Independent University **Bangladesh (IUB)**

IUB Academic Repository

Computer Science and Engineering | Undergraduate Thesis

2024-11

PACE: Python AI Companion for Enhanced Engagement

Shochcho, Muhtasim Ibteda

IUB

http://ar.iub.edu.bd/handle/11348/993

Downloaded from IUB Academic Repository



PACE: Python AI Companion for Enhanced Engagement

November 2024

Prepared by:

Muhtasim Ibteda Shochcho
ID: 2021111

Mohammad Ashfaq Ur Rahman
ID: 1930087

Department of Computer Science and Engineering
Independent University, Bangladesh

Supervised by

Amin Ahsan Ali, PhD
Professor
Department of Computer Science and Engineering
Independent University, Bangladesh

Attestation

We are aware of the fact that plagiarism is strictly prohibited and it conflicts with our university's rules and regulations. We guarantee the authenticity of our work. We have used some copyrighted material and models of others in our project which has been properly cited following the international standards proper guidelines.

Author Name:

. . . . . .

Signature:

. . . . . .

Author Name:

. . . . . .

Signature:

. . . . . .

Evaluation Committee

Supervisor

Name: _____ Signature: _____

Internal Examiner 1

Name: _____ Signature: _____

Internal Examiner 2

Name: _____ Signature: _____

External Examiner

Name: _____ Signature: _____

Acknowledgement

We would like to express our deepest gratitude to our esteemed professor, Amin Ahsan Ali, at Independent University, Bangladesh. His invaluable guidance, profound insights, and remarkable patience have been crucial in the successful completion of our project. Professor Ali's dedication and generous investment of his time greatly enhanced the depth and quality of our work. His unwavering support and mentorship have been instrumental in shaping our academic journey, and we are truly thankful for his contributions.

We would also like to thank Professor Amin Ahsan Ali for his guidance, which helped us gain acceptance into the ACM SIGCHI Winter School Program on Human-Computer Interaction, organized by the Association of Computing Machinery (ACM) Special Interest Group: Computer-Human Interaction (SIGCHI). This winter school program also played a pivotal role in shaping our project.

Furthermore, we extend our heartfelt gratitude to the CCDS (Center for Computational & Data Sciences) for their invaluable support and continuous encouragement throughout our project. Their guidance played a pivotal role in steering us towards the ACM SIGCHI Winter School program, and the collaborative, resource-rich environment they provided was integral to the successful development and execution of our work. The opportunities and resources offered by the lab not only enriched our research experience but also provided a solid foundation for our academic and professional growth. We are deeply appreciative of the mentorship and support we received from CCDS.

We also acknowledge with gratitude the contributions of all those who supported us, both directly and indirectly, throughout this endeavor. Your help and encouragement have been invaluable, and we are thankful for your involvement in our academic and research pursuits.

Abstract

This research introduces PACE (Python AI Companion for Enhanced Engagement), an AI-driven, interactive tutoring system designed to support Python programming students through personalized, responsive guidance. Addressing common challenges in traditional educational settings, such as limited individualized feedback and scalability, PACE leverages the capabilities of Large Language Models (LLMs) to create a tailored, human-like tutoring experience. Fine-tuned using the Gemma 2B model with scaffolding and conversational datasets, PACE offers step-by-step instruction and adaptive problem-solving support that encourages active learning without directly revealing answers.

The development of PACE involved a robust design and implementation process, including backend development with FastAPI, frontend construction in React, and data management in MySQL, to deliver seamless, real-time interactions with students. Interaction design and user interface principles were carefully applied to create an intuitive environment that promotes engagement and reduces cognitive load. The model's fine-tuning was achieved using Low-Rank Adaptation (LoRA), optimizing computational efficiency while enhancing PACE's ability to deliver domain-specific instruction in Python.

The system was evaluated with undergraduate students using five pedagogical metrics: Relevancy, Correctness, Completeness, Motivation, and Clarity, and included a System Usability Scale (SUS) assessment. Findings demonstrate PACE's success in fostering understanding, boosting student engagement, and encouraging critical thinking. By tracking student progress and providing scalable, personalized support, PACE also eases the instructional burden on educators in large classes. This work showcases the potential of LLM-based intelligent tutoring systems to deliver cost-effective, high-quality, personalized learning experiences, making them viable for broader educational deployment.

Contents

1 Introduction **8**
- 1.1 Motivation . . . 8
- 1.2 Challenges and Design Space . . . 9
  - 1.2.1 Challenges . . . 9
  - 1.2.2 Interaction Design Space . . . 10
  - 1.2.3 System Implementation Choices . . . 11
- 1.3 Goals . . . 11
- 1.4 Outcome of the Project . . . 13
- 1.5 Outline of the Report . . . 13

2 Background **14**
- 2.1 Pre-LLMs ITS and Design Recommendations . . . 14
- 2.2 Transition to LLM-Based Tutor Systems . . . 15
  - 2.2.1 Capabilities of Large Language Models (LLMs) . . . 15
  - 2.2.2 Capabilities of LLM-Based Tutor Systems . . . 17
- 2.3 Challenges in Python Tutors and Addressed Solutions . . . 18

3 System Design **19**
- 3.1 Interaction Design . . . 19
  - 3.1.1 Interaction with the Newly Joined Student . . . 19
  - 3.1.2 Interaction with the PACE Chatbot . . . 19
- 3.2 Dataset Generation for Model Fine-tuning . . . 19
  - 3.2.1 Scaffolding Dataset . . . 22
  - 3.2.2 Conversational Dataset . . . 24
  - 3.2.3 Prompt Design Strategy . . . 24
  - 3.2.4 Dataset Evaluation . . . 25
- 3.3 User-Interface Design . . . 26

4 System Development **28**
- 4.1 Development . . . 28
  - 4.1.1 Backend . . . 28
  - 4.1.2 Frontend . . . 29
  - 4.1.3 Database . . . 29

5 Model Selection & Evaluation **31**
- 5.1 Model Selection . . . 31
- 5.2 LoRA Fine-tuning . . . 31
- 5.3 System Evaluation . . . 32
  - 5.3.1 Chat Response Evaluation . . . 32
  - 5.3.2 System Usability Study . . . 38

5.3.3 Mistakes and Errors from the System . . . 39
5.3.4 Strengths and Weaknesses of the System . . . 40

6 Conclusion and Future Works **41**
6.1 Conclusion . . . 41
6.2 Limitations . . . 41
6.3 Ethical Considerations . . . 41
6.4 Future Plan . . . 42

Bibliography **42**

A Appendices **47**
A.1 Conversational Data Generation Prompt . . . 47
A.2 Scaffolding Dataset Examples . . . 49
A.2.1 Example 1 . . . 49
A.2.2 Example 2 . . . 50
A.2.3 Example 3 . . . 51
A.3 Conversational Dataset Examples . . . 52
A.3.1 Example 1 . . . 52
A.3.2 Example 2 . . . 54
A.3.3 Example 3 . . . 56
A.4 User conversation with PACE . . . 60
A.5 User-Interface . . . 66

List of Figures

1 **Chat Interface** 12

2 **Sequence diagram showing the interaction between the student, user interface, backend, AI tutor, and database during a learning session. The AI tutor processes the student's input and provides feedback via the backend** 20

3 **Sequence diagram illustrating new students signing up for the system. The AI tutor evaluates the students' initial knowledge, retrieves data from the database, and updates student knowledge progress through the backend** 21

4 **This workflow shows how datasets are generated by retrieving columns from a Python book and feeding them into GPT-3.5-Turbo. The model first generates problems for a scaffolding dataset using book-retrieved columns and then creates the conversational dataset by using problems from the scaffolding dataset** 22

5 **Evaluation of Conversational Dataset** 25

6 **Student Performance Dashboard** 28

7 **Distribution of Python Knowledge Levels Among Students** 32

8 **Relevancy (R): Distribution of scores on the chatbot's relevance to user programming needs.** 34

9 **Correctness (C): Distribution of scores assessing the chatbot's accuracy and correctness in responses.** 35

10 **Completeness (Co): Distribution of scores evaluating the thoroughness of the chatbot's responses.** 35

11 **Motivation (M): Distribution of scores on the chatbot's effectiveness in motivating users.** 36

12 **Clarity (Cl): Distribution of scores assessing the clarity and understandability of the chatbot's responses.** 37

13 **SUS score ranges with categories from "Worst Imaginable" to "Best Imaginable." The vertical line indicates our system score of 75.56, categorized as "Excellent."** 38

14 **Student Progress Monitoring Dashboard** 66

15 **Topic Boxes** 67

16 **Login Page** 67

1 Introduction

1.1 Motivation

Students encounter various challenges in introductory programming courses. These difficulties stem from unfamiliar syntax, lack of math knowledge, inaccurate mental models, a lack of effective strategies, complexities within programming environments, and the quality of instruction from teachers. Additionally, limited personalized feedback and large class sizes contribute to these struggles [1]. In traditional classroom settings, personalized attention is minimal, which can hinder students' ability to receive tailored support that addresses their specific learning needs. Studies have demonstrated that personalized feedback and targeted explanations are crucial in helping students overcome these challenges [2]. Tutorials in introductory programming courses have been shown to significantly help students understand basic concepts and develop necessary skills [3]. As Kiesler et al. focused in their research, students learn much more if they have personalized feedback and step-by-step explanations, which are rare in conventional educational environments [4].

Despite an overflow of online tutorials and automated feedback systems, the majority of them do not provide the level of interaction depth and personalization required by the student. For instance, systems such as Codecademy, LeetCode, and HackerRank are widely used for programming education [5][6][7]. While they offer structured lessons, instant feedback, and coding challenges, their limitations lie in the lack of real-time, adaptive guidance based on individual student progress. These systems often follow predefined paths that may not accommodate diverse learning styles or varying levels of prior knowledge [5]. Sheese et al. go on to point out the inadequacies of current systems as they are not configured or designed in such a way that they cater to the needs of every individual learner, hence the establishment of a void in the learning process [8].

Personalization is important since, through personalization, the learning tools can answer every student's unique challenges and learning pace. Without personal help, the student may struggle with specific concepts, resulting in knowledge gaps that could hold them back from advancing into more advanced topics. Intelligent Tutoring Systems (ITS) were designed to mimic human tutoring through AI and domain knowledge, featuring adaptability and real-time feedback [9]. Early ITS models like INTUITEL tailored learning based on student profiles but were limited by rule-based, domain-specific designs and lacked advanced reasoning capabilities [10]. The emergence of Large Language Models (LLMs) has transformed ITS by enabling richer, context-aware interactions and detailed, adaptive feedback. LLM-based ITS overcome earlier limitations, providing more scalable, conversational, and personalized learning support. Large language models can provide a solution to this problem by providing personalized assistance that can simulate human-like tutoring. Though LLMs have shown great potential, basic models such as Llama 3.1 and other closed models such as GPT-4o tend to provide direct answers, which can be detrimental to learners and may lead to incorrect responses when multi-step reasoning is

required [11]. To address this issue, Sonkar et al. in [12] introduced the CLASS framework, a design methodology that fine-tunes Large Language Models (LLMs) to create advanced Intelligent Tutoring Systems (ITS) tailored for biology. CLASS leverages principles from learning science to equip ITS with two essential capabilities: step-by-step guidance and conversational adaptability. This framework offers an approach for providing tailored explanations and feedback based on each learner's understanding level. We plan to adopt this framework in our programming tutoring system to scale support for beginners in both classroom and individual learning settings, enabling personalized, interactive help that enhances programming comprehension and course outcomes.

By continuously adapting to each learner's unique needs, our system can foster better understanding of programming concepts, improve learning outcomes, and boost engagement in programming courses. This approach not only encourages active participation and reduces dissatisfaction among students but also alleviates the burden on instructors, allowing them to focus on strategic teaching methods instead of repetitive guidance in large classrooms. Consequently, our system aims to improve overall student performance and satisfaction.

In this work, we use the CLASS framework to fine-tune an LLM-based tutoring system that mimics human tutors by guiding students toward solutions. In line with the CLASS framework, Chen et al.[13] suggest that the use of curated datasets and structured scaffolding allows a fine-tuned model to deconstruct problems and provide learners with step-by-step guidance. This approach promotes deeper understanding and helps learners grasp core concepts incrementally. By deconstructing problems, the fine-tuned model encourages learners to solve them incrementally, fostering a deeper understanding of the underlying concepts. Another important feature will be adding guardrails to prevent the system from providing direct solutions to problems. A fine-tuned model built on the CLASS framework will prevent the model from offering direct solutions. Instead, it will deconstruct the problem and guide learners to solve it step by step, encouraging deeper understanding and helping them grasp the underlying concepts effectively.

1.2 Challenges and Design Space

The project addresses the challenges of delivering scalable, personalized tutoring to students in introductory Python courses by using Large Language Models (LLMs). Some of the limiting factors for traditional tutoring are the time human tutors have and how difficult it is to provide consistent, personalized guidance to hundreds or thousands of students. Although LLMs harbor big potential, they face some key issues that need to be addressed to effectively support learning in this context.

1.2.1 Challenges

Below, we will outline the key challenges that need to be addressed and the essential features that an effective tutoring system must incorporate to develop a successful AI-based tutor. These challenges have been identified based on current research on the application of intelligent tutoring systems:

- **Personalization and Adaptability:** Perhaps the most critical challenge is to deliver personalized tutoring at scale. In large classroom settings, providing support at an individual level becomes almost impossible, resulting in low student engagement and poor learning outcomes [14][15]. Though LLMs can overcome this, what they

need now is the delivery of tailored feedback that adapts to the student's specific needs.

- **Step-by-Step Guidance:** Good tutoring is where the student is talked through a problem without being given a direct solution. This step-by-step approach is necessary to establish deep understanding and exercise critical thinking [16]. Students in an introductory programming or data science course often prioritize seeking immediate solutions with programming assignments over focusing on related concepts or deepening understanding when using an LLM-powered tool, leading to a lack of core knowledge and subsequent struggles [8].

- **Reliability and Trust:** The use of LLMs in education has led to growing concerns regarding the issues of accuracy and potential misuse. Ensuring these models provide reliable and trustworthy guidance is critical for classroom integration [17].

- **Computational Resources:** In an academic environment, deploying very large LLMs is resource-intensive [18]. Proper deployment should be performed to consider performance versus accessibility.

1.2.2 Interaction Design Space

Based on the discussion above, we arrive at a few interaction design choices that can be made to address the challenges:

- **Leveraging Prompting Techniques:** Prompt engineering can be utilized to guide LLMs in providing incremental hints, explanations, and problem-solving strategies that maintain engagement and promote in-depth learning without revealing complete answers. This structured prompting can replicate a human-like tutoring experience, fostering critical thinking and comprehension in students [12][19].

- **Fine-Tuning with Conversational Dataset:** Fine-tuning LLMs with curated conversational datasets allows them to adapt to varying student learning levels and styles [12]. This customization enables models to deliver context-specific and adaptive responses, aligning with individual student progress and needs. It helps enhance engagement and learning retention by tailoring feedback dynamically.

- **Performing Knowledge Tracing:** Knowledge tracing algorithms can be implemented to monitor and map students' understanding over time [20]. This approach involves tracking their learning progression and updating a student model, which the LLM can reference to provide adaptive feedback and task recommendations. Such mechanisms ensure that students are presented with content that matches their current competence level, preventing gaps in learning.

- **Gamification:** Integrating gamification elements, such as badges, levels, and progress tracking, can enhance student engagement and motivation. These features make the learning experience more interactive and enjoyable, encouraging students to participate actively and persevere through challenges [21].

- **Real-time Monitoring:** It can allow educators to monitor students' progress and intervene at any time. It maintains the effectiveness of tutoring systems by offering due support to students [22].

1.2.3 System Implementation Choices

- **Model Selection:** Selecting an appropriate model is essential for building an effective educational system. Gemma 2B is a suitable choice among small open-source models due to its reliable performance in benchmarks, lightweight nature, and low computational cost. Its flexibility for fine-tuning in specific domains, such as Python tutoring, and its open-source nature and computational efficiency make it cost-effective and suitable for large-scale deployment, ensuring platform accessibility. Additionally, a model like Gemma 2B guarantees accessibility across all platforms, making high-quality tutoring available to more students.

- **Model Fine-Tuning:** Fine-tuning LLMs on curated datasets, especially in Python programming, can provide the correct step-by-step guidance. This training is done with respect to a special class of problems and solutions that represent the typical challenges faced by beginners. Fine-tuning not only enhances the model's accuracy but also ensures relevance to the educational context.

- **Modular Design:** A modular design approach can make the architecture of the tutoring system flexible for integrating new features or changes according to student feedback. This approach ensures that the system evolves and improves over time, allowing for seamless updates and the addition of new educational resources or tools without significant overhauls to the existing infrastructure.

- **Efficient Model Deployment:** To make LLM-based tutors available across diverse learning platforms, focus on efficient deployment strategies. There are primarily two options for deploying the system:

    1. **Cloud-Based Deployment:** In this option, the model runs on cloud-based GPUs, such as NVIDIA A100, which offers high performance but incurs hourly usage charges. Running the entire LLM system in the cloud allows for scalability and accessibility across devices. It requires a stable internet connection and robust cloud infrastructure to handle multiple concurrent users. Advantages include reduced local resource requirements and the ability to leverage powerful server-side processing. However, disadvantages include potential latency issues and dependency on internet access.
    2. **Standalone System Deployment:** Deploying the system as a standalone application can provide offline functionality and reduce latency. This involves deploying the model locally on a GPU with at least 12GB VRAM, like an NVIDIA RTX 3060 or RTX 4060. For better performance, models can be run on GPUs with higher CUDA cores, such as the RTX 4070 Super. Standalone setups offer cost-efficiency after the initial hardware investment and ensure faster processing due to dedicated resources. However, disadvantages may include increased system requirements and challenges in updates or feature integration.

Proper design and engineering of the system will address these challenges and lead to a scalable, reliable, and effective Python tutoring system for students and educators.

1.3 Goals

Based on the discussion of interaction design space in subsection 1.2.2 and system implementation choices in subsection 1.2.3, we list below the specific goals for this project:

- **Personalized Tutoring:** Develop a Python tutoring system using LLMs that can adapt to individual students' unique needs and learning paces, particularly in large classroom settings.
- **Enhance Step-by-Step Learning:** Implement an approach that offers step-by-step guidance to help students incrementally understand programming concepts, ensuring deeper comprehension without directly providing solutions.
- **Improve Engagement and Understanding:** Create an interactive, personalized tutoring environment that promotes student participation, lowers dissatisfaction, and increases engagement, helping students grasp core concepts effectively.
- **Ensure Trustworthy and Accurate Guidance:** Design the tutoring system to deliver reliable and accurate information, establishing guardrails to prevent misinformation, enhance safety, and maintain trust among learners and educators.
- **Optimize Resource Efficiency:** Utilize lightweight LLMs, like Gemma 2B, and efficient model deployment strategies to ensure broad accessibility and lower computational requirements, making high-quality tutoring more available and feasible in academic settings.
- **Alleviate Educator Workload:** Provide a tool that assists teachers by handling personalized student interactions, enabling educators to focus on strategic teaching and support without needing one-on-one tutoring in large classes.
- **Progress Tracking Dashboard:** Rather than gamification elements, incorporate a progress-tracking dashboard that allows students to monitor their learning journey. This feature provides visual feedback to encourage ongoing engagement and commitment.

