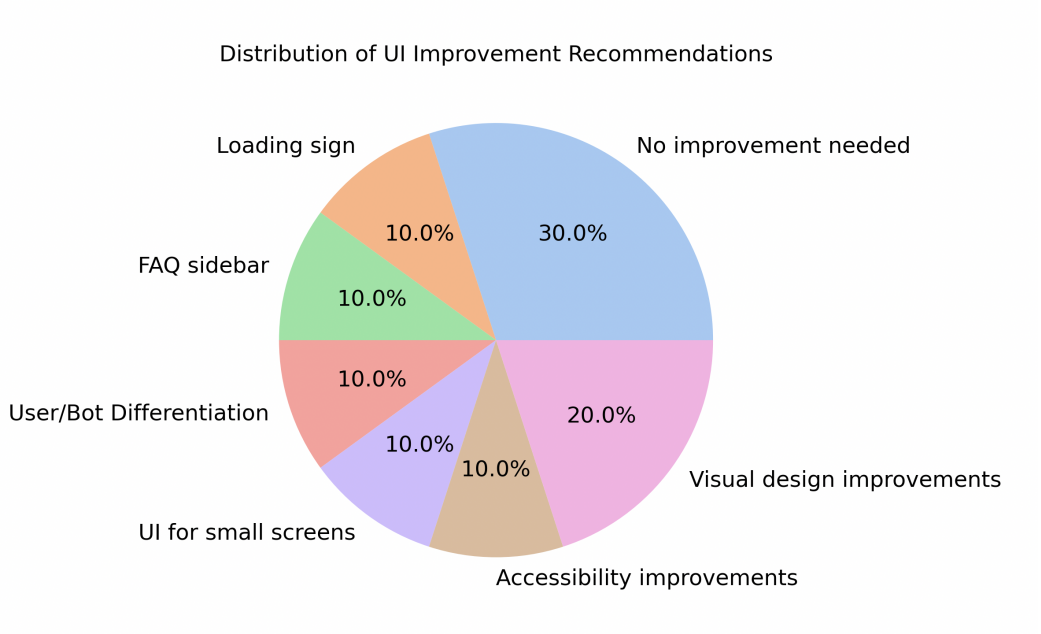


Figure 5.10: Distribution of UI Improvement Suggestion

Chatbot Improvement

We asked the participants if they thought the chatbot could do more. 9 of them answered this question. Among them, 7 of them think that it does not need any improvement. Only 2 of them suggested some improvements. A distribution of chatbot functionality feedback is shown in 5.11. Some additional functionalities the participants proposed are as under:

- One participant suggested based on the participant's question, that the chatbot could ask him more questions to evaluate his knowledge.
- Another participant suggested that the chatbot could have more awareness of other parts of the exercise, such as the difficulty level, exercise type and the answer provided by the user. Now, it is aware of only the question.

