Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Kiran Joy Kulangara

Designing and Building a Platform for Teaching Introductory Programming Supported by Large Language Models

Master's Thesis
Espoo, December 24, 2023

Supervisor:	Senior University Lecturer Arto Hellas
Advisor:	Doctoral Researcher Charles Koutcheme

Aalto University
School of Science

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Kiran Joy Kulangara		
Title: Designing and Building a Platform for Teaching Introductory Programming Supported by Large Language Models			
Date:	December 24, 2023	**Pages:**	77
Major:	Security and Cloud Computing	**Code:**	SCI3084
Supervisor:	Senior University Lecturer Arto Hellas		
Advisor:	Doctoral Researcher Charles Koutcheme		

Large language models (LLMs) have the potential to improve programming education by providing feedback and guidance to students. Despite their potential benefits, the integration of LLMs into education presents unique challenges, including the risk of over-reliance on their feedback and the inconsistency of feedback quality. Addressing these concerns requires research to identify effective ways of integrating LLMs into programming education, which itself is challenging due to the rapid evolution of LLMs.

To meet this challenge, this thesis introduces a flexible platform that can integrate multiple LLMs, providing an experimental space for research and innovative approaches to enhance programming education through LLMs.

Guided by the Design Science Research Methodology framework, the thesis outlines the design, development, and evaluation of this educational platform. Conducted at Aalto University's LeTech research group, the thesis presents an introductory programming learning platform specifically tailored to the group's research objectives. The platform facilitates data collection, and enables the students to have a personalized learning experience with the help of LLM feedback. The work advances our understanding of LLMs in education and feedback mechanisms' importance.

The developed platform effectively demonstrates the feasibility of integrating LLMs into programming education. A small-scale study evaluating the platform's overall usability received an average rating of 4.21 out of 5.00, while the LLM feedback received an average usefulness rating of 4.28 out of 5.00, highlighting its effectiveness and value in assisting students. Though the study sample size was small, the findings are encouraging. Future research could use the platform to explore multiple LLMs and conduct studies to improve the feedback mechanisms.

Keywords:	Large Language Models, Programming Education, Feedback, Interactive Learning Environment
Language:	English

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Arto Hellas, and my advisor, Charles Koutcheme, for their guidance, support, and encouragement throughout my thesis research. Their encouragement and support helped me stay motivated and on track throughout my research. Their expertise and insights have been invaluable, and I am truly grateful for their mentorship.

I would also like to thank my colleagues, Marc Aguado and Nimer Singh, for their collaboration and contributions to the project. Their teamwork and support have been essential to my success.

I am also grateful to our English lecturer, Jan-Mikael Rybicki, from Aalto's Language Centre, for his input and tips on academic writing, which helped me improve my thesis.

I am also grateful to the LeTech research group at Aalto University for the opportunity to present my thesis and for their valuable feedback.

Finally, I would like to thank my friends and family for their love and support throughout my academic journey. They have always believed in me, even when I doubted myself. I am truly grateful for their presence in my life.

Espoo, December 24, 2023

Kiran Joy Kulangara

Abbreviations and Acronyms

CER	Computing Education Research
DSRM	Design Science Research Methodology
FR	Functional Requirement
GPT	Generative Pre-trained Transformer
GPT-3.5	Large Language Model from OpenAI
GPT-4	Large Language Model from OpenAI
LLM	Large Language Model
LBT	Learning By Teaching
NR	Non-functional Requirement

Contents

Chapter 1

Introduction

The emergence of Large Language Models (LLMs) is changing the educational landscape, prompting discussions about the future of teaching, especially in programming [18]. These advanced neural networks, trained on vast amounts of data, can generate high-quality, context-aware text that closely resembles human-generated text [12].

Among other things, LLMs can be used in computing education to solve programming problems, generate assignments and explanations, and simulate human teaching assistants [22, 37, 57]. Studies have shown that the use of LLMs in programming education can help students understand programming concepts better and improve their learning experience [45].

However, we are still learning how to best use LLMs in education, and the rapid evolution of LLMs makes it difficult to conduct research on them. Hence, this thesis aims to facilitate the use and study of LLMs in programming education. In the next sections, we will discuss the context and motivation behind the thesis in more detail.

1.1 Context and Motivation

The research in this thesis was conducted within the LeTech research group at Aalto University. The group studies different aspects of computing education and learning sciences. Their primary focus is the pedagogy of teaching introductory programming and devising methodologies to enhance students' learning experiences. In the last two years, LeTech members have actively investigated the potential of LLMs in programming education, aiming to improve the feedback mechanisms and enrich the overall learning journey for students. Their work has resulted in several noteworthy publications [22, 36, 37, 45, 57].

Conducting research on LLMs and feedback mechanisms is challenging, requiring a suitable environment with essential elements such as data, participants, and relevant contextual settings for conducting experiments. To address this need, the LeTech research group requested the development of a custom platform tailored to their specific research objectives.

This thesis addresses these requirements by presenting the design and development of a platform for teaching introductory programming supported by LLMs. This platform will serve as an experimental space for exploring new ways to enhance programming learning through LLMs. The platform's flexibility will facilitate the integration of newer LLMs into research settings, thereby streamlining research efforts.

1.2 Scope of the Thesis

The research group envisions the platform to have the following features upon its completion:

1. **Provide personalized learning experiences with the help of LLMs:** The platform will provide a personalized learning experience for students with the help of LLMs. Students can use the platform to learn programming and receive personalized feedback from LLMs, which will improve their learning experience.

2. **Provide a single point of access to a variety of LLMs:** The platform will provide a unified interface for researchers to access and interact with different LLMs. This will make it easier for researchers to experiment with different LLMs.

3. **Enable the development of innovative LLM-based learning spaces:** The platform will provide a development environment and tools for researchers to create their own LLM-based learning environments with less effort. This will foster the development of new and innovative ways to use LLMs to enhance student learning.

4. **Support the collection and analysis of user interactions:** The platform will collect data on how users interact with the LLMs and how the LLMs are used to enhance student learning. This data will be used for further research to improve the use of LLMs in programming education.

However, to limit the scope of this thesis, we will focus on developing the foundational architecture of the platform, emphasizing its flexibility, and data collection capabilities. This means that we will design the platform to facilitate the integration of new LLMs, the development of new LLM-based learning spaces, and the collection of user data within the platform.

In addition to developing the foundational architecture, we will integrate one LLM, GPT-4 [50], into the platform. GPT-4 is a state-of-the-art LLM effective for a variety of tasks, including code generation and debugging. We will use GPT-4 to develop an interactive learning space, named programming space, that can provide students with personalized feedback and assistance. In this thesis, the term “learning space” is used to refer to an interactive exercise space within the platform configured to use LLMs in the backend for personalized feedback, and the term “programming space” is used to refer to such a space with programming exercises.

The thesis will also showcase the developed LLM-based programming space, conducting a user study to assess its effectiveness in helping students in learning programming. We will also discuss the results of the user study in detail.

1.3 Structure of the Thesis

The rest of this thesis is structured as follows:

- **Chapter 2:** Reviews literature at the intersection of Computing Education Research and LLMs and emphasizes the significance of automated feedback and the use of LLMs.
- **Chapter 3:** Outlines the Design Science approach to building the platform, from explicating the initial problem statement to evaluating the platform. This chapter also discusses the tools and practices adopted by the development team.
- **Chapter 4:** Highlights both functional and non-functional requirements essential for the platform’s development.
- **Chapter 5:** Provides in-depth insights into the technical aspects of building the platform. This chapter also discusses the design decisions made to achieve the platform’s flexibility and data collection capabilities.
- **Chapter 6:** Showcases the platform, the LLM-based programming space, and its current use cases.
- **Chapter 7:** Presents the outcomes of the user study done on the platform and provides an analysis of the collected data.

- **Chapter 8:** Discusses the thesis and the user study in light of related works and research, highlights the limitations of the work, and presents future research directions.
- **Chapter 9:** Summarizes the work.

Chapter 2

Background

This chapter provides a foundation for the thesis by discussing the complexities of teaching and learning programming, various systems designed for teaching programming, and the possibilities of integrating LLMs into programming education. It emphasizes the role of feedback mechanisms in enhancing the learning experience, sheds light on the potential of LLMs for computing education, and examines specific systems that employ LLMs for programming education. The chapter ends by discussing Design Science Research, a structured framework for system development and evaluation that aligns with the goals of the research.

2.1 Teaching and Learning Programming

Computing Education Research (CER) is an interdisciplinary field that studies how students learn computing concepts, particularly programming. It aims to identify effective teaching strategies, address common learning difficulties, and promote evidence-based practices for programming education. CER focuses on the didactic triangle, which comprises the student, teaching methods, and the curriculum [7]. By examining the interactions among these elements, CER informs pedagogical approaches and contributes to the continual advancement of computing education worldwide.

The learning journey to becoming a proficient programmer is influenced by several key aspects, including the acquisition of disciplinary content knowledge. This foundational knowledge forms the basis for effective programming practices [42]. Within this context, several key aspects influence the learning journey.

First, it is essential to understand how students learn programming concepts. This involves exploring the various mental models through which students grasp programming concepts and identifying common misconceptions within these models. Research shows that novice programmers often exhibit misconceptions even in fundamental programming concepts such as variables [60] and value and reference assignment [43]. By addressing these misconceptions, educators can tailor their teaching strategies to meet the specific needs of students and foster deeper learning.

Second, literacy in programming is essential. This encompasses code reading, writing, and debugging. Students must develop the ability to comprehend existing code, write their own programs, and effectively troubleshoot and rectify errors. Proficiency in tracing and explaining code serves as a foundation for code writing skills [39]. Emphasizing these skills in the early stages of introductory programming courses allows for their mutual reinforcement and development.

Finally, the sentiment aspect towards learning programming is also important, which involves students' attitudes and engagement. Positive attitudes towards programming contribute to motivation, persistence, and a willingness to tackle challenges. Students' self-motivation and engagement in learning activities play a significant role in their success [56]. Therefore, measuring and enhancing student engagement is crucial. Engaged students are more likely to actively participate, explore programming concepts, and develop a deeper understanding of the subject matter [56]. This highlights the importance of fostering a supportive and inclusive learning environment that encourages students to embrace programming as a valuable skill.

By considering these factors and aligning instructional strategies with the principles of CER, educators can create an effective learning environment that supports students' comprehensive acquisition of programming knowledge and skills.

2.2 Systems for Teaching Programming

Introductory programming courses often face the challenge of student retention, primarily due to a lack of engagement [33]. The large enrollments in these courses make it difficult for lecturers to monitor individual student engagement. To address this issue, research has focused on understanding students' perceptions, engagement levels, attitudes, and experiences in learning programming, aiming to enhance students' overall learning experience.

Research on e-learning environments has demonstrated that effectively managing and facilitating such environments can enhance students' motivation to learn and their sense of efficacy [35]. Numerous tools have emerged in the e-learning space, and a systematic literature review on these tools can be found in [21].

One example of a system designed for teaching programming is the Python Classroom Response System (PCRS) [69]. PCRS is a web-based tool that facilitates code-writing and multiple-choice questions in a classroom setting, extending the principles of peer instruction to introductory programming education. PCRS aims to provide real-time feedback to instructors and support active learning techniques in programming classes. It evaluates code submissions using a suite of tests to identify common misconceptions, allowing instructors to identify and address student challenges as they submit code. The system also enables instructors to retrieve specific code submissions for in-class examples through an editor and visualizer.

PCRS supports peer instruction by incorporating code-writing questions alongside multiple-choice questions in programming classes. It enhances the effectiveness of code-writing exercises by providing interactive visualization tools for debugging and program understanding. Instructors receive real-time feedback on students' progress and can identify misconceptions through the analysis of code submissions and test results. PCRS supports various code-writing exercises, including extending starter code, debugging exercises, and defect triage exercises.

Another system used for teaching programming is CodeWrite [14], a web-based tool that offers drill and practice support for Java programming. Its primary goal is to help students develop their programming skills by providing a wide range of short programming exercises. CodeWrite can be accessed from any browser-connected computer with internet access. Notably, CodeWrite emphasizes testing. Exercise authors define sets of tests that code implementations must pass before becoming available to other students. This approach incentivizes students to write tests and ensures that the code submitted by the class is automatically verified and given feedback.

CodeWrite follows a pedagogical approach called constructive evaluation [41], where students take responsibility for developing exercises that can be shared among their peers. This approach exposes students to various solutions for a given problem, helping them recognize multiple correct solutions and deepen their understanding of programming concepts. After successfully completing an exercise, students can engage in a lightweight peer review process, evaluating the exercise's quality, providing feedback to the author, and endorsing solutions contributed by their peers.

The effectiveness of CodeWrite was evaluated in a CS1 course at the University of Auckland, focusing on the coverage of language features and the variety of solutions submitted. The results showed that the exercises in the CodeWrite repository encourage the use of a wide range of Java language features, with a high percentage of exercises involving assignments, conditionals, loops, and arrays. Moreover, students utilized the full range of language features taught in the course, demonstrating that CodeWrite effectively supports their practice. The tool also exposed students to a variety of solutions, showcasing substantial variation in the number of lines of code and control flow complexity. Overall, CodeWrite provides students with ample opportunities to practice language features and build confidence in their programming skills.

An integral aspect of these systems is the integration of timely feedback. This feedback loop enables students to learn from their mistakes, compare different approaches, and gain confidence in their programming abilities.

2.2.1 Feedback in Programming

Feedback is an essential component of both formative and summative assessment in introductory programming courses [51]. Formative assessment, which is typically conducted throughout a course, focuses on providing ongoing feedback to students to help them improve their learning. Summative assessment, on the other hand, is typically conducted at the end of a course or unit to measure student learning.

In formative assessment, feedback fosters self-regulated learning by enabling students to set goals, assess their progress, and adjust their strategies accordingly. Incorporating a feedback-revision-resubmission cycle into homework submissions significantly enhances student engagement and reduces stress, allowing for iterative improvement and mastery over time [24]. By receiving feedback and revising their work, students continually enhance their problem-solving abilities. By reflecting on their mistakes and identifying areas for improvement, students refine their understanding and skills. Whereas in summative assessment, feedback provides students with an overall assessment of their performance and indicates their level of achievement [63]. While it may not provide the same level of detailed guidance as formative feedback, summative feedback serves as a summary of the learning process, enabling students to evaluate their overall progress. It allows for reflection on performance over an extended period and identification of areas that may require further improvement. Summative assessment is particularly valuable for making judgments about competence and determining final grades or certifications.

Providing individual feedback on every submission in large-scale courses such as Massive Open Online Courses (MOOCs) poses a challenge for instructors [20]. With a significant number of students, manually assessing and offering personalized feedback for each assignment becomes impractical. To address this issue, automated assessment approaches have been implemented to reduce the workload of instructors while maintaining the quantity and quality of practical exercises [61]. These automated systems use algorithms and predefined criteria to evaluate student work, providing instant feedback and assessment results.

By incorporating effective feedback mechanisms, instructors can facilitate students' learning and growth throughout their programming education journey.

2.2.2 Automated Assessment Systems and Feedback

Automated assessment systems are computer-based systems that can grade programming assignments automatically. Some systems rely on static analysis tools like PMD [53] and CheckStyle [10] to flag potential issues, but still require human reviewers to make the final judgment. Other systems, such as [23], Web-CAT [16] and HoGG [47], aim to automate the entire assessment process. They offer several advantages for both teachers and students in programming courses. For teachers, automated assessment systems can significantly reduce workload and enable continuous student assessment even in large classes. By automating tasks such as grading, teachers can assign more programming tasks while spending less time on evaluation, leading to increased efficiency [5]. For students, automated assessment ensures consistency and objectivity in evaluating programming assignments. By removing personal biases, students receive grades based solely on their performance, and certain aspects of programming assignments can be more accurately assessed by automated approaches compared to human evaluation [1]. Additionally, computer-based assessment is more effective in evaluating students' programming skills than traditional paper and pencil testing [32].

One of the additional benefit of automated assessment is the generation of automated feedback for students. The feedback provides valuable information about the quality of their solutions and offers guidance on how to improve their work. It helps students to identify their weaknesses and limitations, fostering self-awareness and promoting continuous improvement. Even basic feedback can have a significant impact on students' learning experience, leading to noticeable enhancements in their programming skills [2]. Precise real-time feedback enhances the effectiveness of practice and contributes significantly to the improvement of learning outcomes [54]. Consequently, automated feedback enables personalized guidance, which would be challenging to achieve in large programming courses without the assistance of technology.

Advancements in CER have led to the development of various effective automated tools that support programming learning with automated feedbacks. These tools provide valuable feedback to novice programmers, helping them identify and fix errors, improve program quality, and develop their programming skills. For example, Code Analyser for Pascal (CAP) [58] and Gauntlet [19] identify errors in programming code and provide detailed feedback to improve program quality and reduce grading time. The Prolog Tutor [25] assesses students' solutions and offers coding-level support to address errors and enhance programming techniques. InSTEP [48] provides targeted feedback for common logic errors, while AnalyseC [67] compares students' solutions with accurately modeled solutions to deliver feedback. Ask-Elle [28] generates instant feedback by evaluating students' solutions against a predefined model, and SIPLeS-II [68] provides relevant hints based on the correctness of students' solutions by comparing them against model solutions.

