applied sciences

Article

GPT-Empowered Personalized eLearning System for Programming Languages

Jennifer Jin [1] and Mira Kim [2,*]

[1] School of Computer Science and Engineering, California State University, San Bernardino, CA 92407, USA; jennifer.jin@csusb.edu
[2] Department of Computer Science, California State University, Fullerton, CA 92831, USA
* Correspondence: mira.kim@fullerton.edu; Tel.: +1-657-278-7778

Abstract: The eLearning approach to programming language instruction has gained widespread acceptance due to advantages such as accessibility, temporal flexibility, and content reusability. However, the current eLearning for programming predominantly employs the delivery of one-size-fits-all content, engendering elevated costs in both the development of language coursework and administration of eLearning sessions, which includes the labor-intensive task of grading student submissions. A compelling research question to consider is how to construct an eLearning system capable of delivering personalized, student-centric content, automating the generation of coursework elements, and eliminating the need for instructor involvement in the management of eLearning sessions. Our approach to delivering a definite solution to the question involves the utilization of a suite of advanced software technologies: GPT to dynamically generate course contents/components, prompt engineering to personalize course content for each individual student, and autonomous computing to manage eLearning sessions without the need for human intervention. The research results encompass the design of an eLearning framework covering all programming languages, a fully functional Python-based implementation, seamless integration with ChatGPT for dynamic content generation, a high degree of content personalization, and the elimination of manual effort required for managing eLearning sessions.

Keywords: eLearning; programming languages; personalized learning; ChatGPT; near-zero effort

Citation: Jin, J.; Kim, M. GPT-Empowered Personalized eLearning System for Programming Languages. *Appl. Sci.* **2023**, *13*, 12773. https://doi.org/10.3390/app132312773

Academic Editors: Mourad Oussalah, Zoi Nikiforidou and Jenny Pange

Received: 12 October 2023
Revised: 25 November 2023
Accepted: 25 November 2023
Published: 28 November 2023

1. Introduction

Learning to write programs using languages such as C, C++, Java, Python, and C# has become increasingly important for students at various levels and for developers in information technology. The importance of learning programming in the U.S. and other advanced countries has risen notably over the years, influenced by educational changes and societal shifts that acknowledge the significance and versatility of coding abilities. For example, the Ministry of Education in South Korea announced the Digital Talent Cultivation Plan in August 2022, which mandates coding education in elementary schools [1].

eLearning refers to the use of digital resources to facilitate teaching and learning, often delivered or facilitated via the Internet. It offers a self-paced, customizable approach to learning, often accessible anytime, anywhere. It has been a common teaching approach in higher education, and the recent pandemic notably accelerated the adoption of eLearning. While eLearning for programming languages offers the benefits of accessibility, time flexibility, and content reusability, the effort involved in creating and maintaining the content is significant. This includes preparing the fundamentals of language constructs, providing examples of using the constructs, designing coding exercise problems, and assessing coding exercises submitted by students.

Building an eLearning system to teach various programming languages poses significant challenges due to the following facts:

- There are numerous programming languages that need to be taught.
- Each language has more than 10 distinct units, as appeared as chapters in books, to teach elements such as variables, data types, operators, functions, control structures, looping, class, and inheritance.
- Each unit of a language has about three to five topics to teach, such as for loop, while loop, and repeat-until loop for the topic of 'looping'.
- Each topic of a unit has four types of learning activities: teaching the foundations of the topic, illustrative examples, exercise problems, and grading the exercise problems by instructors. The foundational material includes coding constructs, their usage, syntax, semantics, and guidelines. The task of generating coding exercise problems can be daunting and the process of evaluating students' exercise submissions demands considerable effort.

Given these challenges, creating an eLearning system for programming languages necessitates a substantial investment in both time and resources, underlining its technical intricacy.

On the other hand, a potent strategy for enhancing learning is to personalize the learning contents to each student's individual needs and characteristics. Personalized learning offers numerous benefits compared to a one-size-fits-all conventional approach. These benefits are especially pronounced when learning programming languages. Unlike merely memorizing language constructs, programming requires creativity in the exercise of coding. Personalized coding exercises for each student provide high effectiveness in learning language constructs and promote applying creativity in coding. Therefore, personalizing learning materials, including examples and exercise problems, surely optimizes the learning efficiency of each individual student.

The research questions we formulated include the following:

- How to construct an eLearning system that can cover all the major programming languages, not just a specific one?
- How to deliver learning content that is effectively personalized to each student?
- How to automate the generation of personalized foundational content on the fly?
- How to automate the generation of coding exercise problems on the fly?
- How to automate both quantitative and qualitative assessments of student submissions in coding exercises?
- How to automate the management of learning progress for students?
- How to achieve complete automation in the creation of eLearning content and classroom management, thereby significantly eliminating the need for instructor involvement?

Our research objective was to design and implement an eLearning system that can effectively teach a range of representative programming languages, generate learning contents and coding exercise problems, personalizing these contents for each student, and automate the evaluation of submitted coding exercises while significantly *minimizing* the initial development effort and cost.

We achieve the goal by seamlessly integrating three modern technologies: (1) harnessing the Generative Pre-trained Transformer (GPT) model, (2) *personalizing* learning contents for each student's characteristics and performance, and (3) *autonomously* conducting end-to-end course management using software agents.

The system configuration is shown in Figure 1.

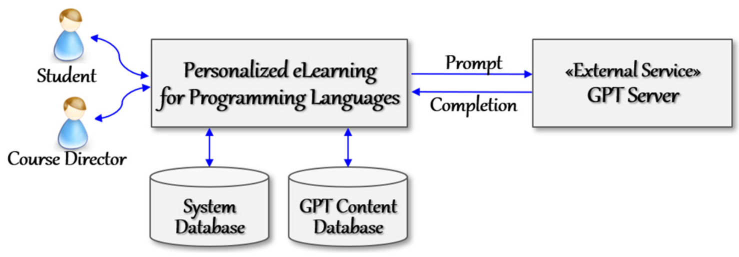

Student
Course Director
Personalized eLearning for Programming Languages
Prompt
Completion
«External Service» GPT Server
System Database
GPT Content Database

Figure 1. Configuration of the personalized eLearning system for languages.

The target system performs all the system functions of eLearning except the generation of learning content and evaluation of exercise submissions. Note that the GPT server does not maintain application-specific sessions; rather, its capability is limited to generating the contents requested through GPT prompts. Hence, the eLearning system itself has to provide application-specific functionality and manage the system database.

This paper is organized as follows: Section 2 summarizes related works. Section 3 presents the system requirements. Section 4 presents the system architecture, design of key components, algorithms, and design tactics. Section 5 presents a proof-of-concept implementation of the system and assessments of the experimental results.

2. Related Works

This section summarizes representative works on eLearning systems for programming languages and educational systems with GPT. The related works are grouped into three categories: works on eLearning systems teaching programming, works on utilizing GPT for education, and works on personalized eLearning.

2.1. Works on eLearning Systems for Programming Languages

There have been studies to design eLearning solutions for teaching programming languages. Mustakerov et al. [2] presented an eLearning system for learning C programming with modules for learning content, tests, exercises, and Q&A. Bashir et al. [3] proposed a problem-based eLearning model that integrates traditional problem-based learning with eLearning features for programming. Rehberger et al. [4] developed a web-based eLearning platform and affiliated teaching techniques, providing exercise programming and modeling languages online and evaluating their usage for future findings about the corresponding learning mechanisms. Wang et al. [5] designed and implemented a method of eLearning for teaching C programming. It finds logic errors which are bugs and cannot be discovered by compilers.

Matthew et al. [6] projected the extracted information to an interactive dashboard and demonstrated its usefulness in allowing database systems professors and teaching staff to quickly identify trends in students' solutions. Harley et al. [7] developed an eLearning and e-assessment tool for a first-year university course on computer programming. This tool presents questions to students and provides immediate feedback on the students' work regarding correctness and score. Dobesova et al. [8] presented an open portal of eLearning for programming including visual programming languages. Hasany et al. [9] devised eLearning for the first computer programming course, including tutoring and assessment. Estévez et al. [10] implemented program coding scaffoldings to teach and experiment with some basic mechanisms of AI systems to high school students using Scratch.

Most of the works in this category proposed eLearning methods and systems for programming languages. However, they do not rely on the automatic generation of course content but conventional ways of creating course content by instructors. And the personalization of course content was not considered in these works.

2.2. Works on Utilizing GPT for Education

There have been works on utilizing GPT for the education domain. Chen et al. [11] designed a ChatGPT-powered programming tool to provide programming code explanations. Hsiao et al. [12] explored a specific personalized guidance technology known as adaptive navigation support. It guides students to appropriate questions in a Java programming course and investigates the effect of personalized guidance. Kris et al. [13] investigated the key motivating factors affecting learning among university undergraduate students taking computer programming courses supported by an eLearning system. Chrysafiadi et al. [14] devised an evaluation method that assesses the results of student modeling in terms of student satisfaction, performance, progress, behavior, and state. Yusupova et al. [15] analyzed the advantages of using eLearning in teaching students programming languages. The authors elaborate on the analysis of collected data and the creation of a platform for online teaching of programming languages, the principles of its operation, and the analysis of the results.

Yilmaz et al. [16] investigated the effect of programming education using ChatGPT on students' computational thinking skills, programming self-efficacy, and motivation. Hosseini et al. [17] conducted a quantitative and qualitative analysis of received responses to questions about the use of ChatGPT in various contexts as well as code discussions. Refining scholarly text or making suggestions to improve existing texts were highlighted among possible positive impacts. Oguz et al. [18] investigated the academic usability of ChatGPT as well as its potential use in undergraduate study. The authors suggested that, while it shows promising potential for academic research in the future, there is a need for further development in certain aspects. Choi et al. [19] utilized ChatGPT to generate answers for four real exams at a law school. Mhlanga et al. [20] analyzed OpenAI regarding the educational sector in developing economies. Mehmet [21] demonstrated how artificial intelligence technologies are integrated into learning management systems using real-world examples, including examples of practical applications as well as integration steps.

Mhlanga [22] analyzed the ethics of using ChatGPT in education, including respect for privacy, fairness and non-discrimination, and transparency in the use of ChatGPT. Biswas et al. [23] asked ChatGPT questions regarding its uses for education to analyze and edit the replies of ChatGPT. Grassini [24] explored the potential and problems associated with applying advanced AI models in education. Kalla et al. [25] examined the advantages and disadvantages of ChatGPT, as well as its limitations and features. Michel-Villarreal [26] studied the unique challenge of the use of AI-generated text to cheat on assignments. Su et al. [27] proposed a theoretical framework for educative AI in education, which includes identifying the desired outcomes, determining the appropriate level of automation, addressing ethical considerations, and evaluating effectiveness. Tlili et al. [28] examined ChatGPT in education through a qualitative instrumental case study. Lo [29] suggested that ChatGPT's performance varied across subject domains, ranging from outstanding (e.g., economics) and satisfactory (e.g., programming) to unsatisfactory (e.g., mathematics).

Most of the works in this category explored the potential and feasibility of GPT models for the purpose of education in various subject areas. They do not directly address eLearning for programming languages.

2.3. Works on Personalizing Learning Content

There exist works on personalizing learning content for each student. Wu [30] proposed a fuzzy tree-structured learning activity model, and a learner profile model to comprehensively describe complex learning activities and learner profiles. They developed an eLearning recommender system based on a recommendation approach. Troussas et al. [31] presented the instruction of computer programming using adaptive learning activities considering students' cognitive skills based on the learning theory of the Revised Bloom Taxonomy. To achieve this, the system utilizes rule-based decision-making and delivers adequate learning activities. Augstein et al. [32] presented an approach based on the modeling of learners' problem-solving activity sequences, and on the use of the models in targeted, and ultimately automated clustering, resulting in the discovery of new, semantically meaningful information about the learners. Murtaza et al. [33] proposed an efficient framework that can offer personalized eLearning to each learner. Gaet et al. [34] proposed a framework for adaptive learning modules and assessment. It describes how personalization can be exploited in eLearning systems.

Rani et al. [35] designed an ontology-driven system to implement the Felder–Silverman learning style model in addition to the learning content, to validate its integration with the semantic web environment. Software agents are employed to provide personalization. Zakrzewska et al. [36] proposed a system in which teaching paths as well as proper layouts are customized to groups of students with similar preferences, created by the application of clustering techniques. Huang et al. [37] described an approach based on the evolvement technique through computerized adaptive testing (CAT). Baylari et al. [38]

proposed a personalized multiagent eLearning system based on item response theory (IRT) and artificial neural network (ANN) which presents adaptive tests (based on IRT) and personalized recommendations (based on ANN). Cakula et al. [39] identified overlapping points of knowledge management and eLearning phases to improve the structure and transfer of personalized course knowledge. Milicevic et al. [40] described a recommendation module of a programming tutoring system, which can automatically adapt to the interests and knowledge levels of learners.

Kausar et al. [41] presented a clustering approach that partitions students into different groups or clusters based on their learning behavior. Alhawiti et al. [42] proposed a personalized eLearning framework, where learning objects are classified, and these learning objects are offered to individual learners according to their personal preferences, skills, and needs. Chen et al. [43] proposed a personalized eLearning system based on Item Response Theory (PEL-IRT) which considers both course material difficulty and learner ability to provide individual learning paths for learners. Chen et al. [44] presented a novel personalized eLearning system with self-regulated learning assistance mechanisms that help learners enhance their self-regulated learning abilities. Ciloglugil [45] implemented an adaptive eLearning system in an "Introduction to Java Programming Language" course, using Learn Square software.