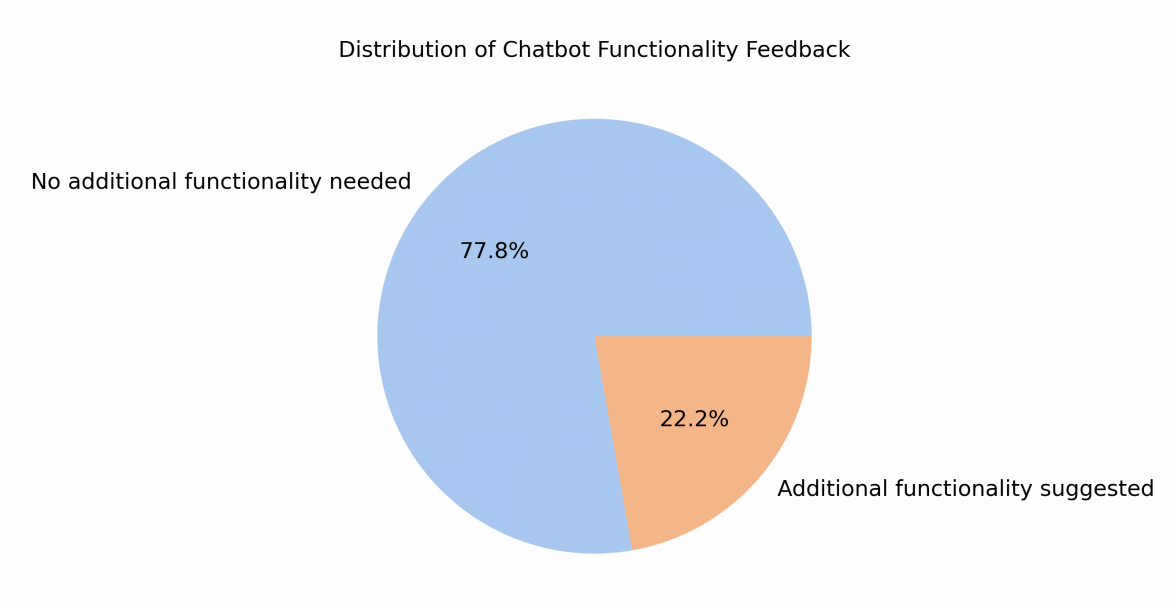


Figure 5.11: Distribution of Users Remarks on Chatbot

Overall Application Improvement

The participants were asked to give feedback about what more features they want to see on CodeElevate in future. The responses were categorised into several key themes which are summarised in Table 5.1.

Category	**Number of Responses**	**Participants Responses**
More programming languages	2	"Yes, more programming languages", "More languages"
User Personalisation	4	"It will be better if every user has their own account", "Maybe an individual account for every user", "To have a history of the prompts, include more examples with the explanation provided by the chatbot", "It would have been better if my answers were saved."
Personal rating and ranking system	1	"Personal rating system, global programmer rank system, FAQ board, Google/GitHub authentication for tracking data or future historical data"
Problem selection and offline use	1	"Choose the type of problems myself; Use a lightweight open source version of GPT to be able to download the app and use it offline"
Expanded fields	1	"Scale it to other fields like math/statistics, language learning, etc."
Improved interface integration	1	"A much more complimentary interface that works together. It felt like I was using three small apps together instead of one app having three parts."
Image generation for ML tasks	1	"Based on the task it did do what it was told, but in the future, I might expect image generation for people who want to perform various Machine Learning tasks"

Table 5.1: Summary of Desired Features for Future App Updates

Detailed Analysis

- **More Programming Languages:** Users expressed a desire to support more programming languages rather than only Python.
- **User Personalisation:** Users requested some additional features such as personal accounts to keep track of their exercises, save answers, and maintain a history of the prompts and examples provided by the chatbot.
- **Personal Rating and Ranking System:** A request was made to include a personal rating system, global programmer rank system, and FAQ board in the application.

- **Problem Selection and Offline Use:** There was a suggestion to give the user the ability to choose their own exercise. The user further suggested to have an offline version of the application as well.
- **Expanded Fields:** A suggestion was made by a user to expand fields such as math, statistics, language learning, etc. For now, it is used for only learning programming.
- **Improved Interface Integration:** One user felt that the interface could be improved more as the user felt they were using three small applications instead of one application.
- **Image Generation for ML Tasks:** A suggestion was made to include image generation for people who want to conduct machine learning tasks.

Chapter 6

Discussion & Future Work

This thesis has explored the integration of Large Language Models (LLMs) in programming education, focusing on developing applications that use LLMs for educational content generation. The main focus of this thesis lies in the development of CodeElevate. Through a systematic approach, we have demonstrated the potential of LLMs in generating programming exercises, providing immediate feedback and using a chatbot as a helper tool to solve the programming exercises.

6.1 Main Outcomes

- **Effectiveness of LLM-generated Content:** Our research has shown great potential in generating programming content through Large Language Models. Through our evaluation process with 10 candidates, we found that the users think that the exercises generated by CodeElevate were well-aligned with the users' skill levels. However, the participants were not able to judge other levels as they were only familiar with their own level. Also, they found the exercise feedback and chatbot helpful and accurate.

- **User Satisfaction:** The user evaluation indicated that the users were highly satisfied with CodeElevate. The participants appreciated the application's easy user interface and the helpfulness of the chatbot to guide to their solution. The majority of the users find the hints provided by the Chatbot helpful. Overall, the participants expressed their willingness to use this application again and recommend others which can be seen from their remark in Figure A.7.