Tools such as Whyline [34] help programmers understand code execution by allowing them to ask specific questions and obtain detailed explanations, improving programming efficiency. AutoLEP [64] combines static analysis with dynamic testing to provide immediate feedback on errors, helping novice programmers develop their skills. Additionally, research also highlights the importance of feedback-on-demand, as demonstrated by an automated programming tutor that integrates guided planning, assisted coding, and dynamic hints for better learning outcomes [30]. Another example is AutoTeach, which offers an incremental hint system for programming exercises, gradually revealing suggestions and solution details through flexible textual and code-revealing hints. AutoTeach [4] incorporates a code visibility model and customizable processing policies to enhance feedback and support in programming education.

With the emergence of LLMs, a new frontier has opened in the field of programming education. LLMs, powered by advanced machine learning algorithms, can analyze and generate code, providing students with not just feedback but also intelligent suggestions and solutions.

The following section explores the role of LLMs in programming education and discusses the benefits and considerations associated with their use.

2.3 Large Language Models in Teaching Programming

Recent advancements in natural language processing have paved the way for the emergence of Large Language Models (LLMs). Early language models such as BERT [15] and CodeBERT [17] laid the foundation for leveraging natural language processing in programming instruction. These models provided valuable insights into code analysis and comprehension, leading to the development of more advanced systems such as Codex [11] and ChatGPT [49].

Codex, developed by OpenAI, has garnered recognition for its ability to generate code snippets from natural language prompts. This capability empowers learners with practical examples, facilitating their understanding of programming concepts. ChatGPT, on the other hand, presents a conversational interface for GPT-3.5 and GPT-4 [50], fostering interactive learning experiences and enhancing student engagement.

Codex stands out as a code-generation model adept at translating natural language into code. Its proficiency extends to over a dozen programming languages. Additionally, Codex offers code completion by automatically completing code snippets based on context. In contrast, GPT-3.5 and GPT-4 are general-purpose language models with a broader range of applications, encompassing code generation, translation, question answering, and crafting various creative content formats. GPT-3.5 and GPT-4 surpass Codex in terms of sophistication, with GPT-4 showcasing a remarkable leap in capability. GPT-4's multimodal nature allows it to analyze text, images, and voice, offering a wider spectrum of understanding compared to the primarily text-based GPT-3.5. Moreover, GPT-4 boasts a significantly larger size, with 10 times more parameters than GPT-3.5. This enhanced capacity enables GPT-4 to better grasp context, distinguish nuances, and deliver more accurate and coherent responses.

Other notable LLMs that showcase the potential for enhancing programming education and productivity include LaMDA [62] by Google, DeepMind's AlphaCode [38], and Amazon's CodeWhisperer [3].

The incorporation of LLMs in programming education has attracted significant attention from researchers and educators alike [6, 18, 37, 46, 57, 65]. Research has demonstrated the effectiveness of LLMs such as Codex in solving introductory programming problems [13, 18, 65]. These studies evaluated the capabilities of Codex using programming problems. The findings indicated that Codex could solve approximately 47.6% of the problems on its initial attempt, and with modified problem description prompts, it was able to solve around 61% of the remaining problems [13]. Comparisons between Codex's performance and student responses revealed that Codex outperformed most students and provided a diverse range of responses for a given problem [18]. Moreover, Codex demonstrated its competence by successfully solving various iterations of the classic Rainfall Problem [18, 59].

LLMs have also been utilized in generating programming exercises and code explanations. One study incorporated Codex into a software development course to generate explanations for code examples in an online e-book, and the students found these code explanations to be helpful for their learning process [45]. This study also highlighted the ability of LLMs to generate code explanations on demand, which can be particularly useful in contexts where explanations do not already exist, such as when students write their own code. Another study used Codex to create programming exercises, including sample solutions, test cases, and code explanations [57]. The study found that the generated content was mostly novel and sensible. However, it emphasized the need for oversight to ensure the quality of the generated content before delivering it to students. Furthermore, LLMs such as GPT-3 have shown potential for generating code explanations that are rated higher in terms of understandability and accuracy compared to those created by students [36]. These LLM-generated explanations can serve as examples for students to improve their own code explanations and adopt a standard format.

These findings highlight the effectiveness of LLMs in supporting programming education. In this thesis, we aim to explore another potential application of LLMs: providing feedback for student-written code in programming assessments. The following section delves into the relevant literature in detail.

2.4 Large Language Models for Automated Feedback

Providing feedback on students' code is essential for programming education, but it can be time-consuming to manually review code, especially in large classes. As we reviewed earlier, automated tools and assessment systems have been developed to address this challenge, but they often rely on static analysis and rule-based systems that may not fully capture the complexities of programming. Assessment systems like Web-CAT [16] and HoGG [47] require manual configurations for adding new assignments and corresponding tests, while systems that compare student code with a model solution necessitate the instructor to provide the model solution. LLMs, on the other hand, offer a promising alternative, leveraging their extensive knowledge of programming languages and problem-solving techniques to provide more sophisticated and context-aware feedback. Unlike static analysis tools, LLMs can grasp the broader context of code changes, including the underlying intent and design decisions. This ability allows them to identify potential issues and provide constructive suggestions that go beyond simply flagging syntax errors or rule violations.

In addition to generating exercises and explanations, LLMs have also been used to provide feedback for student-written code in programming assessments. A recent study [22] collected help requests and code samples from an online programming course and prompted two different LLMs, OpenAI Codex and GPT-3.5, to identify and explain issues in the students' code. The study found that GPT-3.5 outperformed Codex in most respects, with both LLMs frequently identifying at least one actual issue in each student program. However, neither LLM excelled at finding all issues, and false positives were common. Notably, LLMs demonstrated better performance in identifying issues related to program logic compared to output formatting.

Another area where LLMs have shown promise is in enhancing programming error messages, particularly compiler/interpreter error messages. Novice programmers often struggle to understand and interpret these error messages, which are typically designed for experienced programmers. However, research has shown that LLMs can improve compiler/interpreter error messages by providing explanations and suggestions for fixing the errors [37]. The results of the study demonstrate that LLMs can generate novice-friendly enhancements to compiler/interpreter error messages that are more interpretable and actionable than the original messages.

All these studies suggest that by incorporating LLMs into programming assessments, instructors can save time while delivering personalized and detailed feedback to each student. LLMs can analyze student code, identify common errors, suggest alternative solutions, and provide explanations for the detected issues. This automated feedback is particularly valuable in introductory programming courses, where students often struggle with fundamental concepts and syntax. Recognizing the potential of LLMs in programming education, researchers have begun exploring the development of various systems that utilize LLMs to enhance the learning experience.

2.5 Systems Utilizing Large Language Models in Programming Education

AI teaching assistants are being introduced in university-level programming courses, which is part of the broader trend of using AI and technology to improve education. GPT-4 is already being used in university courses and classrooms [55]. Another example is Khanmigo, an AI chatbot tutor developed by Khan Academy, which uses OpenAI's GPT-4 to provide students with assistance in various subjects [9].

Recent studies have explored the creation of systems using LLMs as teachable agents for learning by teaching (LBT) [29, 44]. LBT involves learners teaching a teachable agent to identify their knowledge gaps and acquire new knowledge.

One example of such a system is HypoCompass, a debugging tutor system that aims to train novices in hypothesis construction skills in software development [44]. The system leverages LLMs to generate various types of training materials, including starter test case category hints, descriptive test case hints, buggy codes, explanations of bugs, and codes with the bugs fixed. It uses LLMs to simulate students who write buggy code, with human novices acting as Teaching Assistants guiding the LLM-generated student through the debugging process. The study highlights that this setup supports learning by allowing novices to reason through diverse bugs and reinforce their problem-solving skills.

An example of another LBT-based system is discussed in [29], where the authors developed two components, "TeachYou" and "AlgoBo," which together create an LBT environment for algorithm learning. AlgoBo is an LLM-based tutee chatbot that can simulate misconceptions and unawareness as prescribed in its knowledge state. TeachYou is the web-based algorithm learning system that supports LBT with AlgoBo, providing metacognitive feedback to learners on their teaching methods in real-time.

These studies suggest that LLMs have the potential to enhance programming education. However, further research is needed to optimize their integration, make them safe, and ensure they effectively address the diverse needs of students and instructors in various educational contexts.

2.6 Design Science Research

When building systems in research, it is meaningful to have a framework that guides the process. Design science research is a multidisciplinary approach that focuses on investigating the creation and use of artifacts as solutions to practical problems in various scientific fields [31]. It involves taking an intentional stance towards artifacts and considering them as tools to support practitioners in specific practices. Design science researchers adopt the role of designers, actively creating artifacts that are useful and valuable to practitioners. The ultimate goal of design science is to contribute to solving practical problems of general interest by studying and creating artifacts through scientific inquiry.

Design science research emphasizes the generation of new artifacts and the knowledge about them, with the belief that the world can be improved through the introduction of these artifacts [31]. It goes beyond mere theorizing and conceptualization by actively modeling, making, and building to create new worlds. In this process, design researchers not only develop novel artifacts but also generate knowledge about their impact on the environment.

While design and design science are related in their focus on creating artifacts to address practical problems, they differ in their aims and contributions to knowledge [31]. Design aims to devise functional solutions to specific problems, often within a particular context or situation. On the other hand, design science seeks to produce and disseminate knowledge that has broader applicability and is of general interest. Design science projects aim to create new knowledge through rigorous research methods, ensuring the results are well-founded, original, and grounded in existing knowledge. Moreover, design science projects are expected to communicate their findings to both practitioners and researchers, contributing to the advancement of scientific knowledge.

While design science projects can be conducted within a local practice, contributing to a global practice and scientific knowledge is crucial. Researchers can employ two different strategies [26]: constructing a generic artifact to address a problem in a global practice or solving a specific problem in a local practice and deriving prescriptive knowledge from the experience. The latter approach allows the researcher to remain situated within the local practice throughout most of the project, with generalization to a global practice occurring towards the project's end.

This thesis adopts a bottom-up approach, aligning with the local practice strategy of design science research.

Chapter 3

Methods

This chapter introduces the research methodology and development process used in this study. We begin by discussing Design Science Research Methodology (DSRM), the overarching framework that guides our systematic development process. The subsequent section details the team structure, iterative software development approach, and the use of Kanban boards for project management and communication.

3.1 Design Science Research Methodology

In practice, design science research often follows a structured framework to guide its progression. A comprehensive framework known as the Design Science Research Methodology (DSRM) is proposed in [52]. This methodology lays out a systematic approach comprising a set of procedures, methods, and practices tailored for conducting design science research.

The DSRM encompasses five main activities, as shown in Figure 3.1, each serving a crucial purpose within the research process:

1. **Problem Explication:** This initial stage involves recognizing and defining the problem or challenge that the research seeks to address. It's the foundation upon which the entire research endeavor is built.

2. **Requirement Definition:** Once the problem is identified, the next step is to define the requirements and criteria that the proposed solution must meet. This stage clarifies the objectives and expectations for the research.

3. **Design and Development:** With clear requirements in place, the research moves into the design and development phase. Here, researchers create the artifacts or solutions aimed at solving the identified problem.

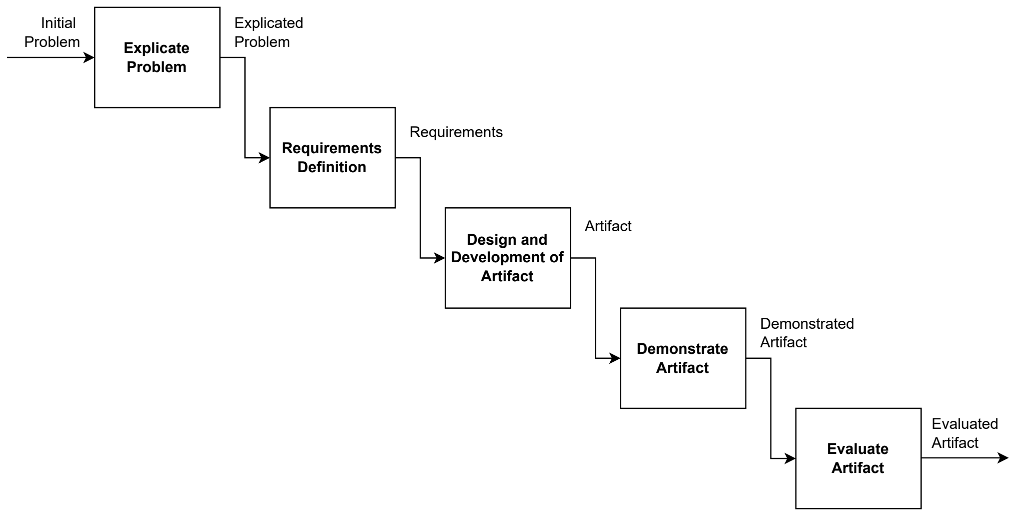


Figure 3.1: Overview of the method framework for design science research adapted from [31]

4. **Demonstration:** Following artifact development, it's crucial to demonstrate how the created solution functions in a real-world context. This step validates the practicality and effectiveness of the designed artifact.

5. **Evaluation:** The final step involves evaluating the solution's performance and its impact on addressing the identified problem. This assessment provides insights into the effectiveness and potential improvements of the solution.

These steps are drawn from existing literature on DSR, and researchers may adapt and use them in various ways, depending on their specific research context. Typically, the findings from the DSR are also communicated to the relevant audience through academic journals, conferences, and workshops. Also, it is important to note that the DSRM process is iterative and can be conducted in a non-linear fashion.

Each of these DSRM steps is further outlined and discussed in detail in the following chapters.

3.2 Process of Development

A five-member team under the LeTech research group, including the thesis author, supervisor, and advisor, worked on the project, with the thesis author leading the effort on the programming space and the use of diverse LLMs. The team used an iterative software development approach to build the platform. Iterative development involves breaking down the software development of a large application into smaller, more manageable segments. With each iteration, additional features are designed, developed, and tested, ultimately resulting in a fully functional software application ready for deployment. This iterative development process comprises a continuous cycle encompassing planning, analysis, implementation, and evaluation, which goes hand in hand with the DSRM.

The team held regular weekly meetings to demonstrate progress on the project and discuss and plan suggestions and changes for the next iteration. We used Kanban boards to visualize and manage the workflow. Kanban is a visual project management tool that helps teams visualize tasks and track their progress. Specifically, we used GitHub Projects[1], a lightweight Kanban tool, to meticulously track individual tasks, identify bottlenecks, and maintain a seamless workflow throughout the development process. Our Kanban boards provided a clear overview of the platform's development status, offering real-time insights into project progress. This transparency fostered effective communication and collaboration among team members and stakeholders, ensuring alignment throughout the development journey.

[1]`https://docs.github.com/en/issues/organizing-your-work-with-project-boards`

Chapter 4

Problem Explication and Requirement Definition

Clearly defining the problem and its requirements is essential before designing and developing a solution. Therefore, following the DSRM framework, this chapter will first explicate the problem that this thesis addresses and then define the requirements for the platform that will be developed to address it.

4.1 Problem Explication

In any design science project, the initial step involves identifying and understanding the issues encountered by individuals within a specific domain. However, often these initial problems are vague, ambiguous, and poorly articulated. Therefore, it is crucial for researchers to analyze, investigate, and reformulate the problem to gain a deep understanding of it. Without a thorough grasp of the issue at hand, there is a risk of developing solutions that fail to effectively address the underlying issue [31].

The *Problem Explication* activity serves as the foundational step within the Design Science Research Methodology (DSRM) framework for design science projects. Its primary objective is to precisely define the initial problem, justify its significance, and delve into the root causes behind it [31]. Diverse research strategies and methods can be employed to explicate the problem.

This thesis was conducted within the LeTech research group at Aalto University, under the guidance of researchers who have been actively exploring the potential of LLMs and feedback mechanisms in programming education. The positioning and justification of the problem were centered around the research objectives of the LeTech research group as detailed in Section 1.1. Several discussions were held with the researchers for the problem explication.

The development of a custom platform equipped with LLMs was justified as a research tool that would facilitate seamless data collection and analysis, providing valuable insights into effective teaching strategies and areas of improvement. There is no root cause for the problem that is being addressed in this thesis. Rather, the motivation for this thesis emerged from the recent advancements and widespread adoption of LLMs within programming education. This thesis addresses the need for a comprehensive evaluation and comparison of LLMs within an educational context, particularly for providing personalized feedback and instructional support. The widespread adoption of these systems in programming education has necessitated a thorough assessment of their effectiveness and suitability for practical use. As LLMs become increasingly integrated into educational technology, it is crucial to understand their impact on student learning outcomes and identify potential areas for improvement.

4.2 Requirement Definition

The second activity in the DSRM framework is *Requirement Definition*, which involves defining the requirements of the artifact that will address the explicated problem. This process involves outlining the artifact at a relatively high and abstract level while gathering requirements from stakeholders, relevant literature, and contextual knowledge.

In this case, the artifact is envisioned as a web application specifically designed to serve as a platform for learning and practicing programming as well as a research tool for researchers. It aims to provide students with a comprehensive environment to independently refine their programming skills while receiving real-time feedback and guidance, and to enable researchers to conduct studies on effective ways to use LLMs in programming education.