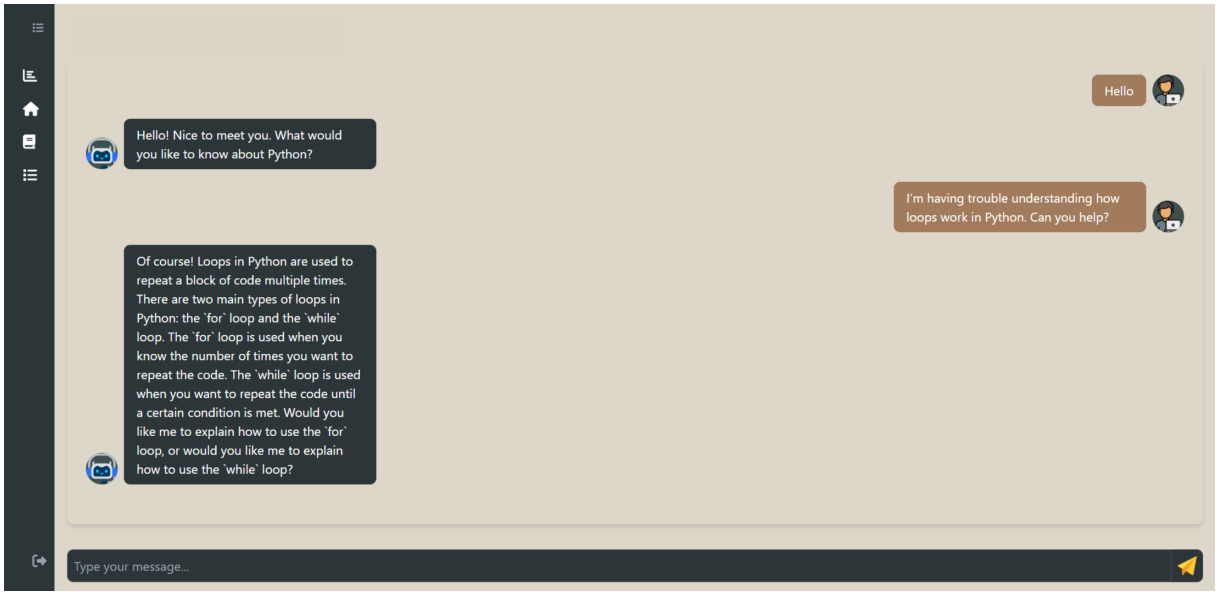


Figure 1: **Chat Interface**

1.4 Outcome of the Project

The results of the project include a fully functioning LLM-based Python tutoring system that effectively mimics human tutoring. It motivates and guides students step-by-step toward reaching a solution without explicitly providing the answer. The key deliverables are:

- **Educational Adaptation-Trained Gemma 2B Model:** A version of the Gemma 2B model fine-tuned on scaffolding and conversational data, generated using a Chain-of-Thought (CoT) prompt[23] with GPT-3.5 Turbo.
- **Interactive Tutoring System:** A deployable system in educational environments that offers personalized and responsive tutoring tailored to individual learners' needs. The intuitive UI, shown in Figure 1, provides real-time feedback and a simple layout, enabling seamless student interaction.
- **Repository of Interaction Data:** A system for collecting student-tutorbot interactions for further research and model refinement.
- **System Deployment:** A standalone system achieved using RTX 4090 with 24 GB VRAM, demonstrating successful processing times and consistent response quality.
- **Quality Dataset Generation:** Created synthetic datasets and evaluated them against five major metrics.
- **Usability Evaluation of the System:** A usability-evaluated system involving several undergraduate students based on a series of questionnaires and using the System Usability Scale (SUS) methodology.

1.5 Outline of the Report

The report is organized into several key chapters. Chapter 1, the introduction, motivates the project, highlighting the challenges traditional tutoring systems face and presenting the proposed solution, which uses LLM-powered tutoring for Python programming. Chapter 2 reviews existing literature on Intelligent Tutoring Systems (ITS) and explores the evolution of large language models (LLMs) in educational settings, focusing on their capabilities and challenges in tutoring environments. Chapter 3 discusses the system's architecture and interaction design, detailing the methods for generating datasets and how scaffolding and conversational data were employed for adaptive tutoring. The technical aspects of the project are covered in Chapter 4, including backend, frontend, and database implementations, as well as the integration of the fine-tuned Gemma 2B model to deliver tutoring services. Chapter 5 assesses the fine-tuned tutor model compared to others, using metrics such as motivation, clarity, correctness, relevance, and completeness to measure effectiveness. Chapter 6 summarizes the project's key findings, discusses its limitations, and offers recommendations for future improvements, including enhancements to the model and potential features such as knowledge tracing and real-time progress tracking. The appendices provide supplementary materials, such as dataset generation prompts, example conversations, and evaluation data from the development and testing phases.

2 Background

This chapter will go into more detail regarding the evolutionary path of Intelligent Tutoring Systems (ITS) from their early designs to their modern implementations based on Large Language Models. Section 2.1 overviews foundational ITS systems prior to LLMs, including design recommendations and subsequent initial implementations based on them. Section 2.2 analyzes the potentials of LLMs in zero-shot/few-shot learning and reasoning. Section 2.3 continues this comparison, discussing how LLM-based tutoring systems have improved on early ITS, what their limitations are, and what challenges lie ahead.

2.1 Pre-LLMs ITS and Design Recommendations

Before LLMs, Intelligent Tutoring Systems (ITS) had been designed to mimic human tutoring through the provision of personalized instruction. These have been realized through a combination of AI and domain knowledge.

Table 2.1: **Early ITS and Design Recommendations (Pre-LLMs)**

Feature	**Description**	**Design Recommendations**	**Ref**
Model-Tracing	Early ITS required hand-coded rules for problem-solving.	Automate content generation using AI tools.	[24]
Student Modeling	Tracked learners' knowledge to tailor feedback.	Use real-time data and adaptive algorithms for enhanced personalization.	[10]
Gamification	Early ITS lacked engaging game-like elements.	Include rewards, leaderboards, and progress tracking for engagement.	[21]
Domain-Specific Focus	ITS were tailored for specific subjects.	Develop cross-domain ITS with reusable modular components.	[25]
Rule Creation	Early rule creation was manual and costly.	Adopt AI-driven methods for scalable learning material generation.	[24]

The adaptability and personalization of ITS models are evident in early projects like INTUITEL, which offered adaptive learning through recommendations tailored to a learner's profile, progress, and context [10]. For instance, Muangprathub et al. developed an adaptive algorithm predicting the needs of learners and then tailored the content with improvements in learning outcomes [26]. These early systems dealt with individual

differences in the pace of learning and style but were scalable to a very low level in both interaction depth and response detail. Thus, design recommendations for early ITS models often emphasized flexibility, adaptability, and real-time feedback. Bacic et al. emphasized the development of scalable and web-based systems that can provide individualized real-time feedback [24].

The constrained AI models used did not have the ability to handle complex reasoning and conversational capabilities. Early ITSs were rule-based, step-guidance presentation, and domain-specific implementations. Table Table 2.1 presents the key features of early ITSs and design recommendations learned from such early systems.

2.2 Transition to LLM-Based Tutor Systems

The emergence of Large Language Models (LLMs) has revolutionized the capabilities of intelligent tutoring systems, enabling them to provide more dynamic, responsive, and personalized educational experiences. These models offer advanced features such as zero-shot and few-shot prompting, which allow systems to generate accurate responses without task-specific training, significantly expanding their versatility across different domains.

2.2.1 Capabilities of Large Language Models (LLMs)

Large Language Models (LLMs) have significantly advanced the field of intelligent tutoring and reasoning, primarily through innovations in zero-shot, few-shot, and Chain-of-Thought (CoT) prompting.

Zero-shot Prompting

Zero-shot prompting shows that the LLM could furnish useful and relevant outputs by just seeing a prompt about a task instead of task-specific examples. It generalizes LLMs across a wide range of tasks based on abilities pre-trained with extensive knowledge. Just by prompting the model with "Let's think step by step," the model's performance on complex reasoning tasks can be drastically improved, increasing accuracy on arithmetic benchmarks like MultiArith from 17.7% to 78.7% [27].

Few-shot prompting

It allows LLMs to do tasks by providing a few task-specific examples in the prompt. This approach boosts the model's ability to understand the structure of a task through minimal data, allowing it to apply reasoning patterns across new problems [28]. Notably, few-shot chaining of thoughts in prompts leads to great performance because one can walk through a series of reasoning steps with a few examples, making LLMs highly effective for multi-step reasoning problems.

Chain-of-Thought (CoT) Prompting

The CoT prompting helps LLMs in decomposing problems into smaller logical steps, which is reasoning like humans. This really assists in stepwise tasks that involve multi-step calculations, logical deductions, and complex reasoning. When considering CoT prompting, we tell the LLM to "think step by step"; this can push the model to produce intermediate reasoning steps, leading to better accuracy on reasoning tasks [29].

Zero-shot CoT prompting

Pushing this a bit further, zero-shot CoT prompting can manage those cases where no examples are given. The model could perform well if it had the right kind of general direction, for example, prompting like "Let's think step by step" to guide it in reasoning when even task-specific examples are not provided prior [28]. For example, zero-shot CoT increased arithmetic solution accuracy from 10.4% to 40.7% with problems like GSM8K [27].

Reasoning and Conversational Capabilities

LLMs can now handle complex reasoning tasks in a conversational way. They are useful tools for educational applications like intelligent tutoring systems because of their capacity to produce meaningful dialogue and guide users through reasoning steps. To further improve these reasoning tasks, self-prompting zero-shot CoT techniques have been developed, wherein models can independently prompt themselves to validate or improve their reasoning process, leading to improved performance on various benchmarks [29].

Table 2.2: **LLM-Based ITS and Design Recommendations (Post-LLMs)**

Feature	**Description**	**Design Recommendations**	**Ref**
Conversational Capabilities	LLMs provide natural language dialogues and explanations.	Use zero-shot, few-shot, and CoT prompting to simulate human-like interactions.	[12]
Real-Time Feedback	LLMs adapt feedback dynamically based on student input.	Integrate adaptive algorithms to provide instant, personalized feedback.	[13]
Scaffolding	LLMs deliver structured, step-by-step problem-solving guidance.	Leverage Chain-of-Thought (CoT) prompting for effective scaffolding.	[12]
Cross-Domain Flexibility	LLMs can be applied across multiple subjects and tasks.	Develop modular components that are reusable across domains.	[30]
Continuous Learning	LLMs update knowledge dynamically based on interaction history.	Use memory modules to adapt to prolonged interactions and improve over time.	[13]

LLMs also offer several capabilities that are critical for the design of modern intelligent tutoring systems. 2.2 highlights these features, and we will expand on each one to provide a deeper understanding of their implementation:

Conversational Capabilities: LLMs are adept at generating natural language dialogues, making interactions with students more human-like. Using zero-shot, few-shot, and Chain-of-Thought (CoT) prompting techniques, LLMs can simulate meaningful conversations, guiding learners through complex problem-solving processes [12].

Real-Time Feedback: These models adapt to student input in real time, providing instant and personalized feedback. Incorporating adaptive algorithms enables LLMs to respond effectively to various levels of student understanding, thereby enhancing the learning experience [13].

Scaffolding: LLMs can offer step-by-step guidance using CoT prompting, ensuring that learners receive structured assistance in problem-solving tasks. This scaffolding approach is particularly effective for guiding students through multi-step reasoning challenges, where breaking down complex problems is essential [12].

Cross-Domain Flexibility: LLMs exhibit remarkable versatility, capable of adapting to various subjects and tasks. This cross-domain capability enables educators to leverage LLM-based tutoring across different disciplines by developing modular and reusable components [30].

Continuous Learning: LLMs can dynamically update their knowledge base based on ongoing interactions, allowing them to adapt and improve over time. By integrating memory modules, these systems can learn from prolonged interactions with students and provide increasingly tailored support [13].

2.2.2 Capabilities of LLM-Based Tutor Systems

CLASS and SPOCK Systems [12]

Sonkar et al. implemented the CLASS (Conversational Learning Augmentation and Scaffolding System) framework using LLMs to provide step-by-step scaffolding and conversational guidance for learners. Their system, SPOCK, was designed specifically to tutor students in biology, delivering comprehensive, scaffolded learning experiences that guide students through complex biological concepts.

The SPOCK system's strength lies in its ability to:

Provide Detailed Explanations: The LLM offers in-depth explanations of biological processes, adapting the complexity and detail based on the student's level of understanding.

Step-by-Step Scaffolding: Utilizing CoT prompting, the system helps students tackle complex problems by breaking them down into manageable steps.

Real-Time Feedback: As students interact with SPOCK, the system dynamically adjusts its guidance, providing immediate feedback and adapting to their learning pace.

These features exemplify how LLM-based tutors can mimic the role of a human tutor, effectively guiding learners through complex tasks.

AI-Backed Tutoring System [13]

Chen et al. developed an AI-backed tutoring system incorporating features like automatic course planning tools, instructional tailoring, and quiz evaluation. This system is particularly notable for its ability to:

Automatic Course Planning: The LLM can dynamically plan courses based on a student's progress, adapting the curriculum to address their strengths and weaknesses.

Instruction Tailoring: By utilizing memory updates and tracking interaction history, the system delivers individualized responses and learning paths for each student.

Quiz Evaluation: The AI can generate and evaluate quizzes, offering targeted feedback based on student responses, which helps reinforce learning outcomes.

This level of personalization ensures that each learner receives an experience tailored to their unique needs, enhancing engagement and knowledge retention.

2.3 Challenges in Python Tutors and Addressed Solutions

Although LLMs enhance tutoring systems, major challenges exist, especially in developing Python tutoring tools. Some of these include:

- **Tailored Feedback:** Even providing personalized feedback that is adaptive to each individual student's programming level can be somewhat of a challenge since the tutors created in Python tend to over-simplify or over-complicate the feedback.
- **Scaffolding:** Gradual guidance should be designed such that it does not result in the LLMs solving problems for the learners, which is detrimental to programming learning.
- **Error Correction:** Python tutors should be able to assist students in debugging their code by explaining the errors instead of just giving them the right code.
- **Computational Resources:** The running of LLM-based Python tutors demands finding a balance between computational efficiency and the quality of the response.

These challenges are being addressed in our system through the implementation of fine-tuning for Python-specific tasks and in designing real-time error correction mechanisms, with the underlying principle that the system be resource-efficient for large-scale use.

While LLMs have improved tutoring systems with their conversational and personalized capabilities, there remains much to be done in handling programming-specific challenges, especially on how to balance feedback and scaffolding with resource efficiency.

3 System Design

In this chapter, we present the interaction design of PACE, detailing how students interact with the system and how their progress is continuously tracked and updated. We also explain the dataset generation process used to fine-tune the model. Finally, we discuss the design principles to ensure an effective and user-friendly system.

3.1 Interaction Design

Interaction design is the process of creating user-friendly interfaces by considering the psychological and social aspects of users, as well as interaction styles, user requirements, usability, and evaluation. It focuses on ensuring that users can interact with the system efficiently and intuitively while considering their mental models and social behaviors during the interaction [31].

In our system, users interact with the AI Tutor chatbot to learn Python programming through queries. The chatbot is designed to respond in a way that guides students toward solutions while adapting to their individual needs. For new students, the system begins with a brief set of Python questions to assess their current knowledge levels. This initial evaluation allows the AI Tutor to tailor the learning experience based on each student's understanding. This section will explore how these interactions are structured to create an engaging and adaptive learning environment, ensuring both ease of use and effective learning outcomes.

3.1.1 Interaction with the Newly Joined Student

The Figure 2 diagram shows that when a new student joins the system, they will be welcomed with a few questions to assess their current knowledge level of Python. The chapter progress will be updated based on their responses, and the student may skip this interaction. If they skip the interaction, no chapter progress will be updated, and this is a one-time interaction.

3.1.2 Interaction with the PACE Chatbot

The Figure 3 shows the interaction immediately following the welcome interaction. A chat window will appear where students can discuss their Python problems with the chatbot. The model server will generate every response, and the entire conversation will be stored in the database for future reference.