- **Perceived Learning Benefits:** User feedback suggests that the tool's ability to provide immediate feedback and also instant help from the chatbot was beneficial.

6.2 Challenges and Limitations

While CodeElevate looks promising in enhancing programming education, it has its drawbacks as well. While developing and evaluating, several challenges and limitations were identified.

- **Accuracy:** Although most of the time CodeElevate's generated response was accurate and helpful, one of the users remarked that the chatbot was explaining the exercise instead of providing hints even after asking for hints from the chatbot. Ensuring this accuracy is very important for novice programmers who might depend on the chatbot's help.
- **Language Support:** For now the application is only generating Python programming exercises as the evaluation was conducted using Python problems. As we did not evaluate using other languages we kept our application limited to generating Python programming exercises. Expanding its capability to more programming languages is necessary to cater for a wider audience.
- **No Personal Account:** Currently the user does not have any personal account. The progress of the user is not saved for which the user cannot track their past solved problems. Adding gamification elements such as achievement badges can help the user stay motivated.

6.3 Future Directions

Based on the successes and limitations of CodeElevate, future improvements will target many key directions to improve its functionality and effectiveness in programming education.

- **Enhance Model Integration:** Currently CodeElevate is running on OpenAI's GPT 4 model. Incorporating more advanced models in future can improve the quality of the exercises and the feedback generation mechanism.
- **Expand Language Support:** Currently it only generates Python programming exercises because our evaluation focused on this language to ensure accuracy and reliability. We did not instruct the LLM to generate other language exercises as we cannot ensure any reliability. Integrating languages like Java, JavaScript, C, C#, C++, and others can help reach a wider audience and also benefit the users.
- **User Account and Progress Tracking:** Implementing a user account system will allow users to track their progress. Additionally, integrating gamification elements such as achievement badges can enhance user engagement and motivation. During the user evaluation, there were also remarks which included integrating a global ranking system.
- **Optimize Performance:** Currently there is a waiting time to generate an exercise, get feedback and get a response from the chatbot. The reason for this is due to the use of GPT-4 which takes some time to respond. To improve the response time, more advanced models like GPT-4o can be used which respond faster as we know that GPT-4o is 2x faster than GPT-4 Turbo [26].

Chapter 7

Conclusion

This thesis presents the development and evaluation of CodeElevate, an application designed to enhance programming education through Large Language Models (LLMs). Traditional methods of programming education while being effective, can be further improved with the integration of LLMs. CodeElevate generates personalised coding exercises, provides instant feedback on the exercise answer and provides chatbot assistance leveraging the power of LLMs.

Our research began by going through existing literature related to LLMs and programming education. Our goal was to study the potential benefits, drawbacks, and challenges of integrating LLMs into programming education. We also reviewed scientific papers on existing programming education tools such as CodeHelp [18] and CodeAid [13] that utilise LLMs.

Following the Design Science Research Methodology (DSRM) [2], we have developed four prototypes. The first prototype, Debugger Lens was designed to provide a step-by-step explanation of JavaScript code. While it was successful in providing explanations, one of its major drawbacks was its very basic debugging capabilities. The second Prototype, Code Analyzer was designed to generate exercises based on code snippets. Although it showed high accuracy in generating relevant exercises, its answer-checking mechanism was less effective, showing only a 46.7% success rate. The third prototype, Code Review Assistant was developed to review user-submitted code snippets and provide feedback. While it performed well with small code snippets, it was not built to handle large code bases with multiple files which was a huge drawback.

The fourth prototype, CodeElevate was selected as the final product with which we continued further development. CodeElevate's user-friendly interface, ability to generate exercises tailored to different skill levels and topics, and immediate feedback mechanism were key strengths. Despite its strength, it has limitations as well. Its current support for only Python programming language is a major limitation. Moreover, its accuracy was also questioned by one user during the evaluation. Future works will focus on these limitations and user feedback.

To conclude we can say that this thesis demonstrates that integration of LLMs into programming education can significantly improve learning outcomes by providing a personalised and interactive learning experience. CodeElevate represents a promising step towards improving programming

education.