The requirements were elicited by considering the unique characteristics of the problem, the proposed solution, technological possibilities, previous research within the group [22, 36, 37, 45, 57], and input from the project team. Additionally, relevant academic literature in the domain were reviewed (as discussed in Chapter 2) to gain a better understanding of the problem and further refine the requirements.

The requirement definition activity resulted in the following functional and non-functional requirements.

4.2.1 Functional Requirements (FR)

FR1 **Programming Exercises:** The platform should provide a variety of programming exercises to help students practice and improve their coding skills.

FR2 **Clear Instructions:** Each exercise should have clear and concise instructions that outline the tasks students are expected to complete.

FR3 **Guided Problem Solving:** The platform should enable students to independently solve programming problems within the platform, including coding, submitting, and receiving feedback and guidance without the need for external software installation.

FR4 **Enhanced Coding Experience:** The platform should provide students with an advanced, familiar, and robust code editing experience that will help them learn and practice their programming skills effectively.

FR5 **Multiple Programming Languages:** The platform should support multiple programming languages to meet the diverse needs of students and educators.

FR6 **Classic Programming Feedback:** The platform should provide classic compiler/interpreter feedback for the students' submissions.

FR7 **LLM-Based Feedback:** The platform should integrate feedback generated by LLMs to enhance students' understanding of their submissions in case of an error and offer guidance for error correction.

FR8 **Submission, Feedback History, & Rating Storage:** The platform should store all previous code submissions for each assignment as well as the feedback received, allowing researchers to analyze students' behavior, identify common mistakes, and assess the generated automated feedback. It should also store user ratings for the generated feedback.

FR9 **Statistics Reporting:** The platform should provide researchers with statistical data on student performance, enabling them to identify areas for improvement in the learning process and experiment with new teaching strategies.

FR10 **Progress Tracking without User Registration:** The platform should implement a progress tracking feature that does not necessitate user authentication or registration. This feature should be designed to function seamlessly, thereby encouraging a broader user base to engage with the system.

FR11 **Mapping Users to LLMs:** To enable researchers to run A/B testing on different LLMs and study the quality of their feedback, each user on the platform should be mapped to a particular LLM, and all feedback generated for that user should be generated by the corresponding LLM.

4.2.2 Non-Functional Requirements (NR)

NR1 **Usability:** The platform should have a user-friendly interface and intuitive navigation to make it easy for students to interact with.

NR2 **Availability:** The platform should maintain high availability with minimal downtime to ensure uninterrupted access for students.

NR3 **Scalability:** The platform should be designed to accommodate a large number of students and support the load of Aalto University introductory classes.

NR4 **Responsiveness:** The platform should be responsive and efficient, even during peak usage times, to provide students with a seamless learning experience.

NR5 **Security:** Robust security measures should safeguard students' personal information, code submissions, and other sensitive data.

NR6 **Flexibility:** The platform's design should allow the easy integration of new types of exercises or sources of feedback. Examples of new exercise types could include parsons problems (where students reorder code fragments) or solution comparison exercises (where students compare different solutions to a problem). Additionally, the platform should be able to accommodate new LLMs released in the future.

In the next chapter, we will discuss how these requirements influenced the design decisions in the platform's development phase.

Chapter 5

Design and Development

Translating the identified requirements into a functional web application requires careful consideration of how to best meet the specified criteria. This chapter discusses the design and development of the proposed artifact: a web application for facilitating programming learning through LLM feedback. It provides an overview of the design and development process, including brainstorming sessions with the LeTech research group at Aalto University to generate ideas for addressing the identified requirements. The chapter then explores the design decisions that emerged from these sessions, providing detailed insight into their implementation. Finally, it describes the overall architecture of the web application, highlighting its key components and their functions.

5.1 Design and Development of Artifact

The activity of *Design and Development of Artifact* involves transitioning from initial solution concepts to a fully realized artifact [31]. This activity can be broken down into four distinct sub-activities: Imagine and Brainstorm, Assess and Select, Sketch and Build, and Justify and Reflect.

In the initial phase, *Imagine and Brainstorm*, designers engage in creative idea generation, either by conceiving new concepts or refining existing ones. Subsequently, in the *Assess and Select* phase, meticulous evaluation of these generated ideas leads to the selection of one or more concepts for further development. This step requires a convergent approach, using analytical assessments to pinpoint the most promising ideas. The preceding *Imagine and Brainstorm* phase can be likened to constructing a solution space brimming with potential ideas, while *Assess and Select* serves to prune and refine these ideas within that space. The dynamic interplay between these sub-activities guides the trajectory towards realizing the final artifact.

The *Sketch and Build* sub-activity involves creating a comprehensive sketch of the artifact based on selected ideas, outlining both core functions and overall structure. This sketch serves as a blueprint guiding the artifact's development and facilitates communication among stakeholders. Conversely, in the *Justify and Reflect* sub-activity, designers validate and introspectively analyze design choices, documenting a rationale that explains decisions made, alternative considerations, and the underlying arguments. This design rationale supports communication within projects and aids future design endeavors by offering insights and lessons from previous decisions.

5.2 Brainstorming

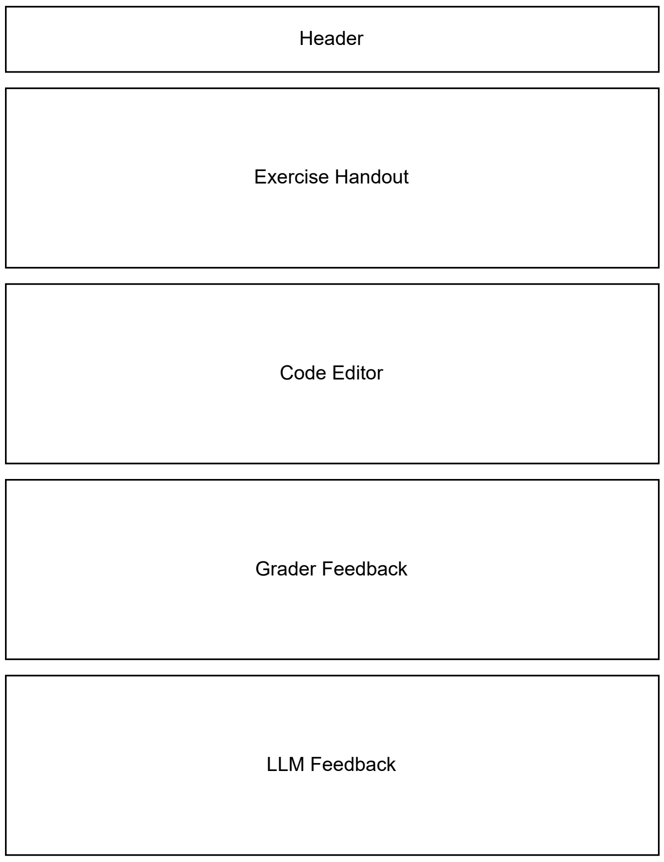


Figure 5.1: Initial UI sketch outlining key components of the platform, including a Header, Exercise Handout, Code Editor, Grader Feedback, and LLM Feedback areas. The components are vertically aligned.

During the development of the platform, the *Imagine and Brainstorm* activities were executed through collaborative brainstorming sessions involving the LeTech research group members. These brainstorming sessions were used to generate ideas to address the requirements identified in Section 4.2. The team used empathetic walkthroughs and sample sketching during these sessions to envision the user experience. *Empathetic thinking* is a practice-based approach in which a designer tries "to see the world through the eyes of others, understand the world through their experiences, and feel the world through their emotions" [8]. The team considered the perspective of an imaginary student when opening the platform and deliberated on what elements the student should encounter and the kind of feedback the platform should deliver. Figure 5.1 depicts an initial sketch of the UI proposed during one of the brainstorming sessions.

5.3 Design Decisions

The following subsections explore the solutions and design decisions that emerged from the brainstorming sessions, providing a detailed insight into their implementation. Each subsection addresses specific functional requirements (FR) or non-functional requirements (NR), outlining the corresponding design decision and implementation details. In the below context, "platform" and "web application" are used as interchangeable terms, referring to the same solution being discussed.

5.3.1 FR1 (Programming Exercises), FR2 (Clear Instructions), FR3 (Guided Problem Solving)

The Functional Requirements FR1, FR2, and FR3 are addressed by developing an end-to-end web application, as shown in Figure 5.2. The application consists of a frontend component that displays programming exercises and facilitates code submissions, a backend component that serves the frontend with necessary data related to exercises, submissions, and feedback, and a database to persist this information. The database stores crucial exercise details, including handouts, starter codes, test cases, and sample solutions. Figure 5.3 shows the design of the database schema.

FR2 is met by ensuring that the instructions are clear, concise, and easy to understand. The platform UI provides detailed handouts and starter codes for each exercise. The handout also contains examples of input and expected output. It also uses markdown format to highlight the relevant details. The application screenshots in Chapter 6 highlights this in detail.

FR3 is facilitated by the platform's ability to provide students with immediate feedback on their code submissions. This feedback, generated by a grader and complemented by LLM-based insights, helps students identify and rectify errors independently. The implementation of these feedback mechanisms are discussed in detail in Section 5.3.4 & Section 5.3.5.

The research group recommended using Astro[1] and Svelte[2] for frontend development, Deno[3] for backend implementation, and PostgreSQL[4] for the database. These technologies were chosen because they are already used to develop other applications in the research group, and they offer seamless integration potential for the future.

5.3.2 FR4 (Enhanced Coding Experience)

To meet FR4 and enhance the coding experience within the platform, the integration of a code editor was deemed necessary. Options like Ace[5] Editor and Monaco Editor[6] were discussed as potential choices due to their efficiency and user-friendly code editing environments.

Monaco Editor was selected because it powers VS Code[7], a widely-used Integrated Development Environment (IDE). This association with VS Code, a highly regarded IDE, underscores Monaco Editor's credibility and robustness, making it a reliable choice for integration. This provides students with an advanced and familiar code editing experience.

5.3.3 FR5 (Multiple Programming Languages)

To limit the scope of the project, it was decided that the platform would initially support a single programming language, Python. This choice aligns with Python's versatility and widespread use in introductory programming courses. However, the platform's architecture is designed to facilitate the seamless integration of additional programming languages in the future.

A Python grader image is used to execute and test Python code. This image utilizes a Python interpreter to run the code and unit tests. This image streamlines the integration of other programming languages by requiring only

[1] `https://astro.build/`
[2] `https://svelte.dev/`
[3] `https://deno.com/`
[4] `https://www.postgresql.org/`
[5] `https://ace.c9.io/`
[6] `https://microsoft.github.io/monaco-editor/`
[7] `https://github.com/microsoft/vscode`

the creation of a new grader image specific to the chosen language. The grader image is further elaborated in Section 5.4.

The platform's design employs a modular approach, where the core functionality is separated from language-specific components. This modularity enables the creation of grader images tailored to specific programming languages without altering the underlying platform infrastructure. The grader image is a self-contained execution environment that encapsulates the language compiler, interpreter, and testing framework, allowing the platform to execute and evaluate code written in the supported language.

To integrate a new programming language, the platform requires the development of a corresponding grader image. This image would encapsulate the language-specific tools and libraries necessary for compiling, interpreting, and testing code written in that language. Once the grader image is integrated, the platform will be able to handle exercises and submissions written in that language, enabling students to learn and practice programming in a diverse range of languages.

5.3.4 FR6 (Classic Programming Feedback)

To assess the correctness of code submissions, the platform uses a combination of the Python interpreter and customized test cases. The test cases, which are stored within the database alongside the exercise handouts, are designed to thoroughly evaluate the functionality of student-written code.

The Python interpreter, the platform's primary tool for executing and evaluating code, runs the student's submissions against these test cases. Upon submission, the platform processes the code and compares the output against the expected results specified in the test cases. Based on this comparison, the platform generates the classic programming feedback, highlighting any syntax errors in the student's code or functional discrepancies in the output.

The grader, a crucial component of the platform's architecture, as mentioned in Section 5.3.3, orchestrates this testing process. It seamlessly integrates the Python interpreter, test cases, and generates the interpreter feedback, ensuring a smooth and efficient evaluation of code submissions.

5.3.5 FR7 (LM-Based Feedback)

To meet this requirement, the research group proposed the use of three candidate Large Language Models (LLMs): GPT-4[8], StarCoder[9], and Llama 2[10]. GPT-4, a proprietary LLM developed by OpenAI, was recommended alongside open-source alternatives, StarCoder and Llama 2. However, due to the limitations of the open-source alternatives, StarCoder and Llama V2, which lacked readily available functional APIs, the decision was made to integrate GPT-4. The availability of a stable and reliable API for GPT-4 made it the most viable choice for incorporating LLM-based feedback into the platform.

5.3.6 FR8 (Submission, Feedback History, & Rating Storage), FR9 (Statistics Reporting)

Requirements FR8 and FR9 were satisfied through a careful design of the platform's database architecture, which seamlessly stores and manages various data elements related to exercises, submissions, feedback, ratings, and user information.

The platform's database table, `exercises`, serves as a repository of problems, storing essential information such as exercise handouts, starter codes, test cases, and sample solutions. The exercise submission data is stored in the `exercise_submissions` table. Feedback data provided by the platform's grader and LLM is stored in the `exercise_submission_feedbacks` table. User ratings for feedback are recorded in the `exercise_submission_feedback_ratings` table. User information is stored in the `users` table.

The database schema (refer Figure 5.3) enables efficient retrieval and analysis of data, facilitating the generation of comprehensive statistics on student performance and progress. The `exercise_submissions` table references the corresponding user UUID for each submission, allowing for seamless cross-referencing and aggregation of data across tables. This data can be utilized to track individual student progress, identify areas of strength and weakness, and personalize the learning experience.

Furthermore, the database schema supports advanced analytics by tracking users' progress using the `knowledge_components`, `exercise_knowledge_components`, and `user_knowledge_component_status` tables. This data can be used for Knowledge Tracing [40], a technique that analyzes student interactions to infer their understanding of specific

[8]https://openai.com/gpt-4

[9]https://huggingface.co/blog/starcoder

[10]https://ai.meta.com/llama/

concepts and provide targeted interventions. However, to limit the scope of the thesis, these functionalities are not implemented in the initial version of the platform, but are planned for future iterations.

5.3.7 FR10 (Progress Tracking without User Registration)

Research has shown that optional user registration can boost user engagement and participation [27]. In the current system, each user is assigned a UUID upon their initial access to the platform, which is then stored in their browser's local storage. This UUID is used to track the user's progress, enabling them to explore the platform without the initial commitment of creating an account. This approach helps in reducing friction and makes the platform more inviting.

5.3.8 FR11 (Mapping Users to LLMs)

To meet this requirement, each user is mapped to a given LLM upon their initial access to the platform. This information is stored in an in-memory Map data structure in the backend. The Map contains the UUIDs of the corresponding users and the name of the mapped LLM as key-value pairs. Whenever the user requests LLM feedback, the appropriate LLM is found by looking up the Map using the user's UUID as a key. Currently, the users are assigned to LLMs in a round-robin fashion.

5.3.9 NR1 (Usability)

To ensure a user-friendly interface and intuitive navigation, we explored the use of UI libraries and CSS frameworks. These resources offer pre-designed components, styles, and layouts that can streamline the frontend development process. Incorporating these libraries enhances the platform usability, providing students with a visually appealing, consistent, and easy-to-navigate interface.

Our team evaluated several CSS component libraries, all of which were Tailwind CSS[11] component libraries. Tailwind CSS is recognized for its utility-first approach, allowing for swift styling and layout adjustments by applying atomic classes directly within the HTML markup. The libraries

[11]https://tailwindcss.com/

we assessed included DaisyUI[12], Preline UI[13], Wind UI[14], and Material Tailwind[15].

During the evaluation phase, we considered the breadth and activity of the component libraries available for platform development. After careful consideration, DaisyUI, a Tailwind CSS plugin, emerged as our choice. This decision was based on several factors, including its vibrant and active community, extensive collection of readily available components, and proven track record as more than just a proof of concept. DaisyUI ensures a consistent and visually appealing design across the entire platform. Furthermore, its open-source nature, active development involving over 100 contributors, substantial 24k GitHub stars, and a user base exceeding 107k contributed to our selection. The choice was also influenced by the presence of an MIT license, which permits unrestricted use and distribution.

5.3.10 NR2 (Availability), NR3 (Scalability)

To ensure high availability and scalability, the platform leverages containerization with Docker[16]. This approach involves packaging platform components into self-contained units called Docker containers. For the current version, we used Docker and Docker Compose to create a manageable containerized environment.

Docker containers offer an additional layer of flexibility, enabling seamless integration with a diverse range of container orchestration solutions that may be needed in the future. Among the potential container orchestration options, Kubernetes[17] (commonly referred to as k8) stands out as a viable choice for scaling and orchestrating containers. It enhances the platform's scalability and reliability by efficiently managing containerized applications.

Docker Swarm is another container orchestration solution that can be explored to address specific scalability and high availability needs. These containerization and orchestration technologies provide the flexibility to adapt and expand the platform's infrastructure as required to meet evolving user demands.