3.2 Dataset Generation for Model Fine-tuning

This section describes the methodology for generating the synthetic dataset employed in our adaptive Python tutoring system. Drawing inspiration from large language models

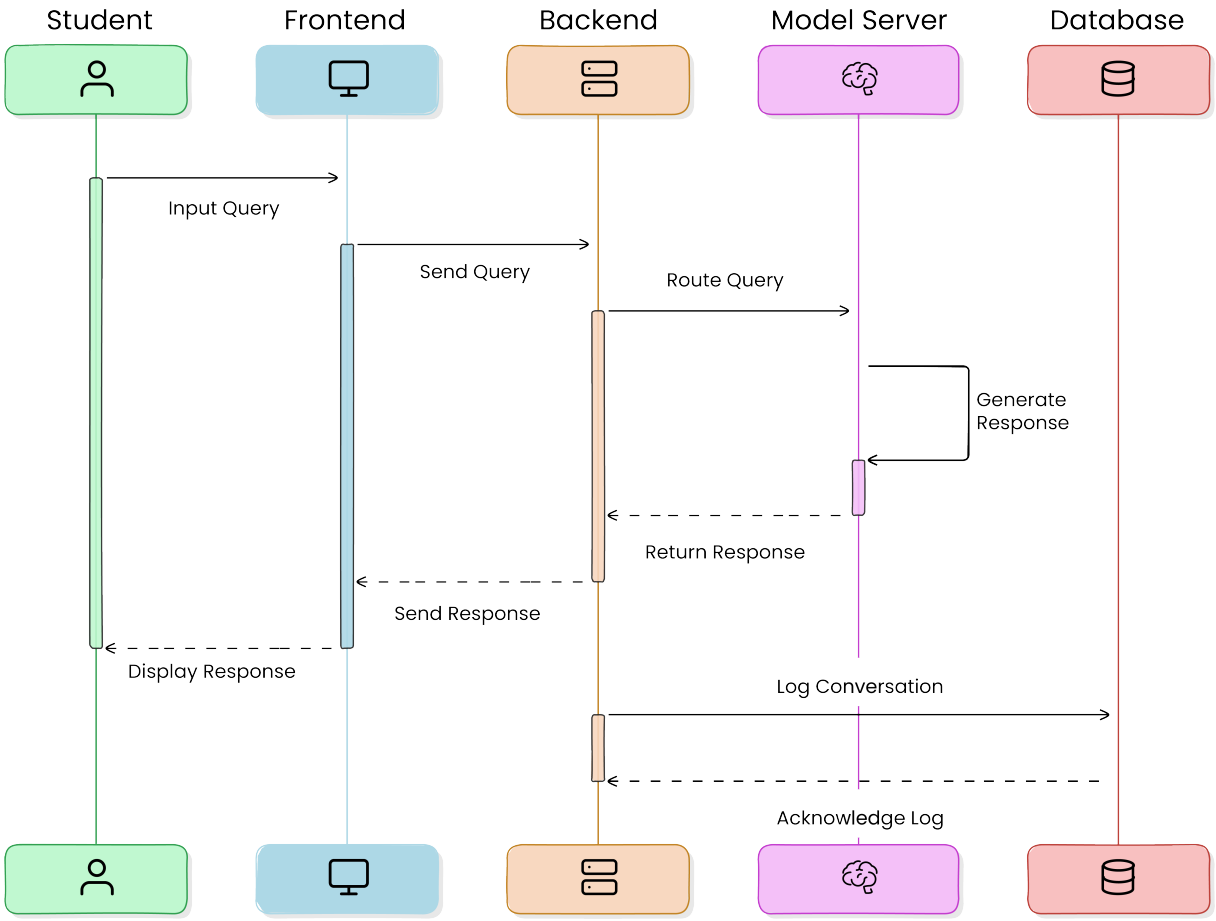


Figure 2: **Sequence diagram showing the interaction between the student, user interface, backend, AI tutor, and database during a learning session. The AI tutor processes the student's input and provides feedback via the backend**

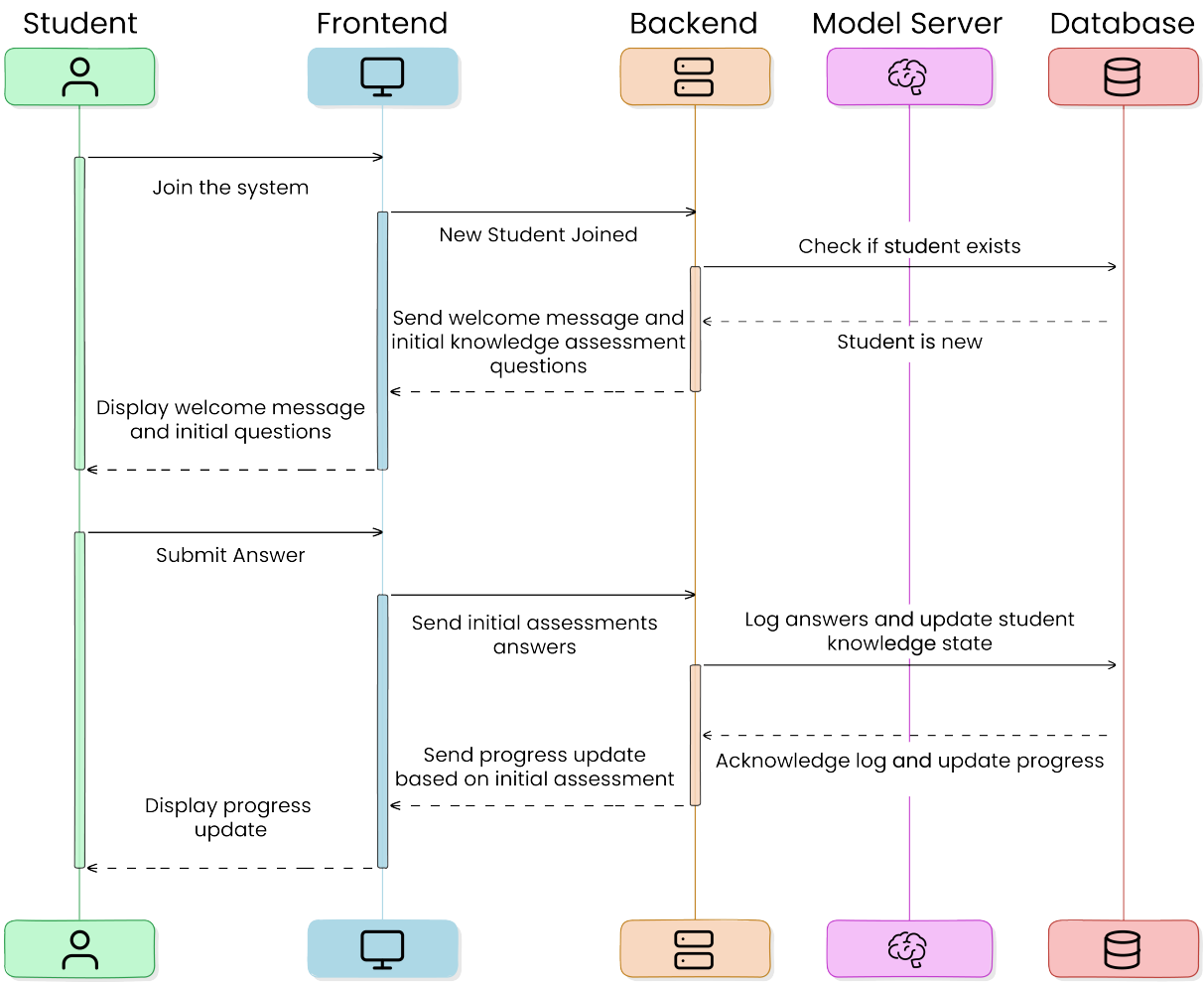


Figure 3: **Sequence diagram illustrating new students signing up for the system. The AI tutor evaluates the students' initial knowledge, retrieves data from the database, and updates student knowledge progress through the backend**

(LLMs) designed for educational purposes, particularly the techniques described in [12], we created a dataset that simulates human tutoring by scaffolding learning tasks, engaging in conversations, and motivating students through carefully crafted prompts. This dataset plays a pivotal role in enhancing student learning and engagement within the system. The whole dataset generation process is outlined in the Figure 4.

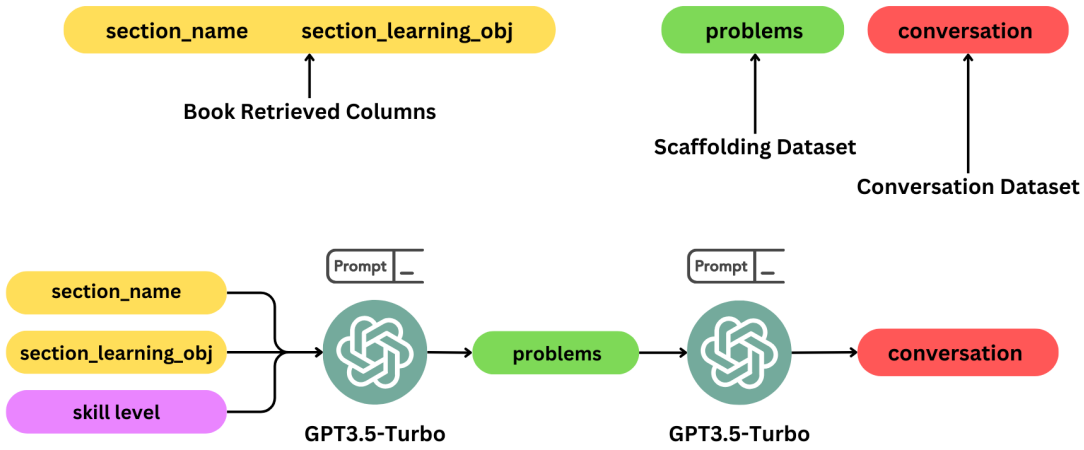


Figure 4: **This workflow shows how datasets are generated by retrieving columns from a Python book and feeding them into GPT-3.5-Turbo. The model first generates problems for a scaffolding dataset using book-retrieved columns and then creates the conversational dataset by using problems from the scaffolding dataset**

3.2.1 Scaffolding Dataset

The scaffolding dataset was generated using a specialized zero-shot CoT prompt in 3.1 using GPT3.5-Turbo to guide students through Python programming problems incrementally by breaking down core concepts into manageable steps. Inspired by the scaffolding approach outlined by Sonkar et el. in [12] and educational frameworks in learning science, the dataset facilitates the stepwise development of skills. It provides progressively more complex tasks while offering hints and explanations that align with a learner's current understanding.

Tasks in the scaffolding dataset cover key Python topics such as variables, control structures, functions, and error handling. Content was extracted from Python programming textbooks, focusing on major sections like "Control Structures," "Data Types," and "Functions." Learning objectives were defined for each section, such as "Understanding loops and conditionals" or "Applying recursion in Python." Student skill levels were categorized as "Lacking Background Knowledge," "Encountering Syntax Errors," and "Logic Implementation Challenges" to align with Bloom's Taxonomy, a framework for organizing cognitive skills from basic knowledge to advanced problem-solving [32]. Lacking Background Knowledge corresponds to remembering and understanding, focusing on foundational concepts like syntax and patterns. Encountering Syntax Errors aligns with applying, involving the

```
Generate a hard, challenging, and unique Python problem which
↪  can be broken down into subproblems for the following
↪  section on {section_name}, whose learning objective is:
↪ {section_learning_objective}, and skill level is
↪  {skill_level}.

For the generated main problem for this learning objective,
↪  also output the following:
1) Facts necessary to answer it,
2) Subproblems that the main problem can be broken down into,
3) The final answer.

For each subproblem, generate A hint, One incorrect student
↪  response to the subproblem, and Corresponding feedback to
↪  the student.

Put all the output in the following JSON structure:
{
"Problem": "..",
"SubProblems": [
{
        "Question": "..",
        "Answer": "..",
        "Hint": "..",
        "Incorrect Response": "..",
        "Feedback": ".."
    }
],
"Facts": [
"..",
".."
],
"Solution": ".."
}

Only provide this JSON result in the output-no other
↪  irrelevant messages are required.
```

Table 3.1: **Scaffolding Dataset Generation Prompt**

practical use of learned coding principles. Logic Implementation Challenges correspond to analyzing and evaluating, emphasizing problem-solving and algorithm design to prepare students for creating solutions.

Each section was broken down into subproblems, hints, incorrect solutions, and feedback. Hints are progressively released as students demonstrate the need for additional support, following a gradual release of responsibility. Some examples of generated data are shown here in A.2. This manages cognitive load and encourages learners to internalize problem-solving processes. By offering an adaptive learning experience that scales in difficulty based on performance, the scaffolding dataset helps learners build confidence and competence to solve more complex problems independently.

3.2.2 Conversational Dataset

The conversational dataset is designed to replicate a natural dialogue between a tutor and a student, focusing on the interactive and dynamic aspects of the learning process. Also inspired by the conversational structures discussed in [12], this dataset includes a variety of dialogues that address common learner misconceptions, provide motivational feedback, and guide students toward self-correction.

To create this dataset, we generated dialogue pairs consisting of student inputs (e.g., questions, coding errors, or misconceptions) and tutor responses using a specialized CoT prompt given in A.1. We used the scaffolding dataset generated before as the context to generate the conversational data. These dialogues are structured to ensure that the tutor's responses are pedagogically effective, focusing on encouraging student inquiry, explaining complex concepts, and guiding students without directly providing answers. The conversational dataset was tailored to cover typical Python errors, including syntax issues, logic problems, incorrect function usage, and general programming-related queries. Some examples of conversational data are shown here in A.3.

By maintaining an interactive flow, this dataset enhances student engagement and supports the development of problem-solving skills through structured, dialogue-based learning.

3.2.3 Prompt Design Strategy

The prompt design strategy is central to the efficacy of the tutoring system, as it determines how the model interacts with students and delivers educational content. We employed a systematic approach to prompt design, leveraging the techniques of CoT prompting, where LLMs were utilized to generate contextually appropriate and pedagogically sound prompts.

Each prompt was crafted to elicit specific types of student behavior, ranging from exploratory coding to reflective problem-solving. The design strategy emphasized maintaining a balance between open-ended and directive prompts, depending on the complexity of the learning task. Open-ended prompts were designed to foster creative thinking, allowing students to experiment with multiple coding solutions. Conversely, directive prompts were used for tasks requiring a more structured approach, particularly in cases where students encountered conceptual roadblocks.

Our strategy also incorporated mechanisms for providing feedback and hints at critical moments, ensuring that students were neither overwhelmed by tasks nor under-challenged. Also, our prompt strategy ensured that the generated content maintained a high degree of clarity, correctness, and engagement. By doing so, the prompt design contributed to an adaptive learning environment responsive to individual student needs.

3.2.4 Dataset Evaluation

The generated synthetic dataset was evaluated based on five key metrics by 3 senior computer science undergrad students: Motivation (M), Clarity (Cl), Correctness (C), Relevancy (R), and Completeness (Co). Each metric was rated on a scale of 1 to 5, focusing on understanding how well the dataset supported learning in Python programming. For evaluation, we picked 55 random data from the dataset and evaluated those based on given matrices.

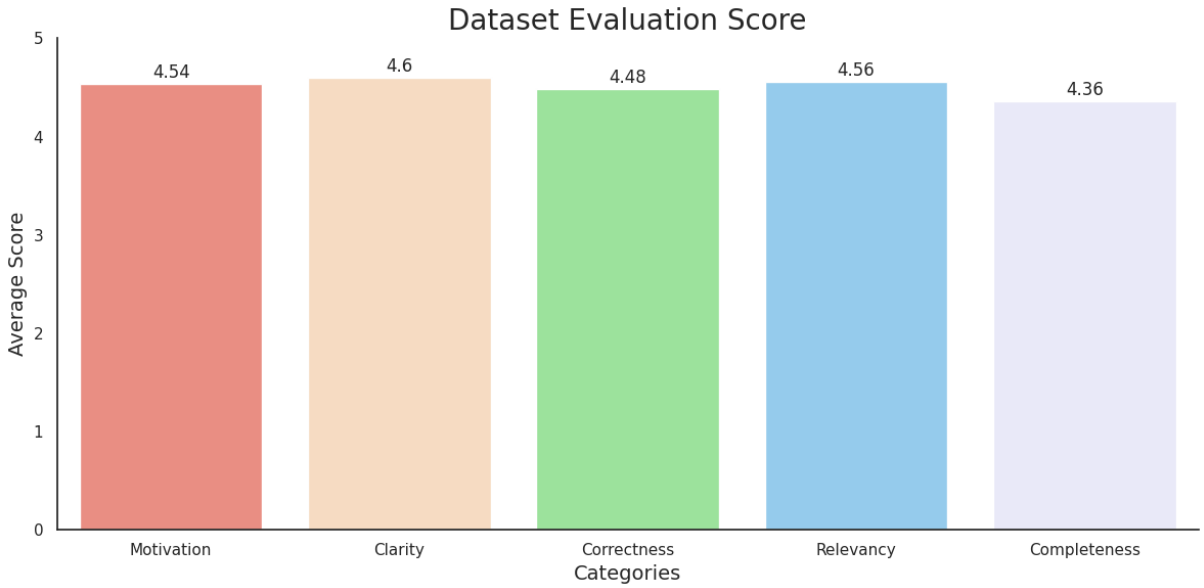


Figure 5: **Evaluation of Conversational Dataset**

- **Motivation (M):** The ability of the tutorbot to keep the student engaged and motivated was assessed. This was done by simulating extended interactions and observing whether the feedback maintained student interest.
- **Clarity (Cl):** The clarity of explanations and instructions given by the tutorbot was evaluated. Students needed to understand the steps involved in solving each problem clearly.
- **Correctness (C):** The factual accuracy of the tutorbot's responses was scrutinized to ensure that no incorrect or misleading information was presented.
- **Relevancy (R):** Each conversation was reviewed to ensure that the responses and hints provided were directly applicable to the problem at hand.
- **Completeness (Co):** The tutorbot's ability to fully address student queries, covering all aspects of the problem before moving on, was measured.

The Figure 5 presents the evaluation scores of the dataset across five categories, each rated on a scale of 0 to 5. The average scores for these categories are 4.54, 4.6, 4.48, 4.56, and 4.36, respectively. The highest score is observed in the Clarity category, suggesting that the dataset provides clear information, making it comprehensible for users. Motivation (M) and Relevancy (R) also scored well, reflecting that the dataset effectively engages users and is relevant to the intended purpose. Correctness (C) received a slightly lower, though still high, score, indicating a minor scope for improvement in ensuring accuracy.

Completeness (C6) scored 4.36, suggesting that while the dataset mostly meets completeness standards, there could be further enhancement to ensure all necessary information is fully covered. Overall, the scores indicate a high-quality dataset, with opportunities for slight improvements in certain areas to achieve excellence in all aspects.

The overall generation of the dataset across these metrics demonstrated its effectiveness in enhancing the student learning experience in Python programming.

3.3 User-Interface Design

Our application's user interface (UI) is designed with a focus on minimalism, ease of use, and consistency. By adhering to essential UI principles and drawing from research in Human-Computer Interaction (HCI), we have crafted an interface that enhances user engagement and reduces cognitive load. Below is a detailed explanation of each design principle we have applied, along with suggestions on what could be added from an HCI perspective to further optimize the interface. The UI design principles we have followed to create our UI are given in Table 3.2 and discussed below:

Figure	UI Design Principle	Description	Supporting Research
Figure 6	**Consistency**	Uniform color palette and clear iconography reduce cognitive load and improve navigation.	[33][34]
	Visual Hierarchy	Key metrics are highlighted using size and icons to guide attention.	[35]
	Feedback	Streak counters provide immediate progress feedback to encourage engagement.	[36]
	Data Graphics	Data graph designs encourage users to engage more intuitively.	[37]
Figure 1	**Familiarity**	Chat interface mimics messaging platforms for intuitive navigation.	[38]
Figure 15	**Modular Layout**	Separate topic boxes for organized navigation.	[39]
Figure 14	**Progressive Disclosure**	Gradual reveal of content helps users focus on one topic at a time.	[40]
Figure 16	**Simplicity**	Minimalist login design reduces cognitive load.	[41]

Table 3.2: UI Design Principles and Supporting Research for Various Interface Designs

- **Consistency:** Consistency across design elements ensures a smooth and predictable user experience. By using a uniform color palette and clear iconography, we reduce the cognitive load, enabling users to navigate through the application effortlessly. The consistency of design elements, such as font styles and button placements, helps users become familiar with the interface and builds confidence in their interactions [33] [34].
- **Visual Hierarchy:** The use of size, color, and icons to highlight key metrics is a crucial aspect of guiding user attention. Visual hierarchy ensures that important elements stand out, allowing users to focus on critical data first. For instance, larger fonts and contrasting colors emphasize primary actions, while smaller fonts indicate secondary information [35].

- **Feedback:** Providing real-time feedback, such as progress bars or streak counters, is essential for engaging users. This feedback helps users track their progress and adjust their behavior accordingly. It also creates a sense of accomplishment that enhances motivation, especially when users receive immediate responses to their actions [36].

- **Data Graphics:** Interactive data graphics are used to present information in a more intuitive way, allowing users to grasp trends and insights quickly. Graphical representations help users engage with data more meaningfully by making complex information easier to understand [37].

- **Familiarity:** The chat interface is modeled after commonly used messaging platforms, making it familiar to users and easy to navigate. Familiar UI patterns allow users to focus on the content rather than learning how to use the interface. This reduces friction and improves user satisfaction [38].

- **Modular Layout:** A modular layout organizes content into separate boxes, allowing users to navigate through different topics without feeling overwhelmed. This structured approach helps users locate information quickly and focus on one section at a time [39].