Appendix A

Appendix

A.1 Misconception Test Results

Misconceptions	Misconception Description	Problems	Intentionally answered with misconception	CodeElevate Replies	Remarks

AssignCompares	The = operator compares two values; e.g., 1 = 2 will evaluate to False, because 1 is not equal to 2.	Exercise:
		
		Write a Python program that prompts the user to enter the name of a historical figure. The program should then print a message stating whether the figure is famous for their contributions to history or not.
		Example:
		Enter the name of a historical figure:
		Abraham Lincoln
		Abraham Lincoln is famous for his contributions to history.
		
		Hint: You can create a list of famous historical figures in the program and compare the user input with the list to determine if the figure is famous or not.
			famous_figures == ["Abraham Lincoln", "Napoleon Bonaparte", "Albert Einstein", "Leonardo da Vinci", "Winston Churchill"]
			
			name == input("Enter the name of a historical figure: ")
			
			if name = famous_figures:
			print(name + " is famous for his/her contributions to history.")
			else:
			print(name + " is not famous for his/her contributions to history.")
				No, the provided answer is not correct. In Python, the equality comparison operator is `==` not `=`. Also, to check if a value is in a list, you should use the `in` operator. Here is the corrected code:
				
				python
				famous_figures = ["Abraham Lincoln", "Napoleon Bonaparte", "Albert Einstein", "Leonardo da Vinci", "Winston Churchill"]
				
				name = input("Enter the name of a historical figure: ")
				
				if name in famous_figures:
				print(name + " is famous for his/her contributions to history.")
				else:
				print(name + " is not famous for his/her contributions to history.")
					Successfully caught the error

ComparisonWith	To determine whether an expression evaluates to True or False, one must use a relational operator (== or !=) to compare with a bool literal (True or False)	Problem statement:
		Write a function is_positive that takes an integer x as input and returns True if x is positive, otherwise False.
			def is_positive(x):
			if (x > 0) == True:
			return True
			else:
			return False
				Your function is_positive works, but it includes an unnecessary comparison (x > 0) == True. The comparison can be simplified because the expression x > 0 itself evaluates to a boolean value (True or False).
				
				Here's a simplified version of your function:
				
				python
				Copy code
				def is_positive(x):
				return x > 0
					Successfully caught the error

Figure A.1: Misconception Tests

Misconceptions	Misconception Description	Problems	Intentionally answered with misconception	CodeElevate Replies	Remarks

Conditional Sequ	if c: A else: B is equivalent to if c: A if not c: B.	**Problem:**
		
		Write a function compare_values that takes two integers x and y as input.
			def compare_values(x, y):
			if x > y:
			return "X is greater"
			if x < y:
			return "Y is greater"
			else:
			return "Equal"
				Your function compare_values works correctly and returns the expected results for the test cases provided. However, there is a minor misconception regarding the use of if-elif-else statements which is not affecting the current behavior, but it's good practice to address it. Specifically, you can use elif instead of a separate if for the second condition to make it clear that the second condition is only checked if the first one is false. Here's a refined version of your function:
				
				def compare_values(x, y):
				if x > y:
				return "X is greater"
				elif x < y:
				return "Y is greater"
				else:
				return "Equal"
					Successfully caught the error

Deferred Return	A return statement in a function does not return immediately, but it waits until the function finishes executing.	**Problem:**
		Write a Python function that takes two numbers as input and returns the sum of these two numbers.
		
		Hint:
		Use the `def` keyword to define a function and use parameters to accept input values. Inside the function, perform the addition operation and return the result using the `return` statement.
			def add_two_numbers(num1, num2):
			# Incorrectly assume return does not return immediately
			if not isinstance(num1, (int, float)) or not isinstance(num2, (int, float)):
			return "Both inputs must be numbers."
			result = num1 + num2
			return result
			print("This print statement will not be executed if the return statement above is executed.")
				The given answer is almost correct, but it has a logical error. The code should return a message as a string when the input values are not numbers, but the return statement is missing quotes around the error message. Here is the corrected code:
				
				python
				def add_two_numbers(num1, num2):
				if not isinstance(num1, (int, float)) or not isinstance(num2, (int, float)):
				return "Both inputs must be numbers."
				result = num1 + num2
				return result
					Successfully caught the error

Figure A.2: Misconception Tests

Misconceptions	Misconception Description	Problems	Intentionally answered with misconception	CodeElevate Replies	Remarks

nitCreates	The method __init__ must cre	**Problem:**
		
		Create a class called "Person" with the following attributes: name and age. Implement the `__init__` method to initialize these attributes when an object of the class is created.
		