[12]`https://daisyui.com/`
[13]`https://preline.co/`
[14]`https://wind-ui.com/`
[15]`https://www.material-tailwind.com/`
[16]`https://www.docker.com/`
[17]`https://kubernetes.io/`

5.3.11 NR4 (Responsiveness)

Achieving responsiveness requires the use of real-time communication mechanisms. Several mechanisms, including Server-Sent Events[18] (SSE), WebSockets [19], and Long Polling, were considered as potential candidates for fulfilling this requirement. These technologies enable communication between the platform and users' browsers, ensuring timely updates and interactions.

SSE allows the server to push updates to clients as they occur, ensuring instant feedback and dynamic content updates. WebSockets establish persistent connections, enabling real-time data exchange between the server and clients. Long Polling involves requesting updates from the server at regular intervals, maintaining an open connection and allowing rapid data exchange.

Among the options discussed above, SSE was chosen. This mechanism aligns well with the application architecture, where grader data flow primarily occurs from the server to the client. SSE efficiently handles this data exchange.

5.3.12 NR5 (Security)

Data security is upheld by storing sensitive information, including students' personal details, code submissions, and feedback, within Aalto University's servers. The University's established infrastructure and security protocols provide a trusted environment for safeguarding user data. The use of secure protocols, access controls, and encryption mechanisms ensures that student information remains confidential and protected from unauthorized access or breaches. By housing data within Aalto servers, the platform fulfils its commitment to maintaining the highest level of security and privacy standards.

5.3.13 NR6 (Flexibility)

Reusable frontend components and a generic database schema make the platform flexible. Reusable frontend components allow developers to create modular and adaptable UI elements that can be reused throughout the platform, reducing development effort and promoting design consistency.

A generic database schema ensures that the platform can accommodate new exercise types and feedback sources without major changes. Additionally, the JSONB[20] data type provides flexibility in representing diverse hand-

[18] `https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events`

[19] `https://developer.mozilla.org/en-US/docs/Web/API/WebSocket`

[20] `https://www.postgresql.org/docs/16/datatype-json.html`

out structures, grading data, prompt data, and rating data without requiring frequent schema adjustments, ensuring efficient data management. This schema flexibility supports the integration of diverse exercise formats and feedback sources in the future, enabling the platform to evolve and expand its offerings while maintaining a cohesive structure. Figure 5.3 shows the database schema.

For example, the *feedback_data* from different *feedback_sources* can have different formats and structures, and the *prompt_data* needed for a given feedback source, such as GPT-4, can differ from what is needed for another LLM. Instead of creating individual tables for each feedback source, we use a single, generic table called `exercise_submission_feedbacks`. This table can accommodate data from different types of feedback sources without any changes to its schema. The configured *feedback_source* type determines what kind of data is stored in the table. We use a similar schema design approach for the `exercises` table, which allows us to add new types of exercises to the platform in the future.

5.4 Artifact Architecture

The problem explication and requirement elicitation made it clear that the artifact would be a web application. Specifically, a web application with a frontend to display exercises and accept code submissions, a backend to serve the frontend with necessary data regarding exercises, submissions, feedback, and a database to store this information is envisioned as the final solution.

Figure 5.2 illustrates the overall architecture of the application and how the users interact with the web application via a browser. Below, we provide detailed descriptions of each component of the application and its function.

nginx: The application is served via an NGINX[21] web server container. NGINX serves as an intermediary between client requests, such as those from web browsers, and the backend servers. It receives incoming requests from clients and forwards them to the appropriate backend servers responsible for fulfilling these requests.

programming-ui: This container serves as the frontend and is developed using Astro and Svelte. The programming-ui container delivers the frontend build to users' web browsers through NGINX. This build includes the JavaScript bundle necessary for user interactions. When users engage with the application, it sends requests to the backend via NGINX to retrieve relevant data, which is then displayed to the user.

[21]`https://www.nginx.com/`

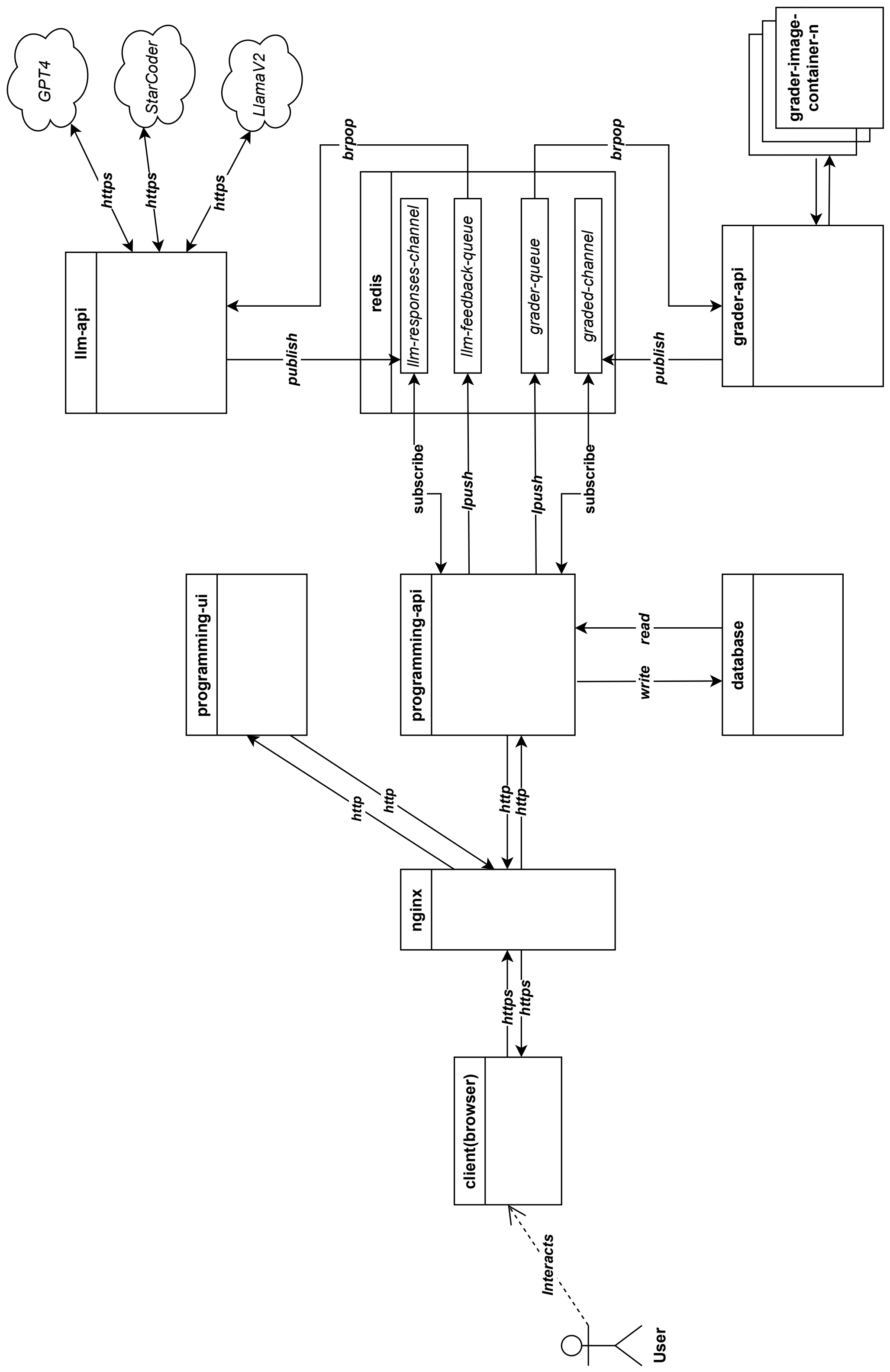


Figure 5.2: Architecture diagram.

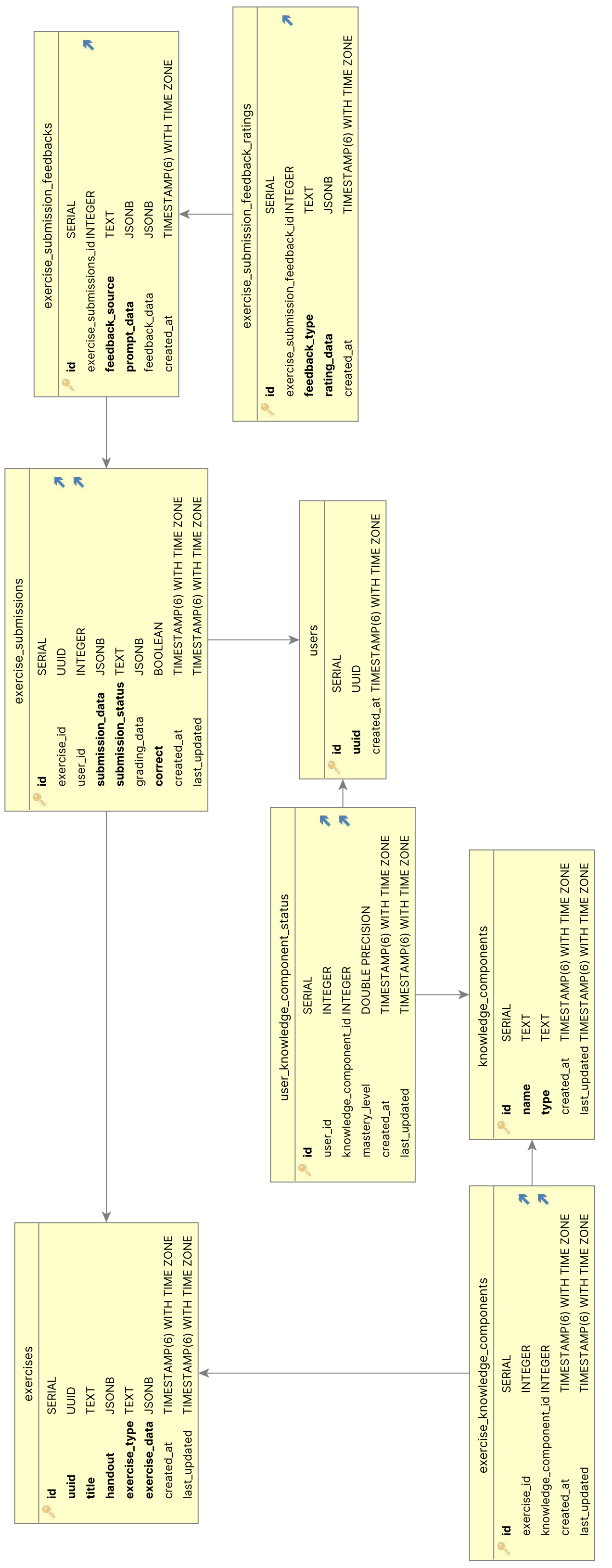

exercise_submission_feedbacks
id SERIAL
exercise_submissions_id INTEGER
feedback_source TEXT
prompt_data JSONB
feedback_data JSONB
created_at TIMESTAMP(6) WITH TIME ZONE
exercise_submission_feedback_ratings
id SERIAL
exercise_submission_feedback_id INTEGER
feedback_type TEXT
rating_data JSONB
created_at TIMESTAMP(6) WITH TIME ZONE
exercise_submissions
id SERIAL
exercise_id UUID
user_id INTEGER
submission_data JSONB
submission_status TEXT
grading_data JSONB
correct BOOLEAN
created_at TIMESTAMP(6) WITH TIME ZONE
last_updated TIMESTAMP(6) WITH TIME ZONE
users
id SERIAL
uuid UUID
created_at TIMESTAMP(6) WITH TIME ZONE
user_knowledge_component_status
id SERIAL
user_id INTEGER
knowledge_component_id INTEGER
mastery_level DOUBLE PRECISION
created_at TIMESTAMP(6) WITH TIME ZONE
last_updated TIMESTAMP(6) WITH TIME ZONE
knowledge_components
id SERIAL
name TEXT
type TEXT
created_at TIMESTAMP(6) WITH TIME ZONE
last_updated TIMESTAMP(6) WITH TIME ZONE
exercises
id SERIAL
uuid UUID
title TEXT
handout JSONB
exercise_type TEXT
exercise_data JSONB
created_at TIMESTAMP(6) WITH TIME ZONE
last_updated TIMESTAMP(6) WITH TIME ZONE
exercise_knowledge_components
id SERIAL
exercise_id INTEGER
knowledge_component_id INTEGER
created_at TIMESTAMP(6) WITH TIME ZONE
last_updated TIMESTAMP(6) WITH TIME ZONE

Figure 5.3: Database schema.

programming-api: This container functions as the backend and is implemented in Deno. The programming-api container is responsible for processing incoming requests received from NGINX. This container handles request processing, forwarding data to the llm-api or grader-api containers through the redis container as needed, and interacts with the database container to store or retrieve data required for request handling.

database: The database container uses a PostgreSQL database to store all relevant information. The database schema is depicted in Figure 5.3.

redis: The redis container utilizes a Redis[22] image. Redis, which stands for Remote Dictionary Server, is an open-source, in-memory data structure store capable of serving as a caching mechanism, database, and message broker. In this implementation, Redis is used as a message broker for passing grading and LLM feedback requests and responses among the respective services. Redis supports various data structures, including strings, lists, sets, sorted sets, hashes, bitmaps, and hyperloglogs, enabling more complex operations beyond simple key-value storage. In our setup, we employ two Redis lists, namely llm-feedback-queue and grader-queue, as a queuing mechanism to store grading and LLM feedback requests originating from the programming-api.

grader-image: The grader-image is based on Alpine Linux[23] and includes essential dependencies like Bash[24], Python3, and pip[25]. This image is used for grading assignment submissions.

grader-api: The grader-api creates and runs a temporary docker container using the grader-image. This temporary grading container is run with the submission code, test cases, and function names. Once the grading results are available the grader-api copies and processes the grading results and subsequently removes the temporary container, performing the necessary cleanup. The grader-api processes submissions one by one from the grader-queue Redis list, grading each submission using the grader-image. After grading, it publishes the results to the graded-channel within the Redis container. The programming-api subscribes to this channel and consumes the graded results.

[22]`https://redis.io/`

[23]`https://www.alpinelinux.org/about/`

[24]`https://www.gnu.org/software/bash/`

[25]`https://pypi.org/project/pip/`

llm-api: The llm-api container manages LLM feedback requests. It reads requests from the llm-feedback-requests Redis list and queries the requested LLM (e.g., GPT-4, StarCoder, Llama 2). The results obtained from the corresponding LLMs are published to the llm-responses-channel. The programming-api, subscribed to this channel, consumes and processes the LLM feedback responses.

The next chapter demonstrates the developed platform and its current capabilities. Furthermore, it also details how the LLMs are prompted.

Chapter 6

Application Demonstration

This chapter discusses the *Demonstrate Artifact* phase of the DSRM, providing a high-level overview of the platform and highlighting its key features and benefits. It then focuses on the programming space UI and how users can interact with its different components. Finally, the chapter discusses how the platform utilizes language models to generate feedback for users.

6.1 Demonstrate Artifact

The *Demonstrate Artifact* phase is the fourth step in the DSRM framework and plays a pivotal role in showcasing the practical utility of a developed artifact within a specific context. This phase involves providing descriptive insights into the functioning of the artifact within the chosen context, alongside explanations for why it operates as it does [31].

The *Demonstrate Artifact* activity involves two sub-activities. The first sub-activity involves selecting or creating a case for artifact application, considering options like fictitious, well-documented, or real-life cases, while the second sub-activity entails applying the artifact to the chosen case and documenting the results [31].

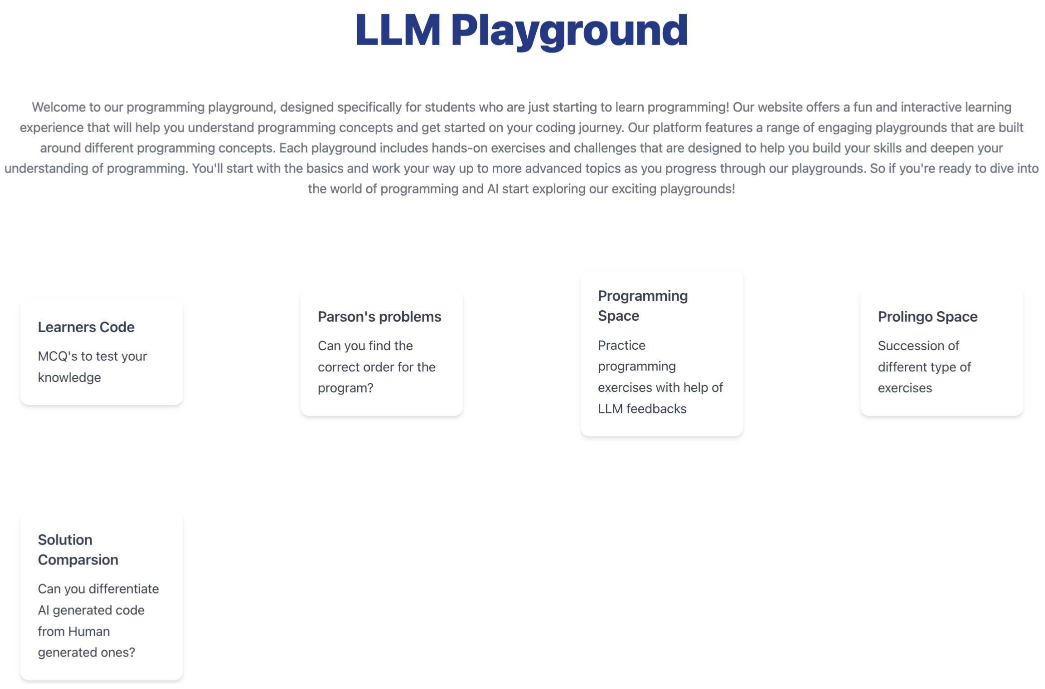
A successful demonstration confirms that the artifact can indeed address specific facets of a problem within a real-world or illustrative case, serving as a preliminary form of evaluation. Demonstrating effectiveness in one case suggests potential applicability in others and is a persuasive tool for communicating the artifact's concept and capabilities to an audience.