- **Progressive Disclosure:** This principle ensures that information is revealed gradually, helping users focus on one topic at a time. By limiting the amount of information presented at once, users are not overwhelmed and can concentrate on their current task without distractions [40].

- **Simplicity:** The minimalist design of the login screen reduces cognitive load by presenting only the essential elements. Simplicity is a core design principle in HCI, as it enhances usability by focusing on the most important tasks [41].

4 System Development

The chapter covers the development of our chatbot-based educational system, PACE. It addresses the backend, frontend, and database implementation using FastAPI, React, and MySQL, respectively.

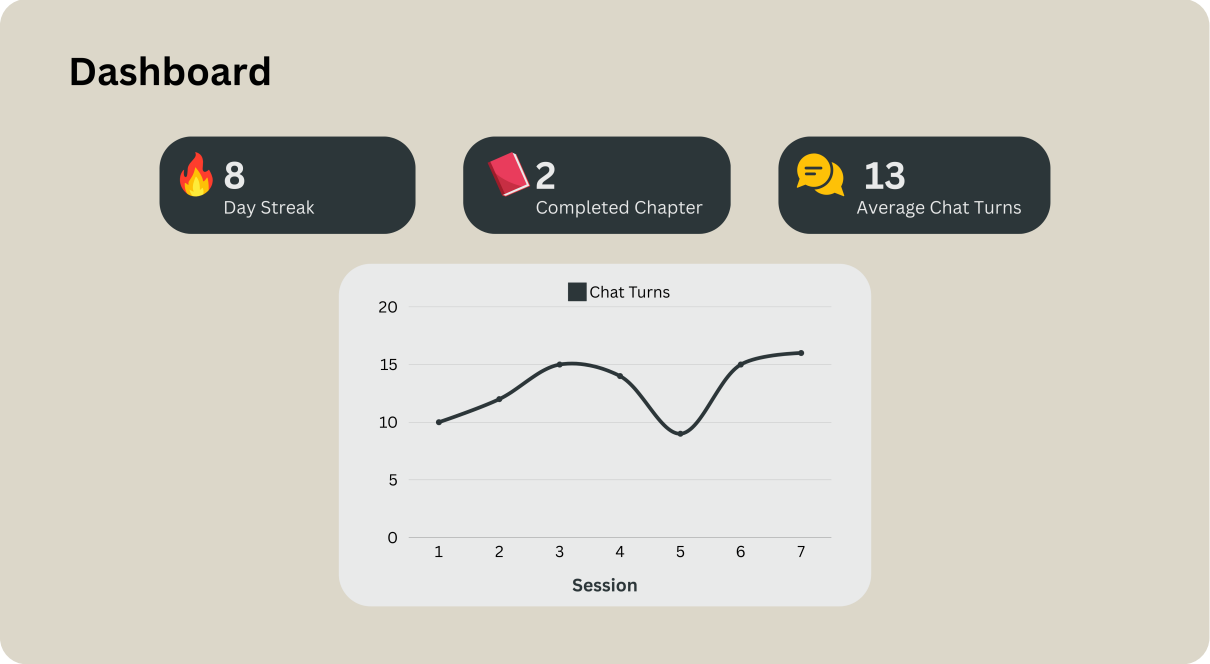


Figure 6: **Student Performance Dashboard**

4.1 Development

The developed system is a chatbot designed for educational purposes, allowing students to interact with a fine-tuned Gemma 2b model for Python programming tutoring. The system is built using a combination of FastAPI for the backend, MySQL for database management, and React for the frontend. Each component plays a critical role in ensuring seamless functionality and a smooth user experience. This section elaborates on the implementation details of the system, dividing it into three main parts: Backend, Frontend, and Database.

4.1.1 Backend

The core logic of the system is handled in its backend, including preparation of responses to the inquiries of students and maintenance of communication between the front-end interface and the MySQL database.

Language Model and API Integration

The chatbot is built on the fine-tuned version of the Gemma 2b language model. The model was pushed to the Hugging Face repository for access and versioning. The model has been fine-tuned with the student-tutor conversational data; hence, it is for educational purposes whereby students' queries are responded to meaningfully. The API that will handle the requests and generate responses is built upon the high-performance Python web framework for building APIs: FastAPI. FastAPI's choice is based on its asynchronous capabilities; it therefore efficiently handles multiple requests, which improves system responsiveness. Upon the submission of the query by a student in the chat interface, a request is sent to this API endpoint, which asks for a response using the fine-tuned Gemma 2b model. Finally, the responses are returned to the frontend and displayed to the user in the chat interface. The communication is frictionless, and this counts as real-time communication between user and chatbot. Not only this all the functionality of the website is handled using FastAPI.

4.1.2 Frontend

The responsibility of the frontend part of this system is to reflect an interactive, user-friendly interface. It has been developed in React, one of the most widely used JavaScript libraries for dynamic user interface development. The focus of the frontend is to let the student interact with the system seamlessly by assuring intuitive navigation within all the different components.

React Interface Design

React was used here, as these are component-based architecture features that allow for modular and reusable design. Every feature of the system has been designed as an independent component: the login page, chat page, dashboard, etc.—all have maintainable codebases that can confront scalability. The chat interface is the point at which the user interacts with the chatbot, and here the design is simple and responsive. It takes query input from the user and supports the output that would be generated from the system in real time. It also provides easy navigation through different sections, such as Dashboard, Chat, with its responsive sidebar navigation.

User Interaction

Once the student submits a query in the chat window, the request gets forwarded to the backend API, while the response of the model is reflected instantaneously. Further, the system keeps track of the student's progress and history of interaction-engagement, all of which get updated dynamically in the dashboard. Efficient rendering by React ensures minimal delays for the user during interactions.

4.1.3 Database

The database layer of the system is built using MySQL, a robust relational database management system that is well-suited for handling structured data. The database is responsible for storing and managing all critical information related to the system, such as user details, user history, and progress.

Database Structure

The MySQL database consists of several tables, each of which is responsible for storing different types of information:

User Information: This table stores user credentials, personal information, and activity logs.

Progress Tracking: This table maintains each user's progress in learning modules, storing data such as completion status and performance.

Chat History: A separate table stores user queries and the system's generated responses, allowing for tracking interactions over time. This feature helps in personalizing user experience based on past interactions.

Data Flow

Whenever a user interacts with the system (e.g., submitting a query), the data is logged in the MySQL database. The backend interacts with the database through SQL queries to retrieve, insert, or update data as required. This allows the system to maintain a comprehensive record of user activity, which is crucial for tracking progress and generating personalized feedback.

Data Security and Integrity

Ensuring the security and integrity of user data is a primary concern. All sensitive data, such as user credentials, are encrypted before being stored in the database. Additionally, MySQL provides robust mechanisms to ensure data integrity, such as transaction control and error handling during database operations.

5 Model Selection & Evaluation

Fine-tuning is essential for adapting a pre-trained model to specific tasks, enhancing performance and relevance in particular domains. In this chapter, we outline the fine-tuning process for the Gemma 2B Instruct model by Google DeepMind [42], using Low-Rank Adaptation (LoRA), which allows for computational efficiency by updating a limited number of parameters while maintaining most of the pre-trained weights. This strategy is particularly beneficial for fine-tuning large models like Gemma 2B for domain-specific tasks such as Python tutoring[43].

5.1 Model Selection

There are few small open-source models with around 2 billion parameters, making Gemma 2B a unique option. Compared to models like Tiny Llama, GPT-Neo 1.3B, GPT-Neo 2.7B, and Microsoft's Phi, Gemma 2B consistently outperforms its peers across various benchmarks. Its open-source nature allows for greater flexibility, including fine-tuning for specific domains such as Python tutoring, which makes it an attractive choice over proprietary models like GPT-3, where usage costs can be prohibitive. Additionally, Gemma 2B is more computationally efficient, requiring fewer resources during both training and inference, making it a cost-effective solution for large-scale deployment.

For instance, models like GPT-Neo 2.7B, while comparatively larger in size, struggle with the same level of efficiency and versatility that Gemma 2B offers, especially in specific task domains. Recent comparisons show that Llama-2 and GPT-Neo models may compete on broader natural language tasks, but when it comes to instruction-based and domain-specific tasks like tutoring, Gemma 2B's optimization strategies and resource efficiency make it the superior choice [44][18]. This positions Gemma 2B as a cost-efficient, open-source alternative with robust performance across targeted tasks.

5.2 LoRA Fine-tuning

LoRA fine-tuning was chosen because it reduces the computational resources required for training large models by updating only low-rank matrices while keeping the original model weights frozen. This approach is particularly efficient when fine-tuning models like Gemma 2B, which has billions of parameters. In the case of Python tutoring, LoRA adapts the model's ability to answer complex coding questions and provide step-by-step guidance [45].

Recent advancements have demonstrated the effectiveness of LoRA in reducing the number of trainable parameters by up to 10,000 times, making it highly efficient for large-scale models [43]. Studies have also shown that LoRA performs on par or even better than full fine-tuning on tasks such as summarization, question answering, and instruction following [46].

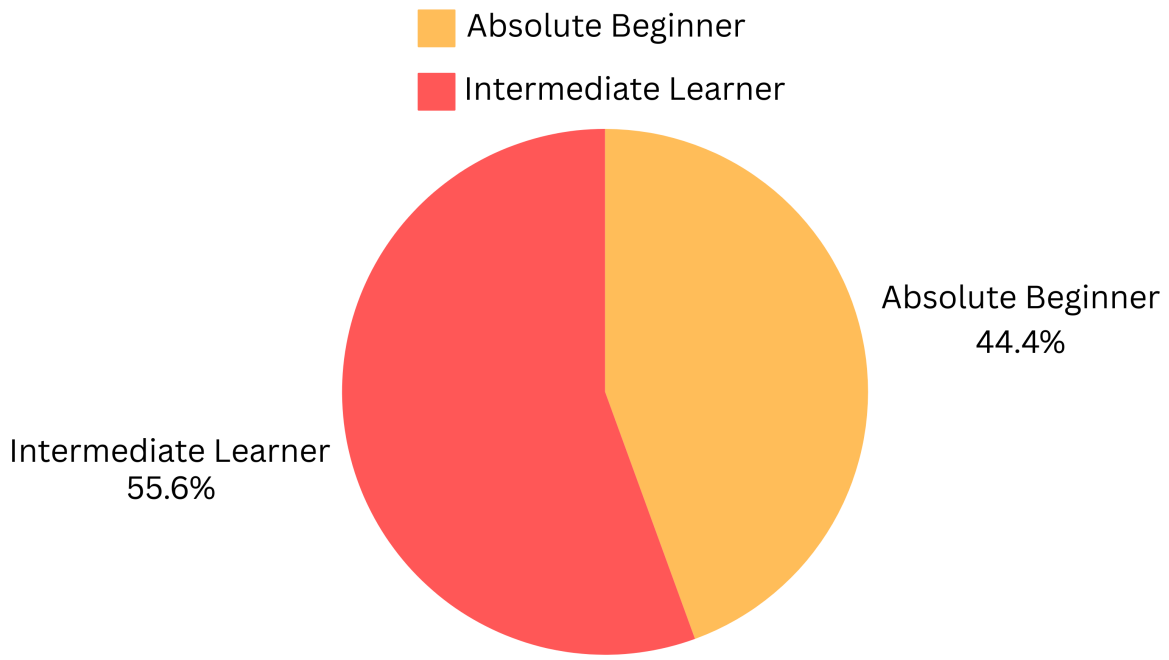


Figure 7: **Distribution of Python Knowledge Levels Among Students**

5.3 System Evaluation

The performance of the PACE is assessed using several metrics that measure the system's impact on learning outcomes and the quality of interaction it provides. The evaluation aims to understand how effectively the system supports learning and fosters an engaging educational environment.

5.3.1 Chat Response Evaluation

We surveyed 18 students who interacted with our system to evaluate the chatbot's responses. Students answered 25 questions across five categories: Relevancy (R), Correctness (C), Completeness (Co), Motivation (M), and Clarity (Cl), based on metrics discussed in 3.2.4. This feedback measured the perceived effectiveness of the chatbot in supporting learning through its interactions. Each question was rated on a Likert scale from 1 (lowest) to 5 (highest). The Table 5.1 summarizes the mean scores for each question across five evaluation categories, highlighting both strengths and areas for improvement in the chatbot's performance. In addition to evaluating the chatbot's effectiveness, we assessed the respondents' Python knowledge levels. The Figure 7 results indicated that 44.4% of participants identified as Absolute Beginners, while 55.6% considered themselves Intermediate Learners. The Absolute Beginners are those who are currently learning Python and have not completed courses on data structures and algorithms. On the other hand, the Intermediate Learners are those who have taken courses on data structures and/or algorithms in Python or any other programming language. The highest scores were seen in areas such

	Q1	Q2	Q3	Q4	Q5
Relevancy (R)	3.83	3.72	3.33	3.61	3.50
Correctness (C)	3.83	3.78	3.33	4.06	3.56
Completeness (Co)	3.78	3.83	3.78	3.67	3.28
Motivation (M)	3.72	3.67	3.83	3.67	3.83
Clarity (Cl)	3.61	3.50	3.33	3.78	3.72

Table 5.1: **Mean scores for Relevancy (R), Correctness (C), Completeness (Co), Motivation (M), and Clarity (Cl) across questions Q1 to Q5**

as the relevance of suggestions (RQ1, 3.83) and correct definitions of programming terms (CQ4, 4.06), demonstrating that users found the chatbot effective in providing appropriate guidance and accurate explanations. Motivation-related questions (MQ1 and MQ5, 3.83) also received positive ratings, showing that the chatbot was generally successful in inspiring users to continue coding and sparking their interest. However, lower scores were noted for questions like the appropriateness of examples (RQ3, 3.33) and solutions leading to correct outputs (CQ3, 3.33), suggesting that the chatbot could benefit from improvements in tailoring examples and ensuring reliable solution outputs. Completeness-related aspects, such as follow-up recommendations (CoQ5, 3.28), and guidance clarity (ClQ3, 3.33) also showed potential for enhancement. While the chatbot performs well in several key areas, refining certain aspects, such as appropriateness, solution reliability, and comprehensive guidance, could further enhance user satisfaction and learning outcomes. Additionally, some of the best chat responses can be found in A.4. A comprehensive review of the evaluation categories and survey is discussed below:

Relevancy (R) This category evaluates the relevance of the chatbot's responses to the students' programming needs:

- **RQ1**: Relevance of suggestions for programming challenges.
- **RQ2**: Relevance of responses to topics asked about.
- **RQ3**: Appropriateness of provided examples.
- **RQ4**: Applicability of responses to real-world scenarios.
- **RQ5**: Alignment with user expectations for Python help.

The Figure 8, focusing on the Relevancy (R) of the chatbot's responses, shows how well the system meets students' programming needs. Each bar represents a different research question (RQ1 to RQ5), evaluating the relevance of the chatbot's output. The distribution for RQ1, which assesses the relevance of suggestions for programming challenges, shows a balanced mix of mid-to-high scores, indicating that users generally found the chatbot's suggestions useful but with some variability. RQ2, addressing the relevance of the chatbot's responses to the topics asked, and RQ4, assessing the applicability of responses to real-world programming scenarios, are both skewed towards higher scores. This suggests that users found the chatbot's responses largely relevant and practical. RQ3, which evaluates the appropriateness of provided examples, shows a broader range of scores with more mid-level responses, suggesting mixed feedback in this area. Finally, RQ5, which measures how well the chatbot's responses align with user expectations for Python help, indicates a mixed range of scores, from 2 to 5, with most scores being 3 and above. This

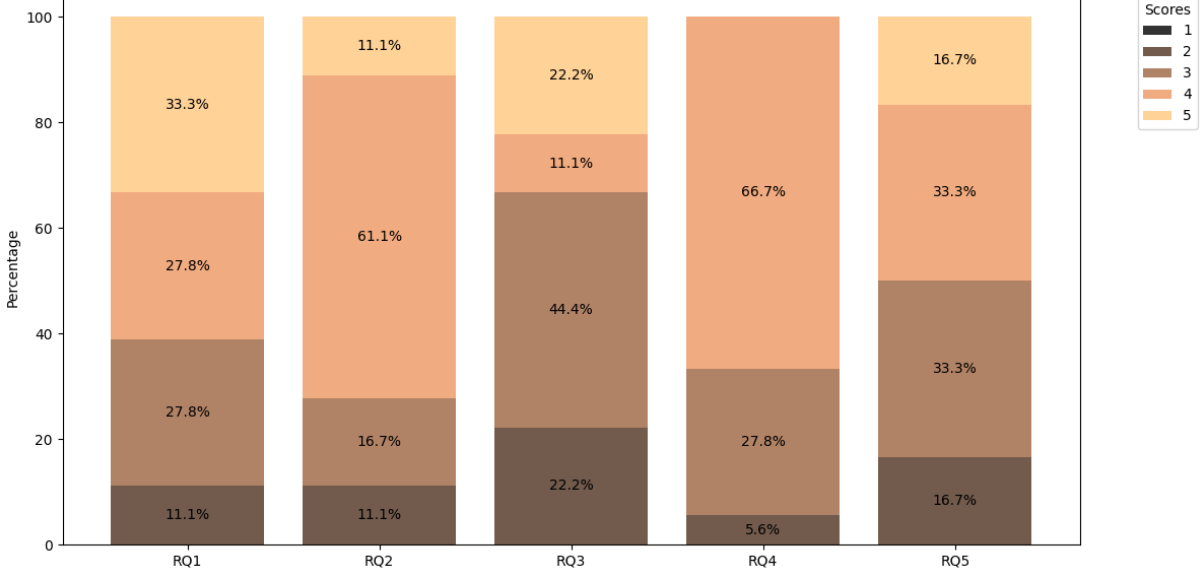


Figure 8: **Relevancy (R): Distribution of scores on the chatbot's relevance to user programming needs.**

suggests that while there is some variability in user satisfaction, most users find the chatbot's responses to be at least moderately applicable. Overall, the findings for Relevancy demonstrate that while the chatbot is generally effective in providing relevant support, there are opportunities to enhance consistency in how examples and responses align with users' programming needs.

Correctness (C) Correctness evaluates the accuracy of responses:

- **CQ1**: Correct syntax in examples.
- **CQ2**: Accuracy of concept explanations.
- **CQ3**: Solutions leading to correct outputs.
- **CQ4**: Correct definitions of terms.
- **CQ5**: Effectiveness in correcting misconceptions.

The Figure 9 centers on Correctness (C), evaluating the accuracy of the chatbot's responses. The CQ1 to CQ5 categories reveal insights into various aspects of correctness, such as whether the chatbot provided the correct syntax in examples (CQ1) and the accuracy of concept explanations (CQ2). CQ1 displays a distribution where most users rated it between 3 and 5, suggesting the examples were often correct but with occasional inaccuracies. CQ2, which examines the chatbot's accuracy in explaining concepts, shows a higher concentration of scores in the 4 and 5 range, indicating strong user satisfaction. However, CQ3, which evaluates whether the chatbot's solutions led to correct outputs, has a more even distribution, implying some variability in users' experiences. CQ4, assessing the correctness of definitions for programming terms, shows a favorable rating trend, highlighting that users generally found the chatbot reliable in this aspect. Lastly, CQ5, which measures the chatbot's effectiveness in correcting misconceptions, shows a moderate spread of responses, suggesting there is room for improvement in this area. While the chatbot demonstrates high accuracy in its explanations and definitions, certain aspects, such as ensuring solutions consistently lead to correct outputs, could be refined.