Most of the works in this category proposed methods or systems for personalizing learning content, not necessarily in programming but in general subject areas. And the personalization in these works does not utilize prompt engineering but is custom-designed for the given objectives and target learning subjects.

The related works presented in this section are summarized and compared to our approach to building an eLearning system, as shown in Table 1.

Table 1. Comparative analysis of related works.

Study	**Dynamic Generation of Learning Content with GPT**	**Coverage of All Programming Languages**	**Personalizing Learning Content for Each Student**	**Autonomous Course Management including Grading**	**Near-Zero Effort for Course Creation**
Works [2–10]		Partial Teaching only 1 language			Partial [5,6,8] supports grading
Works [11–29]	Yes			Partial [21] for managing courses	Partial
Works [30–45]		Partial [31,40,45] for only 1 language	Yes But, not through prompt engineering		Partial [43,44]
Our Approach	Yes With content generated by GPT	Yes All P.L. Content trained in GPT	Yes With prompt engineering	Yes With reinforcement learning loop	Yes Zero cost except course profile

Our work is uniquely distinct from existing works in five aspects: (1) dynamically creating specific learning contents on the fly through the GPT server, (2) scope of covering virtually all programming languages available, (3) personalizing the learning content for various student characteristics including age group, education level, and application domain, (4) autonomously managing the whole learning process and its activities, and (5) more significantly, the near-zero effort involved in creating learning content, evaluating exercises, and operating the eLearning system.

3. Software Requirement Specification

This section defines the requirements for developing the target eLearning system.

3.1. Registering Students

This functionality manages the profiles of students who are interested in learning programming through the eLearning system. The profile includes the following attributes: name, identification information, address, phone, email, affiliation, and level in school.

3.2. Registering Course Directors

This functionality manages the profiles of course directors who specify the profiles of programming languages, courses, and course offerings. The profile includes the following attributes: name, identification, affiliation, director certificate, address, phone, email, department, and specialty.

3.3. Registering Programming Languages

This functionality manages the profiles of programming languages such as C, C++, Java, and Python. The profile serves as a meta-description of a target programming language. Language profiles are defined by course directors. It includes several attributes, including language name, version, and standard document. Examples are ANSI C, Java 8, Java 20, Python 2, and Python 3.

3.4. Registering Courses

This functionality manages the profiles of courses for each programming language. A course here refers to a unit of teaching that covers a specific programming language over a specified period such as a semester. There can be multiple courses for a given language such as 'Fundamental Java' and 'Advanced Java' for Java 8. A course profile includes several attributes, including:

- 'Course Description' includes course objectives, topics, and assessment methods like tests, quizzes, assignments, and tests.
- 'Duration' such as a semester-long, an entire school year, or just a few weeks.
- 'Level' specifies the level of difficulty or prerequisites.
- Chapters and topics.

A course is organized into units and each unit is further organized into topics where a topic represents a specific language construct within the unit's main theme. For example, the unit of 'looping' in C language includes three topics: for loop, while loop, and repeat-while loop.

An example of a course profile for 'Fundamental Java' is shown in Table 2.

The comprehensiveness and accuracy of course profiles are of utmost importance as they serve as the foundation for providing instructions to students.

3.5. Registering Offerings

This functionality manages the course offerings for a given course. An offering refers to a specific instance or section of a course that is scheduled to be taught during a particular academic term such as a semester or quarter. Therefore, a course offering is often documented in the syllabus. For a given course, there can be multiple offerings for different semesters, and also multiple offerings for multiple sections for the same semester. For example, an offering for the course 'Fundamental Java' can be 'CS105 for Fall 2023'.

An *Offering Profile* includes several attributes including Course Identifier, Instructor, Class Schedule, Prerequisites, Credits, Course Format, Textbooks and Materials, Grading Policy, and Office Hours.

The relationships among *Language*, *Course*, and *Offering* are shown in Figure 2.

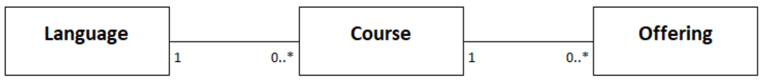


Figure 2. Relationships among language, course, and offering.

Table 2. Course profile of 'Fundamental Java'.

Unit	Title	Topics
1	Introduction to Java	• Overview of Java programming language • Installing Java Development Kit (JDK) • Installing Integrated Development Environment (IDE) • Writing and running a simple Java program
2	Variables and Data Types	• Primitive data types (e.g., int, double, Boolean) • Variable declaration and initialization • Type casting and conversion • String manipulation
3	Control Structures	• Conditional statements (if, else if, else) • Looping (for, while, do-while) • Switch statements
4	Array	• Declaration, initialization, and manipulation of arrays • Array traversal and manipulation • Multidimensional arrays
5	Encapsulation and Information Hiding	• Encapsulation • Information Hiding • Object
6	Class, Constructor, and Destructor	• Class • Constructor and Destructor • Static
7	Inheritance and Polymorphism	• Inheritance and Subclassing • Overloading • Overriding • Dynamic Binding
8	Exception Handling	• Exception as Objects • Handling and Throwing Exceptions • Try-Catch Block • Exception hierarchy • Checked vs. Unchecked Exceptions
9	File Handling	• Reading from and writing to files • File input/output operations
10	Collections	• ArrayList • LinkedList • Other Collection Classes

For a given programming language, there can be zero or more instances of Course. Each course is defined with a distinct objective, teaching coverage, and chapter organization. For a course, there can be zero or more instances of Offering. Students register for a specific offering for a given course.

3.6. Presenting the Foundations

This functionality presents the foundational material of the topics in a unit. The foundation consists of three elements: *Introduction*, *Language Constructs*, and *Code Examples*.

The *Introduction* is an initial overview or explanation that introduces the topic and its significance within the programming language. *Language Construct* refers to a syntactical element or feature provided by a programming language that allows developers to perform specific tasks or operations within their code. These constructs are building blocks that programmers use to create algorithms, define data structures, control program flow, and perform various other functions. *Code Examples* demonstrate the usage of the language constructs.

The system also offers an interactive Q&A feature that simulates in-class teaching. As students delve into the foundational material, they can pose questions at any moment, and the system promptly provides precise and relevant answers. For instance, if a student inquires, "What is the difference between a For loop and a While loop?", the system will present a clear explanation of their distinctions.

3.7. Generating Personalized Coding Exercises

This functionality generates coding exercise problems that are personalized to each user's characteristics, such as age, group, occupation, place, level of language proficiency, and the evaluation results of previous coding exercise submissions. The personalized coding exercise caters to the individual needs, strengths, weaknesses, and learning styles of each student. This tailored approach promotes deeper understanding and retention of material, greater engagement, and often results in better academic outcomes. In contrast, one-way coding exercise problems apply a "one-size-fits-all" model, which might not resonate with all students and can leave some students behind.

3.8. Submitting Exercise Solutions

This functionality allows students to submit their exercise solutions. Once submitted successfully, the system triggers the evaluation of the submission.

3.9. Evaluating Submissions of Coding Exercises

This functionality evaluates the submitted coding exercises. The system evaluates the submitted program codes based on the given evaluation criteria. Initially, the set of evaluation criteria consists of conformance, correctness, efficiency, and extensibility. However, the criteria for evaluation can be altered by the course directors.

The system evaluates exercise submissions using either letter grades (A, B, C, D, and F) or a numerical score ranging from 0 to 100. In addition, it provides feedback on necessary corrections, suggestions for improvement, and alternative approaches to writing the code.

3.10. Generating Learning Progress Reports

This functionality generates Learning Progress Reports and Certificates of Completion for students. The progress report provides comprehensive details of the training sessions conducted. It encompasses the complete history of training sessions conducted for the specific programming language, highlighting the exercises undertaken and the evaluation results of code submissions.

A certificate confirms that the recipient has successfully completed all the required training units and fulfilled the necessary criteria for certification. It is a concise, one-page document that bears the official seal of the training institute.

3.11. Interaction with a GPT Server

As an artificial intelligence model designed for natural language processing tasks, GPT can play a key role in generating learning content. We utilize a GPT as a backend engine for generating the foundations of topics, generating exercise problems, and evaluating exercise submissions. The initial version of our eLearning system utilizes ChatGPT, but the system architecture is designed to utilize alternative GPTs such as Google Bard.

4. Design Specification

This section presents the design of the eLearning system. The system was designed with the assumption of a client-server architecture for effective system maintenance and web client support. Also, the design was conducted using an object-oriented paradigm with UML to deliver high modularity and extensibility.

4.1. Schematic Architecture for GPT Interactions

The schematic architecture of a target system is a structured and visual representation of structural elements, their roles, and relationships among the elements. The key elements in a schematic architecture are tiers, layers, partitions within a layer, and their relationships.

The schematic architecture of the eLearning system is defined with three architecture styles: tiered architecture, MVC, and Microservice, as shown in Figure 3.

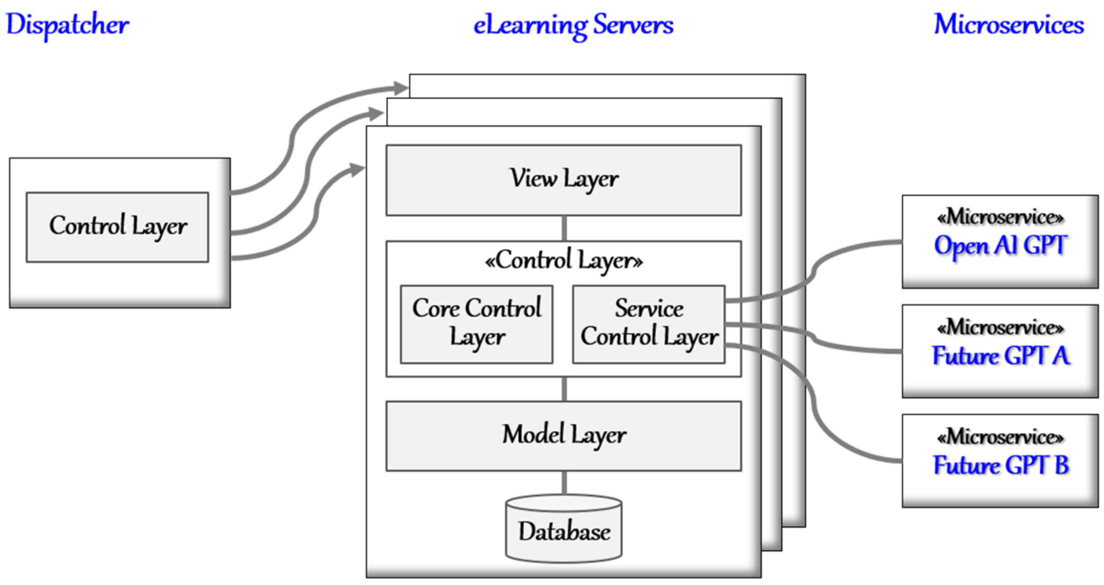


Figure 3. Schematic architecture of the eLearning system.

The role of the Dispatcher tier is to replicate eLearning Servers to ensure high availability and reliability, distributing the invocation load evenly among the replicas. The eLearning Server tier is responsible for delivering the core functionality of the system, while the Microservice tier is designed to represent externally developed and deployed GPT servers.

The 'Service Control Layer' of the eLearning Server dynamically binds a compatible microservice that provides a GPT language model. This design decision is needed to avoid dependency on a particular GPT model since various large language models are available, including OpenAI ChatGPT, Google Bard, and Meta Llama 2. Even newer and better GPT models will appear in the future. Our default GPT model is set to ChatGPT, but the 'Service Control Layer' performs interface adaptation using the Adapter Pattern [46] and mediation using the Mediator Pattern [46] when adopting other GPT models.

In addition, this schematic architecture is designed to offer the advantages of high availability, fault tolerance [47], and dynamic adaptation of GPT models.

4.2. Modeling the System Functionality

A use case diagram captures the key functionality of the system, which becomes the basis for deriving functional components. The whole system functionality was modeled as a set of use cases, and Figure 4 shows the essential part of the use case diagram for the eLearning system.

Each use case is given a two-digit identifier for its functional group, followed by a sequential number such as 'FT01. Get Instruction Scope'. The figure shows only the essential use cases, including use cases for interacting with the GPT model.

Note that we employ three actors of software agent type: *Teaching Agent*, *Exercise Agent*, and *Evaluation Agent*. A software agent in use case diagrams does not model human users or hardware devices; rather, it models a background-running daemon process that invokes its relevant use cases [48]. This type of actor is utilized to eliminate instructors' efforts in managing eLearning sessions and even grading student submissions.

From the use case diagram, functional components can be identified by grouping a set of relevant use cases. Then, the components are allocated to the schematic architecture of the system as shown in Figure 5.