Guidelines for successful demonstration include justifying the choice of the case by explaining its representativeness and suitability as a test bed for assessing the artifact's capabilities. Additionally, it's crucial to clearly define and communicate the extent to which the artifact is being tested, specifying the components and functionalities utilized during the demonstration.

In essence, the *Demonstrate Artifact* phase serves as a critical bridge between the conceptualization and practical application of the artifact, facilitating its validation, communication, and eventual deployment within a real-world context.

The artifact developed as part of this thesis primarily targets students as its main user group. Therefore, in the following context, the terms “user” and “student” are used interchangeably to refer to the individuals using the application.

6.2 Platform Overview



LLM Playground

Welcome to our programming playground, designed specifically for students who are just starting to learn programming! Our website offers a fun and interactive learning experience that will help you understand programming concepts and get started on your coding journey. Our platform features a range of engaging playgrounds that are built around different programming concepts. Each playground includes hands-on exercises and challenges that are designed to help you build your skills and deepen your understanding of programming. You'll start with the basics and work your way up to more advanced topics as you progress through our playgrounds. So if you're ready to dive into the world of programming and AI start exploring our exciting playgrounds!

Learners Code
MCQ's to test your knowledge

Parson's problems
Can you find the correct order for the program?

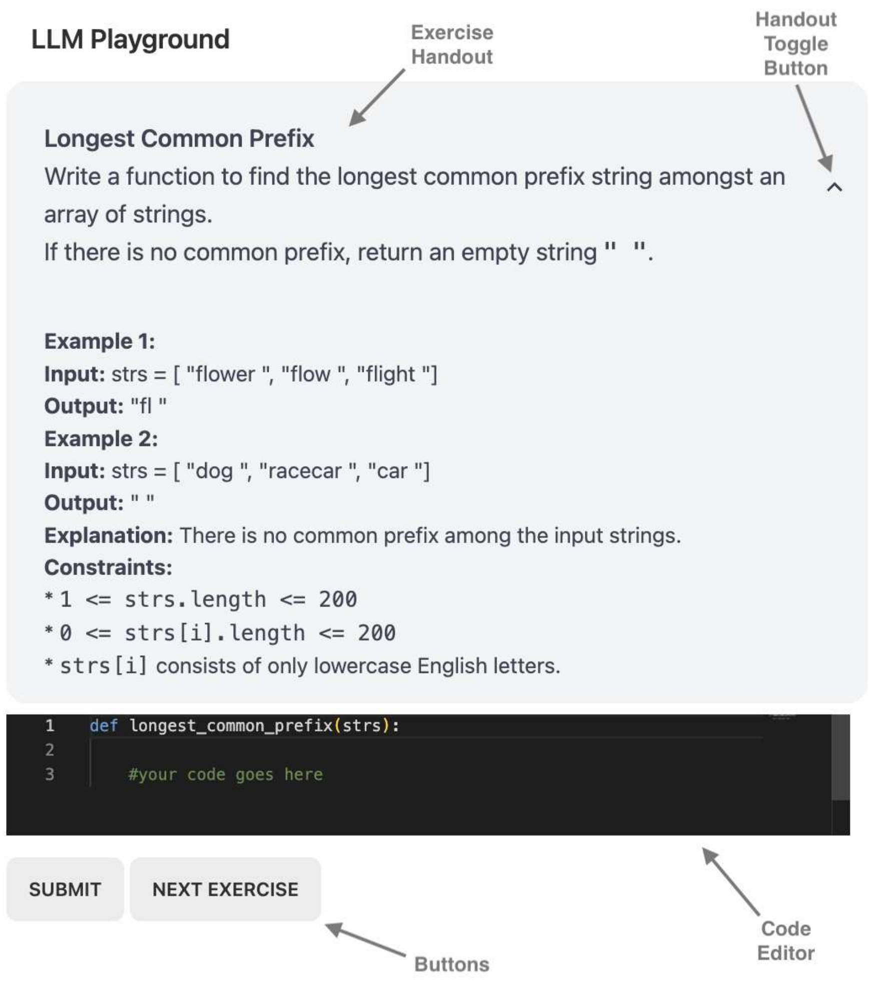
Programming Space
Practice programming exercises with help of LLM feedbacks

Prolingo Space
Succession of different type of exercises

Solution Comparsion
Can you differentiate AI generated code from Human generated ones?

Figure 6.1: Landing page of the application with a brief description of the application and cards representing distinct learning spaces.

Figure 6.1 presents the platform’s landing page, which showcases a variety of cards representing distinct learning spaces specifically designed to facilitate programming education. Each learning space caters to a specific exercise type. For example, the *Parson’s Problem* emphasizes sequence and logic comprehension, while the *Programming Space* focuses on hands-on coding practice. These exercises are accompanied by clear instructions, enabling problem-solving without the need for external software installation. Users receive both classic and LLM-based feedback to enhance their learning experience. It is important to note that this thesis focused primarily on the *Programming Space*, with the subsequent sections delving into its use cases in more detail.



Longest Common Prefix
Write a function to find the longest common prefix string amongst an array of strings.
If there is no common prefix, return an empty string " ".

Example 1:
Input: strs = ["flower ", "flow ", "flight "]
Output: "fl "
Example 2:
Input: strs = ["dog ", "racecar ", "car "]
Output: " "
Explanation: There is no common prefix among the input strings.
Constraints:
* `1 <= strs.length <= 200`
* `0 <= strs[i].length <= 200`
* `strs[i]` consists of only lowercase English letters.

```
def longest_common_prefix(strs):

    #your code goes here
```

Figure 6.2: A view of the programming space with an open exercise handout alongside the code editor. Arrows highlight key elements such as the exercise details in the handout, the code editor for writing code, a handout toggle button for displaying or hiding the detailed exercise descriptions in the handout, a *"SUBMIT"* button for code submission, and a *"NEXT EXERCISE"* button to retrieve the next exercise.

6.3 Programming Space

The programming space is a learning environment that provides students with programming exercises, feedback, and assistance from an AI tutor. The following subsections delve into the different components of the programming space UI and details how the users can interact with it.

6.3.1 Handout & Code Editor

The programming space as depicted in Figure 6.2, presents programming exercises from the database. It displays the relevant exercise details from the handout including examples. This supports FR2 (Clear Instructions) by providing students with clear and concise instructions tailored to each exercise. The handout section also features a "Handout Toggle" button, which enables students to control the visibility of this information, ensuring they can focus on the level of guidance that best suits their learning needs.

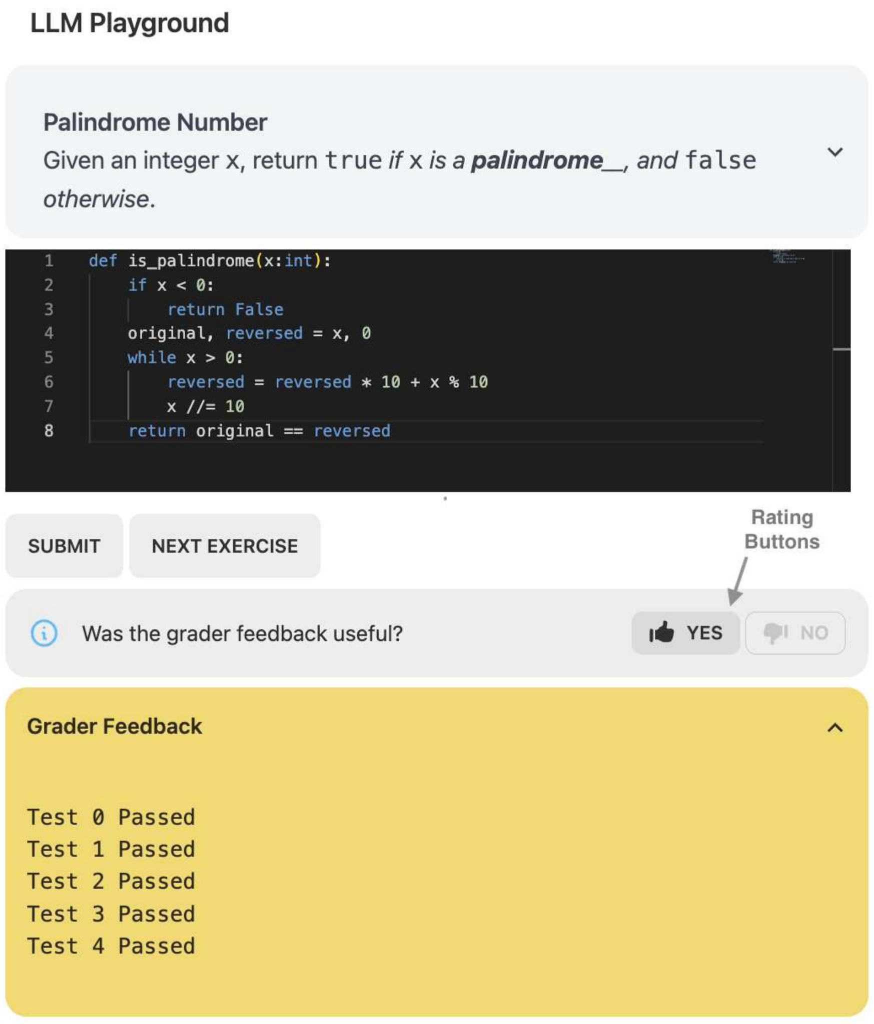
The exercise handout is followed by a Monaco code editor. This editor offers an array of features, including code highlighting, to facilitate a smooth and efficient coding experience, aligning with FR4 (Enhanced Coding Experience). Students can use its capabilities to craft and refine their code solutions.

The programming space incorporates two interactive buttons, namely, *"SUBMIT"* and *"NEXT EXERCISE"*. The *"SUBMIT"* button enables students to submit the code currently present in the code editor, initiating the grading process for their solution while the *"NEXT EXERCISE"* fetches the next exercise from the repository. Loader animations accompanying these buttons serve as visual indicators of the ongoing progress with a submission grading or exercise fetching. In addition to this, both buttons are also enabled by default, which means that the students can skip an exercise if it seems difficult for them to solve. This helps promote a continuous learning journey.

6.3.2 Grader Feedback

The Grader feedback section plays an important role in the programming space, as it is the primary channel through which students receive information about their exercise submissions. Figure 6.3, Figure 6.4, and Figure 6.5 showcases the Grader feedback section within the platform's user interface.

One of the standout features of the Grader feedback section is its ability to pinpoint and highlight syntax errors. When students encounter syntax issues in their code, the interpreter feedback is displayed prominently, helping them identify and rectify these errors. In addition to offering feedback on syntax errors, the Grader feedback section also provides a clear breakdown of the test results. It shows the number of test cases that were passed successfully and those that failed. This breakdown allows students to gauge the overall performance of their code, providing a detailed assessment of their understanding and application of the subject matter.



LLM Playground

Palindrome Number

Given an integer x, return `true` *if* x *is a* ***palindrome__****, and* `false` *otherwise.*

```
def is_palindrome(x:int):
    if x < 0:
        return False
    original, reversed = x, 0
    while x > 0:
        reversed = reversed * 10 + x % 10
        x //= 10
    return original == reversed
```

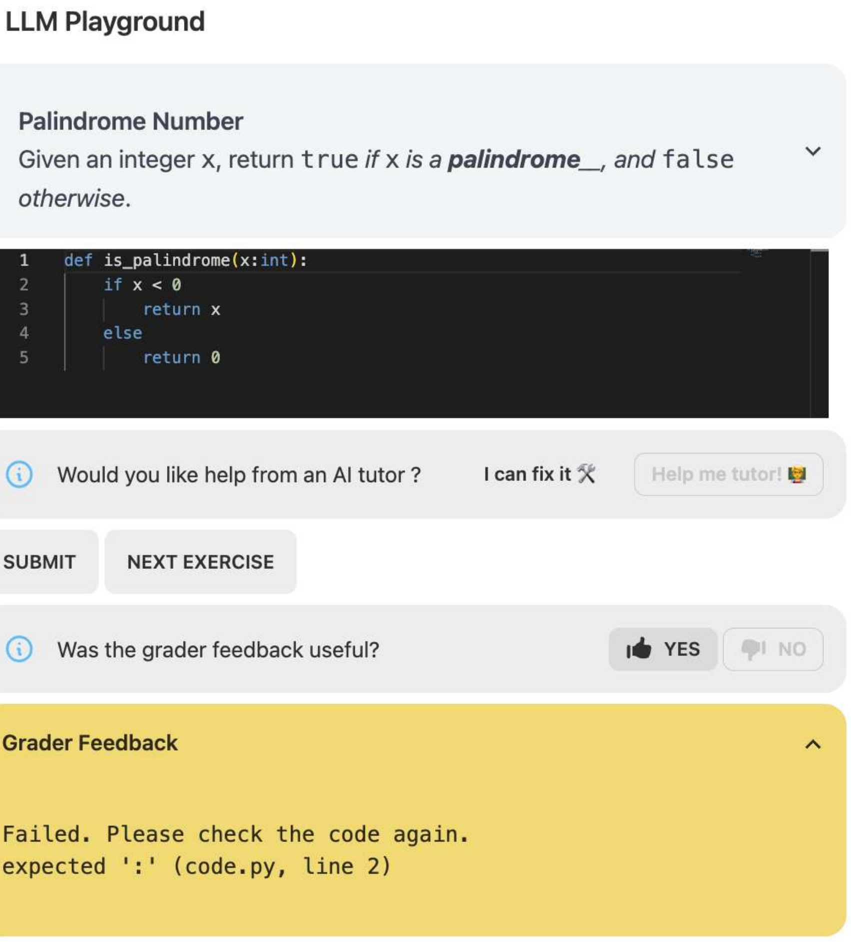
Was the grader feedback useful?

Grader Feedback

```
Test 0 Passed
Test 1 Passed
Test 2 Passed
Test 3 Passed
Test 4 Passed
```

Figure 6.3: A view of the programming space featuring Grader feedback. In this case, the Grader feedback section highlights that the submission is correct and passes all the test cases. The arrow indicates the rating buttons that users can use to rate the generated Grader Feedback.

To enhance the overall user experience, the platform provides users with the option to hide the Grader feedback section if they so desire. This feature ensures that students can tailor their learning experience to their preferences, allowing them to focus on different aspects of the test results or revisit the feedback at their convenience.


LLM Playground

Palindrome Number

Given an integer x, return true *if* x *is a **palindrome**_, and* false *otherwise*.

```
def is_palindrome(x:int):
    if x < 0
        return x
    else
        return 0
```

Would you like help from an AI tutor ? I can fix it 🛠 Help me tutor!

SUBMIT NEXT EXERCISE

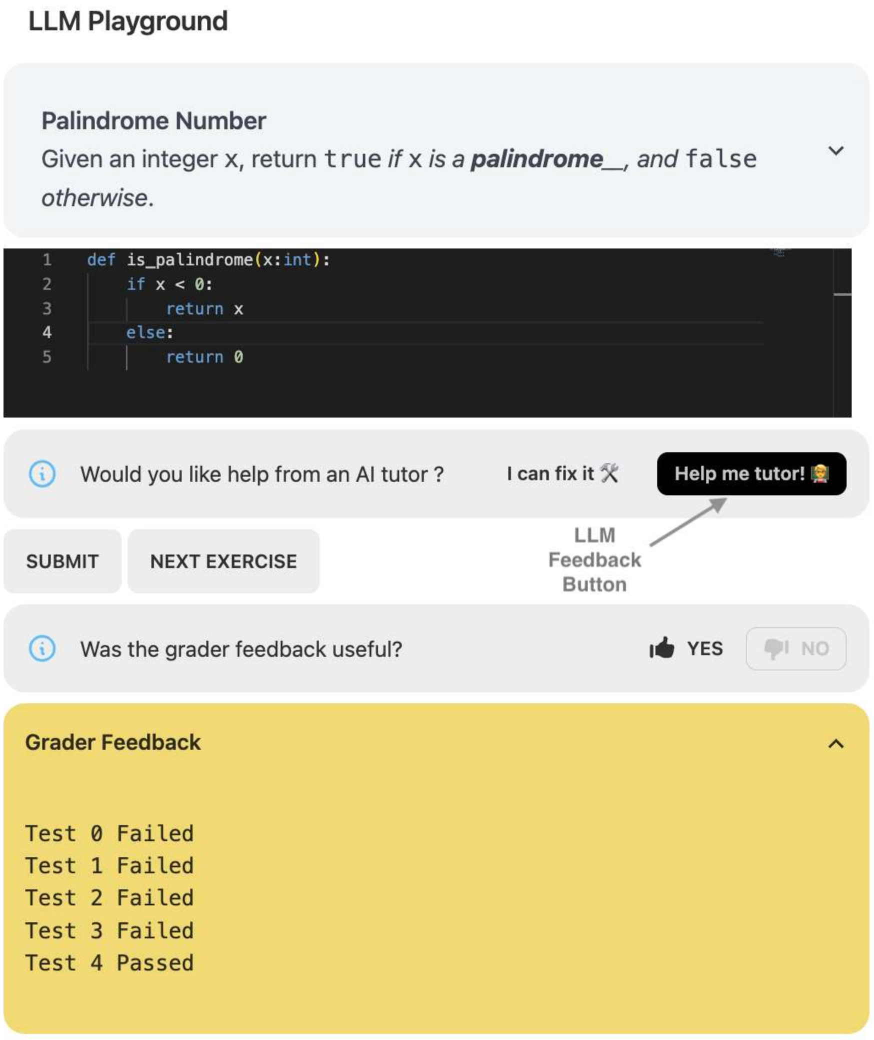
Was the grader feedback useful? YES NO

Grader Feedback

```
Failed. Please check the code again.
expected ':' (code.py, line 2)
```

Figure 6.4: A view of the programming space featuring Grader feedback, indicating a syntactically incorrect submission with guidance to check code for an expected ':' at line 2.