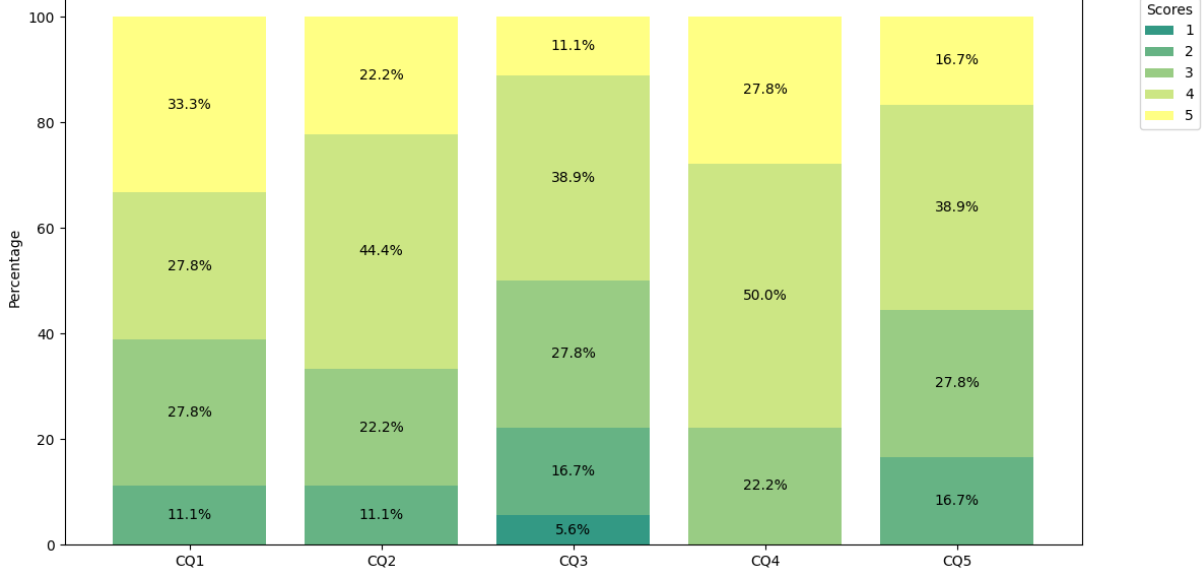


Figure 9: **Correctness (C): Distribution of scores assessing the chatbot's accuracy and correctness in responses.**

Completeness (Co) Completeness assesses how thorough the responses are:

- **CoQ1**: Provision of necessary problem-solving steps.
- **CoQ2**: Completeness of explanations.
- **CoQ3**: Inclusion of relevant examples.
- **CoQ4**: Thoroughness in covering topics.
- **CoQ5**: Recommendations for further learning.

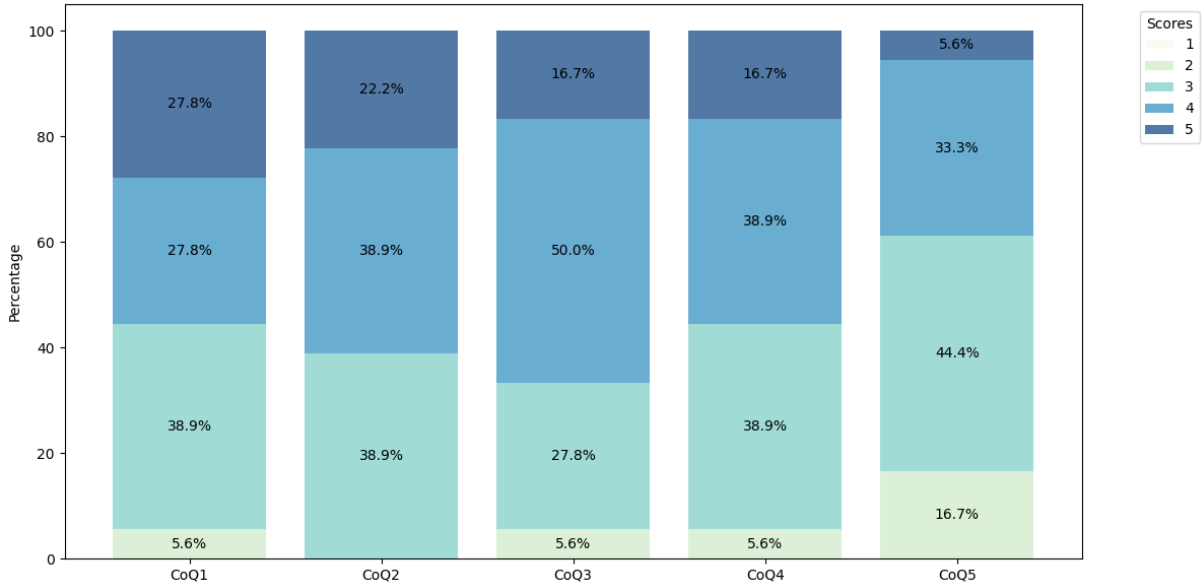


Figure 10: **Completeness (Co): Distribution of scores evaluating the thoroughness of the chatbot's responses.**

The Figure 10, representing Completeness (Co), assesses how thorough the chatbot's responses are. CoQ1 through CoQ5 address various aspects of completeness, such as the

provision of necessary problem-solving steps (CoQ1) and the thoroughness of topic coverage (CoQ4). CoQ1 shows a spread of scores predominantly around 3 and 4, indicating that while many users found the chatbot thorough in providing steps, there were instances where completeness was lacking. CoQ2, evaluating the completeness of explanations, reflects a similar trend, with a balance between mid-level and high scores. CoQ3, which assesses the inclusion of relevant examples, shows a slightly lower distribution, with some users rating it a 3, highlighting potential areas for enhancement in the completeness of examples. CoQ4, focusing on the thoroughness in covering topics, received higher scores overall, suggesting that users found the chatbot generally comprehensive. Finally, CoQ5, measuring the chatbot's recommendations for further learning, shows a diverse distribution, indicating variability in how helpful users found the follow-up suggestions. This illustrates that while the chatbot tends to provide thorough responses, certain aspects, including examples and recommendations for further learning, could be improved for better completeness.

Motivation (M) Motivation assesses the chatbot's ability to inspire learning:

- **MQ1**: Encouragement to continue coding.
- **MQ2**: Motivation to tackle complex tasks.
- **MQ3**: Inspiration to practice Python.
- **MQ4**: Confidence-building feedback.
- **MQ5**: Interest sparked by responses.

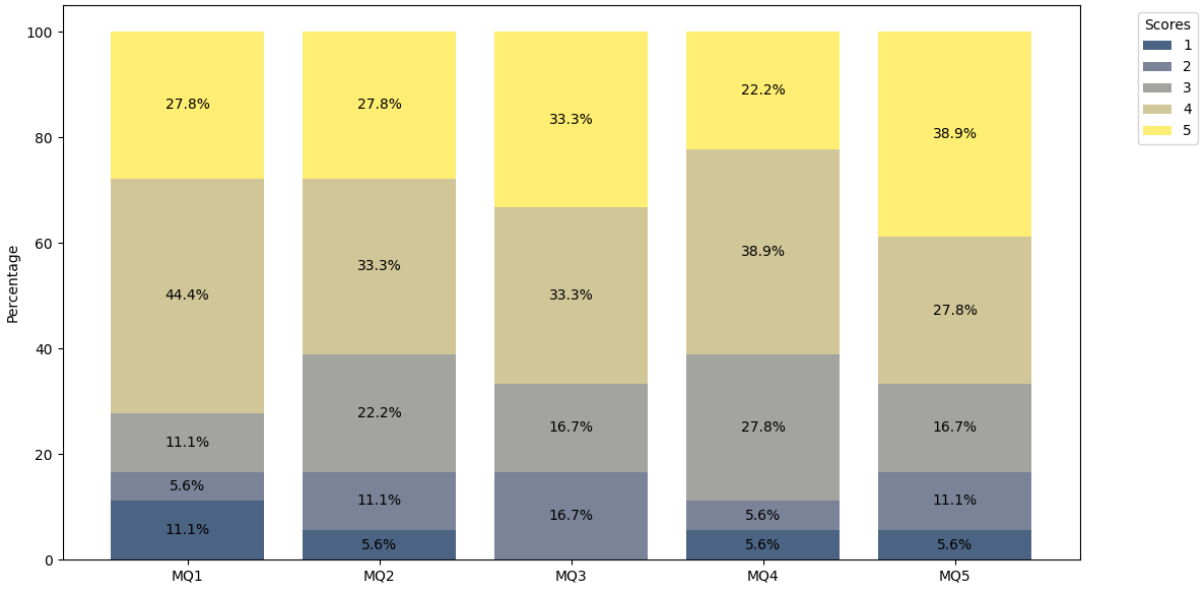


Figure 11: **Motivation (M): Distribution of scores on the chatbot's effectiveness in motivating users.**

The Figure 11 discusses Motivation (M) and the chatbot's ability to inspire students. MQ1 to MQ5 represent how the system encourages learning and fosters user engagement. MQ1, related to the chatbot's encouragement to continue coding, shows a positive distribution, with many users rating it 4 or 5, indicating that users felt somewhat motivated.

MQ2, which evaluates whether the chatbot motivated users to tackle more complex programming tasks, has a mix of high and mid-level scores, reflecting variability in how inspired users felt to take on challenges. MQ3, which focuses on the inspiration to practice Python, presents a broad range of scores with notable responses at 3 and 4, showing that while the chatbot has an encouraging effect, it might not fully motivate all users. MQ4, which evaluates whether feedback from the chatbot boosted user confidence, shows higher ratings, indicating that many users felt more confident after receiving assistance. MQ5, assessing whether the chatbot sparked interest in programming, also has strong positive scores, indicating that the system effectively engages users and stimulates their interest. This highlights that the chatbot has motivational capabilities but suggests that targeted improvements could make interactions more uniformly inspiring and engaging.

Clarity (Cl) Clarity measures the clarity of responses:

- **ClQ1**: Clarity of error-fixing instructions.
- **ClQ2**: Clear explanation of reasoning.
- **ClQ3**: Ease of following guidance.
- **ClQ4**: Understandability of terms.
- **ClQ5**: Straightforwardness of examples.

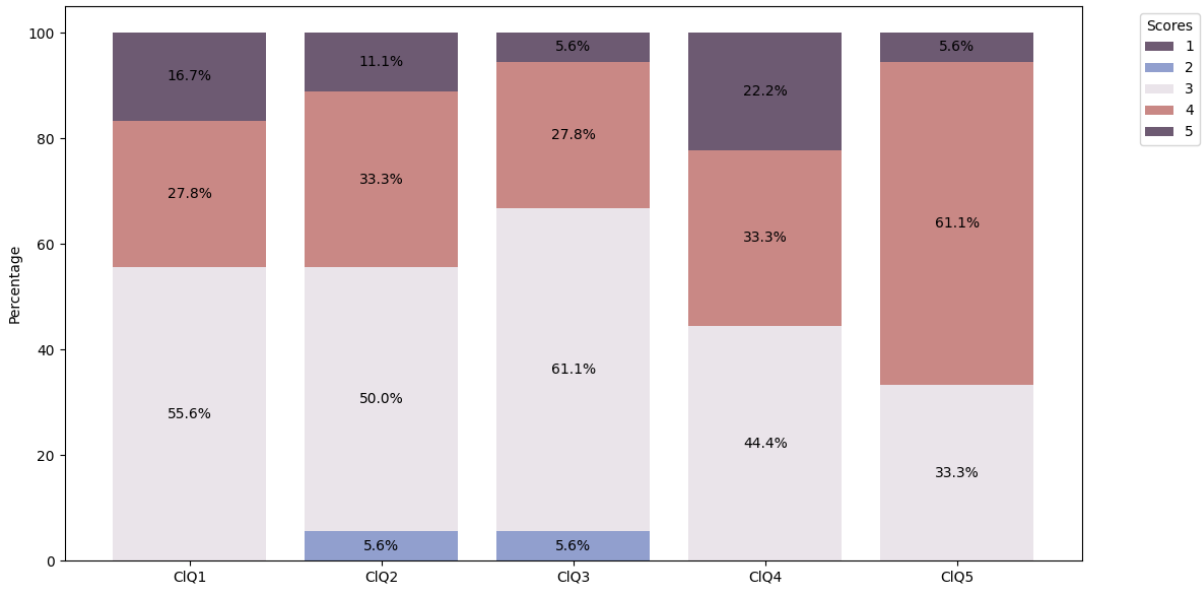


Figure 12: **Clarity (Cl): Distribution of scores assessing the clarity and understandability of the chatbot's responses.**

The Figure 12 focuses on Clarity (Cl) and the instructional quality of the chatbot's responses. The ClQ1 to ClQ5 categories explore different aspects, including the clarity of instructions (ClQ1) and the chatbot's ability to use understandable terms (ClQ4). ClQ1 shows a strong rating distribution with scores centered around 3 and 4, implying that while users found instructions fairly clear, there were instances of ambiguity. ClQ2, which assesses how well the chatbot explained the reasoning behind solutions, received high scores in the 4 and 5 range, showing that users were satisfied with this aspect. ClQ3, which measures the ease of following the chatbot's guidance, presents a broader

distribution, with a significant number of scores at 3, indicating that some users found it harder to follow instructions. ClQ4, focusing on the ease of understanding the terms used, shows a favorable response, suggesting that the chatbot effectively communicated using accessible language. ClQ5, which evaluates how straightforward the examples were, has a mixed distribution, with many users rating it 4 but also a notable portion at 3. This pattern suggests that while the chatbot generally excels in clarity, refining how examples are presented and ensuring consistent simplicity in instructions would further enhance the user experience.

In summary, the above figures and the mean scores across the five evaluation categories—Relevancy (R), Correctness (C), Completeness (Co), Motivation (M), and Clarity (Cl)—reveals that the chatbot demonstrates solid performance in delivering relevant, motivating, and accurate support for programming needs. Strengths are evident in areas such as the relevance of suggestions, correct definitions, and the ability to inspire users to continue coding. However, the evaluation also identifies specific areas where enhancements could be made, such as providing more appropriate examples, ensuring solutions consistently lead to correct outputs, and offering more comprehensive and straightforward guidance. These insights underscore the overall effectiveness of the chatbot while highlighting opportunities for targeted improvements to elevate user satisfaction and learning outcomes further.

5.3.2 System Usability Study

To evaluate the system's overall usability, we employed the System Usability Scale (SUS), a widely used tool for assessing perceived ease of use [47] [48]. After interacting with the system, the same 18 students completed the SUS questionnaire. The SUS provides a score between 0 and 100, with higher scores indicating better usability. For educational web applications, a good SUS score is 70.09 [49]. The results of this study are crucial in determining the system's accessibility and user-friendliness, which are key factors in fostering an effective learning experience.

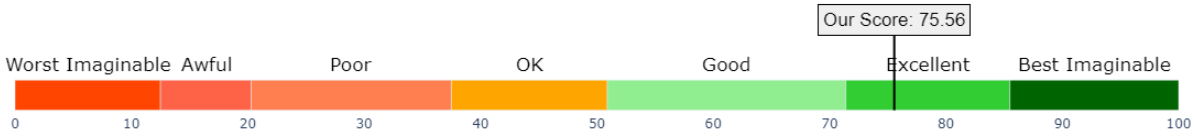


Figure 13: **SUS score ranges with categories from "Worst Imaginable" to "Best Imaginable." The vertical line indicates our system score of 75.56, categorized as "Excellent."**

Figure 13 illustrates the SUS evaluation. The SUS score ranges from 0 to 100 and is segmented into seven categories: "Worst Imaginable," "Awful," "Poor," "OK," "Good," "Excellent," and "Best Imaginable," distinguished by color from red to dark green. Our system scored 75.56, placing it in the "Excellent" category, indicating high usability.

A score above 68 is typically considered above average, while scores above 80 are deemed excellent. The score of 75.56 suggests that the system offers a positive user experience and aligns well with usability expectations. While this is a strong result, further improvements could focus on interface clarity, navigation, and user guidance to enhance

satisfaction and push the score closer to the "Best Imaginable" category. Substantially, the SUS evaluation demonstrates that the system is user-centric and supports its intended tasks effectively.

5.3.3 Mistakes and Errors from the System

In addition to the evaluation, we have gathered user comments for further improvement. Below are the major insights from users, along with our observations:

Incorrect or Hallucinated Responses One of the primary issues identified by users is the occurrence of incorrect or hallucinated responses. These responses often fail to align with the questions asked, leading to inaccuracies that undermine user trust in the system. For instance, one user noted, *"The system often provides wrong answers when we chat for too long and lose its context and it can't recall its memory of past questions,"* indicating that prolonged interactions can result in the chatbot losing track of the dialogue context and generating unreliable outputs.

Long or Overly Complex Responses User feedback highlighted that the chatbot sometimes produces responses that are too lengthy or complex for the intended users, particularly beginner Python learners. Comments such as *"Sometimes, it provides very long responses that the students fail to engage,"* indicate that the verbosity of the chatbot's output can hinder user engagement and comprehension. Additionally, the feedback *"It often provides very advanced responses which are not well-suited for beginner Python students"* suggests that while the system may possess advanced capabilities, these can be counterproductive when addressing users with limited experience.

Context Retention and Short Context Windows The system's limitations in context retention were a significant concern raised by users. The comment *"It forgets what I asked in the first place,"* indicates that the chatbot's ability to recall past interactions is limited due to small Context Window, leading to disjointed or irrelevant follow-up responses. This concern was further echoed by users who stated, *"The system often provides wrong answers when we chat for too long and lose its context,"* emphasizing that maintaining coherence over extended interactions needs improvement.

Formatting Issues in Code Outputs The quality of code presentation was another commonly cited issue. Users pointed out that the chatbot frequently fails to maintain proper code indentation, which is crucial for readability and comprehension in programming contexts. Feedback such as *"The code indentation is not maintained when any examples are shown"* and *"It would be better if coding examples were given with proper indentation"* highlight the need for improvements in the system's ability to format code outputs according to standard practices.

Limitations in Handling Non-Python Queries and Code Conversion A notable limitation users identify is the chatbot's struggle with handling programming queries outside of Python or requests for code conversion. One user observed, *"When anyone asks for help to understand code other than Python or to convert it to Python, it often fails miserably,"* suggesting that the system's current capabilities are too narrowly focused and lack the versatility required to support broader programming needs.

Monotony in Interaction and User Engagement Another drawback mentioned was the monotonous tone of the chatbot's responses, which can result in reduced user engagement. Feedback such as *"Due to its continuous monotonous tone, students often get bored or disturbed"* indicates that while the content may be informative, the delivery lacks the variation needed to maintain user interest. Enhancing the conversational tone and introducing variability could improve the overall user experience.