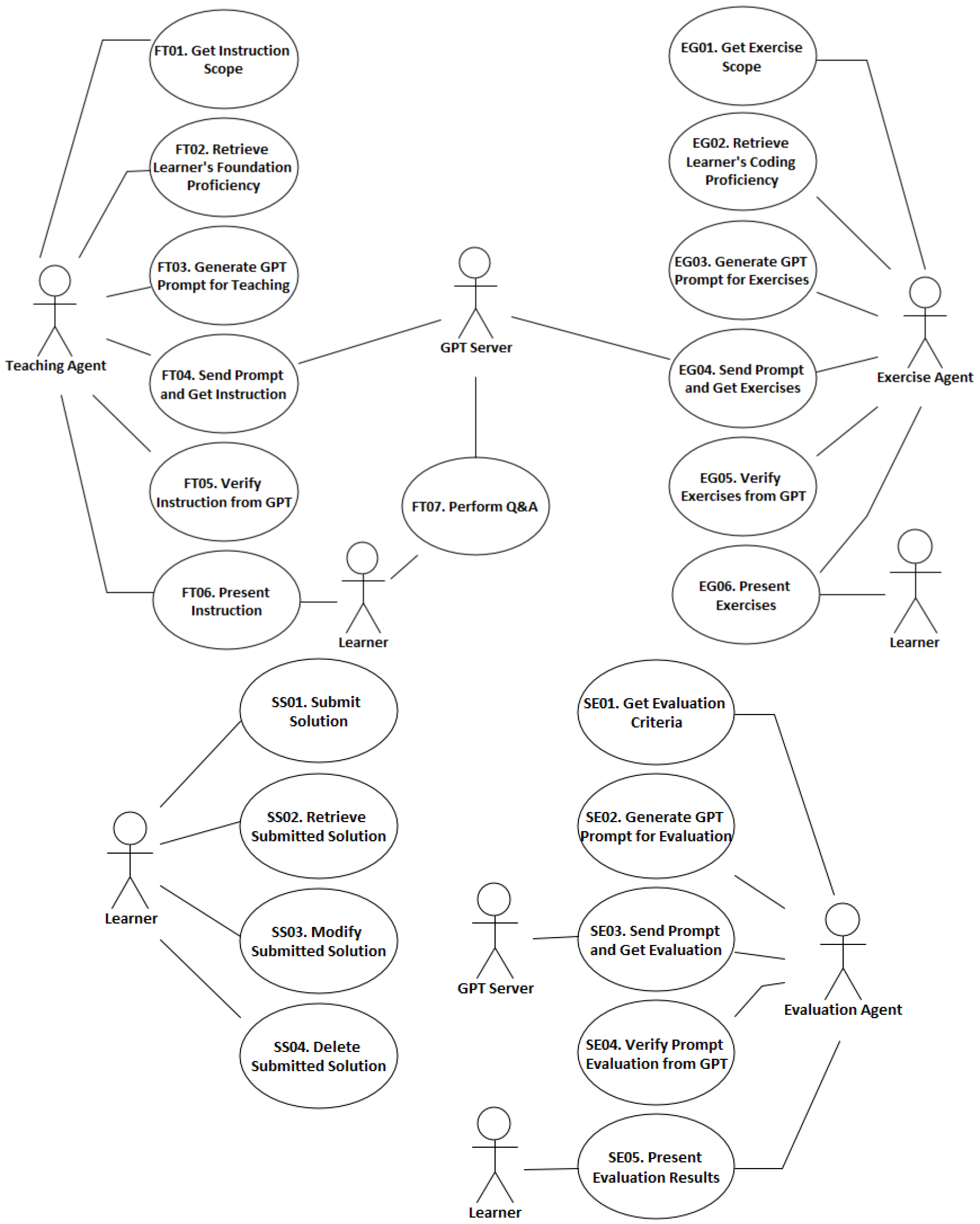

FT01. Get Instruction Scope
FT02. Retrieve Learner's Foundation Proficiency
FT03. Generate GPT Prompt for Teaching
FT04. Send Prompt and Get Instruction
FT05. Verify Instruction from GPT
FT06. Present Instruction
FT07. Perform Q&A
Teaching Agent
GPT Server
Learner
EG01. Get Exercise Scope
EG02. Retrieve Learner's Coding Proficiency
EG03. Generate GPT Prompt for Exercises
EG04. Send Prompt and Get Exercises
EG05. Verify Exercises from GPT
EG06. Present Exercises
Exercise Agent
Learner
SS01. Submit Solution
SS02. Retrieve Submitted Solution
SS03. Modify Submitted Solution
SS04. Delete Submitted Solution
Learner
SE01. Get Evaluation Criteria
SE02. Generate GPT Prompt for Evaluation
SE03. Send Prompt and Get Evaluation
SE04. Verify Prompt Evaluation from GPT
SE05. Present Evaluation Results
GPT Server
Evaluation Agent
Learner

Figure 4. Part of the use case diagram for the eLearning system.

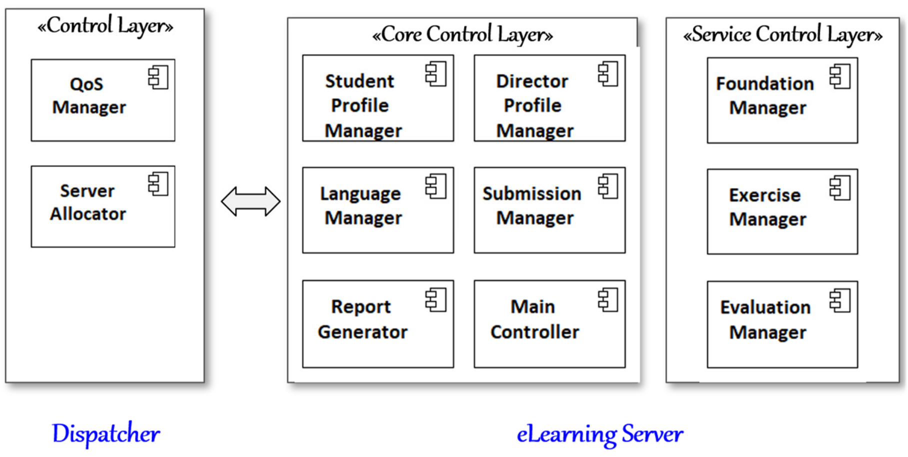


Figure 5. Functional components of the eLearning system.

Note that the three functional components of the *Service Control Layer* interact with a GPT Server to generate learning content.

4.3. Persistent Dataset Modeling

The persistent datasets of the system are modeled as entity-type classes, which map to tables of a relational database. Figure 6 shows the persistent object model in a class diagram.

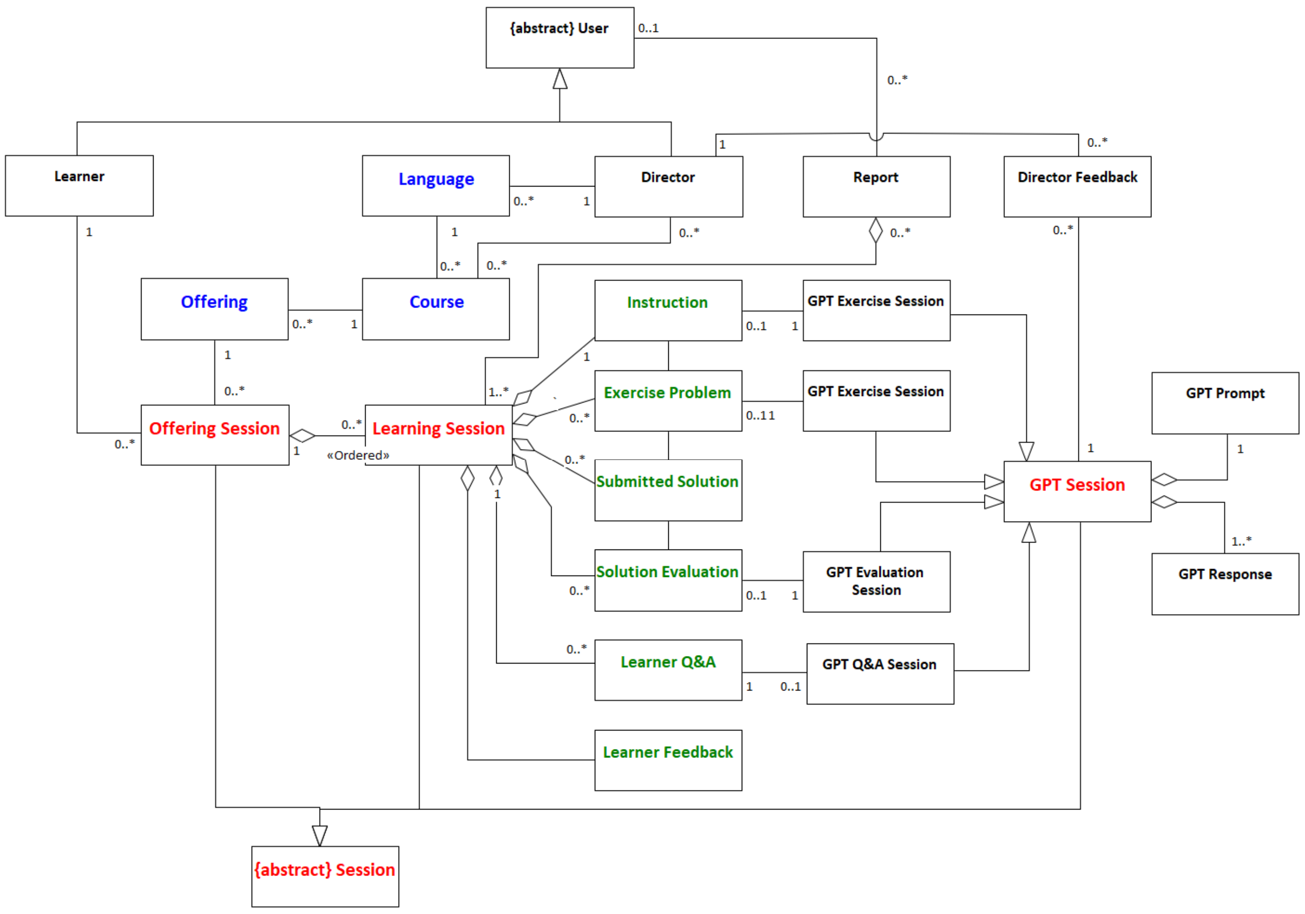


Figure 6. Class diagram modeling persistent object classes.

This class diagram captures three types of persistent datasets: (1) programming language-related classes in blue color, (2) classes of sessions in red color, i.e., logs of learning activities, and (3) classes of generated learning content in green color.

Among the session-related classes, the class 'Learning Session' is defined as an aggregation of six other classes. This class is essential not only for keeping track of current progress but also for autonomously managing the eLearning sessions.

The persistent datasets are maintained in a database through object-relational mapping, and the stored session information is further utilized to enhance the quality of system services.

4.4. Autonomous Management of System Behavior

The main control flow of a system specifies the whole runtime behavior of the system. Figure 7 shows the entire control flow of the eLearning system using an activity diagram.

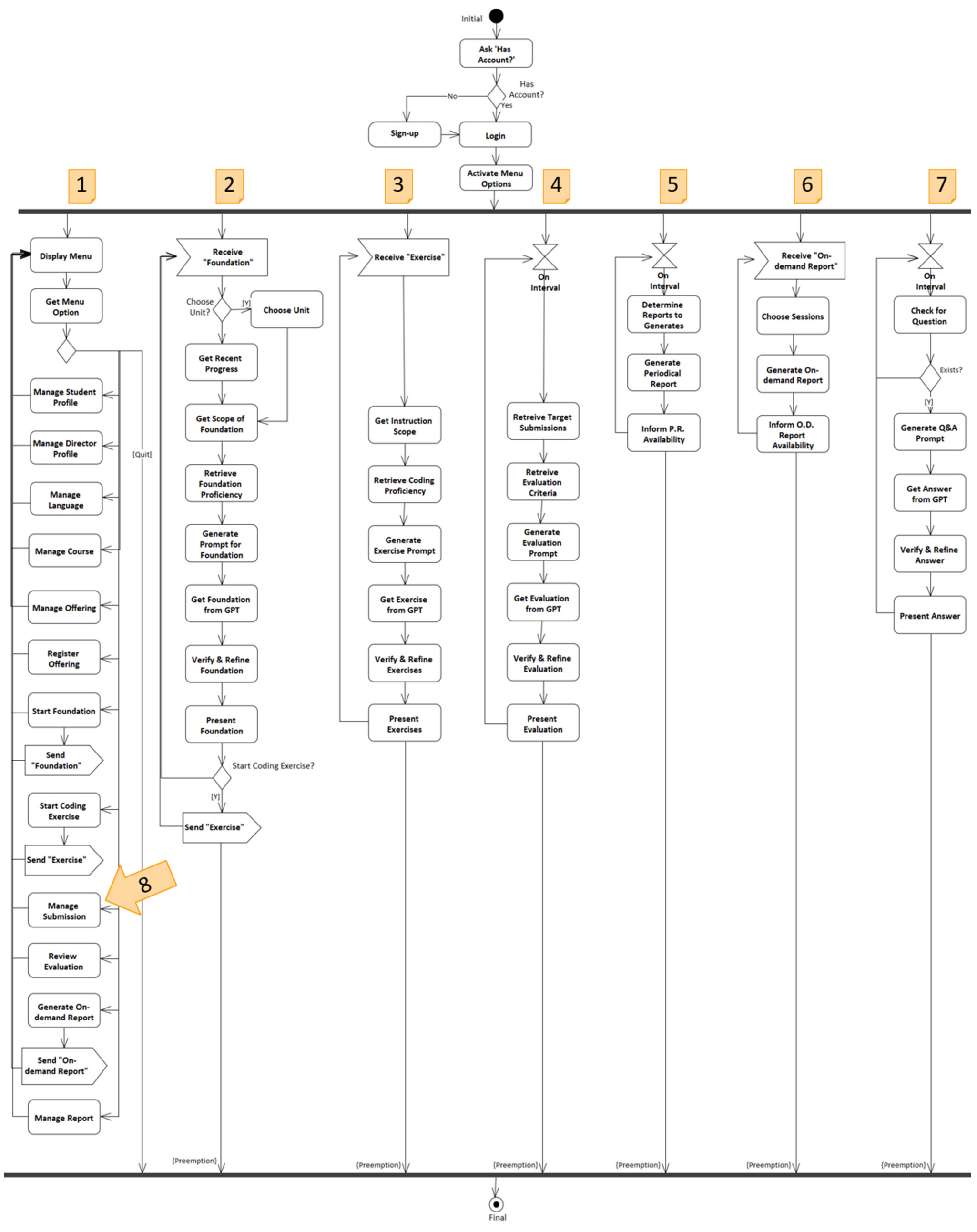


Figure 7. System control flow in an activity diagram.