Moreover, the platform also values student input and offers students the opportunity to rate the feedback they receive. This can be done by selecting the appropriate option in response to the dialog, *"Was the grader feedback useful?"*. As illustrated in Figure 6.3, the dialog is placed right above the displayed Grader feedback and gives the option to choose between *"YES"* or *"NO"* to rate the feedback. If the user wants to give a positive rating for the feedback then they can choose *"YES"*, otherwise, if they want to give a negative rating for the feedback then they can choose *"NO"*. Choosing a rating option closes the dialog. These ratings are invaluable for researchers as they offer insights into the quality of the feedback provided.


LLM Playground

Palindrome Number
Given an integer x, return `true` *if* x *is a* ***palindrome__****, and* `false` *otherwise.*

```
def is_palindrome(x:int):
    if x < 0:
        return x
    else:
        return 0
```

```
Test 0 Failed
Test 1 Failed
Test 2 Failed
Test 3 Failed
Test 4 Passed
```

Figure 6.5: A view of the programming space featuring Grader feedback, highlighting test case failures for logically incorrect code submission. Notably, Test 0, Test 1, Test 2, and Test 3 have failed, and only Test 5 has successfully passed. The arrow indicates the "Help me tutor!" button, facilitating LLM feedback requests on the submitted code.

Figure 6.4 illustrates the Grader feedback on an incorrect submission with a syntax error. It asks the user to check and correct the syntax error, specifying the line number at which it occurred.

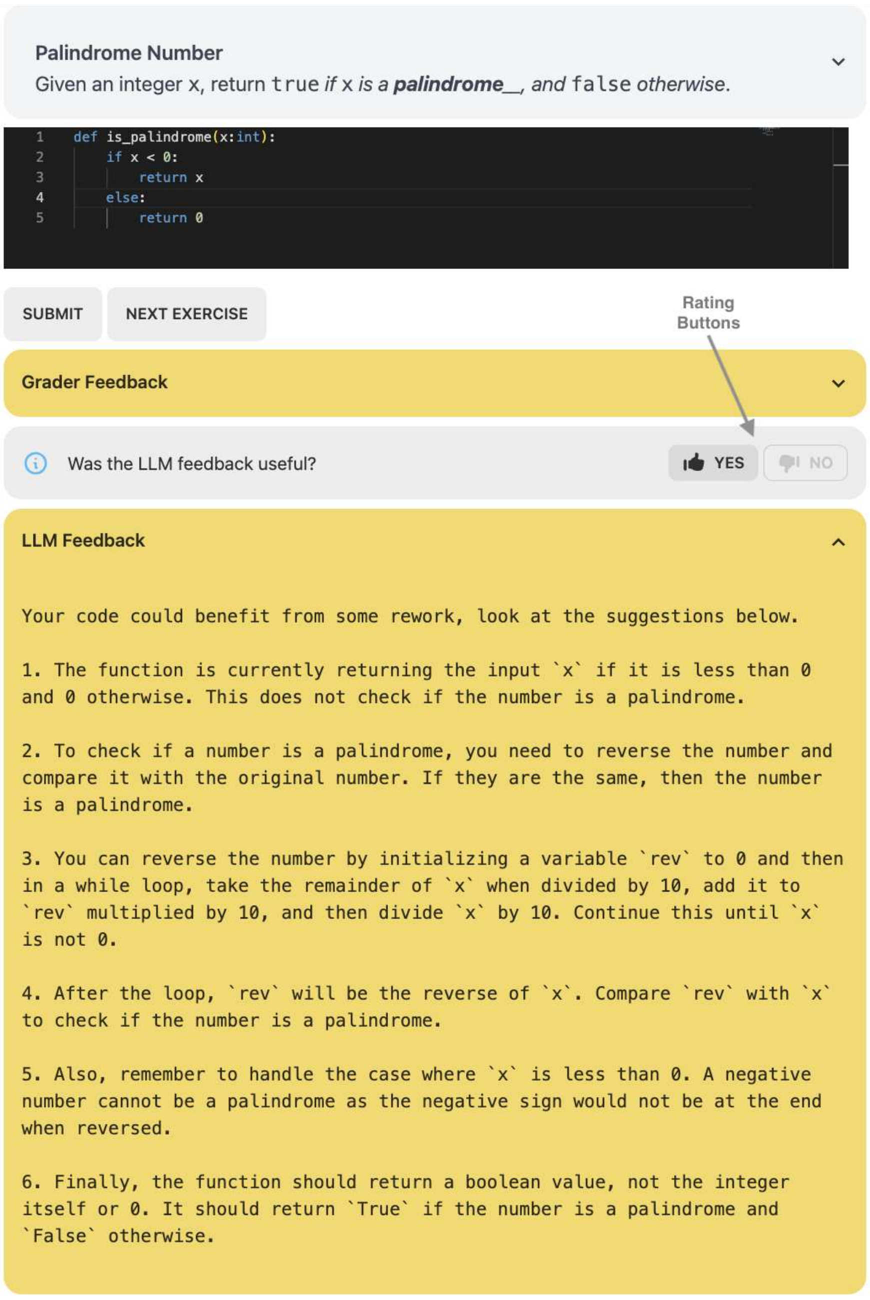
Figure 6.5 illustrates the Grader feedback that students receive when their code submission contains errors and have failed certain test cases. This feedback informs students about the number of test cases that passed and the ones that failed. In the current version of the application, the details of the test cases themselves are hidden from students. This decision is rooted in the belief that revealing them might tempt some students to adjust their submissions solely to pass the test cases, potentially misleading the grading system.

Additionally, in case of submission failures, the user interface includes a feature that allows students to request assistance from an AI tutor. This is made simple through a dialog box that asks, *"Would you like help from an AI tutor?"* This dialog, as depicted in Figure 6.5, is positioned just below the Code Editor display. It presents two options: *"I can fix it"* and *"Help me tutor!"* If a student believes they can resolve the errors independently based on the Grader feedback, they can choose *"I can fix it"*. On the other hand, if they find themselves stuck and in need of additional guidance, they can select *"Help me tutor!"*. Opting for the latter triggers a request to the backend to query the Language Model assigned to the student for further assistance. The dialog closes when the option is chosen.

6.3.3 LLM Feedback

The platform incorporates LLM feedback as an integral feature. This feedback is displayed below the Grader feedback section and is accessible to the students when they click on the *"Help me tutor!"* button. The process involves a backend request to an LLM, querying it with exercise handout, submission, and submission error details. The retrieved feedback from the LLM is then presented on the user interface. Similar to the Grader feedback, users have the option to toggle and hide the LLM feedback.

Figure 6.6 demonstrates LLM feedback from GPT-4 for incorrect submissions where there function is completely incorrect. In such cases, GPT-4 offers higher-level guidance on how to approach the problem. In this particular case, it provides instructions on writing a Python function to find whether a given number is a palindrome. The initial part of the feedback suggestions, numbered as "1," highlights the shortcomings of the current code, pinpointing the errors or issues therein. Subsequent suggestions, labeled as "3 - 6," provide a step-by-step breakdown of how to approach and solve the exercise correctly. Such comprehensive feedback not only identifies mistakes but also empowers the student with a clear path toward rectifying the errors and achieving a correct solution. It serves as a valuable learning aid, fostering



Given an integer x, return true *if* x *is a **palindrome__**, and* false *otherwise.*

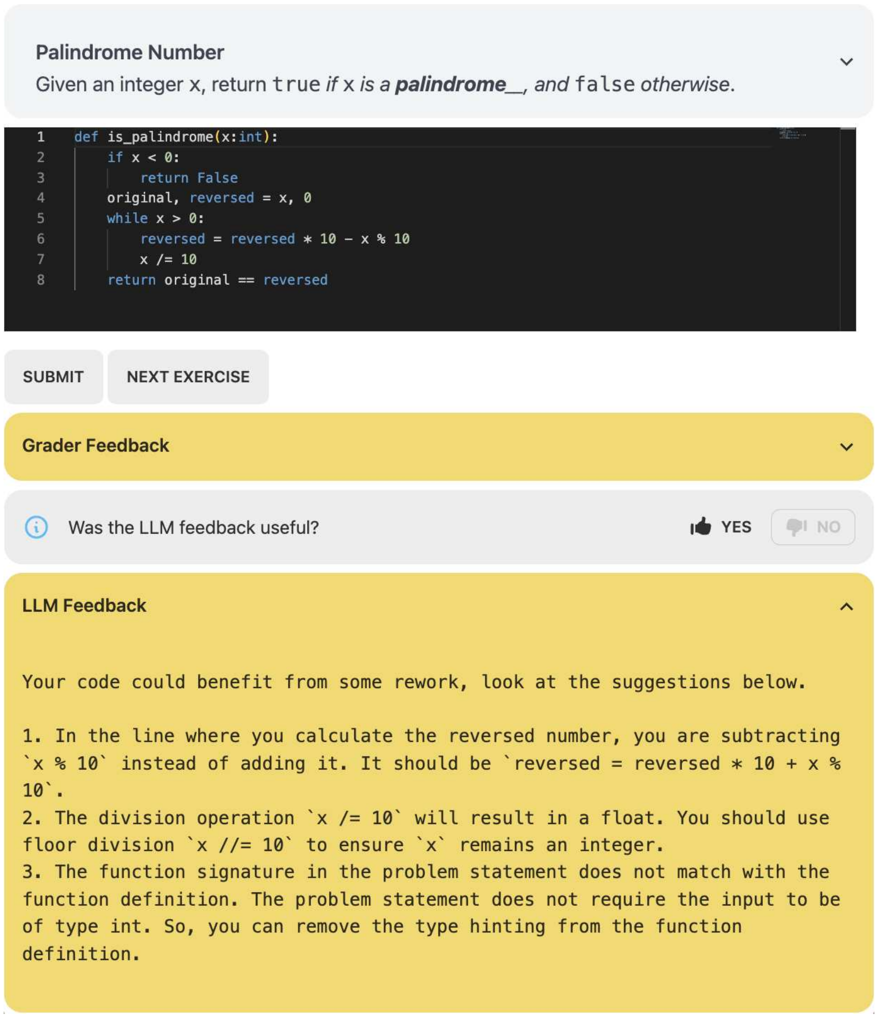
```
def is_palindrome(x:int):
    if x < 0:
        return x
    else:
        return 0
```

Was the LLM feedback useful?

```
Your code could benefit from some rework, look at the suggestions below.

1. The function is currently returning the input `x` if it is less than 0
and 0 otherwise. This does not check if the number is a palindrome.

2. To check if a number is a palindrome, you need to reverse the number and
compare it with the original number. If they are the same, then the number
is a palindrome.

3. You can reverse the number by initializing a variable `rev` to 0 and then
in a while loop, take the remainder of `x` when divided by 10, add it to
`rev` multiplied by 10, and then divide `x` by 10. Continue this until `x`
is not 0.

4. After the loop, `rev` will be the reverse of `x`. Compare `rev` with `x`
to check if the number is a palindrome.

5. Also, remember to handle the case where `x` is less than 0. A negative
number cannot be a palindrome as the negative sign would not be at the end
when reversed.

6. Finally, the function should return a boolean value, not the integer
itself or 0. It should return `True` if the number is a palindrome and
`False` otherwise.
```

Figure 6.6: A view of the programming space with LLM feedback, presenting six generic suggestions to address errors in an incorrect code. The arrow highlights the rating buttons that users can use to rate the generated LLM feedback.

both understanding and skill development in programming concepts.


Palindrome Number

Given an integer x, return `true` *if* x *is a* ***palindrome__****, and* `false` *otherwise.*

```python
def is_palindrome(x:int):
    if x < 0:
        return False
    original, reversed = x, 0
    while x > 0:
        reversed = reversed * 10 - x % 10
        x /= 10
    return original == reversed
```

```
Your code could benefit from some rework, look at the suggestions below.

1. In the line where you calculate the reversed number, you are subtracting
`x % 10` instead of adding it. It should be `reversed = reversed * 10 + x %
10`.
2. The division operation `x /= 10` will result in a float. You should use
floor division `x //= 10` to ensure `x` remains an integer.
3. The function signature in the problem statement does not match with the
function definition. The problem statement does not require the input to be
of type int. So, you can remove the type hinting from the function
definition.
```

Figure 6.7: A view of the programming space with LLM feedback, presenting three specific suggestions to address errors in an incorrect code.

Figure 6.7 illustrates LLM feedback for partially correct submissions containing logical errors. Here, GPT-4 provides specific guidance, addressing the specific logical flaws in the submitted code and offering clear instructions for rectification. The feedback suggestions "1" and "2" point out the exact logical errors in the code and explain how to correct them. However, there's another suggestion "3" that claims a mismatch between the function signature and the problem statement. This suggestion is incorrect because the problem statement clearly says the input is an integer. Similar issue with inaccurate suggestion in LLM response was also seen in earlier research using

GPT-3.5 by the research group [22]. It highlights the need to carefully review and improve LLM-generated feedback.

Similar to the Grader feedback rating, the platform enables users to rate the quality of feedback generated by the LLM, offering the choice of *"YES"* or *"NO"* in response to the question *"Was the LLM feedback useful?,"* which aids in evaluating the LLM's effectiveness in providing meaningful feedback to the users.

6.4 Querying Language Models

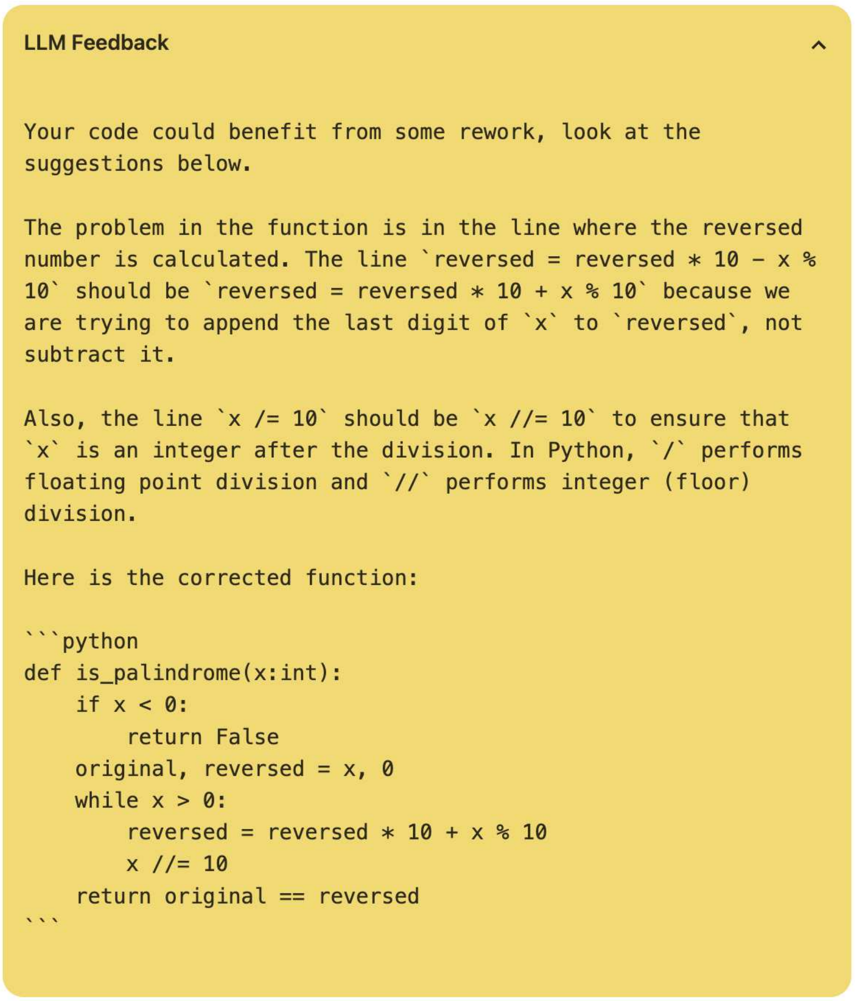
The application currently utilizes GPT-4 as the backbone for generating Language Model feedback. However, it's worth noting that the backend of the system is designed to be flexible and can be easily configured to use alternative language models with minimal adjustments satisfying the FR7 (LLM-Based Feedback). The current backend configuration assigns each user a specific LLM upon sign-up. Subsequently, all feedback requests from that user are directed to the assigned LLM. This approach ensures consistency in evaluating LLMs based on user ratings and interactions, and helps accomplish the research goals of the research group as mentioned in Section 1.1.