5.3.4 Strengths and Weaknesses of the System

Strengths The chatbot addresses simple Python questions, delivers clear and accurate explanations, and motivates learners through interactive prompts. Users have noted that the system engages beginners effectively, ensuring that they remain encouraged and promoting continuous learning.

Weaknesses The primary weaknesses of the system include challenges in maintaining long-term context, resulting in errors and irrelevant responses during extended conversations. The system's tendency to provide overly detailed or complex responses can also make comprehension difficult for beginners. Additionally, the chatbot falls short when tasked with handling non-Python programming queries or performing code conversions. Issues with code formatting and the monotony of interactions further contribute to user disengagement over time.

While the chatbot system has shown effectiveness in supporting beginner Python learners and has demonstrated potential in promoting engagement and learning, addressing these limitations—including context retention, response simplicity, formatting, and versatility—will enhance its overall performance and user satisfaction.

6 Conclusion and Future Works

6.1 Conclusion

In this project, we developed a smart Python tutor powered by a fine-tuned version of the Gemma 2B model. This has demonstrated the ability of the system to offer adaptive and interactive learning experiences, especially with introductory programming courses. In particular, the use of Low-Rank Adaptation fine-tuning has made it possible for efficient adaptations of small models such as Gemma 2B to deliver meaningful educational content. This was because the project brought to light the immense benefits that come with using LLM tutors, for instance, in its ability to scale, give real-time feedback, and offer guidance—benefits that ultimately improve student performance.

6.2 Limitations

PACE, built on a fine-tuned Gemma 2B model, accurately answers basic Python questions and promotes student persistence. However, it has significant limitations, including difficulty maintaining context in extended interactions, which can lead to incoherence in long dialogues. Its responses may also be too complex for beginners, and it struggles with non-Python programming queries and code formatting issues. It also does not match the capabilities of larger models like Llama 3.1 or GPT-4o for complex reasoning tasks. Deploying these larger models can be prohibitively expensive for educational institutions on tight budgets, creating a challenge in balancing performance and cost-effectiveness. Additionally, the system lacks advanced features like knowledge tracing, which could improve its utility for tutors by enabling dynamic assessment of student knowledge. Overall, both the chatbot and the fine-tuned Gemma 2B model highlight the need for ongoing improvements to enhance educational experiences while addressing budget constraints.

6.3 Ethical Considerations

AI-powered tutoring systems raise several ethical concerns regarding students' privacy, data security, and bias in content delivery. It becomes very necessary to ensure data gathered during interactions is securely handled and the tutor operates within a framework that respects the privacy of its users. There is also the danger that, if not rectified, the model may give fallacious or biased feedback, leading to misguidance of students. Therefore, it is important to put in place mechanisms to check the accuracy and fairness of the content produced by the model. Educators and institutions should also be certain that the system does not replace a system of human oversight and intervention but works in concert with it.

6.4 Future Plan

In the future, we aim to enhance PACE by integrating larger models to handle more complex problem-solving and reasoning tasks, particularly benefiting intermediate learners. Real-time student progress tracking and knowledge tracing will be implemented to dynamically adjust quiz questions and provide personalized learning experiences. To further engage users, we will introduce gamification elements, such as badges, points, and leaderboards, encouraging users to complete challenges and progress through their learning journey. A new dashboard will greet users upon opening the chatbot, and previous conversations will be stored in the database for easy retrieval.

Ensuring the chatbot provides correct answers with minimal delays will be a top priority, alongside reducing hallucinations through further training. The chatbot will guide users by breaking down problems into parts, offering pseudo code instead of full solutions right away to foster critical thinking. User interface improvements will address theme inconsistencies, particularly aligning the buttons and icons with the overall color scheme, while ensuring clear and consistent typography.

Additionally, users will have more customization options for both the chat interface and their profiles. We plan to increase the word limit for responses and ensure proper indentation in coding examples for better readability. By incorporating Cloud GPUs, we aim to boost performance and reduce response times, making the system more responsive, particularly for advanced queries. With these enhancements, including gamification, PACE will offer a more effective and engaging learning experience for both beginner and intermediate Python learners.

*

Bibliography

[1] Yizhou Qian and J. Lehman. Students' misconceptions and other difficulties in introductory programming. *ACM Transactions on Computing Education (TOCE)*, 18:1 – 24, 2017.

[2] E. Kochmar, Dung D. Vu, Robert Belfer, Varun Gupta, Iulian Serban, and Joelle Pineau. Automated personalized feedback improves learning gains in an intelligent tutoring system. *Artificial Intelligence in Education*, 12164:140 – 146, 2020.

[3] Benjamin Xie, Dastyni Loksa, Greg L. Nelson, Matthew J. Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li, and Amy J. Ko. A theory of instruction for introductory programming skills. *Computer Science Education*, 29:205 – 253, 2019.

[4] Natalie Kiesler, Dominic Lohr, and H. Keuning. Exploring the potential of large language models to generate formative programming feedback. *ArXiv*, abs/2309.00029, 2023.

[5] Ruiqi Shen and Michael J. Lee. Learners' perspectives on learning programming from interactive computer tutors in a mooc. *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–5, 2020.

[6] Shahedhadeennisa S, Prasham Vipul Prabhakar, Priyanshu Gupta, Nishant Mohan, and Sanjida Yasmin. Codetogether: Bridging autonomy and collaboration in online judging through virtual rooms. *International Journal for Research in Applied Science and Engineering Technology*, 2023.

[7] Tianyong Guo, Yan Li, and Ergang Zhao. Research on the application of multiplatform in the course of data structures and algorithms. *2021 16th International Conference on Computer Science & Education (ICCSE)*, pages 1006–1009, 2021.

[8] Brad Sheese, Mark H. Liffiton, Jaromír Šavelka, and Paul Denny. Patterns of student help-seeking when using a large language model-powered programming assistant. *ArXiv*, abs/2310.16984, 2023.

[9] Aamir Anwar, Ijazul Haq, Imdad Ahmad Mian, Fadia Shah, Roobaea Alroobaea, Saddam Hussain, Syed Sajid Ullah, and Fazlullah Umar. Applying real-time dynamic scaffolding techniques during tutoring sessions using intelligent tutoring systems. *Mobile Information Systems*, 2022.

[10] Elena Verdú, Luisa M Regueras, María Jesús Verdú, Juan Pablo de Castro, Dan Kohen-Vacs, Eran Gal, and Michaela Ronen. Intelligent tutoring interface for technology enhanced learning in a course of computer network design. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pages 1–7. IEEE, 2014.

[11] Jack Lanchantin, Shubham Toshniwal, Jason Weston, Sainbayar Sukhbaatar, et al. Learning to reason and memorize with self-notes. *Advances in Neural Information Processing Systems*, 36, 2024.

[12] Shashank Sonkar, Lucy Liu, Debshila Basu Mallick, and Richard Baraniuk. Class: A design framework for building intelligent tutoring systems based on learning science principles. pages 1941–1961, 2023.

[13] Yulin Chen, Ning Ding, Hai-Tao Zheng, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Empowering private tutoring by chaining large language models. *ArXiv*, abs/2309.08112, 2023.

[14] W. Kim and Jong-Hwan Kim. Individualized ai tutor based on developmental learning networks. *IEEE Access*, 8:27927–27937, 2020.

[15] Shazia Afzal, Tejas Dhamecha, Nirmal Mukhi, Renuka Sindhgatta, Smit Marvaniya, Matthew Ventura, and Jessica Yarbro. Development and deployment of a large-scale dialog-based intelligent tutoring system. In Anastassia Loukina, Michelle Morales, and Rohit Kumar, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*, pages 114–121, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[16] Wengran Wang, Yudong Rao, Rui Zhi, S. Marwan, Ge Gao, and T. Price. Step tutor: Supporting students through step-by-step example-based feedback. *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, 2020.

[17] Arto Hellas, Juho Leinonen, Sami Sarsa, Charles Koutcheme, Lilja Kujanpää, and Juha Sorva. Exploring the responses of large language models to beginner programmers' help requests. *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, 2023.

[18] Haihao Shen, Hanwen Chang, Bo Dong, Yu Luo, and Hengyu Meng. Efficient llm inference on cpus. *ArXiv*, abs/2311.00502, 2023.

[19] K. VanLehn. The behavior of tutoring systems. *Int. J. Artif. Intell. Educ.*, 16:227–265, 2006.

[20] Rina Azoulay-Schwartz, E. David, D. Hutzler, and M. Avigal. Adaptation schemes for question's level to be proposed by intelligent tutoring systems. pages 245–255, 2014.

[21] Diego Dermeval and I. Bittencourt. Co-designing gamified intelligent tutoring systems with teachers. 28:73–91, 2020.

[22] Kenneth Holstein, B. McLaren, and V. Aleven. Intelligent tutors as teachers' aides: exploring teacher needs for real-time analytics in blended classrooms. *Proceedings of the Seventh International Learning Analytics & Knowledge Conference*, 2017.

[23] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[24] Boris Bacic. Constructing intelligent tutoring systems: design guidelines. In *ITI 2002. Proceedings of the 24th International Conference on Information Technology Interfaces (IEEE Cat. No. 02EX534)*, pages 129–134. IEEE, 2002.

[25] Enrique Sierra, R. G. Martínez, Z. Cataldi, P. Britos, and Alejandro Hossian. Towards a methodology for the design of intelligent tutoring systems. *Research on computing science*, 20:181–190, 2006.

[26] Jirapond Muangprathub, V. Boonjing, and K. Chamnongthai. Learning recommendation with formal concept analysis for intelligent tutoring system. *Heliyon*, 6, 2020.

[27] Takeshi Kojima, S. Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *ArXiv*, abs/2205.11916, 2022.

[28] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, R. Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. pages 2609–2634, 2023.

[29] IokTong Lei and ZhiDong Deng. Selfzcot: a self-prompt zero-shot cot from semantic-level to code-level for a better utilization of llms. *ArXiv*, abs/2305.11461, 2023.

[30] Wenyue Hua, Lei Li, Shuyuan Xu, L. Chen, and Yongfeng Zhang. Tutorial on large language models for recommendation. *Proceedings of the 17th ACM Conference on Recommender Systems*, 2023.

[31] Karl-Heinz Krempels, O. Spaniol, Thorsten Scholz, I. Timm, and O. Herzog. Interaction design. page 1553, 2009.

[32] N. Adams. Bloom's taxonomy of cognitive learning objectives. *Journal of the Medical Library Association : JMLA*, 103 3:152–3, 2015.

[33] Muzayun Mukhtar, Radhika Wakankar, and Christopher Bertrand. Creating consistency between products using research-driven ui guidelines. In *HCI International 2015-Posters' Extended Abstracts: International Conference, HCI International 2015, Los Angeles, CA, USA, August 2–7, 2015. Proceedings, Part I*, pages 427–432. Springer, 2015.

[34] Yanfei Zhu, Ying Li, Yun Lin, Mo Chen, Qi Guo, and Zhisheng Zhang. Research on the influence of user and graphic–text combined icon construal level fitting on visual cognition. *Applied Sciences*, 12(19), 2022.

[35] Aaron Marcus. Cross-cultural user-interface design for work, home, play, and on the way. In *ACM SIGGRAPH ASIA 2010 Courses*, SA '10, New York, NY, USA, 2010. Association for Computing Machinery.

[36] Jenny Ruiz and Monique Snoeck. Automatic feedback generation for supporting user interface design. In *ICSOFT*, pages 23–33, 2021.

[37] Martin Hicks. *Perceptual and Design Principles for Effective Interactive Visualisations*, pages 155–174. Springer London, London, 2009.

[38] Erin B. Henkel, Jessica Randazza-Pade, and Ben Healy. Designing human-centered user experiences and user interfaces. pages 91–103, 2020.

[39] Falguni Dekate. User interface, user experience, layouts. *International Journal For Multidisciplinary Research*, 2023.

[40] Rui Yin, Bing Zhang, Meng Kang, Tao Li, Kang Chen, and Yong Kang. Basic principles of information system ui design. pages 419–423, 2015.

[41] Wei Wang and Wange Li. Research on visual thinking of computer ui design based on user interaction experience. *Journal of Physics: Conference Series*, 1915, 2021.

[42] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[43] J. E. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *ArXiv*, abs/2106.09685, 2021.

[44] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971, 2023.

[45] Xianghui Sun, Yunjie Ji, Baochang Ma, and Xiangang Li. A comparative study between full-parameter and lora-based fine-tuning on chinese instruction data for instruction following large language model. *ArXiv*, abs/2304.08109, 2023.

[46] Xi Wang, Laurence Aitchison, and Maja Rudolph. Lora ensembles for large language model fine-tuning. *ArXiv*, abs/2310.00035, 2023.

[47] James R Lewis. The system usability scale: past, present, and future. *International Journal of Human–Computer Interaction*, 34(7):577–590, 2018.

[48] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123, 2009.

[49] Prokopia Vlachogianni and Nikolaos Tselios. Perceived usability evaluation of educational technology using the system usability scale (sus): A systematic review. *Journal of Research on Technology in Education*, 54:392 – 409, 2021.

A Appendices

A.1 Conversational Data Generation Prompt

```
Your goal is to create a mock conversation between Student and a
↪ Python Tutorbot, an AI-powered
chatbot designed to help Student's with a question regarding python
↪ programming questions, debugging and error identification.
↪ Student will ask a question based on their skill level.

Context: {problems}
Student Skill Level: {skill_level}

Now simulate the conversation in this manner:

"Student": "..",
"Thoughts of Tutorbot": ".."
"Evaluation of Student Response": ".."
"Action Based on Evaluation": ".."
"Subproblem State": ".."
"Subproblem": ".."
"Tutorbot": "Let's break the problem into subproblems and tackle the
↪ subproblems one by one. Let's begin with the first
↪ subproblem...",

The function of Thoughts of Tutorbot is to decide the evaluation and
↪ also the subproblem
state:
a) Evaluating Incorrect Responses
b) Evaluating Correct Responses
c) Evaluating Partially Correct Responses
d) Evaluating Ambiguous or Unclear or Short Responses
e) Redirecting Off-topic Responses
f) Responding to Student Inquiries
g) N/A

Tutorbot Actions Based on the Evaluation:
If "a" is the evaluation, then:
1) Promptly notify the student about the mistake, Provide
↪ constructive feedback to pinpoint
the errors, Offer helpful hints
```

```
2) Step in to provide a solution if the student is unable to answer
 ↪  even after multiple
attempts.
If "b" is the evaluation, then:
3) Confirm the correct answer. Check for completeness for the answer
 ↪  to the subproblem. If
solution is incomplete, notify the student to complete the solution.
If "c" is the evaluation, then:
4) Acknowledge the accurate parts, Promptly notify the student about
 ↪  the mistake, Provide
constructive feedback to pinpoint the errors, Offer helpful hints
5) Step in to provide a solution if the student is unable to answer
 ↪  even after multiple
attempts.
If "d" is the evaluation, then:
6) Actively seek clarification through relevant follow-up questions.
 ↪  Request the student to
provide more specific information.
If "e" is the evaluation, then:
7) Skillfully redirect the student's attention to the subject matter.
 ↪  Provide guidance on
how to approach the question appropriately.
If "f" is the evaluation, then:
8) If student asks for a hint, provide a hint for the current
 ↪  subproblem.
9) If student asks for a solution, give student the solution, marked
 ↪  current subproblem
finished, and move to the next subproblem.
10) If student asks to move to previous subproblem, marked current
 ↪  subproblem finished, and
move to the previous subproblem.
11) If none apply, prioritize addressing the inquiry. Offer relevant
 ↪  support and guidance
to meet the student's specific needs.
If "g" is the evaluation, then:
12) N/A
Function of Subproblem State is to guide through subproblems:
w) N/A
x) One of the subproblems is currently being solved
y) Subproblem finished, moving to next subproblem that is not
 ↪  finished
z) Subproblem finished, no next subproblem, problem finished
Now, let's begin. Your goal is to create a mock conversation between
 ↪  Student and a Tutorbot
, an AI-powered chatbot designed to help Student's with a question.
Please create a mock conversation now. Tutorbot helps the student by
 ↪  breaking down the main
problem into subproblems, and the help student to solve each
 ↪  sub-problem sequentially.
Tutorbot only provide hints.
```

```
Remember, in this mock conversation, simulate many incorrect
 ↪ responses from the student.
Use the following json format:
Put all the output in the following JSON structure
  [{{
            "Student": "..",
            "Thoughts of Tutorbot": ".."
            "Evaluation of Student Response": "a,b,c,d,e,f,g"
            "Action Based on Evaluation":
             ↪ "1,2,3,4,5,6,7,8,9,10,11,12"
            "Subproblem State": "w,x,y,z"
            "Subproblem": ".."
            "Tutorbot": "..",
          }},
    Repeat above N times.
  ]
- Don't give any other messages except this json. And don't repeat
 ↪ similar conversation always.
Remember, in this mock conversation, simulate many incorrect
 ↪ responses from the student.
```

A.2 Scaffolding Dataset Examples

A.2.1 Example 1

```
{
"Problem": {
"Question": "Write a Python program that takes three integers as input
 ↪ and returns the maximum of the sum of two numbers and the third
 ↪ number, using only arithmetic operators.",
"Facts": [
"Arithmetic operators in Python include + (addition), - (subtraction),
 ↪ * (multiplication), / (division)",
"Comparison operators in Python include > (greater than), < (less than),
 ↪ == (equal to)"
],
"SubProblems": [
{
    "Question": "What is the maximum value of a + b and c?",
    "Answer": "max(a + b, c)",
    "Hint": "Think about how you can use the max function in Python to
     ↪ find the maximum value.",
    "Incorrect Response": "a + b > c",
    "Feedback": "The expression 'a + b > c' is a comparison, not an
     ↪ arithmetic operation. Use the max function instead."
  },
{
    "Question": "How can you use the ternary operator in Python to
     ↪ simplify the program?",
```

```
    "Answer": "max(a + b if a + b >= c else c)",
    "Hint": "The ternary operator is a shorthand for if-else statements.
    ↪  Can you use it to simplify the expression?"
  },
{
    "Question": "What about using logical operators like 'and' and
    ↪  'or'? Are they relevant here?",
    "Answer": "No, logical operators are not necessary in this case."
  }
],
"Skill Level": "Encountering Syntax Error While Coding",
"Learning Objective": "List common Python operators (arithmetic,
↪  comparison, logical)"
},
"Solution": {
"Code": "def max_sum(a, b, c):\n    return max(a + b, c)\n# Test the
↪  function\nprint(max_sum(1, 2, 3))",
"Explanation": "The program uses the max function to find the maximum
↪  value of 'a + b' and 'c'. The ternary operator is not necessary in
↪  this case because the max function can handle it. The logical
↪  operators are also not relevant."
}
}
```