After login, the system runs seven tasks in parallel. Thread 1 handles a menu for the functionality with explicit invocation. Thread 2 handles the generation of foundational materials for given topics in the current unit. Thread 3 handles the generation of personalized exercise problems for the current unit. Thread 4 requests the GPT model to evaluate the exercise solutions submitted by students. Threads 2, 3, and 4 rely on the GPT model in providing their functionalities. This is enabled by constructing and transmitting customized GPT prompts, and its logic is specified in Algorithm 1. The submission of exercise solutions by students is handled by an action of 'Manage Submission' in thread 1.

Threads 5 and 6 handle the generation of periodical and on-demand reports. Thread 7 handles the interaction with students, i.e., Q&A sessions. The parallel processing with the seven threads allows students to navigate the various options of learning activities flexibly. For example, a student may stop learning a topic and switch to another topic or unit by using a web service interface. The 'Manage Submission' action in the activity diagram (noted as '8' in the figure), is used to submit students' coding exercise solutions.

4.5. GPT Prompt Engineering for Content Personalization

Prompt engineering for GPT entails the design and optimization of input queries to achieve specific and accurate model outputs. This practice is crucial for personalizing content for eLearning systems. By fine-tuning the structure and vocabulary of the prompts, practitioners can enhance model efficiency, reduce computational overhead, and improve accuracy [49].

Each of the threads 2, 3, and 4 in the activity diagram includes an action for prompt engineering. The prompt is customized for each student's characteristics, performance in learning, and learning effectiveness, as follows:

- Level of Compactness

  There exists a right level of instruction detail that fits well with each student's education level, familiar domain, and language proficiency. Beginners would prefer to have an instruction that fully elaborates the topics to be learned, while advanced learners would prefer to have an instruction that only presents the essential content of the topic.

- Types of Examples and Coding Exercises

  There typically exist application domains of the examples and exercise problems that each student is familiar with. Hence, the system should generate examples and exercise problems for the domains for effective learning.

- Number of Exercise Problems

  There exists a right number of incrementally challenging exercises that fit well with each student's education level and language proficiency. Beginners may prefer a large number of incrementally challenging exercises, while advanced learners may prefer a small number of relatively challenging exercises.

Based on these observations, we devise design tactics for personalizing the instructional content from a GPT server.

- Domains of Interest

  This tactic determines the application domains that a learner is familiar with for the examples and coding exercises. This information can automatically be inferred by analyzing student profiles or can explicitly be specified by students. An attribute *domain_list* maintains an ordered list of domains.

- Education Level

  This tactic determines the educational level which will be used for the personalization. This information can automatically be inferred by reviewing the affiliation or school information in the student profiles.

- Language Proficiency

This tactic determines the proficiency of students in the target language, which will be utilized in providing personalized learning. An attribute, *lang_level*, is defined to be an integer between 1 and 5, where level 1 represents beginner-level and level 5 represents expert-level. A new attribute, *compound_level*, is used to indicate the compound proficiency level of students. It is measured as the sum of *Edu_Level* and *Lang_Level*, and the sum will be on a scale of between 1 and 10. The compound level is then utilized in generating GPT prompts.

The process for personalizing learning content for each student is modeled using a data flow diagram as shown in Figure 8.

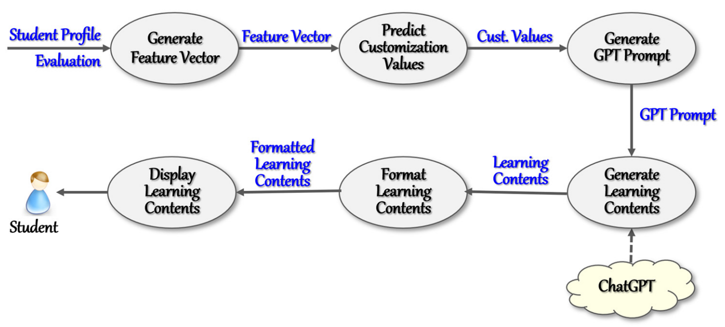
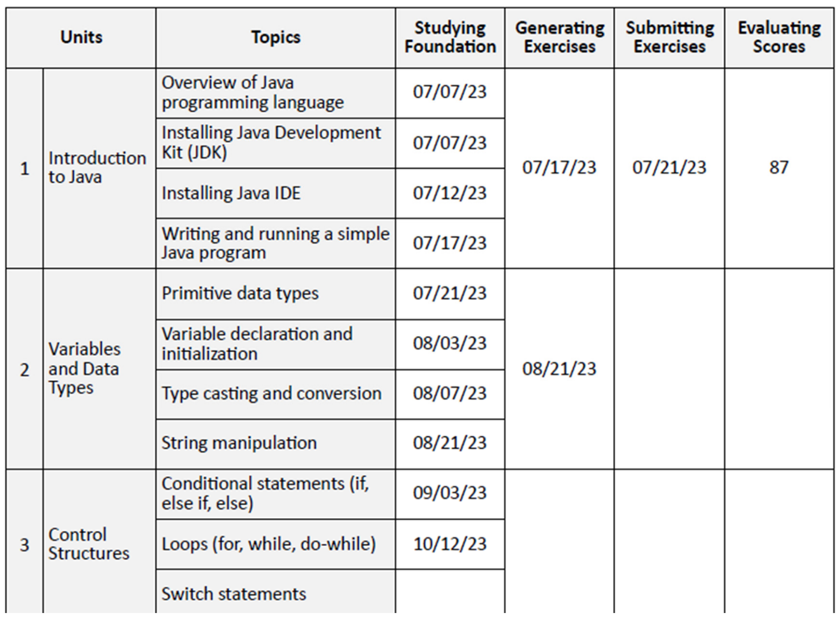


Figure 8. Process of personalizing learning content.

The system generates a feature vector for each student by reading the student profile and evaluation results on submitted exercises. Then, the vector is transformed into numeric values that are appropriate for customizing learning content. Then, a GPT prompt is automatically generated based on the customization values. By sending the prompt to the GPT server, the system acquires learning content and formats the content for students.

4.6. Foundation Manager Algorithm

The Foundation Manager component configures the personalized foundational content for the topics to learn by consulting the GPT server and presents the content on the screen. The system first determines the unit and its topic to learn for each student by referring to the ‘Table of Learning Progress’ as shown in Figure 9.



Units		Topics	Studying Foundation	Generating Exercises	Submitting Exercises	Evaluating Scores
1	Introduction to Java	Overview of Java programming language	07/07/23	07/17/23	07/21/23	87
		Installing Java Development Kit (JDK)	07/07/23			
		Installing Java IDE	07/12/23			
		Writing and running a simple Java program	07/17/23			
2	Variables and Data Types	Primitive data types	07/21/23	08/21/23		
		Variable declaration and initialization	08/03/23			
		Type casting and conversion	08/07/23			
		String manipulation	08/21/23			
3	Control Structures	Conditional statements (if, else if, else)	09/03/23			
		Loops (for, while, do-while)	10/12/23			
		Switch statements				

Figure 9. Table of ‘Learning Progress’.

This table keeps track of the learning progress, and it is created for each registered offering. The system enters the date of completion when a student completes a learning activity. For example, the whole of unit 1 is completed, including exercise submission and its grading. A student finished studying all four topics in unit 2 but did not start the rest of the activities. The student started learning the first two topics in unit 3, even before completely finishing unit 2.

The control flow of 'Foundation Manager' is specified in Algorithm 1.

Algorithm 1: Control Flow of 'Foundation Manager'

```
Input: Table of Progress
Output: Content of Foundation for the Learning Scope
1:genereate_Foundation() {
2:    //Step 1. Generate the scope to learn.
3:    LANG := get_PL();     //Retrieve the target language to learn
4:    If (Auto-Resume) then SCOPE := get_scope() //From Progress Table
5:    else
6:    SCOPE := get_scope_from_Student(); //A student enters the scope.
7:    compound_level:= compute_compound_level();
8:    //Step 2. Generate a personalized GPT prompt
9:    PROMPT := "A student wants to learn about <SCOPE> of a programming language <LANG>.";
10:    PROMPT += "Generate an instructional material for the given topics in the following order:";
11:    PROMPT += "(1) Introduction to the given topics,";
12:    PROMPT += "(2) syntax of the relevant language constructs,"
13:    PROMPT += "(3) examples of using the syntax, and";
14:    PROMPT += "(4) additional guidelines for utilizing the language constructs."
15:    PROMPT += "Personalize the learning contents for this student who has the following features;"
16:    PROMPT += "The student's age is " + toString(current_Student.getAge());
17:    PROMPT += "Applications of interest are in the order of " + toString(domain_list);
18:    PROMPT += "The student's language proficiency level is " + toString(compound_level);
19:    //Step 3. Submit the prompt to and receive the response from GPT server.
20:    RESPONSE := GPT_Server.chat(PROMPT);
21:    RESPONSE := verify_Foundation(RESPONSE);
22:    //Step 4. Display the learning foundation
23:    foundation_content := format(RESPONSE);
24:    display(foundation_content);
25:    //Step 5. Store the current session on database.
26:    store_session(LANG, SCOPE, PROMPT, RESPONSE);
27:END
```

Note that the GPT prompt for generating the foundation to learn is personalized with student-specific features such as age, domain, and proficiency level, as shown in step 2.

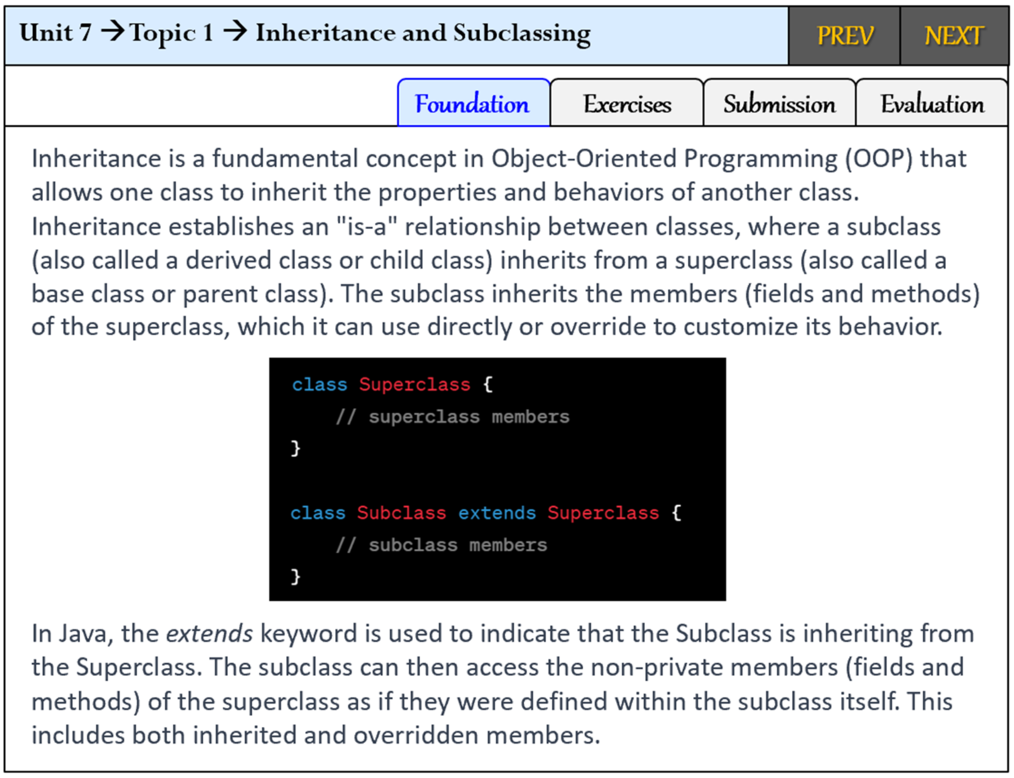
5. Implementation and Experiments

5.1. Proof-of-Concept Implementation

We implemented a proof-of-concept system with its design. The development environment includes Python 3.1, Ubuntu operating system, MySQL for the database, and Django for the web framework.