6.4.1 Query Parameters & Prompt

```
{
    model: "gpt-4",
    messages: [{ role: "assistant", content: <prompt> }],
    temperature: 0,
    max_tokens: 1500,
    top_p: 1.0,
    frequency_penalty: 0.0,
    presence_penalty: 0.0
}
```

Listing 6.1: The query parameters used to query the GPT-4 API

The code snippet in Listing 6.1 illustrates the parameters used when querying the GPT-4 API. The "model" parameter specifies that the GPT-4 model is being utilized. The "messages" parameter, consisting of an array of message objects, defines the conversation structure. The role "assistant" signifies that responses will originate from the assistant, while the content field captures user prompts. Several additional parameters are set to fine-tune the generation process: "temperature" is set to 0 to reduce text randomness, "max_tokens" limits text length to 1500 tokens, and "top_p" controls token diversity. Finally, "frequency_penalty" and "presence_penalty" are both set to 0.0, implying no specific word frequency or presence restrictions.



Your code could benefit from some rework, look at the suggestions below.

The problem in the function is in the line where the reversed number is calculated. The line `reversed = reversed * 10 - x % 10` should be `reversed = reversed * 10 + x % 10` because we are trying to append the last digit of `x` to `reversed`, not subtract it.

Also, the line `x /= 10` should be `x //= 10` to ensure that `x` is an integer after the division. In Python, `/` performs floating point division and `//` performs integer (floor) division.

Here is the corrected function:

```python
def is_palindrome(x:int):
    if x < 0:
        return False
    original, reversed = x, 0
    while x > 0:
        reversed = reversed * 10 + x % 10
        x //= 10
    return original == reversed
```

Figure 6.8: LLM feedback generated based on the initial prompt version (see Listing 6.2), providing detailed error explanations and correct code suggestions.

During the platform's development, different query prompts were experimented with. Initial prompt versions always returned complete answers, which did not align with the desired learning objectives. The code snippet in Listing 6.2 showcases one of these initial prompt versions for the same code submission in Figure 6.7, and the corresponding feedback from the LLM can be seen in Figure 6.8.

```
###
<handout_question>
<handout_description>
###
What is going wrong in the below function?
###
Buggy Python
<submission_code>
###
Suggestions
```

Listing 6.2: The initial version of the query prompt used for LLM feedback generation, comprising a problem handout, a question asking about code issues, the submitted code, and a request for suggestions.

Consequently, prompts were modified to explicitly instruct GPT-4 to provide suggestions rather than full solutions. The current prompt version is shown in the code snippet in Listing 6.3. The LLM feedback in Figure 6.6 & Figure 6.7 are generated using the current prompt version.

```
###
<handout_question>
<handout_description>
###
Give me only suggestions to fix bugs in the below function.
Don't give me a working solution or corrected function.
###
Buggy Python
<submission_code>
###
Suggestions
```

Listing 6.3: The present version of the query prompt used for LLM feedback generation, comprising a problem handout, a question asking suggestions to address code issues without providing the corrected code, the submitted code, and a request for suggestions.

There is potential for further improvement in query prompts, such as tailoring them based on prior conversation history and requesting step-by-step assistance. Additionally, the application could enhance user experience by displaying the option to query the LLM based on specific criteria, rather than offering it by default on submission failures. These areas for improvement are discussed in more detail in Chapter 8.

Chapter 7

Evaluation

This chapter discusses the evaluation of the programming learning platform, focusing on its usability, effectiveness, and user experience. We begin by introducing the concept of artifact evaluation and its importance in the design process. Next, we describe the experiment design used to evaluate the platform, including the participant selection process, the tasks given to participants, and the data collection methods. We then present the results of the evaluation, followed by a discussion of the insights gained.

7.1 Evaluate Artifact

The *Evaluate Artifact* is the fifth step in the method framework, assessing how well the artifact solves the stated problem and meets the requirements. This involves determining the utility of the artifact and possibly explaining why it is effective [31].

The goals of artifact evaluation can vary, including testing its effectiveness in solving the problem it was designed for, assessing its compliance with defined requirements, exploring its underlying theories and principles, comparing it to other similar artifacts, identifying unintended consequences, and finding opportunities for improvement during design [31].

Evaluation can be formative, focusing on improvement during design iterations, or summative, assessing the artifact's final utility. Evaluation strategies can be ex ante (before use) or ex post (after use). They can also be naturalistic (in real-world settings) or artificial (in controlled environments). The chosen strategy influences the research methods [31].

The following subsections discuss the design of the evaluation experiment conducted for the platform including its results.

7.2 Experiment Design

The evaluation of the platform was thoughtfully designed to ensure coverage of its usability, effectiveness, and user experience. The students from the department were recruited as participants.

Participants were asked to solve the following three exercises one by one in the order of increasing difficulty within the platform.

1. **Easy Exercise:** Write a function to calculate the sum of two numbers.
2. **Intermediate Exercise:** Write a function for finding the factorial of a given number.
3. **Advanced Exercise:** Write a function to find whether a given number is a palindrome or not.

To evaluate the platform's ability to guide users through challenging programming problems, participants were intentionally asked to introduce a mistake into their solution. This setup simulates real-world programming challenges where students often encounter errors in their code. By intentionally introducing mistakes, we could observe how the platform effectively identified and assisted users in rectifying these errors.

Data for this evaluation was collected through a user survey administered after participants completed the exercises. The participation of users in this survey-based evaluation was entirely voluntary. Participants were informed that their responses would be used exclusively for research purposes.

The survey was designed using a combination of Likert scale and open-ended questions. This approach allowed for a comprehensive assessment of the platform's usability, effectiveness, and user experience. The survey covered the following aspects:

- **Usability:** Participants were asked to rate the platform's usability on a scale from 1 (not at all usable) to 5 (very usable). This rating provided insights into the platform's overall user-friendliness.
- **Usefulness of Grader Feedback:** Participants were asked to evaluate the usefulness of Grader feedback on a scale from 1 (not at all useful) to 5 (very useful). This measurement aimed to determine the effectiveness of the Grader feedback in assisting users. It is to be noted that the Grader feedback discussed above is just the Python Interpreter feedback, including the number of test cases with success or failure.

- **Usefulness of LLM Feedback:** Participants were asked to evaluate the utility of the LLM feedback on a scale from 1 (not at all useful) to 5 (very useful). This measurement aimed to determine the effectiveness of the LLM feedback in assisting users.
- **Free-Form Improvement Suggestions:** A section of the survey allowed participants to provide open-ended feedback and suggestions for improvements. This qualitative input provided valuable insights into specific areas that could be enhanced.

Participants were provided with a quiet, distraction-free environment and were given ample time to complete the exercises. The survey was administered immediately after participants completed the exercise. This ensured that their feedback was fresh and relevant to their experience with the platform.

7.3 Experiment Data: Analysis and Insights

The experiment involved six participants, including individuals from both Bachelor's and Master's programs, with varying levels of experience in Python coding. While most participants had prior Python coding experience, one participant entered the study with no prior Python coding knowledge.

Table 7.1: Aggregate data from the experiment with six participants assessing platform usability, Grader feedback usefulness ratings, and LLM feedback usefulness ratings across different difficulty levels (Easy, Intermediate, Advanced). Additionally, the table displays the number of "Help Me Tutor!" button clicks for each difficulty category.

Data Category	**Easy**	**Intermediate**	**Advanced**
Platform Usability Ratings	4.16	4.16	4.33
Grader Feedback Usefulness Ratings	3.16	3.33	3.16
LLM Feedback Usefulness Ratings	3.83	4.5	4.33
Number of Help Me Tutor! Button Clicks	6	15	25

Table 7.1 presents a summary of the number of times participants clicked the "Help me tutor!" button and the average ratings they provided for the

platform's usability, usefulness of the Grader feedback, and the LLM feedback. The button clicks indicate the frequency with which participants sought additional assistance from the platform's tutor feature.

7.3.1 Platform Usability & Improvement Suggestions

When considering the data across all exercise categories, it becomes evident that participants found the platform to be generally usable, with a usability rating of above 4.00 for all exercise categories.

In the free form suggestions, the participants also provided valuable feedback regarding the platform's user interface which can significantly enhance the overall user experience. Some of the notable UI suggestions included improving the handling of pop-out instructions to prevent them from displacing the text-code environment and ensuring that dropdown menus remain open when clicked on them anywhere except the toggle arrow button. Additionally, participants recommended adjusting font sizes for question descriptions to improve readability.

7.3.2 Grader Feedback vs. LLM Feedback

In our evaluation of the feedback systems used in this study, we found that Grader feedback and LLM feedback differ significantly in their effectiveness and impact on learners. Grader feedback received an average rating of 3.22, while LLM feedback received an average rating of 4.28, revealing distinct variations in how they are perceived by participants.

Grader feedback, as reflected by the average rating of 3.22, appears to have significant room for improvement in terms of clarity and informativeness. Participants provided mixed reviews of this feedback system, with many considering it primarily useful for identifying syntax errors but lacking the depth and specificity needed for a comprehensive learning experience. This suggests that while Grader feedback serves a fundamental purpose in pinpointing errors, it may not provide learners with the detailed guidance and insights they desire.

On the other hand, LLM feedback stands out with the highest average rating of approximately 4.28, underscoring its effectiveness and value in assisting learners. Participants overwhelmingly praised LLM feedback for its instructive and valuable nature. They found it to be clear in explaining mistakes and offering precise strategies for rectification. However, some participants raised concerns that the LLM often provided complete answers rather than hints, which could potentially hinder the learning process. This feedback system's ability to provide solutions was both an advantage and a drawback, as highlighted by participant comments such as *“LLM gave the answers in most cases, maybe there is a way to provide hints instead of solutions?”* and *“Sometimes LLM is telling me the correct answer instead of hints”*.

Grader feedback, while serving a basic error identification function, falls short in providing the depth of guidance desired by learners. In contrast, LLM feedback excels in clarity and depth but sometimes errs on the side of offering complete solutions, raising questions about the balance between hints and answers in feedback delivery. These findings suggest that both systems have their strengths and weaknesses, emphasizing the need for a balanced approach to feedback in the learning process.

7.3.3 Reliability of LLM feedback

Participant feedback also raised concerns about the reliability of LLM feedback. While participants praised the LLM’s feedback for being instructive and clear, one participant questioned whether learners could always trust it:

> *“I don’t know if I can trust the LLM every time.”*

This statement highlights the need to further refine and enhance the LLM’s responses to ensure that learners have confidence in its guidance. Building trust in LLM feedback is essential for its effective integration into the learning process.

The same participant also emphasized the importance of debugging capabilities within the platform. This feature would allow learners to test and verify the accuracy and reliability of the LLM feedback. Currently, the platform only pinpoints syntax errors and the number of test case failures. It does not show the exact interpreter output, which could be helpful for debugging programs and testing and validating LLM feedback. By incorporating debugging tools, the platform can empower users to validate the guidance they receive from the LLM, thereby addressing concerns about blindly trusting its responses.

By addressing these concerns, the platform can instill greater confidence in the LLM’s feedback and enhance its overall effectiveness in facilitating the learning process.

Chapter 8

Discussion

This chapter discusses the relationship of the thesis to related research, the limitations of the work, and future research and development directions.

8.1 Relationship with Related Work

This section compares our platform to existing systems and studies, highlighting the similarities and differences.

8.1.1 Comparison to Existing Systems

The application of LLMs in programming education is relatively new, and there are only a few published papers discussing systems that use LLMs to teach programming. Learning By Teaching (LBT) systems such as HypoCompass, TeachYou, and AlgoBo are similar to our platform in their goal of helping students learn programming. However, they differ in the way they achieve this goal.

These systems follow an indirect approach of “learning by teaching,” while our platform follows a more direct approach of helping students with errors in their code submissions. In these LBT systems, the LLMs play the role of tutees, while in our platform, the LLMs are the tutors. In these LBT systems, students learn by teaching the teachable agent (the LLM), correcting its errors, and guiding it to the correct solution. In our platform, it is the other way around: the LLM teaches the student by correcting their errors and giving suggestions, guiding them to the correct solution.

Another major difference between these systems and our platform is in terms of its flexibility. Our platform is flexible and can integrate multiple LLMs, while these systems are configured to use a single LLM. Additionally, our platform's flexibility to have different types of exercises gives us the possibility to have similar LBT spaces in our platform as well.

8.1.2 Comparison to Existing Studies

Participants in our user study rated the LLM feedback as useful, which is consistent with the findings of [45], where an LLM was used to generate code explanations in an interactive e-book on web software development.

The concern about the reliability of LLM feedback, expressed in participant comments in our user study, resonates with comments on the performance of LLMs on help requests in [22]. The study found that LLMs sometimes reported issues that did not actually exist in the student's code. Similar instances were also observed in our user study.

8.2 Limitations of the Work

In any research, there are things that can be improved. In the following subsections, we will examine these limitations and highlight areas where further research and development are needed to enhance the efficacy and comprehensiveness of this work.

8.2.1 Single LLM Limitation

One significant limitation of this study is that it uses only a single LLM, GPT-4. While the study effectively demonstrates the platform's potential in integrating LLMs into programming education, it does not explore the platform's full capabilities, particularly regarding the integration of multiple LLMs. This limitation arises from the intentional decision to use GPT-4 within the scope of this project, considering the lack of readily available APIs of other open source alternatives as discussed in Section 5.3.5. However, the platform's ability to integrate multiple LLMs, especially open-source ones, presents a promising opportunity for future research and development. This capability is particularly relevant considering the rapid development of new LLM models and the potential for their integration into educational platforms.

8.2.2 Challenges of Small-Scale User Study

Another limitation of this research is the small scale of the user study, with only six participants. This limited sample size means that the findings and conclusions may not comprehensively represent the opinions and experiences of a larger and more diverse population of potential users.

Furthermore, the platform's primary target audience is novice programmers, but the majority of the study participants were master-level students with prior experience in coding, particularly in Python. Only one participant had no previous experience in Python. This disparity between the intended user group and the actual study participants raises concerns about the generalizability of the findings.

Therefore, a more extensive and diverse user study is needed to evaluate the platform's effectiveness and usability for its target audience. Expanding the study's participant pool to include a broader range of users, particularly those with minimal coding experience, would provide more robust insights.

However, it is important to keep in mind that the main aim of this thesis was to build a versatile platform for educational research purpose, which was successfully completed. Thus, this thesis has much to offer beyond the conducted user study.

8.2.3 Prompt Injection

The current platform is susceptible to prompt injection. Users with knowledge of LLMs can carefully insert instructions into the code editor to bypass the constraints of the original prompts. This manipulation could enable students to receive complete answers, hindering the learning process.

To mitigate this limitation and ensure the platform's educational integrity, an input validation or content moderation mechanism is needed to prevent prompt injections and ensure that users engage with the intended learning materials and feedback.

8.2.4 User Tracking Limitation

The current platform tracks and saves user progress by generating and storing UUIDs in the browser's local storage. This approach allows for basic tracking of user progress, but it has a significant drawback: users risk losing all their progress and data if they clear their local storage.

This limitation is less than ideal from both a usability and a research perspective. From a usability standpoint, it can be frustrating for users to inadvertently lose their progress. From a research perspective, this limitation can result in incomplete or unreliable data.

Implementing an optional registration mechanism could help address this limitation. This would allow users to choose to save their data and progress securely, ensuring that their work is not lost in the event of local storage clearance. This registration option would enhance the user experience and provide researchers with a more consistent and complete dataset for analysis and evaluation.

8.3 Future Directions

The platform developed as part of this thesis opens up several new possibilities for research and development, each with the potential to enhance the experience and effectiveness of programming education. In the following subsections, we highlight a few of these research possibilities, which represent exciting opportunities to build on the existing work and further advance the platform's capabilities and impact.

8.3.1 Balancing Help for Effective Learning

In our user study, some students found the platform gave them too much help, sometimes even giving the answer. This leads to an important question: *How much information is too much information?*

Currently, the "Help me tutor!" button is displayed whenever a student's code submission fails, providing immediate assistance. This mechanism could be enhanced by dynamically displaying the button based on each student's progress and individual needs. Future research could use the platform to conduct similar studies to identify the optimal amount of information or assistance that enables students to progress in their programming education without hindering their overall learning experience.