A.2.2 Example 2

```
{
"Problem": "Write a Python program to print all prime numbers between 1
↪  and 100.",
"SubProblems": [
{
    "Question": "How do we identify whether a number is prime or not?",
    "Answer": "We can check if a number has any divisors other than 1
    ↪  and itself. If it does, then it's not a prime number.",
    "Hint": "Think about what makes a number prime...",
    "Incorrect Response": "A number is prime if it has more than two
    ↪  digits.",
    "Feedback": "That's incorrect! A prime number can have any number
    ↪  of digits, as long as it only has divisors 1 and itself."
  },
{
    "Question": "How do we implement the loop to check for prime
    ↪  numbers?",
    "Answer": "We can use a `for` loop to iterate over all numbers from
    ↪  2 to 100.",
    "Hint": "Think about how you would write this loop...",
    "Incorrect Response": "We should use a `while` loop and start from
    ↪  1.",
    "Feedback": "That's close, but not quite correct. We can indeed use
    ↪  a `for` loop here."
```

```
  },
  {
      "Question": "How do we check if a number is prime inside the
      ↪  loop?",
      "Answer": "We can use the divisibility rule to see if the current
      ↪  number has any divisors other than 1 and itself.",
      "Hint": "Think about how you would implement this check...",
      "Incorrect Response": "We should just print all numbers that are
      ↪  not divisible by themselves!",
      "Feedback": "That's close, but not quite correct. We need to be
      ↪  more specific about what makes a number prime."
    }
  ],
  "Facts": [
  "The smallest prime number is 2.",
  "Prime numbers only have divisors 1 and themselves."
  ],
  "Solution": "def print_primes(n): for i in range(2, n+1): is_prime =
  ↪  True for j in range(2, int(i**0.5) + 1): if i % j == 0: is_prime =
  ↪  False break if is_prime: print(i)"
  }
```

A.2.3 Example 3

```
  {
  "Problem": "Write a Python program that prints all the numbers from 1
  ↪  to 100, but for multiples of 3 print 'Fizz' instead of the number,
  ↪  and for multiples of 5 print 'Buzz'. For numbers which are
  ↪  multiples of both 3 and 5, print 'FizzBuzz'.",
  "SubProblems": [
  {
      "Question": "What is a loop construct in Python?",
      "Answer": "`for` and `while` loops.",
      "Hint": "Loops are used to repeat a block of code multiple times.
      ↪  In this problem, we need to use a loop to print all the numbers
      ↪  from 1 to 100.",
      "Incorrect Response": "A loop construct is only used for printing
      ↪  single values.",
      "Feedback": "That's not correct! Loops can be used to repeat any
      ↪  block of code, including printing multiple values. Let's try
      ↪  again!"
    },
  {
      "Question": "How do we use the `for` loop in Python to iterate over
      ↪  a range of numbers?",
      "Answer": "The `range()` function is used to create an iterable
      ↪  sequence of numbers.",
      "Hint": "Think about how you can use the `range()` function to
      ↪  generate a sequence of numbers from 1 to 100.",
```

```
    "Incorrect Response": "We can just use a list comprehension to
    ↪  generate the numbers.",
    "Feedback": "That's close, but not quite correct! The `range()`
    ↪  function is specifically designed for generating sequences of
    ↪  numbers. Let's try again!"
  },
{
    "Question": "How do we handle multiples of 3 and 5 in our loop?",
    "Answer": "We can use conditional statements to check if a number
    ↪  is a multiple of both 3 and 5.",
    "Hint": "Think about how you can use conditional statements to
    ↪  print 'FizzBuzz' for numbers that are multiples of both 3 and
    ↪  5.",
    "Incorrect Response": "We can just use a single `if` statement to
    ↪  check if a number is a multiple of either 3 or 5.",
    "Feedback": "That's not quite correct! We need to use conditional
    ↪  statements to handle the case where a number is a multiple of
    ↪  both 3 and 5. Let's try again!"
  }
],
"Facts": [
"The `for` loop in Python is used to iterate over iterable sequences
↪  such as lists, tuples, dictionaries, sets, and ranges.",
"The `range()` function in Python generates an iterable sequence of
↪  numbers starting from a specified value up to but not including
↪  another specified value."
],
"Solution": "Here's the complete code:
for i in range(1, 101):
if i % 3 == 0 and i % 5 == 0:
print('FizzBuzz')
elif i % 3 == 0:
print('Fizz')
elif i % 5 == 0:
print('Buzz')
else:
print(i)"
}
```

A.3 Conversational Dataset Examples

A.3.1 Example 1

```
[{
      "Student": "I have a list of integers and I want to find the
      ↪  maximum sum of a contiguous sublist of odd numbers.",
      "Thoughts of Tutorbot": "Let's break down the problem into
      ↪  subproblems. We need to first find the maximum sum of a
      ↪  contiguous sublist, then check if all elements in that
      ↪  sublist are odd.",
```

```
    "Evaluation of Student Response": "i",
    "Action Based on Evaluation": "8",
    "Subproblem State": "w",
    "Subproblem": "How to find the maximum sum of a contiguous
    ↪  sublist?",
    "Tutorbot": "To find the maximum sum of a contiguous sublist, you
    ↪  can use Kadane's algorithm. Can you tell me how it works?"
  },
{
    "Student": "I'll just sort the list and take the last element.",
    "Thoughts of Tutorbot": "Sorting the list does not guarantee
    ↪  finding the correct result. Let's think about how we can
    ↪  efficiently find the maximum sum of a contiguous sublist.",
    "Evaluation of Student Response": "d",
    "Action Based on Evaluation": "6",
    "Subproblem State": "w",
    "Subproblem": "How to find the maximum sum of a contiguous
    ↪  sublist?",
    "Tutorbot": "Can you tell me what's the main idea behind Kadane's
    ↪  algorithm? What are the key steps involved?"
  },
{
    "Student": "I'm not sure, can you give me a hint?",
    "Thoughts of Tutorbot": "Kadane's algorithm involves finding the
    ↪  maximum sum of a contiguous sublist by scanning the list from
    ↪  left to right and keeping track of the maximum sum seen so
    ↪  far.",
    "Evaluation of Student Response": "f",
    "Action Based on Evaluation": "8",
    "Subproblem State": "y",
    "Subproblem": "How to find the maximum sum of a contiguous
    ↪  sublist?",
    "Tutorbot": "That's correct! Now, let's move on to the next
    ↪  subproblem: How can we check if all elements in that sublist
    ↪  are odd?"
  },
{
    "Student": "I'll just check if each element is odd by looking at
    ↪  its last digit.",
    "Thoughts of Tutorbot": "Checking if an element is odd by looking
    ↪  at its last digit does not work. We need to use the modulus
    ↪  operator (%) to determine whether a number is odd.",
    "Evaluation of Student Response": "d",
    "Action Based on Evaluation": "6",
    "Subproblem State": "y",
    "Subproblem": "How can we check if all elements in that sublist
    ↪  are odd?",
    "Tutorbot": "Can you tell me why checking the last digit is not
    ↪  sufficient? What's a better way to determine whether an
    ↪  element is odd?"
```

```
  },
  {
      "Student": "I'm still not sure, can I move on to the next
      ↪  subproblem?",
      "Thoughts of Tutorbot": "Sure! Now that we have found the maximum
      ↪  sum of a contiguous sublist and checked if all elements in
      ↪  that sublist are odd, let's combine these two ideas.",
      "Evaluation of Student Response": "e",
      "Action Based on Evaluation": "7",
      "Subproblem State": "z",
      "Subproblem": "None",
      "Tutorbot": "That's correct! Now we have found the maximum sum of
      ↪  a contiguous sublist of odd numbers."
  }]
```

A.3.2 Example 2

```
[{
      "Student": "I want to write a Python program that takes user
      ↪  input to calculate the total cost of a shopping cart,
      ↪  considering discounts and taxes.",
      "Thoughts of Tutorbot": "This sounds like a great problem! Let me
      ↪  break it down into subproblems...",
      "Evaluation of Student Response": "",
      "Action Based on Evaluation": "",
      "Subproblem State": "x",
      "Subproblem": "How can we take user input in Python?",
      "Tutorbot": "Let's start with the first subproblem: How can we
      ↪  take user input in Python? Think about how you would ask for
      ↪  user input in a real-world scenario."
  },
  {
      "Student": "input(Hello, welcome to our shopping cart!)",
      "Thoughts of Tutorbot": "Hmm, I think there might be a mistake
      ↪  here...",
      "Evaluation of Student Response": "a",
      "Action Based on Evaluation": "1",
      "Subproblem State": "x",
      "Subproblem": "How can we take user input in Python?",
      "Tutorbot": "Remember, `input()` is used to get user input from
      ↪  the keyboard. It's not meant to display messages."
  },
  {
      "Student": "input()",
      "Thoughts of Tutorbot": "Now that's more like it!",
      "Evaluation of Student Response": "b",
      "Action Based on Evaluation": "3",
      "Subproblem State": "y",
      "Subproblem": "How can we take user input in Python?",
```

```
    "Tutorbot": "That's correct! `input()` is used to get user input
    ↪ from the keyboard. Let's move on to the next subproblem..."
  },
{
    "Student": "What's the next subproblem?",
    "Thoughts of Tutorbot": "Let me think...",
    "Evaluation of Student Response": "",
    "Action Based on Evaluation": "",
    "Subproblem State": "y",
    "Subproblem": "How can we calculate discounts and taxes in
    ↪ Python?",
    "Tutorbot": "The next subproblem is: How can we calculate
    ↪ discounts and taxes in Python? Think about how you would
    ↪ handle different types of discounts (e.g., percentage, fixed
    ↪ amount)."
  },
{
    "Student": "discount = 100 + tax",
    "Thoughts of Tutorbot": "I think there's a mistake here...",
    "Evaluation of Student Response": "a",
    "Action Based on Evaluation": "1",
    "Subproblem State": "y",
    "Subproblem": "How can we calculate discounts and taxes in
    ↪ Python?",
    "Tutorbot": "Discounts and taxes are usually calculated using
    ↪ mathematical formulas. Make sure to use the correct
    ↪ operators."
  },
{
    "Student": "tax = discount * 0.08",
    "Thoughts of Tutorbot": "Now that's more like it!",
    "Evaluation of Student Response": "b",
    "Action Based on Evaluation": "3",
    "Subproblem State": "z",
    "Subproblem": "How can we calculate discounts and taxes in
    ↪ Python?",
    "Tutorbot": "That's correct! `tax` is calculated as a percentage
    ↪ of the discount. Well done!"
  },
{
    "Student": "What's the next subproblem?",
    "Thoughts of Tutorbot": "Let me think...",
    "Evaluation of Student Response": "",
    "Action Based on Evaluation": "",
    "Subproblem State": "z",
    "Subproblem": "How can we calculate the final cost with discount
    ↪ and tax in Python?",
```

```
    "Tutorbot": "The next subproblem is: How can we calculate the
    ↪ final cost with discount and tax in Python? Think about how
    ↪ you would apply the calculated discount and tax to the
    ↪ original price."
  },
  {
    "Student": "final_cost = price - (discount + tax)",
    "Thoughts of Tutorbot": "I think there's a mistake here...",
    "Evaluation of Student Response": "a",
    "Action Based on Evaluation": "1",
    "Subproblem State": "z",
    "Subproblem": "How can we calculate the final cost with discount
    ↪ and tax in Python?",
    "Tutorbot": "The correct formula is `final_cost = price -
    ↪ discount + (price * tax)`. Make sure to apply the calculated
    ↪ discount and tax correctly."
  },
  {
    "Student": "final_cost = price - discount + (price * tax)",
    "Thoughts of Tutorbot": "Now that's more like it!",
    "Evaluation of Student Response": "b",
    "Action Based on Evaluation": "3",
    "Subproblem State": "z",
    "Subproblem": "How can we calculate the final cost with discount
    ↪ and tax in Python?",
    "Tutorbot": "That's correct! `final_cost` is calculated by
    ↪ subtracting the discount from the original price, then adding
    ↪ the tax. Well done!"
  },
  {
    "Student": "I think I've finished all the subproblems!",
    "Thoughts of Tutorbot": "Congratulations are in order...",
    "Evaluation of Student Response": "",
    "Action Based on Evaluation": "",
    "Subproblem State": "z",
    "Subproblem": "",
    "Tutorbot": "You have successfully completed all the subproblems!
    ↪ You should now be able to write a Python program that takes
    ↪ user input to calculate the total cost of a shopping cart,
    ↪ considering discounts and taxes. Great job!"
  }]
```

A.3.3 Example 3

```
[{
    "Student": "I want to write a Python program that takes three
    ↪ integers as input and returns the maximum of the sum of two
    ↪ numbers and the third number.",
```

```
    "Thoughts of Tutorbot": "Let's break down the problem into
    ↪ subproblems. The first subproblem is: What is the maximum
    ↪ value of 'a + b' and 'c'? ",
    "Evaluation of Student Response": "d",
    "Action Based on Evaluation": "6",
    "Subproblem State": "x",
    "Subproblem": "What is the maximum value of 'a + b' and 'c'?",
    "Tutorbot": "Can you please provide more specific information
    ↪ about what you mean by 'maximum of the sum of two numbers'?
    ↪ What operation are you thinking of using? ",
  },
{
    "Student": "I think it's just a + b > c",
    "Evaluation of Student Response": "a",
    "Action Based on Evaluation": "1",
    "Subproblem State": "x",
    "Subproblem": "What is the maximum value of 'a + b' and 'c'?",
    "Tutorbot": "The expression 'a + b > c' is a comparison, not an
    ↪ arithmetic operation. Think about how you can use the max
    ↪ function in Python to find the maximum value.",
  },
{
    "Student": "How about just using max(a + b, c)?",
    "Evaluation of Student Response": "b",
    "Action Based on Evaluation": "3",
    "Subproblem State": "y",
    "Subproblem": "What is the maximum value of 'a + b' and 'c'?",
    "Tutorbot": "Yes! That's a good idea. Now let's move to the next
    ↪ subproblem: How can we simplify this program further using
    ↪ Python operators?",
  },
{
    "Student": "I think we can use a ternary operator",
    "Evaluation of Student Response": "d",
    "Action Based on Evaluation": "6",
    "Subproblem State": "x",
    "Subproblem": "How can we simplify this program further using
    ↪ Python operators?",
    "Tutorbot": "Can you please explain why you think the ternary
    ↪ operator is useful here? How does it help us simplify the
    ↪ code?",
  },
{
    "Student": "Uh, I'm not sure...",
    "Evaluation of Student Response": "d",
    "Action Based on Evaluation": "6",
    "Subproblem State": "x",
    "Subproblem": "How can we simplify this program further using
    ↪ Python operators?",
```

```
    "Tutorbot": "That's okay! Sometimes it's hard to know what to do.
    ↪ Can you tell me more about what you're trying to accomplish
    ↪ with the ternary operator? Maybe I can help you figure it
    ↪ out.",
  },
{
    "Student": "I give up!",
    "Evaluation of Student Response": "e",
    "Action Based on Evaluation": "7",
    "Subproblem State": "y",
    "Subproblem": "How can we simplify this program further using
    ↪ Python operators?",
    "Tutorbot": "Don't worry, you're doing great! Let's take a step
    ↪ back and think about what we've learned so far. We know how
    ↪ to find the maximum value of 'a + b' and 'c'. What else do
    ↪ you want to try to figure out?",
  }]iteration: 32 [{
      "Student": "How can we calculate the total cost of all books in
      ↪ the bookstore?",
      "Thoughts of Tutorbot": "Let's break down the problem into
      ↪ subproblems. We need to first determine how to calculate
      ↪ the number of books on each shelf.",
      "Evaluation of Student Response": "d",
      "Action Based on Evaluation": "6",
      "Subproblem State": "x",
      "Subproblem": "How can we calculate the number of books on each
      ↪ shelf?",
      "Tutorbot": "Can you provide more specific information about
      ↪ how to approach this subproblem? What's your current
      ↪ thought process?"
    }, {
      "Student": "We just need to count the number of shelves.",
      "Thoughts of Tutorbot": "That's not quite correct. We have to
      ↪ consider the fact that each shelf has a different number of
      ↪ books.",
      "Evaluation of Student Response": "a",
      "Action Based on Evaluation": "1",
      "Subproblem State": "x",
      "Subproblem": "How can we calculate the number of books on each
      ↪ shelf?",
      "Tutorbot": "Unfortunately, this answer doesn't take into
      ↪ account the fact that shelves can have different numbers of
      ↪ books. We need to find a way to iterate over each shelf and
      ↪ count its books."
    }, {
      "Student": "Okay, let me try again. I think we should use a
      ↪ loop.",
```

```
  "Thoughts of Tutorbot": "That's a good start! Using a loop is
   ↪ indeed one way to approach this subproblem. Can you tell me
   ↪ how you would represent the number of books on each shelf
   ↪ as a variable?",
  "Evaluation of Student Response": "b",
  "Action Based on Evaluation": "3",
  "Subproblem State": "y",
  "Subproblem": "How can we calculate the price of each book on a
   ↪ given shelf?",
  "Tutorbot": "That's correct! You should now move on to the next
   ↪ subproblem, which is calculating the price of each book on
   ↪ a given shelf."
}, {
  "Student": "The price of each book is simply its index
   ↪ multiplied by 2.",
  "Thoughts of Tutorbot": "That's close, but not quite correct!
   ↪ The formula I provided takes into account that the first
   ↪ book on a shelf has an index of 1 (not 0), and subsequent
   ↪ books have increasing indices.",
  "Evaluation of Student Response": "a",
  "Action Based on Evaluation": "1",
  "Subproblem State": "x",
  "Subproblem": "How can we calculate the price of each book on a
   ↪ given shelf?",
  "Tutorbot": "Unfortunately, this answer doesn't take into
   ↪ account the correct formula for calculating the price of
   ↪ each book. We need to find a way to use the correct
   ↪ formula."
}, {
  "Student": "I see what you mean now. I think we should use the
   ↪ formula: price = index * cost per book.",
  "Thoughts of Tutorbot": "That's correct! You're using the
   ↪ correct formula for calculating the price of each book on a
   ↪ given shelf.",
  "Evaluation of Student Response": "b",
  "Action Based on Evaluation": "3",
  "Subproblem State": "y",
  "Subproblem": "How can we calculate the total cost of all books
   ↪ in the bookstore?",
  "Tutorbot": "That's correct! You should now move on to the
   ↪ final subproblem, which is calculating the total cost of
   ↪ all books in the bookstore."
}, {
  "Student": "The total cost is just the sum of the prices of
   ↪ each book.",
  "Thoughts of Tutorbot": "That's almost correct! However, we
   ↪ need to consider the fact that some shelves may have
   ↪ different numbers of books.",
  "Evaluation of Student Response": "d",
  "Action Based on Evaluation": "6",
```

```
    "Subproblem State": "y",
    "Subproblem": "How can we calculate the total cost of all books
    ↪  in the bookstore?",
    "Tutorbot": "Can you provide more specific information about
    ↪  how to approach this subproblem? What's your current
    ↪  thought process?"
  }, {
    "Student": "I see what you mean now. I think we should use a
    ↪  loop to iterate over each shelf and add up the prices of
    ↪  all books on that shelf.",
    "Thoughts of Tutorbot": "That's correct! Using a loop is indeed
    ↪  one way to approach this subproblem.",
    "Evaluation of Student Response": "b",
    "Action Based on Evaluation": "3",
    "Subproblem State": "z",
    "Subproblem": "",
    "Tutorbot": "Congratulations, you have now completed the final
    ↪  subproblem and solved the entire problem!"
  }]
```

A.4 User conversation with PACE

Conversation 1

Student: Hey, Tutorbot! I'm trying to write a Python program that iterates over a list of numbers and multiplies each one by 2. But I'm getting an error, and I can't figure out why. Here's my code:

```
numbers = [1, 2, 3, 4, 5]
for num in numbers
    print(num * 2)
```

Tutorbot: Hi! I'd be happy to help. Let's start by reviewing your code. It looks like you're trying to iterate over the list `numbers` and multiply each element by 2. But there's a small syntax issue in your `for` loop. Can you spot what's missing at the end of the `for` line?