We implemented the eLearning system utilizing the ChatGPT 4.0 API (GPT-4-0613). The primary classes of the API employed in our implementation encompass *ChatGPT-Client* for handling API authentication and session management, *ChatSession* for managing chat sessions and conversation states, *Message* for representing messages sent to and received from the model, *CompletionRequest* for encapsulating parameters for generating text completions, and *FilteringOptions* for specifying content filtering settings.

All the functional components, data components, main control flow, Foundation Manager algorithm, and design for personalizing GPT prompts have been fully implemented as specified. The user interface for the main menu is shown in Figure 10.



Inheritance is a fundamental concept in Object-Oriented Programming (OOP) that allows one class to inherit the properties and behaviors of another class. Inheritance establishes an "is-a" relationship between classes, where a subclass (also called a derived class or child class) inherits from a superclass (also called a base class or parent class). The subclass inherits the members (fields and methods) of the superclass, which it can use directly or override to customize its behavior.

```
class Superclass {
    // superclass members
}

class Subclass extends Superclass {
    // subclass members
}
```

In Java, the *extends* keyword is used to indicate that the Subclass is inheriting from the Superclass. The subclass can then access the non-private members (fields and methods) of the superclass as if they were defined within the subclass itself. This includes both inherited and overridden members.

Figure 10. User interface of the eLearning system.

The user interface keeps displaying the current topics such as 'Unit 7 → Topic 1 → Inheritance and Subclassing' and it offers tabs for choosing an option among the four activities: *Foundation*, *Exercises*, *Submission*, and *Evaluation*.

The system displays the foundational content of the current topic in the content frame. Upon completing the foundations, students proceed to choose the *Exercises* tab, which will trigger the generation of exercise problems. Upon completing coding exercises, students can submit their solutions by using the *Submission* Table. Students can view the results of the evaluation on their submissions by using the *Evaluation* tab.

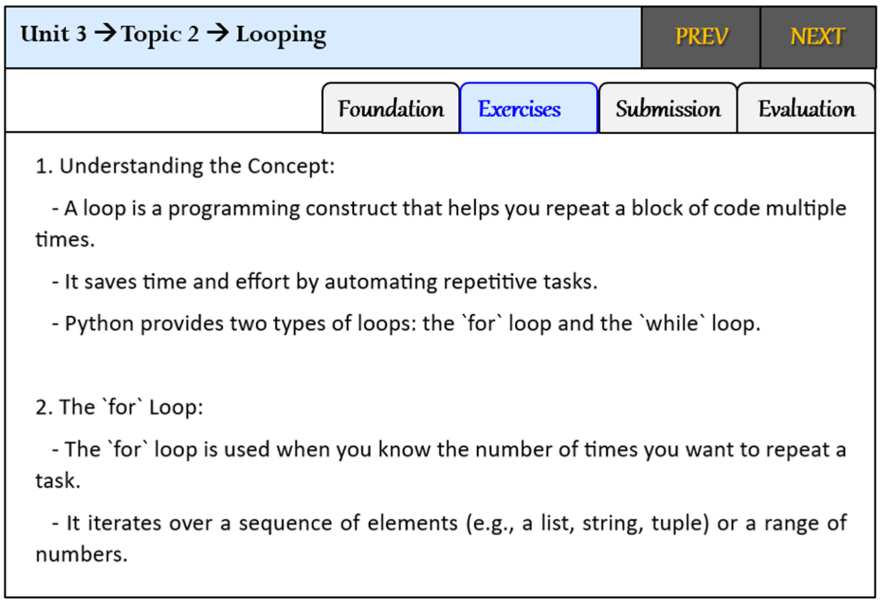
The *PREV* and *NEXT* buttons allow students to freely navigate through units and topics within each unit. A click on the *NEXT* button brings up the next topic in the unit and continues displaying the learning content. However, a click on the *NEXT* button while studying the last topic in the unit will trigger a generation of exercise problems for the student.

5.2. Assessment through Personalized Content

We conducted a number of experiments to evaluate the delivered eLearning system. The implemented eLearning system is provided as a cloud service and hence the only environmental constraint for accessing the system is the use of a web browser.

5.2.1. Experiment of Personalizing Foundations for Different Proficiency Levels

This experiment generates foundational material for a topic for a student at level 1, i.e., novice, and for another student at level 10, i.e., subject expert. The foundational material generated from GPT for the student at level 1 is shown in Figure 11.



1. Understanding the Concept:

- A loop is a programming construct that helps you repeat a block of code multiple times.

- It saves time and effort by automating repetitive tasks.

- Python provides two types of loops: the `for` loop and the `while` loop.

2. The `for` Loop:

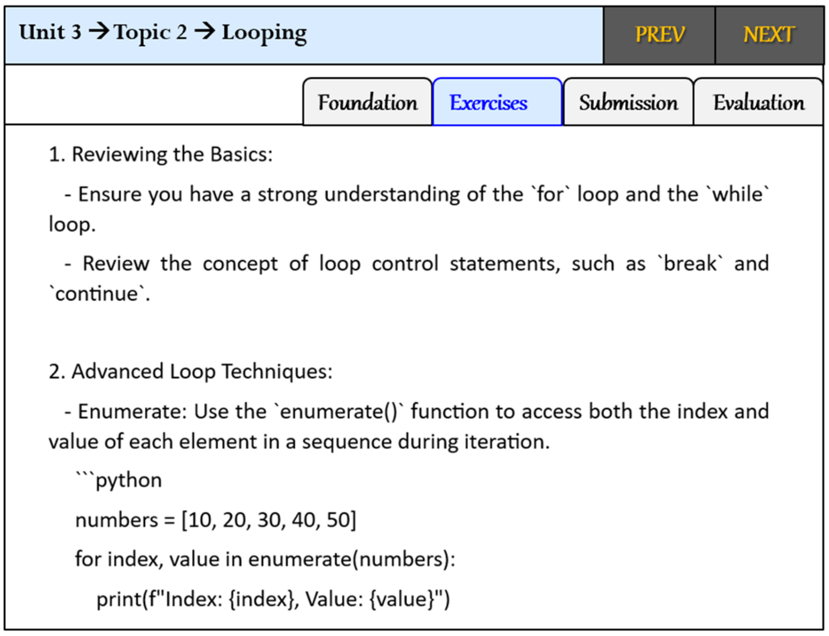
- The `for` loop is used when you know the number of times you want to repeat a task.

- It iterates over a sequence of elements (e.g., a list, string, tuple) or a range of numbers.

Figure 11. Foundational material from GPT for a student at level 1.

As shown in the figure, the foundation only includes a preliminary concept of 'loop' constructs including 'for' loop. This content is self-explanatory, and the example is easy to understand and intuitive.

Now, the foundational material generated from GPT for the student at level 10 is shown in Figure 12.



1. Reviewing the Basics:

- Ensure you have a strong understanding of the `for` loop and the `while` loop.

- Review the concept of loop control statements, such as `break` and `continue`.

2. Advanced Loop Techniques:

- Enumerate: Use the `enumerate()` function to access both the index and value of each element in a sequence during iteration.

~~~
```python
numbers = [10, 20, 30, 40, 50]
for index, value in enumerate(numbers):
    print(f"Index: {index}, Value: {value}")
~~~

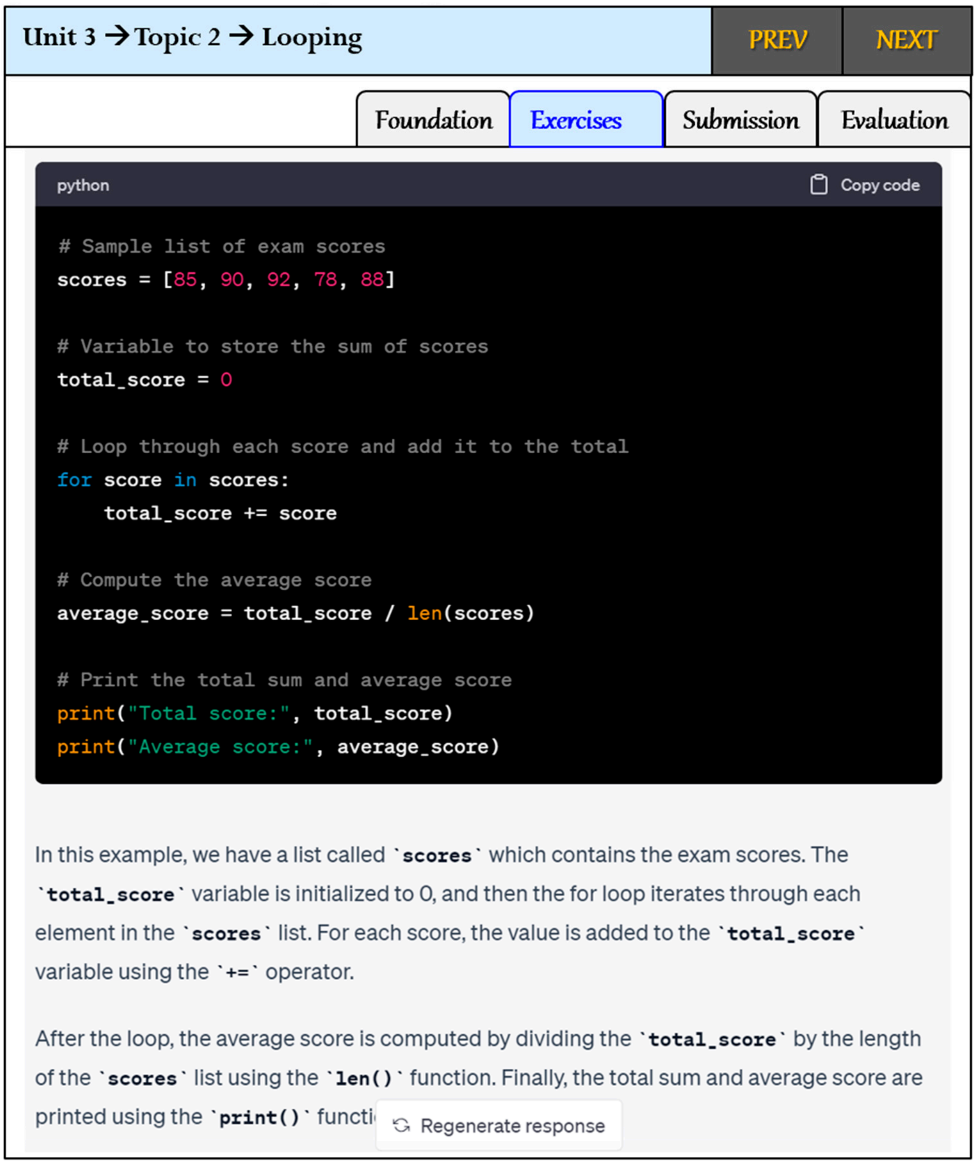
Figure 12. Foundational material from GPT for a student at level 10.

The generated foundation presents a summary of basic constructs including 'for', while', and 'break'. And it also presents advanced techniques for using loop constructs, and understanding the given example requires some basic proficiency in Python language. This foundational content should reasonably challenge students at level 10 but will be too challenging for beginners to understand.

Through the two contents of foundational material generated for different levels of students, we can clearly observe that the GPT content for the foundations is well personalized for each proficiency level.

5.2.2. Experiment of Personalizing Examples for Different Domains

This experiment generates examples of looping for a student in the academic domain and for another student in the finance domain. The examples of looping generated from GPT for the student in academia are shown in Figure 13.