8.3.2 Potential of Prompt Engineering

As discussed in Section 6.4, the responses generated by LLMs can vary significantly based on the prompts used. The rapid evolution of LLMs has led to the emergence of a new field called prompt engineering, which focuses on how to design effective prompts for LLMs [66]. While this thesis explored a limited number of prompt iterations, it did not delve deeply into the realm of prompt engineering. There is substantial potential for further exploration in this domain. For example, we can ask questions such as:

1. *Are there more effective ways to prompt the LLM?*
2. *Are certain prompts better than others at helping students grasp programming concepts?*

The platform could help facilitate further research on these topics.

8.3.3 Comparing LLMs in Programming Education

As previously outlined in Section 8.2.1, the platform currently uses only one LLM, GPT-4. However, its architecture is flexible enough to allow for experiments with multiple LLMs. Future research could explore the use of diverse LLMs and assess their capacity to guide students effectively. This opens up several intriguing research questions:

1. *Do students find feedback from one LLM to be more beneficial than that from another?*

2. *Does one LLM outperform the others at guiding students through their programming education?*

These questions present exciting research possibilities that can be further explored with the help of the platform, contributing to a more comprehensive understanding of the role of different LLMs in programming education.

8.3.4 Fostering Critical Thinking

Concerns regarding the blind trust in LLM responses were voiced by some participants. This issue is not isolated to our study; research on GPT-3.5 has also highlighted limitations in using LLMs for feedback, including instances where the LLMs reported false issues with the code [22]. Therefore, it is important for students to develop a critical mindset towards LLM feedback.

As the platform stores all information regarding exercises, previous code submissions, and LLM feedback, it could help students develop critical thinking skills through various kinds of exercises. For example, students could be presented with a code submission and the corresponding LLM feedback. They could then be asked to identify which suggestions from the LLM are correct and relevant, and which ones are wrong. Another exercise could involve solution comparison. Students could be presented with an exercise and multiple solutions from different LLMs as well as humans. They could then be asked to decide which solution is best.

These types of exercises could be explored further, and more research could be conducted to identify additional ways to help students develop a critical mindset towards LLM feedback.

Chapter 9

Conclusion

The emergence of LLMs has led people to rethink what is possible and what is not in the domain of programming education. These powerful language models, such as GPT-4 and its alternatives, have the potential to transform the way we teach and learn programming, opening up new possibilities and complementing traditional teaching methods.

This thesis has contributed to the exploration of the potential of LLMs for programming education by developing a platform that facilitates both personalized learning and research. The platform provides students with a tailored learning experience, gathers data on student interactions, and offers the unique ability to integrate multiple LLMs into the learning environment.

The findings of this thesis demonstrate the potential of LLMs as valuable tools for programming education. The platform can effectively integrate LLMs to provide personalized feedback and support for students, enhancing their learning experience and promoting a better understanding of programming concepts. The platform's ability to collect data on student interactions offers valuable insights into effective teaching strategies with LLMs, providing a foundation for further research and development in this area.

An evaluation experiment of the platform with students of varying programming experience showed that LLM feedback enhanced the learning experience. However, further refinement is needed to ensure the reliability of LLM feedback.

The practical implications of this research are twofold:

- Researchers can use the platform to collect data and conduct studies on multiple LLMs and how to effectively incorporate them into programming education.

- Educators can use the platform to create more engaging and interactive programming courses for students.

The theoretical implications of this research revolve around prompt engineering, balancing assistance with independent problem-solving, and the comparative analysis of different LLMs in educational settings.

While the initial evaluation of the platform was limited by the use of a single LLM (GPT-4) and a small sample size, the findings are encouraging and pave the way for future research and development.

For future research, we recommend using the platform to:

- Explore optimal prompt engineering techniques for enhancing LLM guidance.
- Conduct comparative analyses of different LLMs in programming education.
- Improve the reliability of LLM feedback to instill confidence in learners.
- Help foster a critical mindset in learners towards the LLM feedback.

While LLMs have the potential to transform programming education, we are still in the early stages of exploring their possibilities. As we move forward, we need more research and evaluation to fully understand the potential and limitations of LLMs in this field. And the platform developed in this thesis can help us on this journey, both as a research tool and as an educational platform.

Bibliography

[1] Ala-Mutka, K. M. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education 15*, 2 (2005), 83–102.

[2] Amelung, M., Krieger, K., and Rösner, D. E-Assessment as a Service. *IEEE Transactions on Learning Technologies 4*, 2 (2011), 162–174.

[3] Ankur, D., and Atul, D. Introducing Amazon Code Whisperer, the ML-powered Coding Companion. `https://aws.amazon.com/blogs/machine-learning/introducing-amazon-codewhisperer-the-ml-powered-coding-companion/`, 2022. Accessed on July 14, 2023.

[4] Antonucci, P., Estler, C., Nikolić, D., Piccioni, M., and Meyer, B. An Incremental Hint System For Automated Programming Assignments. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2015), ITiCSE '15, Association for Computing Machinery, p. 320–325.

[5] Bai, X., Ola, A., Akkaladevi, S., and Cui, Y. Enhancing the Learning Process in Programming Courses through an Automated Feedback and Assignment Management System. *Issues in Information Systems 17*, 3 (2016).

[6] Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., and Santos, E. A. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (2023), pp. 500–506.

[7] Berglund, A., and Lister, R. Introductory Programming and the Didactic Triangle. In *Proceedings of the Twelfth Australasian Confer-*

ence on Computing Education - Volume 103 (AUS, 2010), ACE '10, Australian Computer Society, Inc., p. 35–44.

[8] Brown, T. Change by design. Revised and Updated. *How Design Thinking transforms organizations and inspires innovation. HarperBusiness* (2019).

[9] Cahn, A. F., and Sriram, S. Will AI Be Monitoring Kids in Their Classrooms?, August 2 2023. Accessed on September 27, 2023.

[10] Checkstyle. Checkstyle, 2023. `https://checkstyle.sourceforge.io/`. Accessed on December 04, 2023.

[11] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374* (2021).

[12] Dasgupta, D., Venugopal, D., and Gupta, K. D. A Review of Generative AI from Historical Perspectives.

[13] Denny, P., Kumar, V., and Giacaman, N. Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (New York, NY, USA, 2023), SIGCSE 2023, Association for Computing Machinery, p. 1136–1142.

[14] Denny, P., Luxton-Reilly, A., Tempero, E., and Hendrickx, J. CodeWrite: Supporting Student-Driven Practice of Java. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2011), SIGCSE '11, Association for Computing Machinery, p. 471–476.

[15] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).

[16] Edwards, S. H., and Perez-Quinones, M. A. Web-CAT: Automatically Grading Programming Assignments. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2008), ITiCSE '08, Association for Computing Machinery, p. 328.

[17] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *arXiv preprint arXiv:2002.08155* (2020).

[18] Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., and Prather, J. The Robots Are Coming: Exploring the Implications of OpenAI Codex on Introductory Programming. In *Proceedings of the 24th Australasian Computing Education Conference* (New York, NY, USA, 2022), ACE '22, Association for Computing Machinery, p. 10–19.

[19] Flowers, T., Carver, C., and Jackson, J. Empowering Students and Building Confidence in Novice Programmers through Gauntlet. In *34th Annual Frontiers in Education, 2004. FIE 2004.* (2004), pp. T3H/10–T3H/13 Vol. 1.

[20] Gabbay, H., and Cohen, A. Investigating the Effect of Automated Feedback on Learning Behavior in MOOCs for Programming. *International Educational Data Mining Society* (2022).

[21] Garcia, R., Falkner, K., and Vivian, R. Systematic literature review: Self-Regulated Learning strategies using e-learning tools for Computer Science. *Computers Education 123* (2018), 150–163.

[22] Hellas, A., Leinonen, J., Sarsa, S., Koutcheme, C., Kujanpää, L., and Sorva, J. Exploring the Responses of Large Language Models to Beginner Programmers' Help Requests. *arXiv preprint arXiv:2306.05715* (2023).

[23] Higgins, C., Hegazy, T., Symeonidis, P., and Tsintsifas, A. The CourseMarker CBA System: Improvements over Ceilidh. *Education and Information Technologies 8* (2003), 287–304.

[24] Holland-Minkley, A. M., and Lombardi, T. Improving Engagement in Introductory Courses with Homework Resubmission. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (New York, NY, USA, 2016), SIGCSE '16, Association for Computing Machinery, p. 534–539.

[25] Hong, J. Guided Programming and Automated Error Analysis in an Intelligent Prolog Tutor. *International Journal of Human-Computer Studies 61*, 4 (2004), 505–534.

[26] Iivari, J. Distinguishing and Contrasting Two Strategies for Design Science Research. *European Journal of Information Systems 24*, 1 (2015), 107–115.

[27] Jay, C., Dunne, R., Gelsthorpe, D., and Vigo, M. To Sign Up, or Not to Sign Up? Maximizing Citizen Science Contribution Rates through Optional Registration. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2016), CHI '16, Association for Computing Machinery, p. 1827–1832.

[28] Jeuring, J., van Binsbergen, L. T., Gerdes, A., and Heeren, B. Model Solutions and Properties for Diagnosing Student Programs in Ask-Elle. In *Proceedings of the Computer Science Education Research Conference* (2014), pp. 31–40.

[29] Jin, H., Lee, S., Shin, H., and Kim, J. "Teach AI How to Code": Using Large Language Models as Teachable Agents for Programming Education, 2023.

[30] Jin, W., Corbett, A., Lloyd, W., Baumstark, L., and Rolka, C. Evaluation of Guided-Planning and Assisted-Coding with Task Relevant Dynamic Hinting. In *Intelligent Tutoring Systems* (Cham, 2014), S. Trausan-Matu, K. E. Boyer, M. Crosby, and K. Panourgia, Eds., Springer International Publishing, pp. 318–328.

[31] Johannesson, P., and Perjons, E. *An Introduction to Design Science*, vol. 10. Springer, 2014.

[32] Kalogeropoulos, N., Tzigounakis, I., Pavlatou, E. A., and Boudouvis, A. G. Computer-Based Assessment of Student Performance in Programming Courses. *Computer Applications in Engineering Education 21*, 4 (2013), 671–683.

[33] Kinnunen, P., and Malmi, L. Why Students Drop out CS1 Course? In *Proceedings of the Second International Workshop on Computing Education Research* (New York, NY, USA, 2006), ICER '06, Association for Computing Machinery, p. 97–108.

[34] Ko, A. J., and Myers, B. A. Debugging Reinvented: Asking and Answering Why and Why Not Questions about Program Behavior. In *Proceedings of the 30th International Conference on Software Engineering* (New York, NY, USA, 2008), ICSE '08, Association for Computing Machinery, p. 301–310.

[35] Law, K. M., Lee, V. C., and Yu, Y. Learning Motivation in E-Learning Facilitated Computer Programming Courses. *Computers Education 55*, 1 (2010), 218–228.

[36] Leinonen, J., Denny, P., MacNeil, S., Sarsa, S., Bernstein, S., Kim, J., Tran, A., and Hellas, A. Comparing Code Explanations Created by Students and Large Language Models. *arXiv preprint arXiv:2304.03938* (2023).

[37] Leinonen, J., Hellas, A., Sarsa, S., Reeves, B., Denny, P., Prather, J., and Becker, B. A. Using Large Language Models to Enhance Programming Error Messages. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (2023), pp. 563–569.

[38] Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. Competition-Level Code Generation with AlphaCode. *Science 378*, 6624 (2022), 1092–1097.

[39] Lister, R., Fidge, C., and Teague, D. Further Evidence of a Relationship between Explaining, Tracing and Writing Skills in Introductory Programming. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2009), ITiCSE '09, Association for Computing Machinery, p. 161–165.

[40] Liu, Q., Shen, S., Huang, Z., Chen, E., and Zheng, Y. A Survey of Knowledge Tracing, 2023.

[41] Luxton-Reilly, A., and Denny, P. Constructive Evaluation: A Pedagogy of Student-Contributed Assessment. *Computer Science Education 20*, 2 (2010), 145–167.

[42] Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., and Szabo, C. Introductory Programming: A Systematic Literature Review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2018), ITiCSE 2018 Companion, Association for Computing Machinery, p. 55–106.

[43] Ma, L., Ferguson, J., Roper, M., and Wood, M. Investigating the Viability of Mental Models Held by Novice Programmers. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2007), SIGCSE '07, Association for Computing Machinery, p. 499–503.

[44] Ma, Q., Shen, H., Koedinger, K., and Wu, T. HypoCompass: Large-Language-Model-based Tutor for Hypothesis Construction in Debugging for Novices, 2023.

[45] MacNeil, S., Tran, A., Hellas, A., Kim, J., Sarsa, S., Denny, P., Bernstein, S., and Leinonen, J. Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book. *arXiv preprint arXiv:2211.02265* (2022).

[46] MacNeil, S., Tran, A., Mogil, D., Bernstein, S., Ross, E., and Huang, Z. Generating Diverse Code Explanations Using the GPT-3 Large Language Model. In *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 2* (New York, NY, USA, 2022), ICER '22, Association for Computing Machinery, p. 37–39.

[47] Morris, D. S. Automatic Grading of Student's Programming Assignments: An Interactive Process and Suite of Programs. In *33rd Annual Frontiers in Education, 2003. FIE 2003.* (2003), vol. 3, IEEE, pp. S3F–1.

[48] Odekirk-Hash, E., and Zachary, J. L. Automated Feedback on Programs Means Students Need Less Help from Teachers. In *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education* (2001), pp. 55–59.

[49] OpenAI. Introducing ChatGPT. `https://openai.com/blog/chatgpt`, 2022. Accessed on 12 Jul 2023.

[50] OpenAI. GPT-4 Technical Report. *arXiv e-prints* (Mar. 2023), arXiv:2303.08774.

[51] Ott, C., Robins, A., and Shephard, K. Translating Principles of Effective Feedback for Students into the CS1 Context. *ACM Trans. Comput. Educ. 16*, 1 (jan 2016).

[52] Peffers, K., Tuunanen, T., Rothenberger, M. A., and Chatterjee, S. A Design Science Research Methodology for Information

Systems Research. *Journal of management information systems 24*, 3 (2007), 45–77.

[53] PMD. PMD, 2023. `https://pmd.github.io/`. Accessed on December 04, 2023.

[54] Rafique, W., Dou, W., Hussain, K., and Ahmed, K. Factors Influencing Programming Expertise in a Web-based E-learning Paradigm. *Online Learning 24*, 1 (2020), 162–181.

[55] Rai, S. World's Most Popular Online Computer Class Turns to AI for Help, June 3 2023. Accessed on September 27, 2023.

[56] Robey, M., Von Konsky, B. R., Ivins, J., Gribble, S. J., Loh, A., and Cooper, D. Student Self-Motivation: Lessons Learned from Teaching First Year Computing. In *Proceedings. Frontiers in Education. 36th Annual Conference* (2006), pp. 6–11.

[57] Sarsa, S., Denny, P., Hellas, A., and Leinonen, J. 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* (New York, NY, USA, 2022), ICER '22, Association for Computing Machinery, p. 27–43.

[58] Schorsch, T. CAP: An Automated Self-Assessment Tool to Check Pascal Programs for Syntax, Logic and Style Errors. *SIGCSE Bull. 27*, 1 (mar 1995), 168–172.

[59] Soloway, E. Learning to Program = Learning to Construct Mechanisms and Explanations. *Commun. ACM 29*, 9 (sep 1986), 850–858.

[60] Sorva, J. The Same but Different Students' Understandings of Primitive and Object Variables. In *Proceedings of the 8th International Conference on Computing Education Research* (New York, NY, USA, 2008), Koli '08, Association for Computing Machinery, p. 5–15.

[61] Staubitz, T., Klement, H., Renz, J., Teusner, R., and Meinel, C. Towards Practical Programming Exercises and Automated Assessment in Massive Open Online Courses. In *2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)* (2015), pp. 23–30.

[62] Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H.-T., Jin, A., Bos, T., Baker, L., Du, Y., et al. LaMDA: Language Models for Dialog Applications. *arXiv preprint arXiv:2201.08239* (2022).

[63] Trotter, E. Student Perceptions of Continuous Summative Assessment. *Assessment & Evaluation in Higher Education 31*, 5 (2006), 505–521.

[64] Wang, T., Su, X., Ma, P., Wang, Y., and Wang, K. Ability-Training-Oriented Automated Assessment in Introductory Programming Course. *Computers Education 56*, 1 (2011), 220–226. Serious Games.

[65] Wermelinger, M. Using GitHub Copilot to Solve Simple Programming Problems. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (New York, NY, USA, 2023), SIGCSE 2023, Association for Computing Machinery, p. 172–178.

[66] White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., and Schmidt, D. C. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT, 2023.

[67] Wu, W., Li, G., Sun, Y., Wang, J., and Lai, T. AnalyseC: A Framework for Assessing Students' Programs at Structural and Semantic Level. In *2007 IEEE International Conference on Control and Automation* (2007), IEEE, pp. 742–747.

[68] Xu, S., and San Chee, Y. Transformation-Based Diagnosis of Student Programs for Programming Tutoring Systems. *IEEE Transactions on Software Engineering 29*, 4 (2003), 360–384.

[69] Zingaro, D., Cherenkova, Y., Karpova, O., and Petersen, A. Facilitating Code-Writing in PI Classes. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2013), SIGCSE '13, Association for Computing Machinery, p. 585–590.