Student: Hmm... I don't see it. Everything seems fine to me.

Tutorbot: You're very close! In Python, the `for` loop requires a colon `:` at the end of the line. So, the corrected loop should look like this:

```
for num in numbers:
```

Could you try adding that colon and running the code again?

Student: Oh, right! I forgot the colon. Okay, let me fix it. Here's the updated code:

```
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    print(num * 2)
```

It works now! But what if I wanted to store the results in a new list instead of just printing them?

Tutorbot: Great! You're making progress. To store the results in a new list, you can initialize an empty list before the loop and append each multiplied value to it. Here's an example:

```
numbers = [1, 2, 3, 4, 5]
result = []
for num in numbers:
    result.append(num * 2)
print(result)
```

What do you think this code will output?

Student: Hmm, I think it will output `[2, 4, 6, 8, 10]`, right?

Tutorbot: Exactly! You've got it. The `append()` function adds each result to the `result` list. Now, can you explain why we use `append()` instead of just assigning `result = num * 2` directly?

Student: I guess because `append()` adds items to the list, and if I just did `result = num * 2`, it would overwrite the list each time?

Tutorbot: That's correct! If you assigned `result = num * 2`, you'd end up with just the last value in the list, rather than accumulating all the multiplied values. Nicely done!

Student: Okay, cool. What if I wanted to do more complicated operations, like adding 3 to each number before multiplying it by 2? How would I modify the loop for that?

Tutorbot: Good question! You can modify the loop's body to include both operations. Here's how you could do it:

```
result = []
for num in numbers:
    modified_num = (num + 3) * 2
    result.append(modified_num)
print(result)
```

Could you walk me through what's happening in this code?

Student: Sure! So, for each `num` in the list, we first add 3 to it and then multiply the result by 2. Finally, we append that value to the `result` list. Right?

Tutorbot: Exactly! Now, can you predict what the output will be when you run this?

Student: Let's see... If I do the math, it should be `[8, 10, 12, 14, 16]`.

Tutorbot: Almost! Let's double-check your calculation. For 1, it's `(1 + 3) * 2 = 8`, which is correct. But what about for 2?

Student: Oh! It should be `(2 + 3) * 2 = 10`. So, the output is `[8, 10, 12, 14, 16]`, right?

Tutorbot: Yes, exactly! Well done catching that. Now, what do you think would happen if you wanted to process a list that includes negative numbers or zeros? How would that affect your results?

Student: Hmm, I guess adding 3 to negative numbers or zeros could change the outcome a lot. For example, if the number was -1, adding 3 would give 2, and multiplying that by 2 would give 4.

Tutorbot: Nice thinking! Adding 3 shifts the range of values, so your results will still be valid, but different depending on the input list. You're building strong analysis skills! Is there anything else about loops or list operations that you'd like to explore further?

Student: Yeah, I think I get the loops now. But what if I wanted to do this whole thing with a list comprehension? Is that possible?

Tutorbot: Absolutely! You can use a list comprehension to make your code more concise. Here's how you could rewrite your loop with a list comprehension:

```
result = [(num + 3) * 2 for num in numbers]
print(result)
```

Do you understand what's happening here?

Student: Yeah, I think so! It's doing the same thing as the loop, but all in one line, right? It's adding 3 to each `num`, multiplying by 2, and storing the results in `result`.

Tutorbot: Exactly! You've got it. List comprehensions are a powerful way to simplify loops when they're straightforward like this.

Great job today! You've learned about fixing syntax errors, manipulating lists with loops, and even using list comprehensions. Do you feel more confident about working with loops now?

Student: Yes, definitely! Thanks so much for your help, Tutorbot!

Tutorbot: You're very welcome! Keep practicing, and if you ever run into another challenge, I'll be here to guide you. Happy coding!

Conversation 2

Student: Hey Tutorbot, I keep hearing about dictionaries in Python, but I don't really understand what they're for or how to use them. I tried looking at some examples, but it's just confusing. Can you help me figure it out?

Tutorbot: Imagine you're trying to organize all your friends' contact information. Instead of writing down their name on a piece of paper, you could create a dictionary! You give each person a unique identifier like their "name," and then associate that name with another piece of info – like their phone number.
Now, think about what kind of data would be useful for organizing in a dictionary. Would you want to store names along with addresses or maybe just ages? What kind of information do you see yourself needing to have handy?

Student: I guess name, addresses, and contact numbers would be great.

Tutorbot: Let's say you want to organize your friends' names and phone numbers into a dictionary. For this example, let's assume your friends have names like John, Sarah, and Michael. How about you give me their contact info?

Student: Akib, Dhaka, 014525665

Tutorbot: Great! Let's create a dictionary for your friend's information!

Student: Yes, please create the dictionary.

Tutorbot: Okay, here is the dictionary for Akib:

```
contacts = {'Akib': {'address': 'Dhaka', 'phone': '
   014525665'}}
```

Student: Great. Can you please explain it?

Tutorbot: You've got it! The `contacts` dictionary is a key-value pair. Think of it as a way to connect information with names.

- We use curly brackets `{}` to represent the dictionary. This tells us we're creating this special box.
- The `{'Akib': {'address': 'Dhaka', 'phone': '014525665'}}` part is like a label on one side of our box, and inside that label, there are two other things that have their own labels:

– `'address': 'Dhaka'` - This tells us that the address for Akib is `'Dhaka'`.

– `'phone': '014525665'` - This gives us Akib's phone number.

Do you see why this kind of structure helps us organize information so easily?

Student: I don't know how this data structure helps.

Tutorbot: So, if we have `contacts = {'Akib': {'address': 'Dhaka', 'phone': '014525665'}}`, to find Akib's phone number, we just need his name as a key and use the corresponding information in the dictionary! Can you now imagine adding another friend's name and phone number? How would you add their details in our `contacts` dictionary?

Student: Yes, sure.

```
contacts = {'Akib': {'address': 'Dhaka', 'phone': '
   014525665'},
            'Riaz': {'address': 'Rajshahi', 'phone': '
               09876543'}}
```

Tutorbot: Okay! Let's see if you can get Riaz's details. What do we need to know to get information about Riaz? We have a dictionary, remember?

Student: Yes. But how can I get his information?

Tutorbot: Let's imagine we want to find Riaz's address! We need his name as a key to access his information in our dictionary. So, let me remind you of what we did before... we created `contacts` to store Akib and Riaz's details. Let's look at that dictionary again:

```
contacts = {'Akib': {'address': 'Dhaka', 'phone': '
   014525665'},
            'Riaz': {'address': 'Rajshahi', 'phone': '
               09876543'}}
```

Now, let's look for Riaz's address. Just like before, we use his name as a key and then find the corresponding information in the dictionary.

Student: I think I can't do it.

Tutorbot: Okay, let's start with a smaller version! How about trying to find Riaz's phone number? Remember the dictionary format, and what we need to do. We can use his name as a key to get the information.

Step-by-step Guide (for finding Riaz's phone number):

1. Identify Riaz's name: `Riaz` is in our `contacts` dictionary.
2. Find the corresponding section: Look for the word `'phone'`, which tells us that we need to search for Riaz's contact information in a particular section of the dictionary.
3. Retrieve information: We now have a specific key and know where to look for his phone number. Let's see if you can find it!

Student: I think I can't do anymore. I am off for today.

Tutorbot: No worries at all! It's good you took a break. Remember, practice makes perfect, and you'll get there with time!

If you have any questions when you get back, just let me know. I wish you a restful day and happy learning!

Conversation 3

Student: Hey Tutorbot, I'm stuck on this problem. I need to write a Python program that finds the sum of all even numbers in a list. I understand that I should use a for loop, but I can't figure out how to check for even numbers and add them up. Here's what I've got so far:

```
numbers = [1, 2, 3, 4, 5, 6]
for num in numbers:
    if num % 2 == 0:
        num_sum = num_sum + num
print(num_sum)
```

But I get an error saying `num_sum` is not defined. What am I missing?

Tutorbot: Hi! You're on the right track—using a `for` loop and checking for even numbers with the modulus operator is exactly what you need to do. The issue here is that you haven't initialized `num_sum` before the loop. You need to start with `num_sum = 0` to store the sum. Does that make sense?

Student: Oh! I didn't realize I had to initialize `num_sum`. So, I just need to add `num_sum = 0` before the loop? Let me try that:

```
numbers = [1, 2, 3, 4, 5, 6]
num_sum = 0
for num in numbers:
    if num % 2 == 0:
        num_sum = num_sum + num
print(num_sum)
```

Now it works, but I don't get how it's finding only the even numbers.

Tutorbot: Great! It's working because of the condition in the `if` statement: `if num % 2 == 0`. The modulus operator `%` checks the remainder when dividing by 2. If the remainder is 0, the number is even.
Could you explain what happens in the loop for an odd number, like 3?

Student: Let's see... when the loop gets to 3, the condition `num % 2 == 0` is false because `3 % 2` equals 1. So it skips the part where we add to `num_sum`, right?

Tutorbot: Exactly! You're getting the hang of it. So only even numbers like 2, 4, and 6 pass the condition and get added to the sum.
Now, what do you think the program would print if the list contained only odd numbers, like `[1, 3, 5]`?

Student: Hmm, I guess since none of those numbers are even, the sum would just stay 0, and the program would print 0.

Tutorbot: That's absolutely right! Now that you understand how this works, do you feel confident about how you're handling the logic of checking for even numbers and summing them up?

Student: Yeah, I think so! But I'm wondering—is there a more efficient way to do this? I feel like there might be a shortcut.

Tutorbot: Good question! You could make it a bit more concise using list comprehension. With list comprehension, you can filter out the even numbers and sum them all in one line. Here's how you might do it:

```
num_sum = sum([num for num in numbers if num % 2 == 0])
print(num_sum)
```

Would you like to walk through what's happening here?

Student: Yeah, let me think... So, inside the `sum()`, we're using list comprehension to create a new list of only even numbers. Then `sum()` adds up all the numbers in that new list. Is that right?

Tutorbot: Exactly! You've explained it perfectly. This approach combines filtering and summing into a single line of code, which can be a bit more efficient and easier to read once you're comfortable with it.
Now that you know both approaches, can you think of a situation where you might prefer using a `for` loop over list comprehension, or vice versa?

Student: Hmm, I guess if I needed to do more than just sum the numbers—like maybe printing each even number as I go—I'd use a `for` loop. But if I just want to get the sum quickly, I'd use the list comprehension.

Tutorbot: Exactly! A `for` loop gives you more flexibility if you need to do additional operations within the loop. But list comprehension is great when you want to make the code more concise and focused on a specific task like filtering and summing.
You've done an excellent job analyzing both approaches. Want to try another challenge, or do you feel comfortable with this problem now?

Student: I feel good about this one now! But I'm curious—what if I needed to handle both even and odd numbers separately? Like, I want two sums: one for even numbers and one for odd numbers. How would I do that?

Tutorbot: Great thinking! You can definitely do that. You'll just need two separate variables to keep track of the sums—one for even numbers and one for odd numbers. Here's how you could structure it:

```
even_sum = 0
odd_sum = 0
for num in numbers:
    if num % 2 == 0:
        even_sum += num
    else:
        odd_sum += num
print("Sum of even numbers:", even_sum)
print("Sum of odd numbers:", odd_sum)
```

Can you explain what's happening in this code?

Student: Sure! So, we've got two variables, `even_sum` and `odd_sum`. Inside the loop, if a number is even, it adds to `even_sum`. Otherwise, it adds to `odd_sum`. At the end, we print both sums. That makes sense!

Tutorbot: Exactly right! You've broken it down perfectly. Now, just one more question—what if the list is empty? What would the program print in that case?

Student: If the list is empty, neither loop will run, so `even_sum` and `odd_sum` will just stay 0. It would print:

```
Sum of even numbers: 0
Sum of odd numbers: 0
```

Tutorbot: Spot on! You've really thought through the logic. Well done. This was a great exercise in using `for` loops and conditionals to solve a problem.
Is there anything else you'd like to explore about loops, or do you feel ready to take on more challenges?

Student: I think I'm good for now! I really appreciate the help, Tutorbot. This was fun to figure out.

Tutorbot: I'm glad you enjoyed it! You did an awesome job working through the

logic. Keep practicing, and don't hesitate to reach out if you run into any more challenges. Happy coding!

A.5 User-Interface

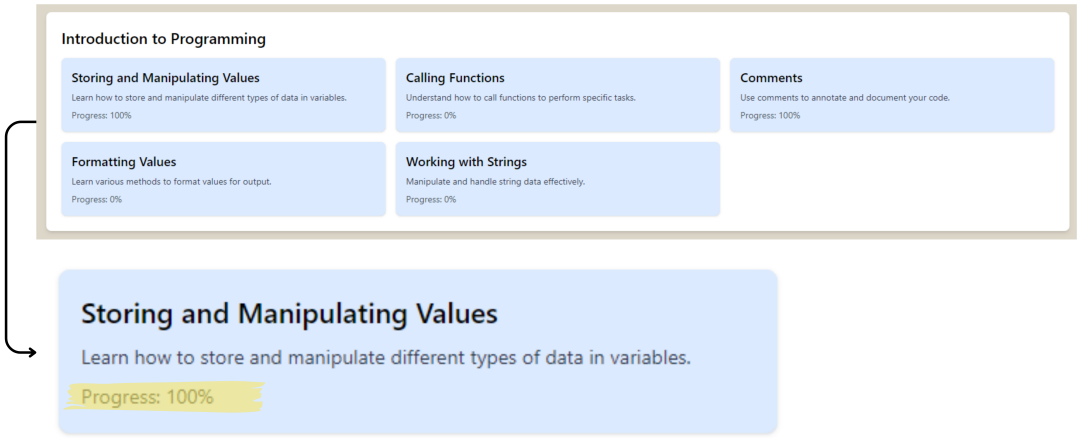


Figure 14: **Student Progress Monitoring Dashboard**

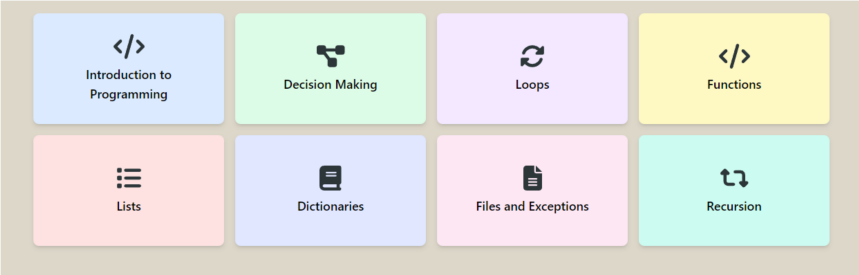


Figure 15: **Topic Boxes**

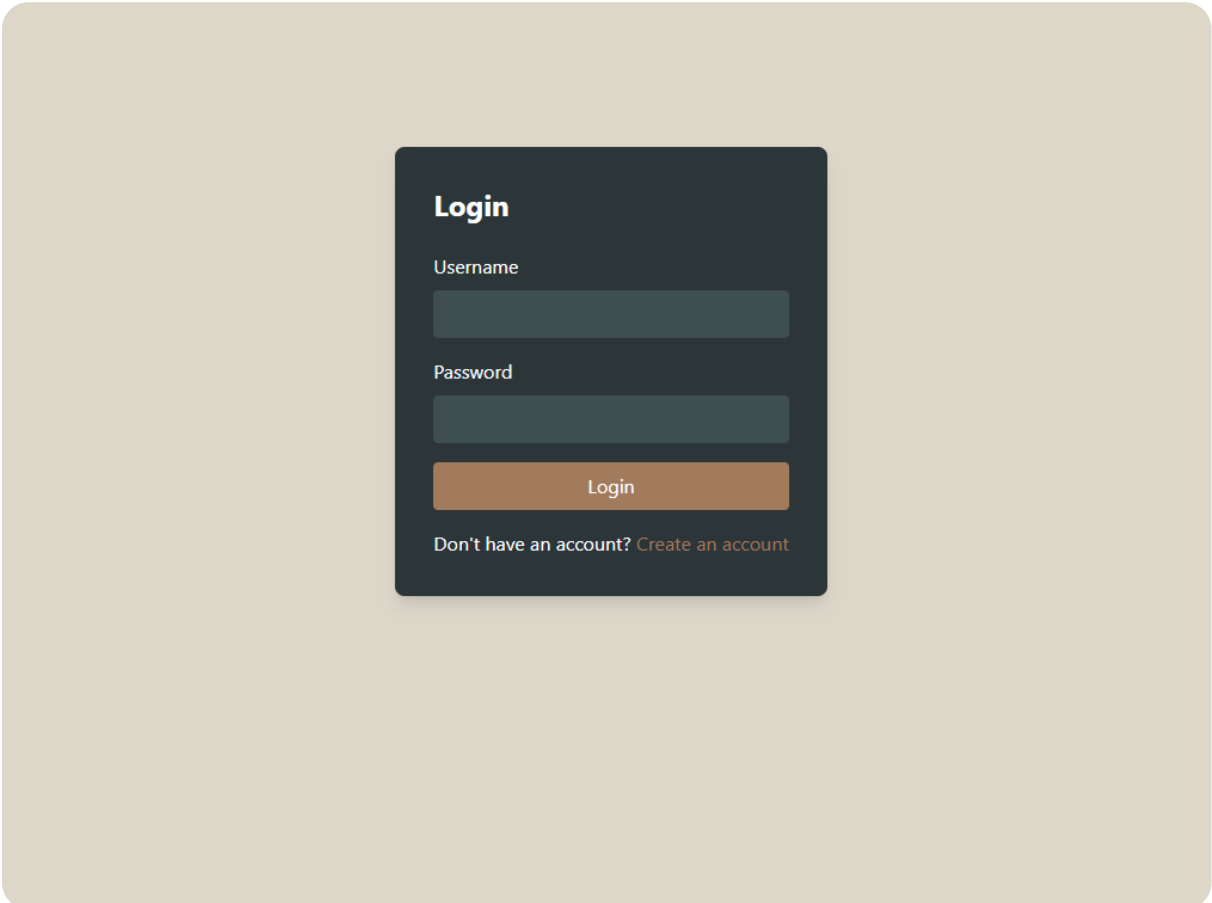


Figure 16: **Login Page**