```python
# Sample list of exam scores
scores = [85, 90, 92, 78, 88]

# Variable to store the sum of scores
total_score = 0

# Loop through each score and add it to the total
for score in scores:
    total_score += score

# Compute the average score
average_score = total_score / len(scores)

# Print the total sum and average score
print("Total score:", total_score)
print("Average score:", average_score)
```

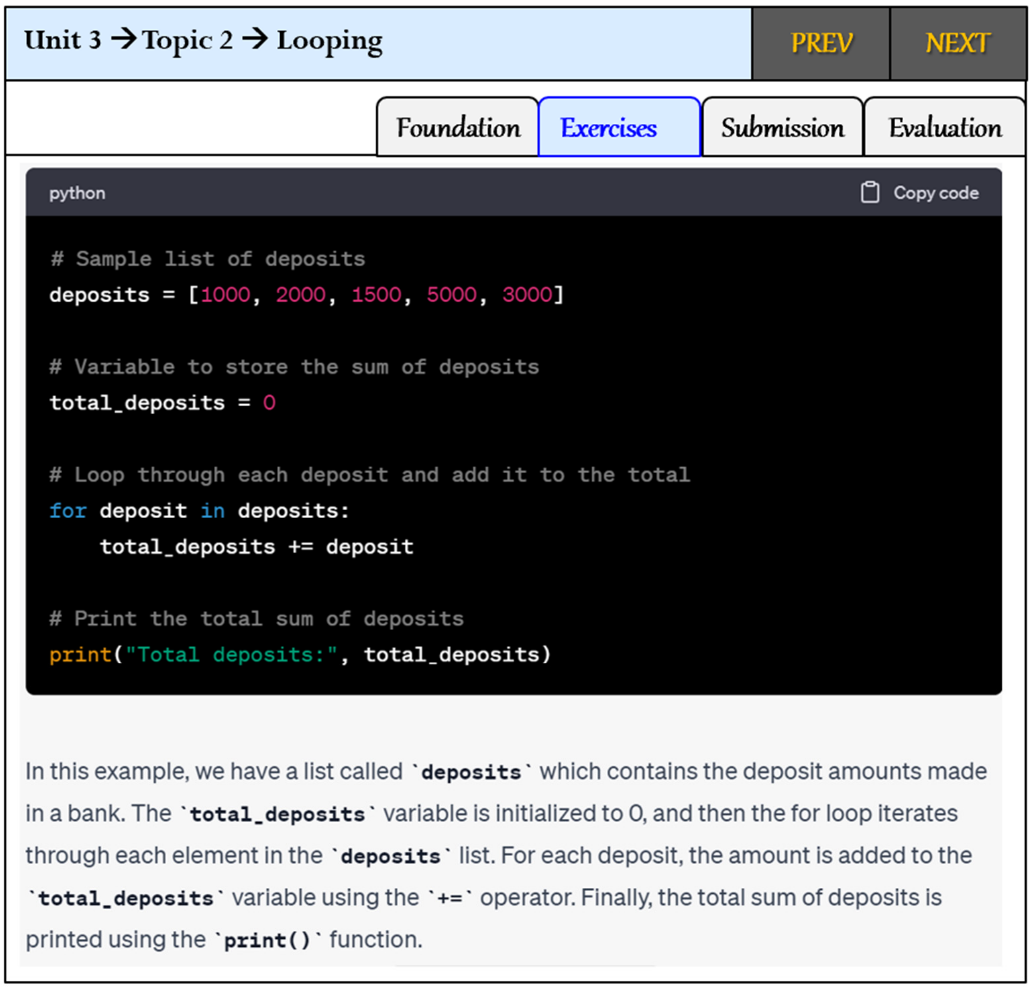
In this example, we have a list called `` `scores` `` which contains the exam scores. The `` `total_score` `` variable is initialized to 0, and then the for loop iterates through each element in the `` `scores` `` list. For each score, the value is added to the `` `total_score` `` variable using the `` `+=` `` operator.

After the loop, the average score is computed by dividing the `` `total_score` `` by the length of the `` `scores` `` list using the `` `len()` `` function. Finally, the total sum and average score are printed using the `` `print()` `` functi

Figure 13. Generated example for a student in academia.

As shown in the screen dump, the example of computing the total and average of some numbers is simple enough for any student to understand.

Now, the foundation generated from GPT for the student in the finance domain is shown in Figure 14.



```python
# Sample list of deposits
deposits = [1000, 2000, 1500, 5000, 3000]