		Hint:
		
		Use the `__init__` method in the class definition to initialize the attributes with the values provided as arguments when creating an object of the class.
			class Person:
			def __init__(self, name, age):
			
			self = Person()
			self.name = name
			self.age = age
				No, the provided answer is not correct. The correct answer is:
				
				python
				class Person:
				def __init__(self, name, age):
				self.name = name
				self.age = age
				Copy Code
				
				This code defines a class called "Person" with attributes "name" and "age" initialized in the `__init__` method when an
					Successfully caught the error

nitReturnsObjec	An __init__ method needs to return an object, and the object requires to be the same type as the surrounding class.	**Problem:**
		
		Create a class called `Car` with attributes `make`, `model`, and `year`. Implement the `__init__` method to initialize these attributes when a new `Car` object is created.
		
		Hint:
		
		Use the `__init__` method in the class to set the initial values of the attributes `make`, `model`, and `year` when a new object of the class is created.
			class Car:
			def __init__(self, make, model, year):
			self.make = make
			self.model = model
			self.year = year
			return self
				No, the provided answer is not correct. The `__init__` method in Python does not return any value explicitly. Here is the correct implementation for the `Car` class:
				
				python
				class Car:
				def __init__(self, make, model, year):
				self.make = make
				self.model = model
				self.year = year
				Copy Code
				
				This implementation correctly initializes the attributes `make`, `model`, and `year` when a new `Car` object is created.
					Successfully caught the error

Figure A.3: Misconception Tests

Misconceptions	Misconception Description	Problems	Intentionally answered with misconception	CodeElevate Replies	Remarks

MapToBooleanW	An if statement is necessary for converting an expression such as a > b into a bool.	Write a Python function that takes two integers as input and returns True if the first integer is greater than the second integer, and False otherwise.	def is_greater(a, b):
			  if a > b:
			  return True
			  else:
			  return False
				Your answer is correct in the sense that it achieves the objective of returning True if the first integer is greater than the second integer and False otherwise. However, the implementation can be simplified. In Python, comparison expressions like a > b directly evaluate to boolean values (True or False), so you can return the result of the comparison directly.
				
				Here is the corrected code:
				
				def is_greater(a, b):
				  return a > b
					Successfully caught the error

MapToBooleanW	The ternary conditional operator is necessary for converting an expression such as a > b into a bool.	Problem:
		Write a Python program that takes a user input for their age and checks if they are eligible to vote (age 18 or above). The program should output either "You are eligible to vote" or "You are not eligible to vote".
		
		Hint: Use the boolean comparison operator to check if the user's age is greater than or equal to 18.
			age = int(input("Enter your age: "))
			
			eligible_to_vote = "True" if age >= 18 else "False"
			
			if eligible_to_vote == "True":
			  print("You are eligible to vote.")
			else:
			  print("You are not eligible to vote.")
				The answer provided has a minor issue. The variable `eligible_to_vote` is being assigned string values "True" and "False" instead of boolean values True and False. While this will work in this case due to how Python evaluates truthy values, it is not the recommended approach.
				
				Here is the corrected code:
				
				python
				age = int(input("Enter your age: "))
				
				eligible_to_vote = age >= 18
				
				if eligible_to_vote:
				  print("You are eligible to vote.")
				else:
				  print("You are not eligible to vote.")
					Successfully caught the error

Figure A.4: Misconception Tests

Misconceptions	Misconception Description	Problems	Intentionally answered with misconception	CodeElevate Replies	Remarks

MultipleValuesR	A return statement can return multiple values.	Problem:
		Create two functions in Python that each take an integer as input and return a value. The first function should square the input integer and return the result, while the second function should cube the input integer and return the result. Write a program that takes an integer as input, calls both functions with the input integer, and then prints the results.
		