# Variable to store the sum of deposits
total_deposits = 0

# Loop through each deposit and add it to the total
for deposit in deposits:
    total_deposits += deposit

# Print the total sum of deposits
print("Total deposits:", total_deposits)
```

In this example, we have a list called `deposits` which contains the deposit amounts made in a bank. The `total_deposits` variable is initialized to 0, and then the for loop iterates through each element in the `deposits` list. For each deposit, the amount is added to the `total_deposits` variable using the `+=` operator. Finally, the total sum of deposits is printed using the `print()` function.

Figure 14. Generated example for a student in finance.

The generated example uses a loop for calculating the total amount of deposits made on bank accounts. This example is intuitive and easy to understand for users in the finance domain.

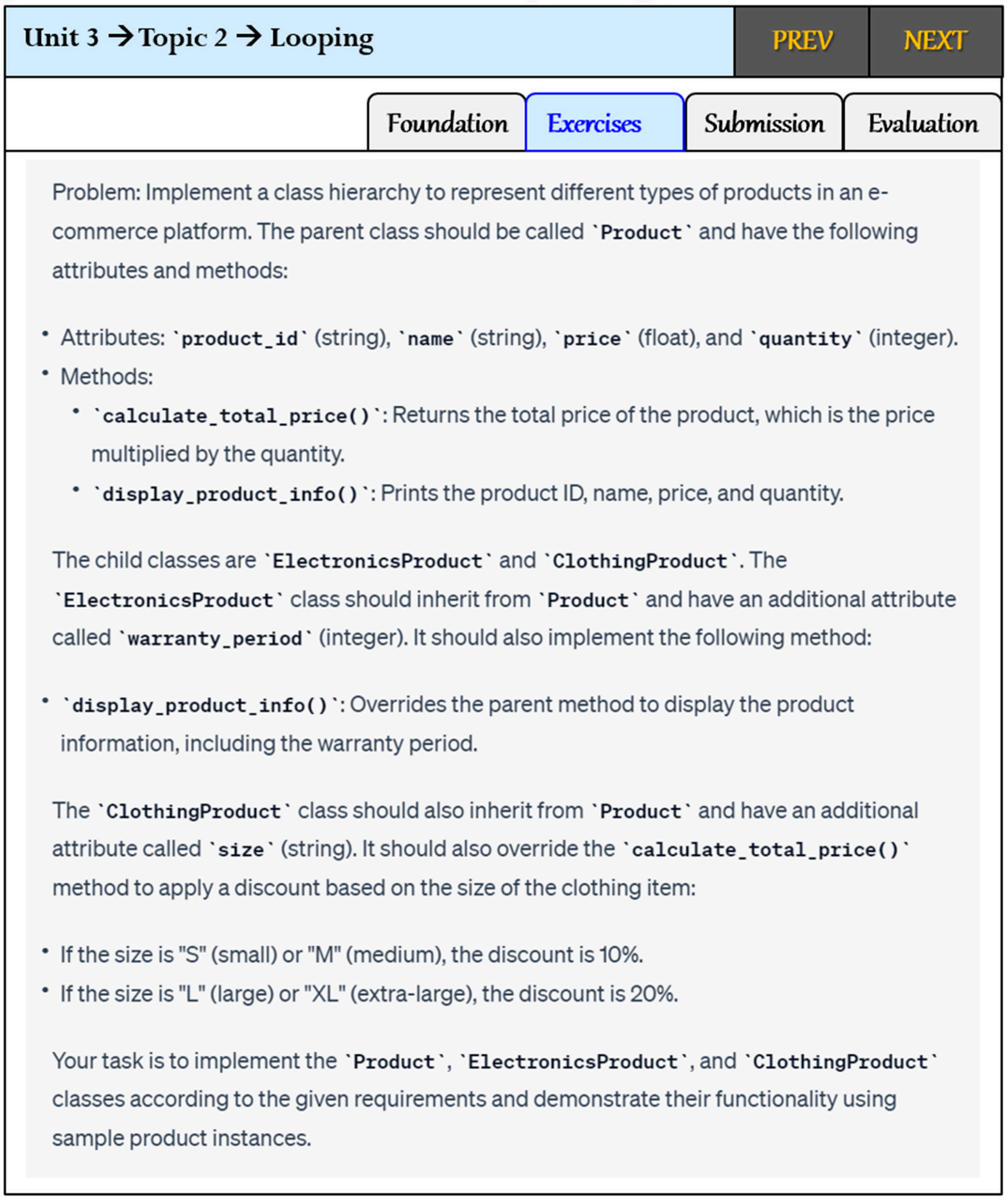
Through the two contents of examples generated for different domains, we can clearly observe that the GPT content for generated examples is well personalized for each domain of interest.

5.2.3. Experiment of Personalizing Coding Exercises for Domains of Interests

This experiment generates coding exercises using 'inheritance' for a student in the eCommerce domain and for another student in the finance domain. The coding exercise for the student in eCommerce is shown in Figure 15.

As shown in the exercise problem, the three classes, *Product*, *ElectroicsProducts*, and *ClothingProduct* are entities in the eCommerce domain. In addition, the methods of each class are the common operations, such as *display_product_info()*, commonly used in the eCommerce domain. Hence, users in the eCommerce domain should feel comfortable understanding the exercise.

Now, the coding exercise generated for the student in the finance domain is shown in Figure 16.



Problem: Implement a class hierarchy to represent different types of products in an e-commerce platform. The parent class should be called `Product` and have the following attributes and methods:

- Attributes: `product_id` (string), `name` (string), `price` (float), and `quantity` (integer).
- Methods:
  - `calculate_total_price()`: Returns the total price of the product, which is the price multiplied by the quantity.
  - `display_product_info()`: Prints the product ID, name, price, and quantity.

The child classes are `ElectronicsProduct` and `ClothingProduct`. The `ElectronicsProduct` class should inherit from `Product` and have an additional attribute called `warranty_period` (integer). It should also implement the following method:

- `display_product_info()`: Overrides the parent method to display the product information, including the warranty period.

The `ClothingProduct` class should also inherit from `Product` and have an additional attribute called `size` (string). It should also override the `calculate_total_price()` method to apply a discount based on the size of the clothing item:

- If the size is "S" (small) or "M" (medium), the discount is 10%.
- If the size is "L" (large) or "XL" (extra-large), the discount is 20%.

Your task is to implement the `Product`, `ElectronicsProduct`, and `ClothingProduct` classes according to the given requirements and demonstrate their functionality using sample product instances.

Figure 15. Generated coding exercises for a student in eCommerce.

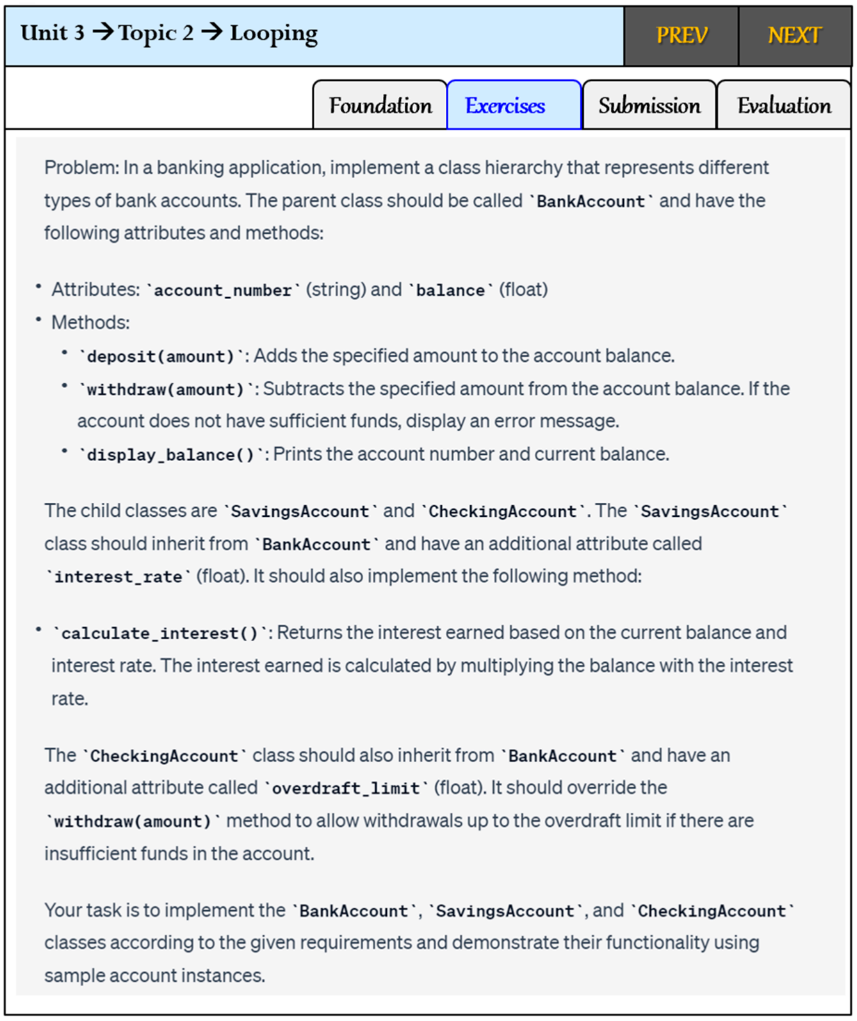
The generated coding exercise is about managing banking accounts, including a savings account and a checking account. And, the methods of each class, such as deposit() and withdraw(), are common operations in banks. Hence, this exercise problem is intuitive and easy to understand for users in the finance domain.

Through the two contents of coding exercises generated for different domains, we can clearly observe that the GPT content for generated coding exercises is well personalized for each domain of interest.

5.3. Assessment through Experiments

Evaluating the quality of educational content and its associated eLearning system cannot be adequately measured using quantitative metrics. Instead, a qualitative approach, such as feedback from users, is more appropriate.

We utilize two criteria for this evaluation: effectiveness of learning and cost of developing and operating the system. These criteria are aligned with our formulated research objectives and questions.



Problem: In a banking application, implement a class hierarchy that represents different types of bank accounts. The parent class should be called `BankAccount` and have the following attributes and methods:

- Attributes: `account_number` (string) and `balance` (float)
- Methods:
  - `deposit(amount)`: Adds the specified amount to the account balance.
  - `withdraw(amount)`: Subtracts the specified amount from the account balance. If the account does not have sufficient funds, display an error message.
  - `display_balance()`: Prints the account number and current balance.

The child classes are `SavingsAccount` and `CheckingAccount`. The `SavingsAccount` class should inherit from `BankAccount` and have an additional attribute called `interest_rate` (float). It should also implement the following method:

- `calculate_interest()`: Returns the interest earned based on the current balance and interest rate. The interest earned is calculated by multiplying the balance with the interest rate.

The `CheckingAccount` class should also inherit from `BankAccount` and have an additional attribute called `overdraft_limit` (float). It should override the `withdraw(amount)` method to allow withdrawals up to the overdraft limit if there are insufficient funds in the account.

Your task is to implement the `BankAccount`, `SavingsAccount`, and `CheckingAccount` classes according to the given requirements and demonstrate their functionality using sample account instances.

Figure 16. Generated coding exercises for a student in finance.

5.3.1. Evaluating the Effectiveness of Learning

This criterion evaluates the effectiveness of learning based on feedback from users. We created a questionnaire with 10 questions by considering the objectives of the system as shown in Table 3.

The first three questions, Q1, Q2, and Q3, evaluate the content coverage of a target programming language's constructs. The next three questions, Q4, Q5, and Q6, evaluate the extent to which the foundational content, examples, and coding exercises are personalized for each student. The next three questions, Q7, Q8, and Q9, evaluate the quality of evaluation, i.e., grading of the submitted exercise solutions. The last question, Q10, evaluates the usability of the systems.

The main user groups of the system are identified as faculty, students, and industry developers. Faculty members would use the system to generate the content for teaching and coding exercises, students would use the system to learn programming skills, and industry developers would use the system to learn the language from scratch or improve their language proficiency.

Accordingly, we formed three distinct participant groups for the experiment: the faculty group, the student group, and the industry developer group. Each group used tailored evaluation criteria. The faculty group assessed the system from an instructor's perspective, comparing the provided learning content to their personal lecture materials and teaching experiences. The student group evaluated the system from a learner's perspective, reflecting on the effectiveness and productivity of their learning experiences. Meanwhile, the developer group judged from a practitioner's standpoint, focusing on the practicality of the learning experience with the eLearning system.

Table 3. Assessment questionnaire for evaluating learning effectiveness.

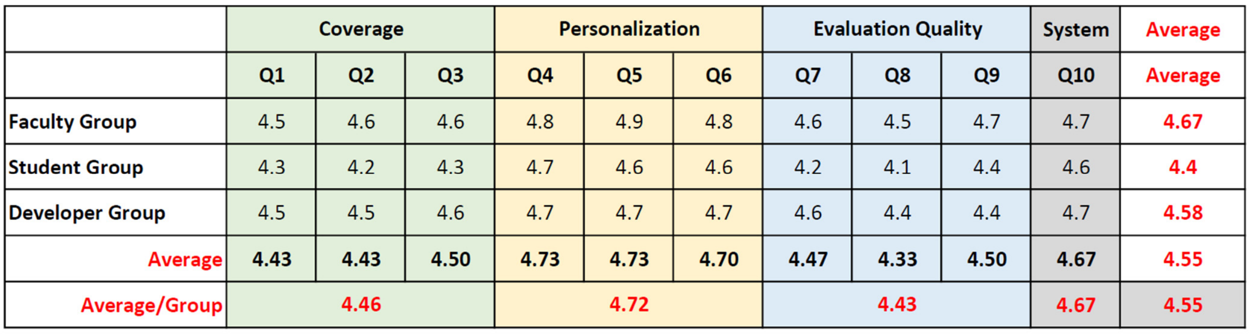
Categor	Questions
Coverage of PL Constructs	
Q1	Does the entire foundation content represent the essential features and constructs of the target programming language?
Q2	Does the entire example content align with the provided foundational content?
Q3	Does the entire exercise content align with the provided foundational content?
Personalization of Contents	
Q4	Is the entire foundational content tailored to your background and proficiency level?
Q5	Is the entire example content tailored to your background and domains of your interest?
Q6	Is the entire exercise content tailored to your background and proficiency level?
Evaluation of Exercise Submissions	
Q7	Is the overall assessment of your exercise submissions conducted accurately?
Q8	Is the overall explanatory feedback on assessments both reasonable and comprehensive?
Q9	Is the overall assessment of exercise submissions consistent and equitable for all students?
Usability of the System	
Q10	Revised: Does the system offer high usability through its user interface and functionality?

The experiment was conducted with a total of 73 participants. The student group consisted of 42 freshman and sophomore undergraduate students in Computer Science from our universities in California and a university in Korea. The faculty group consisted of six university professors who are currently teaching or have taught programming courses. The industry group consisted of 25 software developers and programmers from software development companies.

Each participant took at least one course, mostly Java or Python. Each participant engaged in all learning activities including attending online classes and completing the exercise problems. Participants allocated an average duration of approximately 3.5 months for course completion.

Consequently, the experiment required considerable time and effort from the participants. This requirement made the recruitment of additional subjects a practical challenge. Additionally, it imposed a significant logistical burden on the research team.

The questionnaire items were rated on a scale of 1 to 5, with '5' indicating the highest level of satisfaction and '1' indicating significant dissatisfaction. The results of experiments using the survey form are shown in Figure 17.



	Coverage			Personalization			Evaluation Quality			System	Average
	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Average
Faculty Group	4.5	4.6	4.6	4.8	4.9	4.8	4.6	4.5	4.7	4.7	4.67
Student Group	4.3	4.2	4.3	4.7	4.6	4.6	4.2	4.1	4.4	4.6	4.4
Developer Group	4.5	4.5	4.6	4.7	4.7	4.7	4.6	4.4	4.4	4.7	4.58
Average	4.43	4.43	4.50	4.73	4.73	4.70	4.47	4.33	4.50	4.67	4.55
Average/Group	4.46			4.72			4.43			4.67	4.55

Figure 17. Evaluation of learning effectiveness.

From the evaluation results, we make the following observations:

- The faculty group gave the highest overall rating of 4.67, equivalent to 93.4%, while the student group gave the lowest at 4.4, translating to 88%.
- The capability of personalizing learning contents was rated the highest at 4.72, equivalent to 94.4%. This suggests that the system's personalization feature, specifically ChatGPT, is highly effective.
- The evaluation quality of coding exercise submissions received the lowest rating of 4.43, which translates to 87%. Nonetheless, as academic authors, we fully understand the challenges associated with grading coding exercises and ensuring consistent evaluation across student submissions. Therefore, this rating of 87% should be regarded as commendably high.
- The cumulative average for learning effectiveness across the three groups is 4.55 out of 5.0, or 91%. This overall rating of 91% is much higher than the authors' initial anticipation. This suggests that GPT-based personalized eLearning is viable and holds potential for official implementation in university courses.

The experimental results are visualized in a line chart, shown in Figure 18.

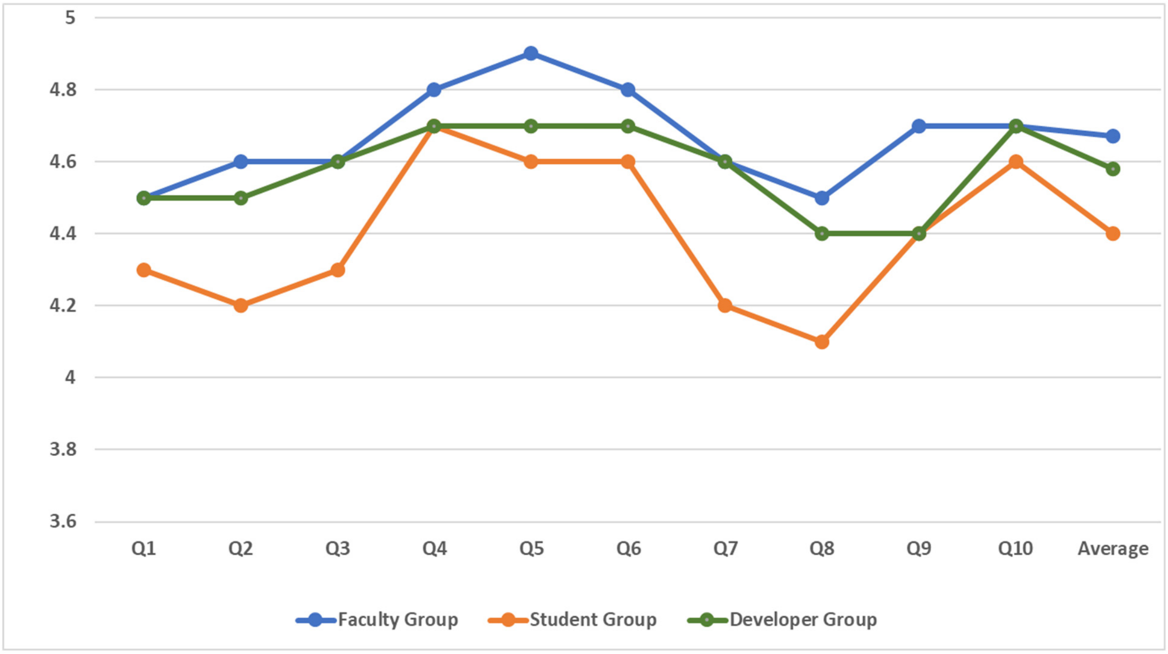


Figure 18. Evaluation of learning effectiveness in a line chart.

The chart visualizes the difference between the three groups. The faculty group, having professional experience and in-depth knowledge of programming languages, gave the highest overall evaluation. The student group, who began to acquire knowledge of the programming language and coding experience, gave the lowest overall evaluation. The developer group, having rich programming experience, gave a relatively high level of evaluation.

5.3.2. Evaluating the Cost of Developing the Learning Content

This criterion evaluates the cost associated with creating content to teach, i.e., the foundational content, examples, and coding exercises.

We posed a question to the six faculty members in the experimental group regarding the cost of creating learning content for a programming language. From their feedback, we could summarize the average efforts as the following:

- Average number of units in a programming language: 12 units/P.L.;
- Average number of topics per unit: 3 topics/unit;
- Average time spent creating foundational content per topic: 3.5 hours/topic;

- Average number of students in a programming class: 28 students/class;
- Average number of coding exercises per unit: 3.5;
- Average time spent creating *a personalized coding exercise*: 0.5 h;
- Average time to grade an exercise submission: 0.25 h.

The time required for operating a conventional eLearning system is the following:

- The total time for creating all of the foundational content for a language is computed as follows: *Time_for_Foundation_Content* = $12 \times 3 \times 3.5 = 126$ *h*
- The total time for creating personalized coding exercise problems for a language is computed as follows: *Time_for_Coding_Exercises* = $28 \times 12 \times 3.5 \times 0.5 = 588$ *h*
- The total time for evaluating exercise submissions for a language is computed as follows: *Time_for_Evaluating_Exercises* = $28 \times 12 \times 3.5 \times 0.25 = 294$ *h*

Hence, the total time for creating learning content for a programming language and operating the class with personalized exercise problems and grading exercise submissions is computed as (126 + 588 + 294) which amounts to just 1008 h.

Now, the time required for operating a ChatGPT-empowered eLearning system is the following:

- The total time for creating all the *foundational content* for a language is nearly zero, except for the effort to create a course profile as described in Section 3.4.
- *Time_for_Foundation_Content = 0 h*
  Time_for_Course_Profile = 2 h
- The total time for creating *personalized coding exercise* problems for a language is computed as follows:
  Time_for_Coding_Exercises = 0 h
- The total time for evaluating exercise submissions for a language is computed as follows:
  Time_for_Evaluating_Exercises = 0 h
- Hence, the total time for creating learning content for a programming language and operating the class by utilizing ChatGPT is computed as (0 + 2 + 0 + 0) which amounts to just 2 h.

The difference in total effort for generating the learning content and evaluating exercise submissions between conventional eLearning and GPT-based eLearning is substantial: 1008 h versus 2 h. This significant disparity serves as a compelling indicator of the feasibility and potential of GPT-based eLearning.

6. Concluding Remarks

While eLearning for programming languages offers benefits such as accessibility, temporal flexibility, and content reusability, it also exposes limitations of delivering a pre-fixed one-size-fits-all content to all students, engendering high costs for developing course contents and managing eLearning sessions including grading student submissions.

Our research was motivated by the need to construct an eLearning system capable of delivering personalized, student-centric content, automating the generation of coursework content for all programming languages, and eliminating the need for instructor involvement in the management of eLearning sessions.

Our strategy and software methodology was to build the system by applying a suite of advanced software technologies. (1) We utilize GPT to dynamically generate all course components, (2) apply GPT prompt engineering to personalize course content for each individual student, and (3) apply the closed loop of autonomous computing to automate eLearning sessions.

The research results encompass the design of an eLearning framework covering all programming languages, a fully functional Python-based implementation, seamless integration with ChatGPT for dynamic content generation, a high degree of content personalization, and the elimination of manual effort required for managing eLearning sessions.

The PoC implementation of this eLearning system was deployed on Amazon AWS for experiments. The experiment was conducted with faculty members, students, and developers and demonstrated a learning effectiveness of 91%. Moreover, the effort required to operate eLearning for programming languages was reduced by 99.8%.

The usefulness of our proposed eLearning system may be summarized as follows:

- GPT models can generate comprehensive course content for programming languages. The dynamically generated course contents for a language are comparable to the course contents created by a highly professional university instructor with years of experience in teaching programming languages.
- The content generated by GPT is highly responsive to the prompts provided by client applications. Consequently, crafting precise GPT prompts plays a crucial role in personalizing course content.
- The control loop inherent in autonomous computing finds practical application in automating the administration of eLearning systems. By seamlessly integrating autonomous control with GPT's grading capabilities for student exercise submissions, it becomes feasible to entirely replace the need for human instructors.
- A single development of this eLearning system can be used to teach an array of programming languages. This is in contrast to the conventional eLearning system where instructors have to create course content for every programming language.

The only limitation we observed at this time was the occasional latency ranging from 3 to 8 s in the generation of course content and the grading of student submissions. The latency is attributable to the network-based interaction with the GPT server, and the processing time incurred during peak usage of the GPT server.

Through this research, we observe several ethical issues in using GPT in education. These include the potential for bias and fairness concerns, as GPT models inherit and amplify biases present in their training data. Accountability is essential, with a need for clear guidelines on who is responsible for evaluating submitted coding exercises. Additionally, there is a risk of over-dependence on GPT technology, which might impact the development of critical thinking and creativity of programming language instructors.

We believe that our personalized eLearning approach for programming sets the stage for configuring cost-effective, personalized programming education. More specifically, we propose to utilize this eLearning system in programming classes at the college level. The system can also be used in various existing eLearning systems of programming languages for the public.

7. Future Works

Our plan for extending this research comprises three aspects: (1) supporting more types of learning schemes, (2) technical advancement of this eLearning system with machine learning, and (3) comprehensive quality assessments involving a significantly larger number of instructors and students.

For the learning schemes, the current eLearning system supports learning schemes of studying foundational materials and examples, solving exercise problems, and reviewing the graded exercise solutions. We plan to add two more useful learning schemes: generating pop-quiz problems and generating personalized programming laboratories.

For the technical advancement, our plan involves training a multi-variable multi-class deep learning model to generate student- and context-specific feature values for foundational concepts, examples, and exercise problems. We anticipate that personalizing GPT prompts with feature values from this deep learning model will outperform our current rule-based approach to prompt personalization.

Regarding the quality-in-use assessment of the system, we intend to enlist a larger cohort of students and instructors from a minimum of 10 universities. This endeavor is expected to demand significant effort and time, based on our past experiences. Additionally, we plan to formally offer a programming course at the freshman or sophomore levels in computer science at our universities. At the end of the semester, we will gather valuable evaluations from the students and instructors and compare their learning effectiveness and course preparation costs with conventional in-classroom teaching.

Author Contributions: Conceptualization, J.J. and M.K., methodology, J.J.; software, J.J.; formal analysis, J.J. and M.K.; investigation, J.J. and M.K.; resources, M.K.; data curation, J.J.; writing—original draft preparation, J.J.; writing—review and editing, J.J. and M.K.; visualization, M.K.; supervision, M.K.; project administration, J.J. and M.K.; funding acquisition, M.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ministry of Education. *Digital Talent Cultivation Plan*; South Korean Government: Seoul, Republic of Korea, 2022.
2. Mustakerov, I.; Borissova, D. A Framework for Development of e-learning System for computer programming: Application in the C programming Language. *J. e-Learn. Knowl. Soc.* **2017**, *13*, 89–101.
3. Bashir, G.M.M.; Hoque, A.S.M.L. An effective learning and teaching model for programming languages. *J. Comput. Educ.* **2016**, *3*, 413–437. [CrossRef]
4. Rehberger, S.; Frank, T.; Vogel-Heuser, B. Benefit of e-learning teaching C-programming and software engineering in a very large mechanical engineering beginners class. In Proceedings of the 2013 IEEE Global Engineering Education Conference (EDUCON), Berlin, Germany, 13–15 March 2013; pp. 1055–1061.
5. Wang, J.; Chen, L.; Zhou, W. Design and Implementation of an Internet-Based Platform for C Language Learning. In Proceedings of the International Conference on Web-Based Learning 2008, Jinhua, China, 20–22 August 2008; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2018; Volume 5145, pp. 187–195.
6. Weston, M.; Sun, H.; Herman, G.L.; Benotman, H.; Alawini, A. Echelon: An AI Tool for Clustering Student-Written SQL Queries. In Proceedings of the 2021 IEEE Frontiers in Education Conference (FIE), Lincoln, NE, USA, 13–16 October 2021; pp. 1–8.
7. Harley, E.; Harley, Z. E-learning and E-assessment for a Computer Programming Course. In Proceedings of the Third International Conference on Education and New Learning Technologies, Barcelona, Spain, 4–6 July 2011; pp. 2074–2080.
8. Dobesova, Z. E-learning for visual programming language. In Proceedings of the 2014 IEEE 12th International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, Slovakia, 4–5 December 2014; pp. 103–108.
9. Hasany, S.N. E-Learning Student Assistance Model for the First Computer Programming Course. *Int. J. Integr. Technol. Educ.* **2017**, *6*, 1–7. [CrossRef]
10. Estévez, J.; Garate, G.; Grana, M. Gentle Introduction to Artificial Intelligence for High-School Students Using Scratch. *IEEE Access* **2019**, *7*, 179027–179036. [CrossRef]
11. Chen, E.; Huang, R.; Chen, H.; Tseng, Y.; Li, L. GPTutor: A ChatGPT-powered programming tool for code explanation. In Proceedings of the 2023 International Conference on Artificial Intelligence in Education, Tokyo, Japan, 3–7 July 2023.
12. Hsiao, I.; Sosnovsky, S.; Brusilovsky, P. Guiding students to the right questions: Adaptive navigation support in an E-Learning system for Java programming. *J. Comput. Assist. Learn.* **2010**, *26*, 270–283. [CrossRef]
13. Law, K.M.Y.; Lee, V.C.S.; Yu, Y.T. Learning motivation in e-learning facilitated computer programming courses. *Comput. Educ.* **2010**, *55*, 218–228. [CrossRef]
14. Chrysafiadi, K.; Virvou, M. PeRSIVA: An empirical evaluation method of a student model of an intelligent e-learning environment for computer programming. *Comput. Educ.* **2013**, *68*, 322–333. [CrossRef]
15. Yusupova, S.B.; Sultanov, O.R.; Baltayev, R.S.; Bekchanov, F.A. The Advantage of Using e-Learning in Teaching Students Programming Languages. In Proceedings of the IEEE International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON), Yekaterinburg, Russia, 11–13 November 2022; pp. 1910–1913.
16. Yilmaz, R.; Yilmaz, F.G.K. The effect of generative artificial intelligence (AI)-based tool use on students' computational thinking skills, programming self-efficacy and motivation. *Comput. Educ. Artif. Intell.* **2023**, *4*, 100147. [CrossRef]
17. Hosseini, M.; Gao, C.A.; Liebovitz, D.M.; Carvalho, A.M.; Ahmad, F.S.; Luo, Y.; MacDonald, N.; Holmes, K.L.; Kho, A. An exploratory survey about using ChatGPT in education, healthcare, and research. *PLoS ONE* **2023**, *18*, e0292216. [CrossRef]
18. Oguz, F.E.; Ekersular, M.N.; Sunnetci, K.M.; Alkan, A. Can Chat GPT be Utilized in Scientific and Undergraduate Studies? *Ann. Biomed. Eng.* 2023. *advance online publication*. [CrossRef]
19. Choi, J.H.; Hickman, K.E.; Monahan, A.; Schwarcz, D.B. ChatGPT Goes to Law School. *J. Leg. Educ.* **2023**. [CrossRef]
20. Mhlanga, D. ChatGPT in Education: Exploring Opportunities for Emerging Economies to Improve Education with ChatGPT. *SSRN Electr. J.* **2023**. [CrossRef]

21. Firat, M. Integrating AI Applications into Learning Management Systems to Enhance e-Learning. *Instr. Technol. Lifelong Learn.* **2023**, *4*, 1–14. [CrossRef]
22. Mhlanga, D. *Open AI in Education, the Responsible and Ethical Use of ChatGPT towards Lifelong Learning*; FinTech and Artificial Intelligence for Sustainable Development. Sustainable Development Goals Series; Palgrave Macmillan: Cham, Switzerland, 2023.
23. Biswas, S. Role of Chat GPT in Education. *J. ENT Surg. Res.* **2023**, *1*, 1–3.
24. Grassini, S. Shaping the Future of Education: Exploring the Potential and Consequences of AI and ChatGPT in Educational Settings. *Educ. Sci.* **2023**, *13*, 692. [CrossRef]
25. Kalla, D.; Smith, N. Study and Analysis of Chat GPT and its Impact on Different Fields of Stud. *Int. J. Innov. Sci. Res. Technol.* **2023**, *8*.
26. Michel-Villarreal, R.; Vilalta-Perdomo, E.; Salinas-Navarro, D.E.; Thierry-Aguilera, R.; Gerardou, F.S. Challenges and Opportunities of Generative AI for Higher Education as Explained by ChatGPT. *Educ. Sci.* **2023**, *13*, 856. [CrossRef]
27. Su, J.; Yang, W. Unlocking the Power of ChatGPT: A Framework for Applying Generative AI in Education. *ECNU Rev. Educ.* **2023**, *6*, 1–12.
28. Tlili, A.; Shehata, B.; Adarkwah, M.; Bozkurt, A.; Hickey, D.; Huang, R.; Agyemang, B. What if the devil is my guardian angel: ChatGPT as a case study of using chatbots in education. *Smart Learn. Environ.* **2023**, *10*, 15. [CrossRef]
29. Lo, C.K. What Is the Impact of ChatGPT on Education? A Rapid Review of the Literature. *Educ. Sci.* **2023**, *13*, 410. [CrossRef]
30. Wu, D.; Lu, J.; Zhang, G. A Fuzzy Tree Matching-Based Personalized E-Learning Recommender System. *IEEE Trans. Fuzzy Syst.* **2015**, *23*, 2412–2426. [CrossRef]
31. Troussas, C.; Krouska, A.; Sgouropoulou, C. A Novel Teaching Strategy Through Adaptive Learning Activities for Computer Programming. *IEEE Trans. Educ.* **2021**, *64*, 103–109. [CrossRef]
32. Augstein, M.; Paramythis, A. Activity sequence modelling and dynamic clustering for personalized e-learning. *User Model. User-Adapt. Interact.* **2011**, *21*, 51–97.
33. Murtaza, M.; Ahmed, Y.; Shamsi, J.; Sherwani, F.; Usman, M. AI-Based Personalized E-Learning Systems: Issues, Challenges, and Solutions. *IEEE Access* **2022**, *10*, 81323–81342. [CrossRef]
34. Gaeta, M.; Miranda, S.; Orciuoli, F.J.; Paolozzi, S.; Poce, A. An Approach to Personalized e-Learning. *J. Educ. Inform. Cyber.* **2013**, *11*, 15–21.
35. Rani, M.; Vyas, O. An Ontology-based Adaptive Personalized E-learning System, Assisted by Software Agents on Cloud Storage. *J. Knowl.-Based Syst.* **2015**, *90*, 33–48. [CrossRef]
36. Zakrzewska, D. Cluster Analysis in Personalized E-Learning Systems. *Intell. Syst. Knowl. Manag.* **2009**, *252*, 229–250.
37. Huang, M.; Huang, H.; Chen, M. Constructing a personalized e-learning system based on genetic algorithm and case-based reasoning approach. *J. Expert Syst. Appl.* **2007**, *33*, 551–564. [CrossRef]
38. Baylari, A.; Montazer, G.A. Design a personalized e-learning system based on item response theory and artificial neural network approach. *Expert Syst. Appl.* **2009**, *36*, 8013–8021. [CrossRef]
39. Cakula, S.; Sedleniece, M. Development of a Personalized e-learning Model Using Methods of Ontology. *Procedia Comput. Sci.* **2013**, *26*, 113–120. [CrossRef]
40. Milicevic, A.; Vesin, B.; Ivanovic, M.; Budimac, Z. E-Learning personalization based on hybrid recommendation strategy and learning style identification. *Comput. Educ.* **2011**, *56*, 885–899. [CrossRef]
41. Kausar, S.; Xu, H.; Hussain, I.; Wenhao, Z.; Zahid, M. Integration of Data Mining Clustering Approach in the Personalized E-Learning System. *IEEE Access* **2018**, *6*, 72724–72734. [CrossRef]
42. Alhawiti, M.; Abdelhamid, Y. A Personalized e-Learning Framework. *J. Educ. e-Learn. Res.* **2019**, *4*, 15–21. [CrossRef]
43. Chen, C.; Lee, H.; Chen, Y. Personalized E-Learning System Using Item Response Theory. *Comput. Educ.* **2005**, *44*, 237–255. [CrossRef]
44. Chen, C.; Huang, T.; Li, T.; Huang, C. Personalized E-Learning System with Self-Regulated Learning Assisted Mechanisms for Promoting Learning Performance. In Proceedings of the Seventh IEEE International Conference on Advanced Learning Technologies, Niigata, Japan, 18–20 July 2007; pp. 637–638.
45. Ciloglugil, B.; Inceoğlu, M. User Modeling for Adaptive E-Learning Systems. In *Computational Science and Its Applications (ICCSA 2012)*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7335, pp. 550–561.
46. Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*; Addison-Wesley: Boston, MA, USA, 1994.
47. *ISO 9126*; Information Technology—Software Product Evaluation—Quality Characteristics and Guidelines for Their Use. International Organization for Standardization: Geneva, Switzerland, 1991.
48. Smith, J.; Williams, R. *Software Engineering Principles: From Requirements to Deployment*; Chapter 7, Use Case Diagrams and Software Agents; Academic Press: Cambridge, MA, USA, 2018.
49. Liu, P.; Yuan, W.; Fu, J.; Jiang, Z.; Hayashi, H.; Neubig, G. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.* **2023**, *55*, 1–35. [CrossRef]