		Hint:
		1. Define two separate functions, one for squaring and one for cubing the input integer.
		2. Call each function with the input integer to calculate the square and cube.
		3. Print the results obtained from both functions.
			def square_and_cube(n):
			  return n * n, n * n * n
			
			n = int(input("Enter an integer: "))
			
			square_result, cube_result = square_and_cube(n)
			
			print(f"The square of {n} is {square_result}")
			print(f"The cube of {n} is {cube_result}")
				The provided answer is incorrect. The functions should be defined separately to square and cube the input integer. Here is the correct code:
				
				python
				def square(n):
				  return n * n
				
				def cube(n):
				  return n * n * n
				
				n = int(input("Enter an integer: "))
				
				square_result = square(n)
				cube_result = cube(n)
				
				print(f"The square of {n} is {square_result}")
				print(f"The cube of {n} is {cube_result}")
					Successfully caught the error

NoEmptyInit	It is not okay for a class to have an __init__ with an empty body.			Answer by AI chatbot User: is it okay for a class to have an __init__ with an empty body? AI: Having an `__init__` method with an empty body in a class is allowed in Python	

Figure A.5: Misconception Tests

Misconceptions	Misconception Description	Problems	Intentionally answered with misconception	CodeElevate Replies	Remarks
NoReservedWord	Anything that starts with a letter or underscore and consists of just letters, digits, and underscores can be used as an identifier (name of a variable, method, or class).	Problem statement: Create a function called `calculate_power` that takes two parameters, base and exponent, and returns the result of raising the base to the exponent power. Hint: You can use the `**` operator in Python to calculate the power of a number.	def calculate_power(base, return): return = base ** return return return	No, the provided answer is not correct. The mistake in the code is that the parameter name "return" is a reserved keyword in Python, so it should not be used as a variable name. Here is the correct code for the function `calculate_power`: python def calculate_power(base, exponent): result = base ** exponent return result Copy Code By using a different variable name like "exponent" instead of "return", we avoid conflicts with reserved keywords and make the code more readable.	Successfully caught the error
ParenthesesOnly	When calling a function without arguments, it is not necessary to use parentheses.	Problem: Write a Python function called `greet` that prints "Hello, World!" when called without any arguments. Hint: Remember to always use parentheses when calling a function, even if it does not have any arguments.	def greet: print("Hello, World!") greet	The provided answer is almost correct, but there is a syntax error in the function definition. The correct answer should include parentheses after the function name. Here is the corrected code: python def greet(): print("Hello, World!") greet()	Successfully caught the error
ReturnCall	A return statement, similar to a method call, needs parentheses around the return value, such as return(3).	Problem statement: Write a Python function that calculates the square of a given number. Hint: Be mindful of how you write the return statement. Remember that the return statement in Python does not require parentheses around the return value.	def square(num): return(num * num)	The provided answer is not correct. Here is the correct answer: python def square(num): return num * num	Successfully caught the error

Figure A.6: Misconception Tests

A.2 Interview Results

Did you enjoy the app? Would you use this app again?	Would you recommend this app to others?
Yes	Yes
Yes	Yes
Yes	Yes
Yes	Yes
Yes	Yes
Yes	Yes
Yes	Yes
Yes	Yes
Yes	Yes
Yes	Yes

Figure A.7: User Satisfaction Results

Did the app help you improve your problem-solving skills? Did you learn anything?
Yes. They show you the explanations and everything.
Yes
Yes it was very simple and efficient.
Yes
Yes
yes, I learned the importance of commas and method names
It did help in helping me think a certain way. I did learn something new.
Yes
Yes, Input, sting to int converstion and if else statement
yes

Figure A.8: Users Response to if they learnt something new

Bibliography

[1] Jennifer Ashley Banda. *Benefits of LLMs in Education*. URL: https://publish.illinois.edu/teaching-learninghub-byjen/benefits-of-llms-in-education/.

[2] Jan vom Brocke, Alan Hevner, and Alexander Maedche. "Introduction to Design Science Research". In: Sept. 2020, pp. 1–13.

[3] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. 2021.

[4] Luca Chiodini et al. "A Curated Inventory of Programming Language Misconceptions". In: *Proceedings of the 2021 ACM Conference on Innovation and Technology in Computer Science Education*. 2021, pp. 243–249.

[5] CodeRabbit. *CodeRabbit*. URL: https://coderabbit.ai/.

[6] Paul Denny et al. *Can We Trust AI-Generated Educational Content? Comparative Analysis of Human and AI-Generated Learning Resources*. 2023.

[7] Wensheng Gan et al. *Large Language Models in Education: Vision and Opportunities*. 2023.

[8] Google. *Chrome DevTools*. Accessed: 2024-05-14. Accessed: 2024. URL: https://developer.chrome.com/docs/devtools/.

[9] Mark Guzdial and Benedict du Boulay. "The History of Computing Education Research". In: Cambridge University Press, Feb. 2019.

[10] Xinyi Hou et al. *Large Language Models for Software Engineering: A Systematic Literature Review*. 2024.

[11] Jaeho Jeon and Seongyong Lee. "Large language models in education: A focus on the complementary relationship between human teachers and ChatGPT". In: *Education and Information Technologies* (May 2023), p. 0000.

[12] *Evaluating LLM-generated Worked Examples in an Introductory Programming Course*. 2024, pp. 77–86.

[13] Majeed Kazemitabaar et al. *CodeAid: Evaluating a Classroom Deployment of an LLM-based Programming Assistant that Balances Student and Educator Needs*. 2024.

[14] Natalie Kiesler, Dominic Lohr, and Hieke Keuning. *Exploring the Potential of Large Language Models to Generate Formative Programming Feedback*. 2023.

[15] Juho Leinonen et al. "Comparing Code Explanations Created by Students and Large Language Models". In: *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. ITiCSE 2023. ACM, June 2023.

[16] Juho Leinonen et al. “Using Large Language Models to Enhance Programming Error Messages”. In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. Mar. 2023.

[17] Cheng Li et al. *Teach LLMs to Personalize – An Approach Inspired by Writing Education*. 2023. arXiv: 2308.07968 [cs.CL].

[18] Mark Liffiton et al. *CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes*. 2023.

[19] Wenhan Lyu et al. *Evaluating the Effectiveness of LLMs in Introductory Computer Science Education: A Semester-Long Field Study*. 2024.

[20] Stephen MacNeil et al. “Automatically Generating CS Learning Materials with Large Language Models”. In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*. Mar. 2022.

[21] Stephen MacNeil et al. *Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book*. 2022.

[22] *Generating Diverse Code Explanations using the GPT-3 Large Language Model*. 2022.

[23] Stephen MacNeil et al. “The Implications of Large Language Models for CS Teachers and Students”. In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*. New York, NY, USA, 2023, p. 1255.

[24] Microsoft. *Visual Studio Code: Debugging*. Accessed: 2024-05-14. Accessed: 2024. URL: https://code.visualstudio.com/docs/editor/debugging.

[25] Steven Moore et al. “Empowering Education with LLMs - The Next-Gen Interface and Content Generation”. In: June 2023, pp. 32–37.

[26] OpenAI. *How can I access GPT-4, GPT-4 Turbo, and GPT-4o*. 2024. URL: https://help.openai.com/en/articles/7102672-how-can-i-access-gpt-4-gpt-4-turbo-and-gpt-4o.

[27] Tung Phung et al. *Generative AI for Programming Education: Benchmarking ChatGPT, GPT-4, and Human Tutors*. 2023.

[28] Lianne Roest, Hieke Keuning, and Johan Jeuring. *Next-Step Hint Generation for Introductory Programming Using Large Language Models*. 2023.

[29] Sami Sarsa et al. “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*. Aug. 2022.

[30] Amazon Web Services. *Amazon CodeGuru*. Accessed: 2024-05-14. Accessed: 2024. URL: https://docs.aws.amazon.com/codeguru/latest/reviewer-ug/welcome.html.

[31] Shen Wang et al. *Large Language Models for Education: A Survey and Outlook*. 2024.